M20 PERSONAL COMPUTER

Linguaggio PASCAL

Manuale Generale

olivetti

INDICE

PAGINA

1. INTRODUZIONE AL LINGUAGGIO PASCAL

SOMMARIO

Lo scopo di questo capitolo e' di riassumere le caratteristiche principali del linguaggio Pascal, le sue origini e la sua collocazione tra altri linguaggi.

Viene prima presentata una breve descrizione del linguaggio nella storia di altri linguaggi di programmazione; poi sono riassunte le principali caratteristiche del linguaggio Pascal standard. Vengono anche fornite alcune delle principali estensioni del Pascal, descritte dettagliatamente nei capitoli successivi del manuale.

INDICE

NOTE STORICHE E COLLOCAZIONE DEL LINGUAGGIO

Il linguaggio Pascal e' stato ideato da Wirth e Jensen nel 1968 come linguaggio con scopo generale fortemente orientato verso l'utente, e si e' rivelato un evento chiave nella storia dei linguaggi di programmazione.

Si possono delineare alcuni orientamenti fondamentali nella storia di tali linguaggi:

- l'allocazione dei dati e' diventata sempre piu' un compito affidato al sistema invece che all'utente;
- l'insieme delle istruzioni e la struttura del linguaggio in generale si sono sempre piu' orientati verso modelli umani piuttosto che verso caratteristiche della macchina;
- la rappresentazione dei dati, un tempo legata alla configurazione hardware, e' sempre piu' definibile dall'utente per mezzo di tipi astratti di dati: i dati vengono considerati piu' dal punto di vista del loro valore logico che della loro rappresentazione fisica in memoria;
- il supporto runtime per l'esecuzione dei programmi e' diventata sempre piu' potente rendendo i linguaggi piu' indipendenti dalle caratteristiche del sistema operativo.

Questa evoluzione puo' essere notata attraverso un breve sommario dei principali linguaggi di programmazione:

LINGUAGGI ASSEMBLER	L'allocazione dei dati e' compito del programmatore; i tipi dei dati sono limitati; gli operatori appartengono a tipi definiti; non ci sono strutture di controllo o di dati;
FORTRAN	L'allocazione dei dati e' statica; i tipi dei dati sono limitati; anche le strutture di dati e di controllo sono limitate;
COBOL	L'allocazione di dati e' statica; i tipi di dati e le strutture di controllo sono limitati; numerose le strutture di dati; limitato il supporto runtime;
PL1	L'allocazione di dati e' dinamica; limitati i tipi di dati; buone le strutture di controllo; numerose le strutture di dati ed esteso il supporto runtime;
PASCAL	L'allocazione dei dati e' dinamica; esteso l'insieme di tipi di dati definibili dall'utente, delle strutture di dati e di controllo; esteso il supporto runtime;

PASCAL MICROSOFT	Estensione del Pascal: presenta prestazioni e funzionalita' di package e la possibilita' di compilazioni separate;
ADA	Estensione del Pascal: presenta prestazioni e funzionalita' di package, la possibilita' di compilazioni separate, programmazione concorrente, facile gestione delle eccezioni, strutture astratte di dati.

PRINCIPALI CARATTERISTICHE DEL LINGUAGGIO

Il linguaggio Pascal dispone di un insieme di strumenti per la strutturazione dei programmi basato sul concetto di programmazione strutturata, comprese le forme piu' note di strutture condizionali e iterative.
Ha due tipi di routine: procedure e funzioni che possono essere utilizzate ricorsivamente (il concetto di ricorsivita' viene spiegato piu' in dettaglio oltre in tale capitolo) ed accettano differenti tipi di parametri: valore, di riferimento e di routine.
Tuttavia la principale caratteristica del linguaggio e' la capacita' di definire tipi astratti di dati enumerando i loro possibili valori. Inoltre i tipi strutturati possono essere creati per mezzo di uno schema di base di strutturazione: array, record, file e set; si possono infine costruire, tramite puntatori, strutture dinamiche di dati di qualunque configurazione.

Nonostante le sue capacita', il linguaggio e' caratterizzato da una grande concisione e chiarezza di descrizione, che ne facilitano l'apprendimento, la scrittura e la lettura.

Esistono inoltre delle versioni estese del Pascal (quali ad esempio il Pascal M20) che supportano compilazioni separate di programmi con condivisione di risorse, e capacita' di racchiudere le relative operazioni all'interno di tipi strutturati (moduli). Si ha anche un certo numero di procedure e funzioni predefinite che sono inerenti alla costruzione del software di sistema .

LA PROGRAMMAZIONE TRAMITE MACCHINE ASTRATTE

Lo scopo di questo paragrafo e' di illustrare la filosofia del linguaggio ed alcuni aspetti metodologici sulla programmazione in Pascal; inoltre viene trattata la struttura gerarchica di un programma.

Supponendo di avere un problema e una macchina capace di risolverlo, si vuole cercare di utilizzare tale macchina.

Per esempio, se si considera una macchina in grado di giocare a scacchi, essa puo' essere usata per una partita. Mentre, se si ha una macchina incapace di giocare, ma capace di calcolare una mossa durante una partita di scacchi, le si possono dare appropriate istruzioni per giocare una partita:

```
RIPETI FINCHE' PARTITA-TERMINATA:
     ESAMINA LA MOSSA AVVERSARIA,
       CALCOLA LA PROSSIMA MOSSA,
      EFFETTUA LA PROSSIMA MOSSA.
```

Cosi', dal punto di vista del giocatore, le due macchine menzionate possono essere considerate identiche.

Piu' in generale si puo' dire che, avendo bisogno di una macchina che non esiste, essa puo' essere costruita con l'ausilio di macchine piu' elementari e di un linguaggio che esprime il modo in cui esse devono essere connesse per eseguire le operazioni richieste.

Ogni componente della macchina puo' naturalmente essere prodotto in questo modo, per mezzo di macchine di piu' basso livello.

Questo processo termina quando sono raggiunte le capacita' delle macchine che sono a disposizione.

L'approccio sopra descritto costituisce il metodo fondamentale per la costruzione di un programma in Pascal: esistono alcune caratteristiche del linguaggio che costituiscono le "macchine" elementari, ovvero le risorse predefinite, ed altre che consentono la definizione di risorse ad alto livello (per esempio procedure e funzioni).

Questo metodo e' anche noto come programmazione top-down, e consiste nell'analizzare il problema da risolvere in termini di sottoproblemi di minore complessita'.

In seguito si descrive piu' in dettaglio la struttura gerarchica di un programma Pascal, anche da un punto di vista sintattico.

LA STRUTTURA GERARCHICA DI UN PROGRAMMA

Come affermato precedentemente, un programma Pascal e' costituito da risorse, sia fornite dal linguaggio stesso sia definite dal programmatore.

Da un punto di vista strutturale il programma e' diviso in due parti: la prima descrive le risorse ad alto livello, la seconda comprende le istruzioni che implementano le risorse nel modo richiesto.

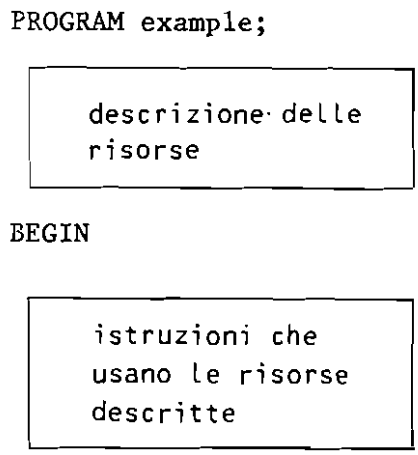


Fig. 1-1 Schema di un Programma Pascal

Con il termine "risorsa" si intende indicare ogni elemento utilizzato dalle istruzioni di programma, per esempio variabili, procedure e funzioni. Ciascun tipo di risorsa e' trattato piu' avanti: tra questi hanno un ruolo fondamentale le procedure e le funzioni, poiche' costituiscono le "istruzioni" ad alto livello su cui si basa la sezione istruzioni.

Poiche' procedure e funzioni possono essere ulteriormente specificate con l'uso di risorse di piu' basso livello, la loro struttura e' piuttosto simile a quella dei programmi.

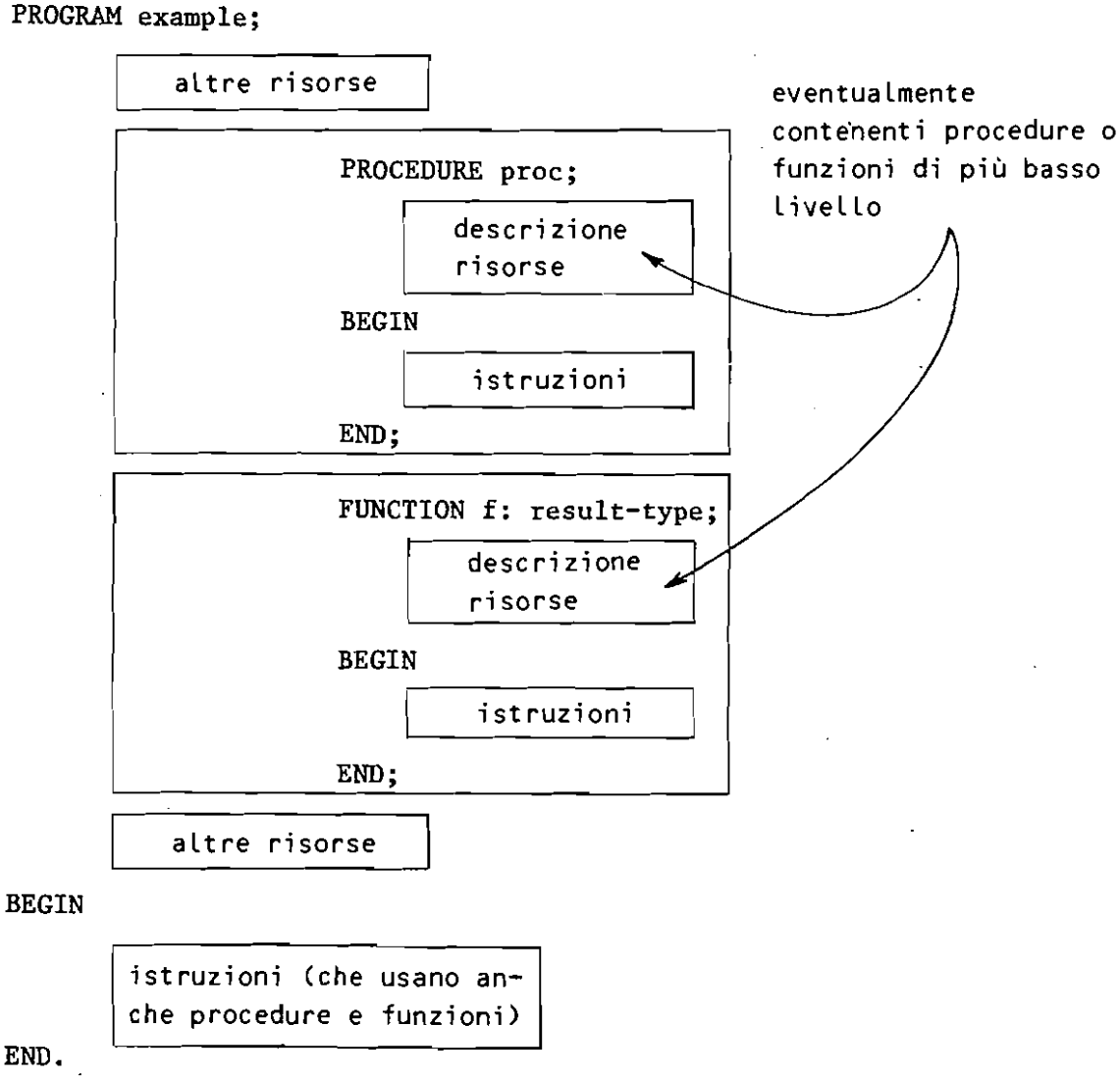


Fig. 1-2 Risorse Nidificate

Ogni procedura o funzione contiene la descrizione delle sue risorse di livello inferiore ed una parte di istruzioni per il loro uso.

Si noti che tra queste risorse di livello inferiore ci possono essere altre procedure e funzioni, strutturate, a loro volta, nello stesso modo.

La differenza tra procedure e funzioni e' trattata piu' avanti in questo manuale; per il momento si puo' loro assegnare il nome comune di "routine".
Nella programmazione in Pascal e' fondamentale la "nidificazione" di risorse; nella parte che segue vengono chiarite alcune regole su tale argomento.

SEQUENZA DI SCRITTURA E DI COMPILAZIONE

Quando si scrive un programma Pascal in termini di gerarchia di risorse, si inizia da quella piu' ad alto livello: cio' implica come primo passo la definizione della parte istruzioni che si riferisce alle routine, ai dati e alle altre risorse che vengono definite in dettaglio successivamente.

Questo metodo top-down di scrittura e' suggerito dal linguaggio stesso.

Il compilatore Pascal, d'altra parte, scandisce il programma solo una volta e per tale motivo e' necessario che le risorse di livello inferiore siano specificate prima che si faccia riferimento ad esse. Quindi la sequenza di scrittura di un programma non coincide con quella fornita al compilatore: scrivendo, prima si definiscono le istruzioni del programma principale, in seguito le entita' cui nel programma principale si e' fatto riferimento:

```
PROGRAM P;

   BEGIN
     (*istruzioni del programma principale
      con riferimento a procedure Q e R*)
   END.
```

Poi si definiscono in dettaglio le risorse di secondo livello, in questo caso le routine Q e R:

```
PROGRAM P;

  PROCEDURE Q;
     BEGIN
       (*istruzioni della procedura Q*)
     END;

  PROCEDURE R;
     BEGIN
       (*istruzioni della procedura R*)
     END;

  BEGIN
    (*istruzioni del programma principale
     con riferimento a Q ed R*)
  END.
```

Ogni routine e' costruita con lo stesso criterio, cioe' specificando prima la definizione delle risorse e poi la parte istruzioni:

```
PROGRAM P;

  PROCEDURE Q;
     (*descrizione risorse della procedura Q*)
    BEGIN
     (*istruzioni della procedura Q*)
    END;

  PROCEDURE R;
     (*descrizione risorse della procedura R*)
    BEGIN
     (*istruzioni della procedura R*)
    END;

  BEGIN
   (*istruzioni del programma principale*)
  END.
```

REGOLE DI VISIBILITA' E LORO ENUNCIATO

Vengono ora espresse alcune regole generali chiamate "regole di visibilita'" (scope rules), relative a tutti i tipi di risorse, incluse quelle predefinite cioe' fornite dal linguaggio stesso.

Il raffinamento in modo top-down di un programma Pascal richiede di avere una corretta visibilita' della nidificazione delle risorse.

Ogni risorsa si deve considerare come assegnata ad una specifica parte del programma. Per esempio nello scrivere una routine per la ricerca di un valore in un array, si deve inserire al suo interno una variabile indice per scandire l'array; questa variabile costituisce una risorsa privata della routine: e' scorretto e inutile usarla all'esterno.

La disciplina relativa all'uso delle risorse e' governata da poche e semplici regole di visibilita'.

Poiche' tali regole sono basate sul livello di nidificazione delle risorse, e' opportuno introdurre una semplice terminologia relativa a questo argomento.

Si consideri l'esempio in cui un programma P sia scritto in termini di due risorse: una variabile A ed una procedura Q.

```
PROGRAM P;
   VAR A:    (* descrizione di A *)
   PROCEDURE Q;
       (* risorse di Q *)
   BEGIN
       (* istruzioni di Q *)
   END;
BEGIN
     (* istruzioni del programma principale *)
END.
```

Fig. 1-3 Risorse del Programma Principale

Poiche' A e Q sono scritte specificatamente per P, si dice che esse sono risorse "figlie" di P, e che P e' "padre" di A e Q; inoltre A e Q sono detti "fratelli".

La risorsa Q e' una routine, e come tale puo' contenere risorse proprie, ad esempio due routine R e S:

```
PROGRAM P;
   VAR A:  (* descrizione di A *);
   PROCEDURE Q;
      PROCEDURE R;
         (* risorse di R *)
      BEGIN
         (* istruzioni di R *)
      END;
      PROCEDURE S;
         (* risorse di S *)
      BEGIN
         (* istruzioni di S *)
      END;
   BEGIN
       (* istruzioni di Q *)
   END;
BEGIN
     (* istruzioni di P *)
END.
```

Fig. 1-4 Nidificazione di Risorse

Esistono quindi le seguenti relazioni: R ed S sono figlie di Q e P e' un antenato di R ed S, in quanto padre del loro padre Q.

Si noti il particolare metodo di rientranza usato per le righe: esso e' un modo utile e piuttosto diffuso per evidenziare la nidificazione di risorse nel testo di un programma.

Quanto detto e' sufficiente per formulare le seguenti tre regole di visibilita':

- regola gerarchica
- regola sequenziale
- regola dell'ombra

Ciascuna di queste regole specifica alcune restrizioni sulla parte di testo del programma in cui una risorsa puo' essere utilizzata, tale parte di testo e' detta "area di visibilita'" della risorsa.

La Regola Gerarchica

La regola gerarchica stabilisce che una risorsa puo' essere utilizzata solo all'interno della sua risorsa-padre (compresi i discendenti del padre), mai in una parte di testo esterna ad essa.

In riferimento all'esempio precedente, la regola gerarchica stabilisce che R ed S non possono essere usate al di fuori della procedura Q.

La Regola Sequenziale

La regola sequenziale stabilisce che una risorsa non puo' essere utilizzata in una parte di testo che precede la sua stessa definizione.

Nell'esempio precedente, la procedura R puo' essere utilizzata all'interno di S, mentre S non puo' apparire all'interno di R.
La variabile A puo' essere usata in Q, e percio' anche in R e in S.

Tuttavia c'e' un' eccezione a tale regola, riguardante i tipi puntatori che vengono trattati in seguito.

La Regola dell' Ombra

Le relazioni tra le risorse sono state precedentemente descritte in analogia con le relazioni familiari; continuando con l'analogia si puo' considerare il problema dei nomi in una famiglia. Comunemente si evita di dare lo stesso nome a fratelli, perche' questo puo' provocare confusione nel chiamarli.

D'altra parte il problema non si pone se a due persone appartenenti a famiglie diverse viene dato lo stesso nome, poiche' il contesto e' sufficiente a distinguerle.

Cio' si verifica anche per gli identificatori di risorse del Pascal, ed e' con la regola dell'ombra che si elimina ogni possibile ambiguita'.

Si vuole aggiungere ora all'esempio precedente una variabile "A" all'interno della procedura R:

```
PROGRAM P;
   VAR A:   (* descrizione di A *)
   PROCEDURE Q;
      PROCEDURE R;
         VAR A: (* altra descrizione di A diversa dalla precedente *)
      BEGIN
         (* istruzioni di R *)
      END;
      PROCEDURE S;
         (* risorse di S *)
      BEGIN
         (* istruzioni di S *)
      END;
   BEGIN
      (* istruzioni di Q *)
   END;
BEGIN
   (* istruzioni di P *)
END.
```

Fig. 1-5 La Regola dell'Ombra

Queste due risorse (le variabili A) sono completamente distinte anche se hanno lo stesso nome.

Normalmente la variabile globale A (le risorse figlie del programma principale sono chiamate globali), e'accessibile all'interno di R; pero' la presenza di un' altra risorsa con lo stesso nome all'interno di R "mette in ombra" la variabile globale nella parte di R che segue la nuova definizione di A. Si tratta comunque di una condizione relativa solo ad R: la variabile globale A e' quindi accessibile in S e nelle parti rimanenti di Q e P.

In altre parole la regola dell'ombra stabilisce che se in una routine compare una risorsa piu' interna avente lo stesso nome di una globale, quest'ultima non puo' essere usata in quel contesto.

ROUTINE RICORSIVE E RISORSE PREDEFINITE

Molti problemi di programmazione sono facilmente risolvibili con l'impiego di algoritmi ricorsivi. Pur non esaminandoli ora da un punto di vista matematico (li si puo' studiare, se necessario, su pubblicazioni di informatica di base), si puo' pero' osservare che ogni routine del Pascal (procedura o funzione) puo' essere ricorsiva, cioe' contenere al proprio interno, nella parte istruzioni, una chiamata a se stessa.

Se le regole di visibilita' sono rispettate, non c'e' alcuna limitazione alle chiamate di routine; percio' una routine puo' eventualmente chiamare anche le routine del proprio padre (se stessa compresa) ed ogni altra routine antenata.

C'e' tuttavia il problema della mutua ricorsivita', che si presenta quando due routine dello stesso livello si richiamano l'un l'altra ricorsivamente, per cui una di esse risulta chiamata prima ancora di essere definita. Questo problema viene risolto per mezzo della direttiva "FORWARD":

```
PROGRAM P;
   PROCEDURE Q;
      FORWARD;
   PROCEDURE S;
      (* risorse di S *)
   BEGIN
      (* istruzioni di S *)
      (* chiamata di Q *)
   END;
   PROCEDURE Q;
      (* risorse di Q *)
   BEGIN
      (* istruzioni di Q *)
      (* chiamata di S *)
   END;
BEGIN
   (* istruzioni di P *)
   (* chiamata di Q o S o di entrambe *)
END.
```

Fig. 1-6 Routine Mutuamente Ricorsive

La procedura Q e' citata per la prima volta in una definizione dove compare la direttiva "FORWARD" anziche' la descrizione per esteso di Q: cio' consente di compilare correttamente la chiamata di Q che compare all'interno di S. La descrizione per esteso di Q e' compilata dopo quella di S.

Si accenna ora brevemente alle risorse utilizzabili in un programma.

Il linguaggio Pascal fornisce, gia' ad un livello elementare, un certo numero di risorse predefinite.

Si consideri l'esempio:

```
PROGRAM P;
   VAR I: INTEGER;
BEGIN
   WRITE (I)
END.
```

Fig. 1-7 Risorse Predefinite

WRITE costituisce una procedura predefinita (descritta nei dettagli piu' avanti), cioe' e' una risorsa che si puo' usare nel programma senza una sua precedente definizione. Lo stesso si puo' dire dell'identificatore INTEGER usato nella definizione della variabile "I"; INTEGER e' il nome della risorsa predefinita che specifica il tipo di valore che "I" puo' assumere.

L'uso di risorse predefinite risulta compatibile con le regole di visibilita', se si immagina un altro ambiente che comprende il programma stesso. Tale ambiente e' il sistema Pascal, che contiene la definizione di risorse quali WRITE o INTEGER.

Tutte queste sono risorse dello stesso livello e precedono il programma, in modo da poter essere utilizzate in tutto il testo, a meno di essere messe in ombra da risorse piu' interne aventi lo stesso nome.

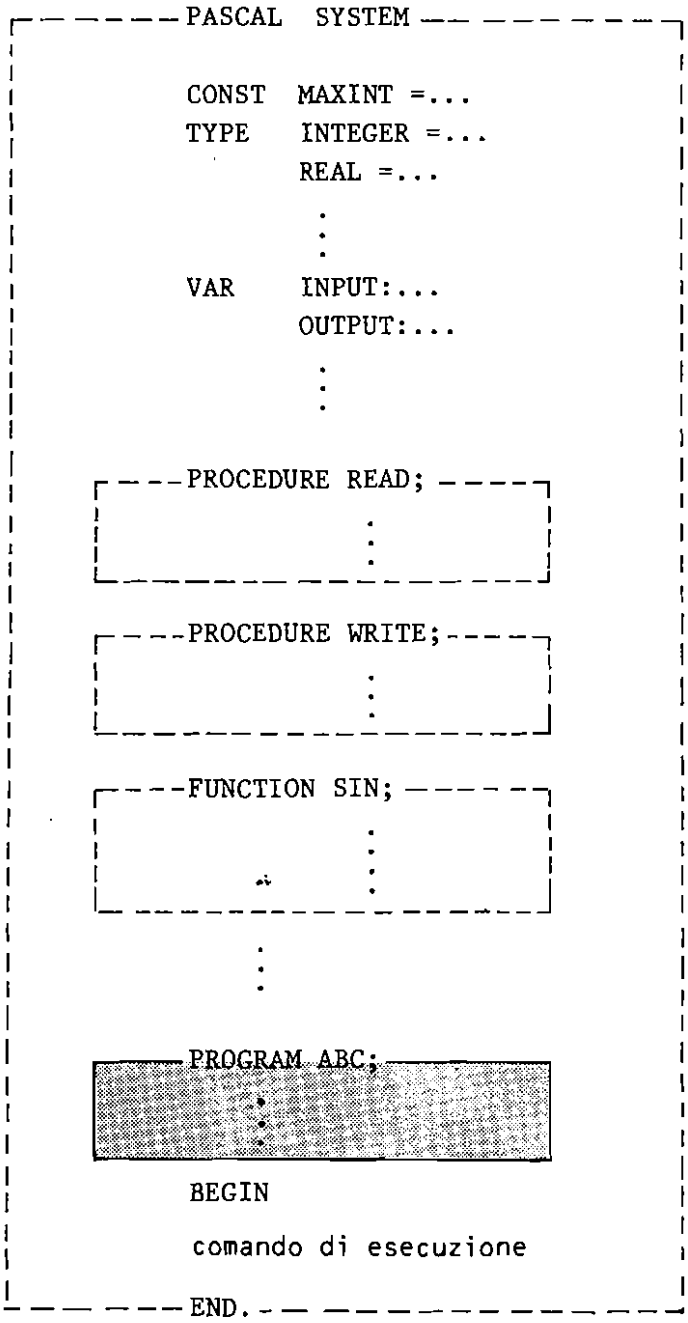


Fig. 1-8 Risorse ed Istruzioni Predefinite

Anche se consentito, mettere in ombra tali risorse puo' comportare qualche problema ed e' consigliabile evitare l'applicazione di tale regola.

IL CONCETTO DI TIPO

Si considera ora un'importante caratteristica di tutti i linguaggi ad alto livello: il concetto di tipo di dato.

Viene inoltre introdotta una classificazione approssimativa dei tipi, distinguendoli in semplici e strutturati; e dei piu' importanti strumenti linguistici per la loro definizione: gli "enumeratori" ed i "costruttori".

E' noto che i programmi non operano direttamente sul mondo reale, ma su sue rappresentazioni simboliche. Come esempio si puo' considerare un programma di gioco degli scacchi, che ovviamente non opera su una scacchiera ma su una sua rappresentazione, localizzata fisicamente nella memoria del calcolatore.

Quindi, per scrivere un programma, si deve analizzare il problema e scegliere la migliore rappresentazione delle entita' implicate (le risorse).

E' evidente che la rappresentazione migliore e' quella piu' vicina al problema reale. Per esempio, riguardo alla scacchiera non conviene considerare 64 variabili che contengono numeri interi; e' certamente meglio considerare una tabella 8 * 8 costituita da "caselle" ognuna delle quali puo' assumere uno dei seguenti valori: "vuoto", "pedone", "alfiere", "regina", ecc.

Il linguaggio Pascal consente la stesura di programmi le cui entita' sono espresse negli stessi "termini" usati nel mondo reale; cosi', invece di rendere il problema conforme al programma, si puo' modellare il programma in conformita' del problema.

Cio' e' possibile se le variabili sono viste come contenitori non di numeri, ma di valore di ogni tipo. Una variabile deve essere considerata come un oggetto contenente alcuni valori definiti nel programma e denotati da identificatori, oltre che da numeri o stringhe di caratteri.

L'insieme di tutti i valori che una variabile puo' assumere e' detto il "tipo" di quella variabile.

Percio' definire un tipo significa definire un insieme di valori.

In riferimento all'esempio precedente, si puo' definire il tipo "casella" elencando i suoi valori: "re", "regina", "alfiere", "cavallo", "torre", "pedone", "vuoto". Una variabile di tipo "casella" puo' assumere solo uno di questi valori: non ha senso assegnarle un valore diverso.

Come ulteriore esempio si puo' considerare il tipo "scacchiera"; l'insieme dei valori di questo tipo e' costituito dall'insieme di tutte le possibili combinazioni dei 64 valori delle sue caselle.

E' evidente che per definire questo tipo, non conviene elencare tutti i possibili valori, ma usare una definizione piu' concisa: il tipo "scacchiera" e' una matrice 8 * 8 di elementi di tipo "casella".

Spesso e' necessario definire alcune operazioni associate al tipo in considerazione.
La definizione di un tipo implica generalmente quella di alcune operazioni che agiscono sui suoi valori.

Per esempio si puo' considerare il tipo INTEGER come l'insieme dei valori che variano da -infinito a +infinito, fornito di un insieme di operazioni come la somma, il prodotto, la divisione, e cosi' via.

Come altro esempio, si puo' considerare il tipo "colore" come l'insieme dei valori: "bianco", "giallo", "arancione", "rosso", "blu", "viola", "nero"; e lo si puo' immaginare dotato dell'insieme di operazioni:

"piu'" definito come

```
PIU' (ROSSO, BLU)= VIOLA
PIU' (GIALLO, ROSSO) = ARANCIONE
```

"inverso" definito come

```
INVERSO (NERO)= BIANCO
INVERSO (GIALLO)= VIOLA
```

"tonalita'" definito come

```
TONALITA' (BIANCO)= 0
TONALITA' (GIALLO)= 1
 :
TONALITA' (NERO)  = 6
```

Una variabile di tipo colore puo' assumere solo uno dei valori sopra elencati, sui quali si puo' agire con le operazioni definite precedentemente. Si consideri ora una parte di un programma Pascal che utilizza questo tipo:

```
TYPE COLORE = (BIANCO, GIALLO, ARANCIONE, ROSSO, BLU, VIOLA, NERO);
VAR X, Y : COLORE;
       I : INTEGER;
BEGIN
:
X:= ROSSO;
Y:= INVERSO (X);
I:= TONALITA' (PIU'(Y, BIANCO));
:
END.
```

Gli identificatori usati X, Y, I sono nomi di variabili, poiche' appaiono dopo la parola chiave "VAR". Essi possono essere usati in un' istruzione di assegnazione (X:= ...) oppure in un' espressione (...:= INVERSO(X)).

BIANCO, GIALLO, ..., NERO sono nomi di costanti, o meglio nomi di valori del tipo colore: non devono essere usati come oggetti di un' assegnazione, ma possono essere usati in un'espressione (...:= ROSSO). COLORE e INTEGER sono nomi di tipi: non si devono usare nella parte istruzioni, ma solo nella parte di descrizione delle risorse (VAR...: COLORE).

Qualsiasi altro uso di questi identificatori, diverso da quello descritto, non ha significato.

Nei successivi due paragrafi viene introdotta una classificazione di tipi, distinguendo i concetti di tipo "semplice" e tipo "strutturato".

TIPI SEMPLICI

Un tipo semplice e' un insieme di valori ciascuno dei quali e' considerato come un elemento informativo elementare ed indivisibile.

Ogni valore e' "costante" all'interno del suo tipo, ed e' denotato da un identificatore , da un numero o da una stringa, in relazione al modo in cui il tipo e' stato definito.

Esistono due strumenti linguistici di base per la definizione del tipo semplice:

- enumerazione, che consiste nell'elencare tutti i valori di un tipo nuovo definito dall'utente; questi valori sono distinti tramite nuovi identificatori e sono considerati in ordine ascendente secondo l'ordine in cui sono scritti.

- "subrange", che consiste nello specificare il primo e l'ultimo valore di un sottoinsieme compatto di un tipo semplice precedentemente definito.

Esempi di tipi enumerativi:

```
TYPE   CASELLA    = (VUOTO, PEDONE, CAVALLO, ALFIERE,
                      TORRE, REGINA, RE);

       COLORE     = (BIANCO, GIALLO, ARANCIONE, ROSSO,
                      BLU, VIOLA, NERO);

       MISURA     = (TEMPERATURA, PRESSIONE, UMIDITA',
                      FUMOSITA');

       GIORNO     = (LUNEDI, MARTEDI, MERCOLEDI, GIOVEDI,
                      VENERDI, SABATO, DOMENICA);

       SEME       = (PICCHE, FIORI, QUADRI, CUORI);

       CIFRE_ROM  = (I, V, X, L, C, D, M);
```

Esempi di tipi "subrange":

```
TYPE   COLORE_CHIARO    = BIANCO..ARANCIONE;

       GIORNI_FERIALI   = LUNEDI..VENERDI;

       SEMI_ROSSI       = QUADRI..CUORI;

       INDICE           = 1..100;

       LETTERA          = "A".."Z";
```

Inoltre si hanno alcuni particolari tipi semplici propri del linguaggio, che sono risorse predefinite, indicate con i seguenti identificatori:

INTEGER	indica l'insieme dei numeri interi;
REAL	indica l'insieme dei numeri reali;
BOOLEAN	indica l'insieme dei valori: falso e vero;
CHAR	indica l'insieme dei caratteri (che dipende dall'implementazione).

Da un punto di vista concettuale questi tipi non differiscono da quelli che l'utente puo' definire per enumerazione, poiche' sono costituiti da insiemi ordinati di valori, denotati non solo da identificatori ma anche in altro modo (numeri, stringhe). Pero' l'insieme di valori definiti dai tipi INTEGER, REAL e CHAR dipendono dalla particolare implementazione.

Le operazioni che si possono applicare ad un tipo definito da programma devono essere definite utilizzando le routine, mentre si hanno alcune operazioni, applicabili ai tipi predefiniti, anch'esse predefinite (+, -, *, /, EXP, LN,...).

Inoltre vi sono alcune operazioni predefinite per tutte le varieta' di tipi semplici, anche definite da programma, come gli operatori relazionali (=, >, <, >=, ...), successore e predecessore (SUCC, PRED), e cosi' via.

TIPI STRUTTURATI

Un tipo strutturato e' un insieme di valori ciascuno dei quali costituito da un aggregato di valori di altri tipi.

Un tipo strutturato e' caratterizzato dai tipi dei valori che lo compongono, chiamati tipi base, e da un metodo di strutturazione.

Non e' possibile definire un tipo strutturato con l'enumerazione dei suoi valori, sia perche' non ci sono identificatori che denotano i valori, sia perche' spesso la cardinalita' e' elevata.

I valori strutturati sono trattati per lo piu' a livello di componente: il riferimento ai componenti e' effettuato tramite le operazioni associate al tipo, dette "selezioni".

Uno strumento linguistico che definisce sia il metodo di strutturazione che l'operazione di selezione si chiama "costruttore"; esso pero' non definisce completamente il tipo strutturato, poiche' non specifica il tipo base. Un costruttore puo' essere visto come un tipo strutturato, i cui tipi base sono parametrizzati e dotati di un' operazione di selezione.

Si considerano ora alcuni esempi di costruttori.

Il primo esempio riguarda la rappresentazione, in un programma Pascal, di una carta d'identita'. L'intera informazione viene suddivisa in un numero di porzioni di informazione piu' dettagliate (e piu' elementari): nome, cognome, data di nascita e luogo di nascita.
L'ordine di tali porzioni di informazione sulla carta d'identita' non e' rilevante; ciascuna e' selezionata individualmente tramite un identificatore: "nome", "luogo di nascita", ecc.

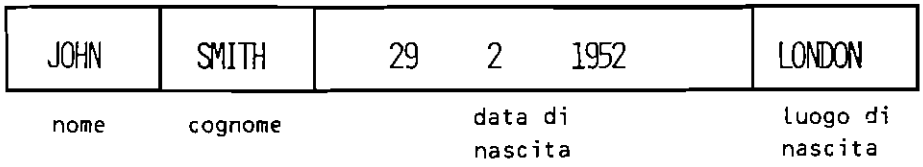


Fig. 1-9 Rappresentazione Grafica di una Carta d'Identita'

Questo metodo di strutturazione e di operazione di selezione costituisce il costruttore RECORD, in cui i tipi base (detti campi) sono i tipi di ciascuna porzione di informazione.
Il costruttore RECORD puo' essere usato per definire il tipo strutturato "CARTA_IDENTITA'", descritto nel modo seguente:

```
TYPE CARTA_IDENTITA' = RECORD
                         NOME: STRING;
                         COGNOME: STRING;
                         DATA_NASCITA: RECORD;
                                         GIORNO: 1..31;
                                         MESE: 1..12;
                                         ANNO: INTEGER
                                       END;
                         LUOGO_NASCITA: STRING
                       END;
```

Quindi si puo' dire che l'insieme dei valori del tipo CARTA_IDENTITA' e' dato dalla combinazione di tutti i possibili valori (cioe' tipi) di ogni porzione di informazione.
Si deve notare che il campo DATA_NASCITA e' a sua volta di tipo record.

Come secondo esempio, si vuole rappresentare l'insieme di valori che una scacchiera puo' assumere.

Si tratta di una matrice 8 * 8 , costituita da caselle che possono contenere, ognuna, un pezzo di un giocatore. Si vuole inoltre selezionare le singole caselle con una coppia di coordinate, ad esempio "C7" o in modo simile.

In altri termini si ha a disposizione un certo numero di valori componenti, tutti dello stesso tipo (CASELLA) il cui ordine e' rilevante.

Ogni valore e' selezionato tramite indici: il costruttore che indica tale metodo di strutturazione e tale operazione di selezione e' chiamato ARRAY.

Quindi si puo' definire il tipo strutturato "SCACCHIERA" specificando il tipo base nel modo seguente:

```
TYPE SCACCHIERA = ARRAY [1..8, 'A'..'H'] OF CASELLA
```

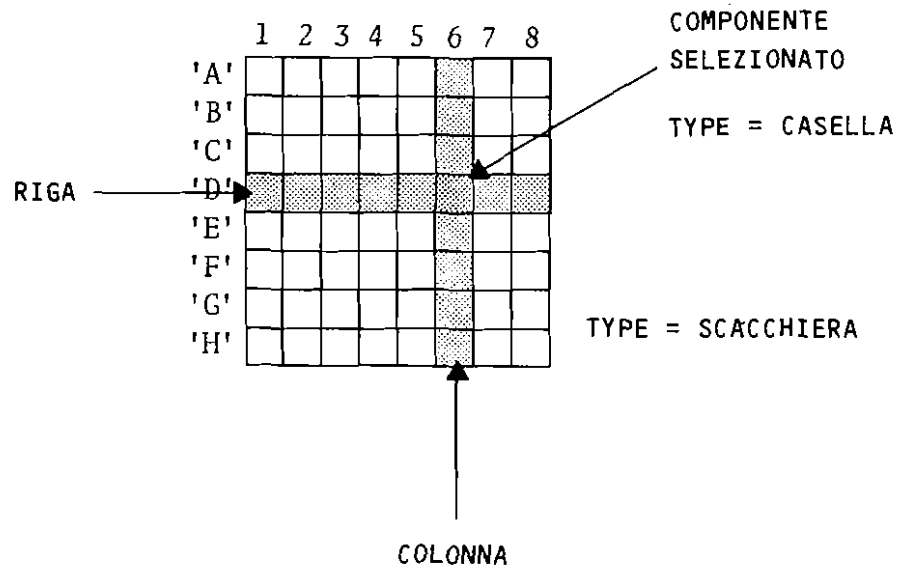


Fig. 1-10 Rappresentazione del Tipo Strutturato SCACCHIERA Definito Tramite il Costruttore ARRAY

Come terzo esempio si puo' considerare la rappresentazione di un mazzo di carte durante una partita di poker. Si tratta della sequenza di un numero variabile di carte, in cui solo quella superiore puo' essere estratta e sono consentiti possibili inserimenti soltanto dall'ultima carta del mazzo.

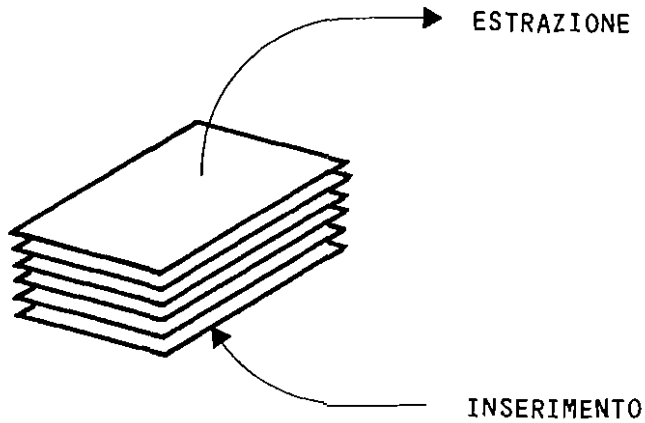


Fig. 1-11 Un Mazzo di Carte in una Partita di Poker

Questo metodo di strutturazione ed operazione di selezione e' detto costruttore QUEUE (le operazioni di selezione sono generalmente chiamate ENQUEUE e DEQUEUE). Il tipo strutturato MAZZO_CARTE puo' essere definito specificando il tipo base CARTA.

Un costruttore simile e' lo STACK. Esso differisce dalla coda (queue) poiche' inserimenti ed estrazioni sono effettuati nella stessa direzione (tali operazioni di selezione sono comunemente chiamate PUSH e POP).

Un ultimo esempio puo' essere la rappresentazione di un libro, esso infatti e' costituito da un numero variabile e ordinato di pagine, scritte e/o lette sequenzialmente a partire dalla prima pagina.

Il costruttore che consente la definizione di un simile tipo strutturato e' chiamato FILE, e la definizione del tipo LIBRO e' effettuato nel modo seguente:

```
TYPE PAGINA = ...;
     LIBRO  = FILE OF PAGINA;
```

Si deve osservare che il tipo base puo' a sua volta essere un tipo strutturato: il tipo PAGINA, per esempio, puo' essere definito come un array di righe.

```
TYPE RIGHE  = ...(*eventualmente strutturato*);
     PAGINA = ARRAY [1..30] OF RIGHE;
     LIBRO  = FILE OF PAGINA;
```

I costruttori piu' comunemente usati sono predefiniti in Pascal cioe': array, record, file ecc.

Tuttavia costruttori particolari (stack, queue, tree,...) possono essere definiti da programma per mezzo delle librerie di risorse.

ESECUZIONE DEI PROGRAMMI E GESTIONE DELLA MEMORIA

Viene ora illustrata l'esecuzione di un programma Pascal; si esaminano soltanto le chiamate di routine (procedure o funzioni), le quali costituiscono delle chiamate a macchine astratte; non vengono invece illustrati la sintassi e il formalismo di tali chiamate.

Comunque, poiche' le chiamate di routine implicano l'allocazione di memoria, e' necessario indicare le tecniche di gestione della memoria con il Pascal, cioe' l'allocazione automatica nel Run-Time-Stack e l'allocazione da parte dell'utente nell'area Heap.

EFFETTO DI UNA CHIAMATA DI ROUTINE

La chiamata di una routine causa il trasferimento del controllo a tale routine; al termine della sua esecuzione il controllo viene nuovamente trasferito all'istruzione che segue quella di chiamata.

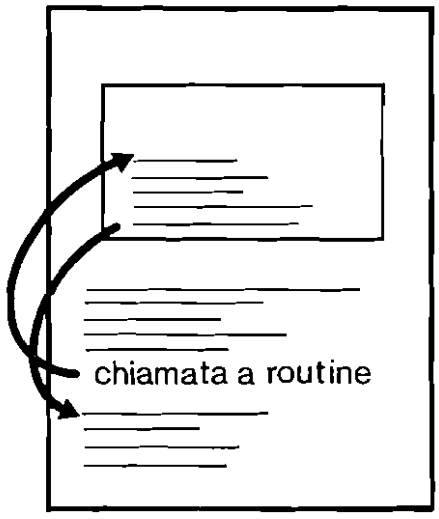


Fig. 1-12 Ordine di Esecuzione

Tuttavia la chiamata di una routine non comporta soltanto trasferimenti di controllo: come gia' osservato precedentemente, essa puo' essere considerata come la chiamata di una macchina astratta, costituita da alcune risorse e da alcune istruzioni.

In particolare, alcune di queste risorse possono essere delle variabili; in base alle regole di visibilita', tali variabili non sono visibili al momento della chiamata, per cui non e' necessaria la loro allocazione nella memoria del calcolatore.

Pero' esse devono essere disponibili durante l'esecuzione della routine; pertanto la loro allocazione avviene mentre la routine e' in esecuzione.

Quando invece la routine e' stata eseguita, queste variabili vengono deallocate, in quanto non visibili dall'ambiente esterno.

Riassumendo, le variabili interne di una routine sono allocate dinamicamente e rimangono in memoria solo durante l'esecuzione della routine.

Cio' consente un' efficiente gestione della memoria, poiche' vengono allocate soltanto le variabili significative, e la stessa area puo' essere usata, in tempi diversi, per le altre variabili delle varie routine chiamate.

IL RUN TIME STACK

L'area di memoria riservata alle variabili di un programma Pascal e' chiamata Run Time Stack (RTS). Essa e' logicamente organizzata come uno stack, poiche' l'allocazione e la deallocazione agiscono come gli operatori "push" e "pop".

Viene presentato ora un esempio di esecuzione di programma.

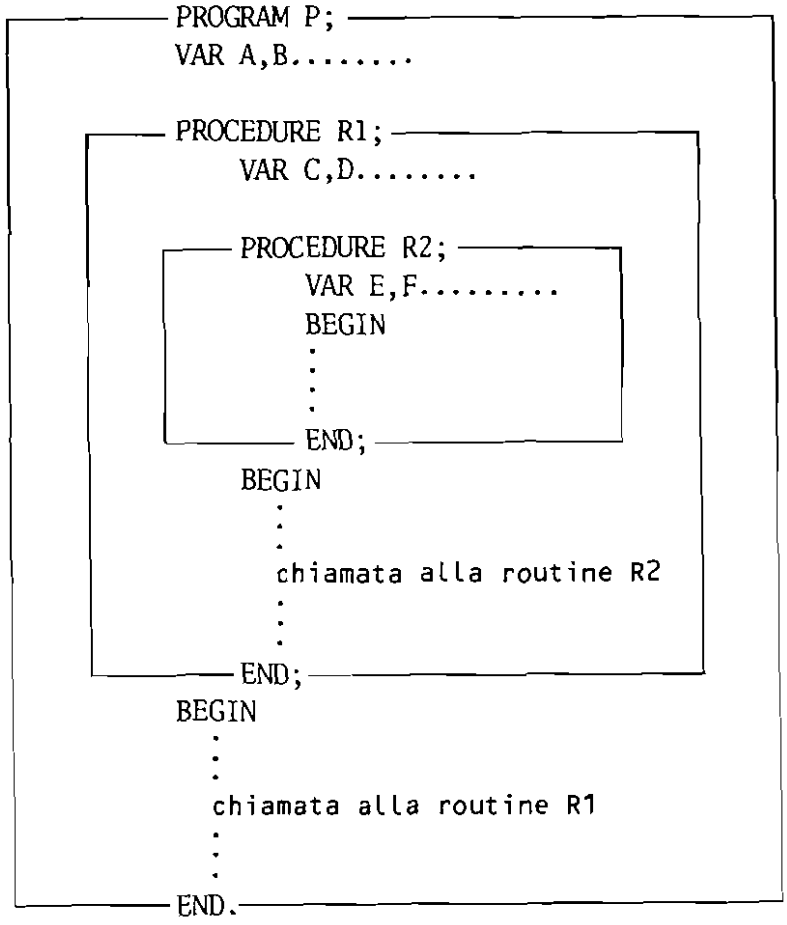


Fig. 1-13 Esempio di Allocazione delle Variabili

Prima di iniziare l'esecuzione di un programma, l'area RTS e' vuota; mentre si eseguono le istruzioni di P (il programma principale), l'area RTS contiene solo le variabili A e B; quando si esegue una chiamata ad R1, vengono allocate le sue variabili C e D; analogamente per le chiamate ad R2.

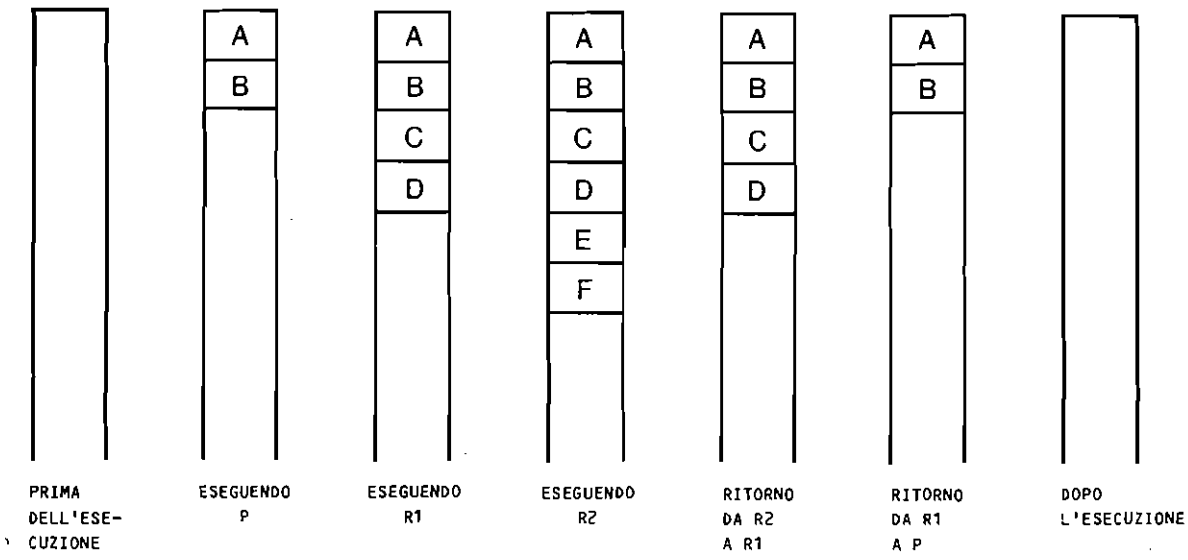


Fig. 1-14 Evoluzione Temporale del Run Time Stack

Inoltre ciascun rientro da routine provoca la deallocazione delle variabili ad essa relative.

Si considera ora il caso di routine "fratelli" chiamate sequenzialmente dalla routine "padre":

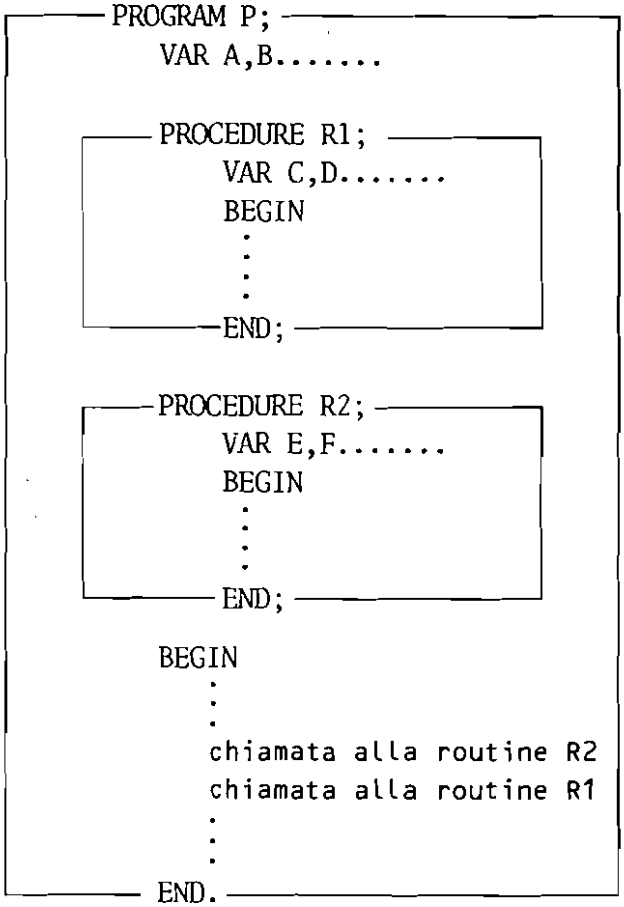


Fig. 1-15 Esempio di Allocazione delle Variabili

In questo caso le variabili C e D di R1 sono allocate nella stessa area in cui erano precedentemente allocate le variabili E ed F di R2.

2. INTRODUZIONE AL PASCAL M20

SOMMARIO

Questo capitolo descrive le principali caratteristiche del linguaggio Pascal Microsoft, e definisce i tre livelli ai quali il linguaggio puo' essere usato.

INDICE

INTRODUZIONE AL PASCAL M20

Il Pascal M20 e' basato sul Pascal Microsoft (anche detto MS-Pascal), che e' una versione altamente estesa e portabile del linguaggio Pascal. Tuttavia in tale contesto l'MS-Pascal viene indicato solamente come Pascal. Le sue estensioni, compatibili con lo standard proposto dall' Organizzazione Internazionale degli Standards (ISO), facilitano la programmazione dei sistemi e delle applicazioni.

Si puo' usare il Pascal al livello dello standard ISO per trasportare i programmi a/da altre macchine, oppure, per usare totalmente le capacita' di un particolare calcolatore, si possono rendere piu' efficienti i programmi usando il linguaggio ai suoi livelli Esteso e di Sistema.

Con il Pascal si ottengono i vantaggi di programmazione di un linguaggio ad alto livello, senza ridurre la velocita' di esecuzione. Poiche' vi sono molte conversioni di basso livello al livello macchina, i programmi scritti in Pascal sono paragonabili in velocita' a quelli scritti in linguaggio assembler.

Il Capitolo 3, "Generalita' del linguaggio" da' un'ampia descrizione del linguaggio, dagli elementi piu' semplici del linguaggio a quelli piu' complessi. I capitoli successivi completano tale visione generale trattando gli elementi, capitolo per capitolo, iniziando dai minori per terminare con una descrizione dei programmi e delle altre unita' compilabili.

Per avere informazioni sull'uso del Compilatore Pascal e dettagli sulla versione specifica del Pascal qui descritto, si puo' vedere l'Appendice A, "Caratteristiche di Implementazione" nel manuale "Linguaggio Pascal Guida Utente".

LIVELLI DEL PASCAL

Il linguaggio Pascal e' organizzato in tre livelli: Standard, Esteso e di Sistema. Le caratteristiche di ciascun livello sono descritte in dettaglio nell'Appendice B, "Prestazioni del Pascal e lo Standard ISO". In breve, le differenze fra i tre livelli sono le seguenti:

1. Livello Standard

   I programmi scritti in base al livello Standard devono essere conformi allo standard ISO; inoltre sono portabili a/da macchine su cui operano altri compilatori Pascal compatibili con l'ISO. Tuttavia vi sono alcune alcune estensioni minori rispetto allo standard che, pero', a questo livello non causano errori. Per avere ulteriori dettagli su tali estensioni e maggiori caratteristiche riguardanti lo standard ISO, si puo' vedere l'Appendice B, "Prestazioni del Pascal e lo Standard ISO". Nel presente manuale le espressioni: "Pascal Standard", "lo standard ISO" e "al livello standard del Pascal" sono generalmente sinonimi.

2. Livello Esteso

   Tale livello e' costituito dalle estensioni strutturate dello standard ISO. I programmi cosi' creati risultano portabili fra tutte le macchine che hanno la possibilita' di operare in Pascal.

3. Livello di Sistema

   Il livello di sistema include tutte le caratteristiche disponibili al livello esteso, oltre ad alcune estensioni non strutturate, orientate verso la macchina, come i tipi indirizzo e la capacita' di accedere a tutti i campi dei file control block, utili per la programmazione di sistemi.

Nel presente manuale le estensioni al Pascal Standard sono dette "caratteristiche". Un elenco completo di queste caratteristiche ed il livello a cui esse sono disponibili, e' riportato nell'Appendice B, "Prestazioni del Pascal e lo Standard ISO". Qui di seguito, brevemente, vengono descritte alcune caratteristiche selezionate.

Oltre ai tre livelli gia' visti, il Pascal ha un gran numero di "metacomandi", cioe' di direttive con le quali e' possibile controllare il compilatore. A tale riguardo si veda il Capitolo 19, "Metacomandi".

CARATTERISTICHE SELEZIONATE

La seguente lista comprende solo alcune delle caratteristiche disponibili ai livelli Esteso e di Sistema. Per un elenco completo si puo' vedere la sezione sulle caratteristiche Pascal, nell'Appendice B.

- Carattere di sottolineatura negli identificatori, che aumenta la leggibilita'

- Numerazione non decimale (esadecimale, ottale e binaria), che facilita la programmazione al livello di byte e di bit

- Costanti strutturate, che si possono sia dichiarare nella sezione dichiarativa di un programma sia usare nelle istruzioni

- Stringhe di lunghezza variabile (tipo LSTRING) e particolari procedure e funzioni predichiarate per i tipi LSTRING, che superano le capacita' di trattamento di stringhe del Pascal Standard

- Super array, uno speciale array di lunghezza variabile la cui dichiarazione consente il passaggio di array di lunghezze differenti a un parametro di riferimento e l' allocazione dinamica di array di lunghezze diverse

- Tipi predichiarati senza segno BYTE (0-255) e WORD (0-65535), che facilitano la programmazione a livello di Sistema

- Tipi indirizzo (segmentato e non segmentato), che consentono la manipolazione degli attuali indirizzi di macchina a livello di Sistema

- Lettori di stringa, che consentono alle procedure READ e READLN di leggere le stringhe come strutture invece che carattere per carattere

- Interfaccia con il linguaggio assembler, fornita dalle procedure PUBLIC ed EXTERN, da funzioni e da variabili, la quale consente di interfacciare a basso livello il linguaggio assembler e le routine di libreria

- Sezione VALUE, dove e' possibile dichiarare i valori costanti iniziali delle variabili di un programma

- Valori di ritorno di una funzione di tipo strutturato e di tipo semplice

- File diretti (ad accesso casuale) accessibili con la procedura SEEK, che migliora le capacita' standard del Pascal di accedere ai file.

- Valutazione "lazy", uno speciale meccanismo interno per file interattivi che consente il normale input interattivo da terminale

- Istruzioni strutturate BREAK e CYCLE, che consentono uscite strutturate da un ciclo FOR, REPEAT o WHILE; e l' istruzione RETURN, che consente un'uscita strutturata da una procedura o funzione

- OTHERWISE nell'istruzione CASE, per mezzo del quale si evita esplicitamente di specificare ogni costante CASE. OTHERWISE e' anche consentito in record con varianti

- Attributo STATIC per variabili, che consente di indicare che una variabile deve essere allocata ad una fissata locazione di memoria invece che nello stack

- Attributo ORIGIN, che puo' essere dato a variabili, procedure e funzioni per indicare la loro locazione assoluta in memoria

- Attributo INTERRUPT per procedure, che avverte il compilatore di dare alla procedura una speciale sequenza di chiamata per salvare lo stato della macchina nello stack in entrata e ripristinare lo stato della macchina in uscita

- Compilazione separata di parti di un programma (unita' e moduli)

- Compilazione condizionale, che usa metacomandi condizionali nel file sorgente per abilitare o disabilitare la compilazione di parti del sorgente.

CARATTERISTICHE NON IMPLEMENTATE

Le seguenti caratteristiche non sono attualmente implementate oppure lo sono soltanto come descritto qui di seguito:

- OTHERWISE non e' accettato nelle dichiarazioni RECORD

- Per le funzioni PURE e' generato codice, ma non e' eseguito alcun controllo

- Gli operatori di livello Esteso SHL, SHR e ISR non sono disponibili

- Le routine di libreria ENABIN, DISBIN e VECTIN non sono disponibili; l'attributo INTERRUPT e' ignorato

- Per le istruzioni GOTO non valide e per i valori REAL non inizializzati non e' eseguito alcun controllo

- READ, READLN e DECODE non possono avere M e N parametri

- L'I/O enumerato, per la lettura e scrittura di costanti enumerate come stringhe, non e' disponibile

- Possono essere dati i metacomandi $TAGCK, $STANDARD, $EXTEND e $SYSTEM, pero' non procurano alcun effetto

- Il metacomando $INCONST non accetta costanti stringa.

3. GENERALITÀ DEL LINGUAGGIO

SOMMARIO

Questo capitolo contiene una descrizione sommaria del linguaggio Pascal, dagli elementi più complessi a quelli più elementari. Ciascuno degli altri capitoli seguenti descrive questi argomenti in dettaglio, iniziando pero' da quelli piu' elementari (la notazione) per terminare con quelli piu' complessi del linguaggio (i metacomandi).

INDICE

METACOMANDI

I metacomandi Pascal forniscono un linguaggio di controllo per il compilatore Pascal. I metacomandi permettono di specificare le opzioni che riguardano l'operazione totale di una compilazione. Per esempio si possono compilare in modo condizionato differenti file sorgenti, si puo' generare un file listing, abilitare o disabilitare un codice di controllo di errori runtime.

I metacomandi sono inseriti entro le istruzioni di commento, ed iniziano tutti con un segno di dollaro ($). Quando si chiama il compilatore, alcuni possono essere dati anche sotto forma di commutatori.

Anche se la maggior parte delle implementazioni del Pascal possiede qualche mezzo di controllo del compilatore, tali metacomandi Pascal non fanno parte del Pascal Standard e quindi non sono portabili.

Questi sono i metacomandi generalmente disponibili:

$BRAVE
$DEBUG
$ENTRY
$ERRORS
$EXTEND
$GOTO
$INCLUDE
$INCONST
$INDEXCK
$INTICK
$IF $THEN $ELSE $END
$INTEGER
$LINE
$LINESIZE
$LIST
$MATHCK
$MESSAGE
$NILCK
$OCODE
$OPTBUG
$PAGE
$PAGEIF
$PAGESIZE
$POP
$PUSH
$RANGECK
$REAL
$ROM
$RUNTIME
$SIMPLE
$SIZE
$SKIP
$SPEED
$STACKCK
$STANDARD
$SUBTITLE
$SYMTAB
$SYSTEM
$TAGCK
$TITLE
$WARN

Per una descrizione completa dei metacomandi si puo' vedere il Capitolo 19, "Metacomandi".

PROGRAMMI E PARTI COMPILABILI DEI PROGRAMMI

Il compilatore Pascal elabora programmi, moduli e implementazioni di unita'. Tutti questi programmi compilabili e parti di programmi sono indicati come unita' di compilazione. Si possono compilare moduli e implementazioni di unita' separatamente, e poi collegarli ad un programma senza dover ricompilare il modulo o l'unita'.

L'unita' fondamentale di compilazione e' il programma. Un programma e' costituito da tre parti:

1. L'intestazione del programma identifica il programma e fornisce una sua lista di parametri.

2. La sezione dichiarativa segue l'intestazione e contiene dichiarazioni di etichette, costanti, tipi, variabili, funzioni e procedure. Alcune di queste dichiarazioni sono opzionali.

3. Il corpo del programma segue tutte le dichiarazioni. E' racchiuso tra le parole riservate BEGIN e END e termina con un punto. Il punto serve a segnalare al compilatore che la fine del file sorgente e' stata raggiunta.

Il programma seguente illustra queste tre parti della struttura di un programma:

```
{Intestazione del programma}
PROGRAM FRIDAY (INPUT, OUTPUT);

{Sezione dichiarativa}
LABEL 1;
CONST DAYS_IN_WEEK = 7;
TYPE  KEYBOARD_INPUT = CHAR;
VAR   KEYIN: KEYBOARD_INPUT;

{Corpo del programma}
BEGIN
   WRITE(ÍIS TODAY FRIDAY? ');
1: READLN(KEYIN);
   CASE KEYIN OF
    'Y', 'y' : WRITELN('It''s Friday.');
    'N', 'n' : WRITELN('It''s not Friday.');
   OTHERWISE
      WRITELN('Inserire Y oppure N.');
      WRITE('Per favore reintrodurre: ');
      GOTO 1
   END
END.
```

Questa struttura suddivisa in tre parti (intestazione, sezione dichiarativa, corpo) e' usata in tutto il linguaggio Pascal. Procedure, funzioni, moduli e unita' sono tutti simili, nella struttura, ad un programma.

I moduli sono unita' di compilazione simili a un programma e contengono la dichiarzione di variabili, costanti, tipi, procedure e funzioni ma non istruzioni di programma. Si puo' compilare un modulo separatamente e poi collegarlo ad un programma, ma il modulo non puo' essere eseguito da solo.

Esempio di modulo:

```
{Intestazione di modulo}
MODULE MODPART;

{Sezione dichiarativa}
CONST PI = 3.14;

PROCEDURE PARTA;
   BEGIN
     WRITELN ('parta')
   END;

{Corpo}
END.
```

Un modulo, come un programma, termina con un punto; pero' a differenza di esso un modulo non contiene alcuna istruzione di programma.

Un'unita' di compilazione ha due sezioni: un'interfaccia e un'implementazione. Come un modulo, un'implementazione puo' essere compilata separatamente e piu' tardi collegata al resto del programma. L'interfaccia contiene le informazioni che permettono di collegare un'unita' ad altre unita', moduli e programmi.

Esempio di unita' di compilazione:

```
{Intestazione per l'interfaccia}
INTERFACE;
UNIT MUSIC (SING, TOP);

{Dichiarazione per l'interfaccia}
VAR TOP : INTEGER;
PROCEDURE SING;

{Corpo dell'interfaccia}
BEGIN
END;

{Intestazione per l'implementazione}
IMPLEMENTATION OF MUSIC;

{Dichiarazione per l'implementazione}
PROCEDURE SING;
BEGIN
   FOR I:= 1 TO TOP DO
   BEGIN
     WRITE ('FA '); WRITELN ('LA LA')
   END
END;

{Corpo dell'implementazione}
BEGIN
   TOP := 5
END.
```

Un'unita', come un programma o un modulo, termina con un punto.

I moduli e le unita' permettono di sviluppare grandi programmi strutturati che possono essere suddivisi in parti. Questo risulta vantaggioso nelle seguenti situazioni:

- Se un programma e' di grandi dimensioni, e' piu' facile svilupparlo, testarlo e mantenerlo suddividendolo in parti.

- Se un programma e' grande e la ricompilazione dell'intero file sorgente richiede troppo tempo, con la suddivisione del programma in parti si guadagna del tempo per la compilazione.

- Se si intende includere determinate routine in un certo numero di differenti programmi, si puo' creare un singolo file oggetto che contiene queste routine e poi collegarlo a ciascuno dei programmi in cui le routine sono utilizzate.

- Se determinate routine hanno differenti implementazioni, le si puo' collocare in un modulo per testare la validita' di un algoritmo e poi creare e implementare routine simili in linguaggio assemblativo per aumentare la velocita' dell'algoritmo.

Per una descrizione piu' completa di programmi, moduli e unita' si puo' consultare il Capitolo 18 su "Unita' compilabili di un programma".

PROCEDURE E FUNZIONI

Le procedure e le funzioni si comportano come sottoprogrammi che operano sotto la supervisione di un programma principale. Tuttavia, a differenza dei programmi, le procedure e le funzioni possono essere nidificate le une dentro le altre e possono anche chiamare se stesse. Inoltre posseggono sofisticate capacita' di passaggio dei parametri che i programmi invece non hanno.

Una procedura e' chiamata da un'istruzione che consiste semplicemente nel nome della procedura; una funzione puo' essere chiamata in una espressione ogni volta che e' richiesto un valore.

Una dichiarazione di procedura ha, come un programma, un'intestazione, una sezione dichiarativa ed un corpo.

Esempio di dichiarazione di procedura:

```
{Intestazione}
PROCEDURE COUNT_TO(NUM : INTEGER);

{Sezione dichiarativa}
VAR I : INTEGER;

{Corpo}
BEGIN
  FOR I := 1 TO NUM DO  WRITELN (I)
END;
```

Una funzione e' una procedura che ritorna un valore di un tipo particolare; quindi una dichiarazione di funzione deve indicare il tipo del valore di ritorno.

Esempio di dichiarazione di funzione:

```
{Intestazione}
FUNCTION ADD (VAL1, VAL2 : INTEGER): INTEGER;

{Sezione dichiarativa vuota}

{Corpo}
BEGIN
   ADD := VAL1 + VAL2
END;
```

Le procedure e le funzioni si presentano differentemente dai programmi, per il fatto che i loro parametri hanno tipi ed altre opzioni. Come il corpo di un programma, il corpo di una procedura o di una funzione e' compreso tra le parole riservate BEGIN e END; pero' in tal caso la parola "END" e' seguita da un punto e virgola piuttosto che da un punto.

La dichiarazione di una procedura o di una funzione e' una cosa completamente distinta dal suo uso in un programma. Per esempio, la procedura e la funzione sopra dichiarate possono effettivamente presentarsi cosi' in un programma:

```
TARGET_NUMBER := ADD (5, 6);
COUNT_TO (TARGET_NUMBER);
```

Per una completa descrizione delle procedure e delle funzioni si puo' vedere il Capitolo 15 su "Introduzione a Procedure e Funzioni".

Invece per una descrizione delle procedure e delle funzioni predichiarate come parte del linguaggio Pascal, si puo' vedere il Capitolo 16 su "Procedure e Funzioni Disponibili", e il Capitolo 17 su "Procedure e Funzioni Orientate su File".

ISTRUZIONI

Le istruzioni compiono azioni come calcoli, assegnazioni, alterazione del flusso di controllo, lettura e scrittura di file. Le istruzioni si trovano all'interno del corpo di programmi, procedure e funzioni, e sono eseguite mentre il programma e' in fase di esecuzione. Le istruzioni del Pascal eseguono le azioni descritte nella Tabella 3-1.

Istruzione	Azione
Assegnazione	Sostituisce il valore corrente di una variabile con un nuovo valore.
BREAK	Consente di uscire dal ciclo attualmente in esecuzione.
CASE	Permette la scelta di un'azione tra le tante disponibili, basata sul valore di un'espressione.
CYCLE	Da' inizio alla successiva iterazione di un ciclo.
FOR	Esegue un'istruzione ripetutamente mentre viene assegnata una progressione di valori ad una variabile di controllo.
GOTO	Continua l'elaborazione di un'altra parte del programma.
IF	Insieme a THEN e ELSE permette l'esecuzione condizionale di un'istruzione.
Chiamata di procedura	Chiama una procedura con valori attuali del parametro.
REPEAT	Ripete una sequenza di istruzioni una o piu' volte, finche' un'espressione booleana diventa vera.
RETURN	Consente di uscire dalla procedura, funzione, programma o implementazione correnti.
WHILE	Ripete un'istruzione zero o piu' volte, finche' un'espressione booleana diventa falsa.
WITH	Permette di includere nel dominio di un'istruzione i campi di uno o piu' record, in modo da poter far diretto riferimento ai campi.

Tabella 3-1 Sommario delle Istruzioni Pascal

Per una descrizione dettagliata di ciascuna di queste istruzioni si puo' vedere il Capitolo 14 su "Istruzioni".

ESPRESSIONI

Un'espressione e' una formula per calcolare un valore. E' costituita da una sequenza di operatori (che indicano l'azione da eseguire) e di operandi (il valore su cui l'operazione e' eseguita). Gli operandi possono contenere chiamate di funzioni, variabili, costanti o anche altre espressioni. Nella seguente espressione il piu' (+) e' un operatore, mentre A e B sono operandi:

```
A + B
```

Vi sono tre tipi fondamentali di espressioni:

1. Le espressioni aritmetiche, che compiono operazioni aritmetiche sugli operandi in un'espressione.

2. Le espressioni booleane, che compiono operazioni logiche e di confronto con risultati booleani.

3. Le espressioni set, che compiono operazioni di combinazione e di confronto su insiemi, con risultati booleani o set.

Le espressioni ritornano sempre valori di un tipo specifico. Per esempio se A, B, C e D sono tutte variabili di tipo REAL, allora la seguente espressione da' un risultato REAL:

```
A * B + (C / D) + 12.3
```

Le espressioni possono anche includere indicatori di funzione:

```
ADDREAL (2, 3) + (C / D)
```

ADDREAL e' una funzione dichiarata precedentemente nel programma. Ha due parametri valore di tipo REAL, che somma insieme per ottenere il totale. Questo totale e' il valore di ritorno della funzione, che e' poi aggiunto a (C / D).

Le espressioni non sono istruzioni, ma possono essere componenti di istruzioni. Nell'esempio seguente l'intera linea e' un'istruzione; solo la parte dopo il segno di uguale e' un'espressione:

```
X := 2 / 3 + A * B
```

Per una descrizione dettagliata delle espressioni si puo' vedere il Capitolo 13 su "Espressioni".

VARIABILI

Una variabile e' un valore che cambia durante il corso di un programma; ogni variabile deve essere di uno specifico tipo di dati.

Dopo aver dichiarato una variabile nell'intestazione o nella sezione dichiarativa di un'unita' compilativa, procedura o funzione, puo' essere usata in tutti questi modi:

- si puo' inizializzarla nella sezione VALUE di un programma
- si puo' assegnarle un valore con un'istruzione di assegnazione
- si puo' passarla come parametro a una procedura o funzione
- si puo' usarla in un'espressione

La sezione VALUE e' una caratteristica del Pascal che si applica soltanto a variabili allocate in modo statico (variabili con un indirizzo fissato in memoria). Come prima cosa si dichiarano le variabili come indicato nell'esempio seguente:

```
VAR I, J, K, L :  INTEGER;
```

Poi si assegnano ad esse valori iniziali nella sezione VALUE:

```
VALUE I := 1;  J := 2;  K := 3;  L := 4;
```

Piu' tardi, nelle istruzioni, le variabili possono essere assegnate e usate come operandi nelle espressioni:

```
I := J + K + L;
J := 1 + 2 + 3;
K := (J * K) + 9 + (L DIV J);
```

Per una descrizione completa delle variabili si puo' vedere il Capitolo 12 su "Variabili e Valori".

COSTANTI

Una costante e' un valore che non cambia durante il corso di un programma. Al livello Standard un costante puo' essere:

- un numero, come 1.234 e 100
- una stringa racchiusa tra segni di apice, come 'Pascal' o 'A1207'

- un identificatore di costante sinonimo di una costante numerica o stringa

Gli identificatori di costanti si dichiarano nella sezione CONST di un'unita' compilativa, procedura o funzione

```
CONST  REAL_CONST = 1.234;
       MAX_VAL    = 100;
       TITLE      = 'Pascal';
```

Poiche' nel Pascal l'ordine delle dichiarazioni e' flessibile, si possono dichiarare costanti in ogni punto della sezione dichiarativa della parte compilabile di un programma, per un numero indefinito di volte.

Le costanti sono strettamente collegate al concetto di variabile e tipo. Le variabili sono tutte di qualche tipo; i tipi, a loro volta, indicano un intervallo di valori assumibili. Questi valori, infine, sono tutti costanti.

Due potenti estensioni del Pascal sono le costanti strutturate e le espressioni costanti.

- VECTOR, nell'esempio seguente, e' un array costante:

```
CONST VECTOR = VECTORTYPE (1,2,3,4,5);
```

- MAXVAL, nell'esempio seguente, e' un'espressione costante (A, B, C e D devono anche essere costanti):

```
CONST MAXVAL = A * (B DIV C) + D - 5;
```

Per una descrizione completa di questi e di altri aspetti sulle costanti si puo' vedere il Capitolo 6 sulle "Costanti".

TIPI

Molta della potenza e della flessibilita' del Pascal risiede nella sua capacita' di dare dei tipi ai dati. Pur essendo disponibile una grande varieta' di tipi di dati, questi possono essere suddivisi in tre grandi categorie: semplici, strutturati e di riferimento.

1. Un tipo di dati semplice rappresenta un singolo valore; un tipo strutturato rappresenta una raccolta di valori. I tipi semplici comprendono i seguenti:

INTEGER	enumerato
WORD	subrange
CHAR	REAL
BOOLEAN	INTEGER4

2. I tipi strutturati comprendono i seguenti:
   ARRAY
   RECORD
   SET
   FILE

3. I tipi di riferimento consentono in modo molto efficace la definizione ricorsiva dei tipi.

In Pascal a tutte le variabili deve essere assegnato un tipo di dati. Un tipo puo' essere predichiarato (per esempio INTEGER e REAL) oppure definito nella sezione dichiarativa di un programma. Nel seguente esempio di dichiarazione di tipo si crea un tipo in grado di memorizzare delle informazioni su uno studente:

```
TYPE
   STUDENT = RECORD
       AGE      : 5..18;
       SEX      : (MALE, FEMALE);
       GRADE    : INTEGER;
       GRADE_PT : REAL;
       SCHEDULE : ARRAY [1..10] OF CLASSES
   END;
```

Per una descrizione dettagliata dei tipi di dati e' utile vedere i seguenti capitoli:

Capitolo 7, "Introduzione ai tipi di dati"
Capitolo 8, "Tipi semplici"
Capitolo 9, "Array, record e set"
Capitolo 10, "File"
Capitolo 11, "Tipi di riferimento e altri tipi"

IDENTIFICATORI

Gli identificatori sono nomi che indicano costanti, variabili, tipi di dati, procedure, funzioni ed altri elementi di un programma Pascal. Le procedure e le funzioni devono avere identificatori; le costanti, i tipi e le variabili possono avere identificatori (ed e' utile per loro averli).

Il programmatore costruisce la maggior parte degli identificatori del programma ed assegna loro un certo significato nelle dichiarazioni. Altri identificatori sono i nomi delle variabili, dei tipi di dati, delle procedure e delle funzioni: non occorre dichiararli, poiche' sono gia' presenti nel linguaggio.

Un identificatore deve iniziare con una lettera (da A fino a Z e da a fino a z). La lettera iniziale puo' essere seguita da qualunque numero di lettere, cifre (da 0 a 9) o di caratteri di sottolineatura. Il compilatore tratta allo stesso modo le lettere maiuscole e minuscole, quindi "A" e "a" sono equivalenti.

Nel Pascal il carattere di sottolineatura e' significativo, per cui i seguenti identificatori non sono uguali:

```
FOREST

FOR_EST
```

L'unica limitazione che riguarda gli identificatori e' questa: non si puo' scegliere come identificatore una parola riservata del linguaggio (per una descrizione delle parole riservate si puo' vedere la sezione sulle "Parole riservate", nel Capitolo 4; mentre per un elenco completo si puo' vedere l'Appendice E, "Elenco delle Parole Riservate del Pascal").

Inoltre la maggior parte dei compilatori hanno alcune limitazioni riguardo la lunghezza assoluta di un identificatore o il numero di caratteri considerati significativi. Per le limitazione imposte dal compilatore di questo linguaggio si puo' vedere l'Appendice A su "Caratteristiche di implementazione", nel manuale "Linguaggio Pascal Guida Utente".

Per una descrizione completa degli identificatori si puo' consultare il Capitolo 5 sugli "Identificatori".

NOTAZIONE

Alla base di tutti i programmi Pascal c'e' un insieme di simboli elementari con i quali sono create le piu' alte componenti sintattiche del linguaggio.

La notazione fondamentale e' costituita dall'insieme di caretteri ASCII, diviso nei seguenti gruppi sintattici:

- gli identificatori sono i nomi assegnati ad istanze individuali di componenti del linguaggio
- i separatori sono i caratteri che delimitano numeri, parole riservate e identificatori adiacenti
- i simboli speciali includono la punteggiatura, gli operatori e le parole riservate
- alcuni caratteri non sono utilizzati dal Pascal ma sono disponibili per l'uso in un commento o in una stringa letterale

Una buona conoscenza di questa notazione consente di aumentare la produttivita' riducendo il numero di piccoli errori sintattici in un programma. Per una descrizione dettagliata della notazione Pascal si puo' vedere il Capitolo 4 sulla "Notazione".

4. NOTAZIONE

SOMMARIO

Questo capitolo descrive come costruire gli identificatori, usando i componenti, i separatori, i simboli speciali ed i caratteri non usati nel linguaggio Pascal.

INDICE

INTRODUZIONE

Tutti i componenti del linguaggio Pascal sono costruiti tramite l'insieme standard di caratteri ASCII. I caratteri costituiscono linee, ciascuna delle quali e' separata da un carattere specifico del sistema operativo; a loro volta le linee costituiscono file.

All' interno di una linea, i singoli caratteri o gruppi di caratteri rientrano in una (o piu') delle seguenti quattro grandi categorie:

1. componenti di identificatori
2. separatori
3. simboli speciali
4. caratteri non utilizzati

COMPONENTI DI IDENTIFICATORI

Gli identificatori sono nomi che denotano costanti, variabili, tipi di dati, procedure, funzioni ed altri elementi di un programma Pascal.

L' uso di identificatori e' descritto in ampio modo nel Capitolo 5, "Identificatori".
Questa sezione descrive soltanto come essi vengono costruiti.

Gli identificatori devono iniziare con una lettera; i componenti che seguono possono includere lettere, cifre e caratteri di sottolineatura.

Anche se in teoria non vi e' limite al numero di caratteri di un identificatore, la maggior parte delle implementazioni ne limitano in qualche modo la lunghezza. Per la specifica limitazione che puo' essere qui applicata, si puo' vedere l' Appendice A, "Caratteristiche di Implementazione", nel manuale "Linguaggio Pascal Guida Utente".

LETTERE

Negli identificatori sono significative soltanto le lettere maiuscole da A a Z. Si possono usare lettere minuscole negli identificatori in un programma sorgente. Tuttavia il Compilatore Pascal trasforma tutte le lettere minuscole degli identificatori nelle corrispondenti lettere maiuscole.

Le lettere nei commenti o nelle stringhe letterali possono essere maiuscole o minuscole: la differenza e' pero' in tal caso significativa, poiche' non si verifica la precedente corrispondenza tra lettere minuscole e maiuscole.

CIFRE

Le cifre sono costituite dai numeri compresi fra zero e nove. Si possono trovare negli identificatori (per esempio, AS129M) oppure nelle costanti numeriche (per esempio 1.23 e 456).

CARATTERE DI SOTTOLINEATURA

Il carattere di sottolineatura (_) e' l'unico carattere non alfanumerico consentito negli identificatori. In Pascal esso e' significativo; lo si deve usare come spazio per migliorare la leggibilita'.
Per esempio gli identificatori della colonna di destra sono piu' facili da leggere di quelli della colonna di sinistra.

POWEROFTEN	POWER_OF-TEN
MYDOGMAUDE	MY_DOG_MAUDE

SEPARATORI

I separatori dividono numeri adiacenti, parole riservate e identificatori, nessuno dei quali deve avere separatori compresi al loro interno.

Un separatore puo' essere:

- il carattere di spazio
- Il carattere di tabulazione
- il carattere di avanzamento carta
- il segnale di nuova riga
- il commento

Nel Pascal Standard i commenti assumono una delle seguenti forme:

```
{Questo e' un commento, racchiuso tra parentesi.}

(*Questa e' una forma alternativa di commento*)
```

La seconda forma e' generalmente usata se le parentesi graffa non sono disponibili su una determinata macchina. In ciascuna di queste forme i commenti possono occupare piu' di una riga.

Al livello Esteso il Pascal consente anche commenti che iniziano con un punto esclamativo:

```
! Il resto di questa riga e' un commento.
```

Nei commenti che assumono tale forma, il carattere di nuova riga separa il commento.

Nel Pascal sono consentiti commenti nidificati, ma ciascun livello deve avere differenti separatori. Cosi', quando si inizia un commento, il compilatore ignora il testo successivo finche' non trova il simbolo di fine commento corrispondente. Tuttavia tale nidificazione non e' portabile.

Occorre usare sempre separatori tra gli identificatori e i numeri. In caso contrario il compilatore genera un errore o un messaggio di avvertimento. In pochi casi il compilatore Pascal accetta l'assenza di un separatore senza generare un messaggio di errore.

Per esempio, al livello Esteso

```
100MOD#127
```

e' accettato come 100 MOD #127, dove #127 e' un numero esadecimale. Tuttavia

```
100MOD127
```

e' assunto come 100 seguito dall'identificatore MOD127.

SIMBOLI SPECIALI

I simboli speciali rientrano in tre categorie:

1. simboli di punteggiatura
2. operatori
3. parole riservate

SIMBOLI DI PUNTEGGIATURA

Nel Pascal i simboli di punteggiatura servono a diversi scopi, compresi quelli descritti nella Tabella 4-1.

Simbolo	Scopo
{ }	Le parentesi graffa separano i commenti
[ ]	Le parentesi quadre separano indici, insiemi e attributi; esse possono anche sostituire le parole riservate BEGIN e END in un programma
( )	Le parentesi tonde separano espressioni, e liste di parametri e parametri di programma
'	Gli apici includono stringhe letterali
:=	Il simbolo di due punti-uguale assegna valori alle variabili nelle istruzioni di assegnazione e nelle sezioni VALUE
;	Il punto e virgola separa istruzioni e dichiarazioni
:	I due punti separano le variabili dai tipi, e le etichette dalle istruzioni
=	Il segno di uguale separa gli identificatori e le clausole di tipo in una sezione TYPE
..	Il doppio punto indica un subrange
.	Il punto indica sia la fine di un programma che la parte frazionale di un numero reale e separa i campi di un record
^	La freccia in alto (puntatore) indica un valore cui si fa riferimento
#	Il segno di numero indica numeri non decimali
$	Il segno di dollaro è usato come prefisso per metacomandi

Tabella 4-1 Sommario della Punteggiatura del Pascal

OPERATORI

Gli operatori sono una forma di punteggiatura che indica alcune operazioni da eseguire; alcuni sono alfabetici, altri sono costituiti da uno o due caratteri non alfanumerici. Ogni operatore costituito da piu' di un carattere non deve avere un separatore tra i caratteri.

Gli operatori formati soltanto da caratteri non alfabetici sono i seguenti:

```
+    -    *    /    >    <    =    < >    < =    > =
```

Alcuni operatori (per esempio NOT e DIV) sono parole riservate, invece di caratteri non alfabetici.

Per un elenco completo degli operatori non alfabetici e per una descrizione dell'uso degli operatori nelle espressioni, si puo' vedere il Capitolo 13 sulle "Espressioni".

PAROLE RISERVATE

Le parole riservate sono una componente fissa del linguaggio Pascal. Esse includono, per esempio, nomi di istruzioni (per esempio BREAK) e parole come BEGIN e END che mettono tra parentesi il corpo principale del programma. Per un elenco completo si puo' vedere l' Appendice E :"Elenco delle Parole Riservate del Pascal".

Non si puo' creare un identificatore uguale ad una parola riservata; tuttavia puo' essere dichiarato un identificatore che contiene al suo interno le lettere di una parola riservata.

Vi sono diverse categorie di parole riservate:

- parole riservate per il Pascal a livello Standard
- parole riservate aggiunte per le caratteristiche del Pascal a livello Esteso
- parole riservate aggiunte per le caratteristiche del Pascal a livello di Sistema
- nomi degli attributi
- nomi delle direttive

Per un elenco completo delle parole riservate, si puo' vedere la Appendice E: "Elenco delle Parole Riservate del Pascal".

CARATTERI NON UTILIZZATI

Pochi caratteri di stampa non sono utilizzati nel Pascal:

```
%   &   "   |   ~   `   \
```

Tuttavia essi possono essere usati all'interno di commenti o di stringhe letterali.

Un certo numero di altri caratteri ASCII non stampabili generano errore se vengono usati in un file sorgente piuttosto che in un commento o in una stringa letterale:

- i caratteri da CHR (0) a CHR (31), tranne il carattere di tabulazione e quello di avanzamento carta, rispettivamente CHR (9) e CHR (12)

- i caratteri da CHR (127) a CHR (255)

Il carattere di tabulazione, CHR (9), e' trattato come uno spazio ed e' passato al file listing. Il carattere di avanzamento carta, CHR (12), e' trattato come uno spazio e da' inizio ad una nuova pagina nel file listing.

NOTE SUI CARATTERI

Questa sezione descrive le speciali proprieta' notazionali del linguaggio Pascal, non trattate altrove in questo capitolo.

I caratteri all'interno di un commento o di una stringa letterale sono sempre consentiti e non danno luogo a particolari effetti.

Il Pascal consente le seguenti sostituzioni:

Cio' che manca sulla tastiera...	puo' essere sostituito da...
[	(.
]	.)
^	@ or ?
@	^ or ?

Il punto interrogativo (?) puo' sostituire la freccia in alto (^), ma si tratta di un uso di estensione minore dello standard ISO.

La tabella 4-2 fornisce un elenco di coppie di caratteri stampabili, che sono gli stessi caratteri ASCII.

ASCII	Carattere	Sostitutivo
CHR (94)	^	freccia in alto, puntatore
CHR (95)	_	sottolineatura, freccia sinistra
CHR (35)	#	segno di numero, segno di sterlina
CHR (36)	$	segno di dollaro, scarabeo (cerchio con quattro punte)

Tabella 4-2 Caratteri ASCII Equivalenti

5. IDENTIFICATORI

SOMMARIO

In questo capitolo e' descritto, in modo piu' dettagliato, cos'e' un identificatore, il suo uso e la sintassi utilizzata per dichiararlo. Inoltre si ha un elenco di tutti gli identificatori predichiarati, suddivisi in base ai tre livelli di linguaggio.

INDICE

COS' E' UN IDENTIFICATORE ?

Gli identificatori sono nomi che denotano costanti, variabili, tipi di dati, procedure, funzioni e altri elementi di un programma Pascal. Le procedure e le funzioni devono avere identificatori; le costanti, i tipi e le variabili possono avere identificatori (ed e' utile per essi averli).

Alcuni identificatori sono predichiarati, altri vengono dichiarati nella sezione dichiarativa. Il Pascal Standard consente l'uso di identificatori per i seguenti elementi del linguaggio:

tipi
costanti
variabili
procedure
funzioni
programmi
campi e identificatori di campo nei record

Anche le seguenti caratteristiche del Pascal, a livello Esteso, richiedono l'uso di identificatori:

tipi super array
moduli
unita'
etichette di istruzioni

Un identificatore e' costituito da una sequenza di caratteri alfanumerici o di caratteri di sottolineatura, pero' e' essenziale che il primo carattere sia alfabetico. Negli identificatori i caratteri di sottolineatura sono consentiti e sono significativi a tutti i livelli del Pascal; inoltre e' possibile usare due caratteri di sottolineatura in una riga, e far terminare un identificatore con un carattere di sottolineatura.

Nonostante le limitazioni prima osservate, gli identificatori possono essere di qualunque lunghezza, ma devono comunque essere posizionati su una riga.
Risultano significativi i primi 19 caratteri di un identificatore, (in alcune versioni 31).

Un identificatore che supera la lunghezza significativa genera un segnale di avvertimento, non un messaggio di errore; i caratteri oltre quelli significativi sono ignorati dal compilatore. Per la lunghezza significativa nelle specifiche implementazioni si puo' vedere l'Appendice A "Caratteristiche di Implementazione", nel manuale "Linguaggio Pascal Guida Utente".

Il Pascal Standard consente interi senza segno come etichette di istruzioni. Tali etichette hanno le stesse regole di visibilita' degli identificatori; a tal proposito si puo' vedere "Dominio degli Identificatori", piu' avanti in questo capitolo. Nelle etichette gli zeri iniziali non sono significativi.

Il Pascal di livello Esteso consente etichette che sono normali identificatori alfabetici.

Gli identificatori usati per un programma, modulo o unita', cosi' come gli identificatori con attributo PUBLIC o EXTERN, sono passati al linker. Il sistema operativo di una macchina su cui si vuol collegare (linking) o far eseguire un programma Pascal compilato, puo' imporre ulteriori restrizioni alla lunghezza degli identificatori usati come simboli globali dal linker.

Il listing del codice oggetto e la tabella dei simboli del debugger possono troncare gli identificatori di variabili e di procedure ai primi 6 caratteri.

La scrittura di programmi destinati ad essere usati con altri compilatori e sistemi operativi, richiede una ulteriore limitazione sul programma. In qualsiasi caso, un tale programma deve essere conforme alle limitazioni sugli identificatori.

Per la portabilita', in genere, si consigliano i seguenti accorgimenti:

- rendere tutti gli identificatori univoci nei loro primi otto caratteri
- rendere gli identificatori esterni univoci nei loro primi sei caratteri
- limitare le etichette di istruzioni a quattro cifre senza zeri iniziali

L'uso di identificatori di sette o meno caratteri permette di risparmiare spazio durante la compilazione.

Nota: Tutti gli identificatori usati internamente dal sistema runtime sono costituiti da quattro caratteri alfabetici seguiti da "QQ". Pero' questa forma deve essere evitata quando si creano nuovi nomi.

DICHIARAZIONE DI UN IDENTIFICATORE

Gli identificatori sono dichiarati e associati agli elementi del linguaggio nella sezione dichiarativa di un programma, modulo, interfaccia, implementazione, procedura o funzione. Esempi di identificatori, degli elementi che essi rappresentano e della sintassi utilizzata per dichiararli sono forniti nella seguente Tabella 5-1. Anche se variano i dettagli, la forma fondamentale della dichiarazione di un identificatore per ciascuno di questi elementi e' simile.

Elemento identificato	Identificatore	Dichiarazione
Programma	Z	PROGRAM Z (INPUT, OUTPUT)
Modulo	XXX	MODULE XXX
Interfaccia	UUU	INTERFACE; UNIT UUU
Implementazione	UUU	IMPLEMENTATION of UUU
Costante	DAYS	CONST DAYS = 365
Tipo	LETTERS	TYPE LETTERS = 'A'..'Z'
Campi di record	X, Y, Z	TYPE A = RECORD X, Y, Z : REAL END
Variabile	J	VAR J : INTEGER
Etichetta	1	LABEL 1
Etichetta	HAWAII	LABEL HAWAII
Procedura	BANG	PROCEDURE BANG
Funzione	FOO	FUNCTION FOO: INTEGER

Tabella 5-1 Dichiarazione di un Identificatore

DOMINIO DEGLI IDENTIFICATORI

Un identificatore e' definito dalla durata della procedura, funzione, programma, modulo, implementazione o interfaccia in cui esso e' dichiarato. Cio' e' vero per qualsiasi procedura o funzione nidificata. L'associazione ad un identificatore deve essere unica all'interno del suo dominio.

Una procedura o una funzione nidificata puo' ridefinire un identificatore solo se questo non e' gia' stato usato in esse. Comunque il compilatore non considera tale ridefinizione un errore, ma utilizza generalmente la prima definizione finche' non trova la seconda. Si ha un'eccezione che riguarda i tipi di riferimento: essa viene trattata in "Note sui Tipi di Riferimento", nel Capitolo 11.

IDENTIFICATORI PREDICHIARATI

Un certo numero di identificatori fanno gia' parte del linguaggio Pascal. Questa categoria include gli identificatori di tipi predichiarati, tipi super array, costanti, variabili di file, funzioni e procedure. Li si puo' usare liberamente senza dichiarazione. Tuttavia essi, a differenza delle parole riservate, possono essere ridefiniti quando si desidera.

Al livello Standard i seguenti identificatori sono predichiarati:

ABS	FALSE	OUTPUT	ROUND
ARCTAN	FLOAT	PACK	SIN
BOOLEAN	GET	PAGE	SQR
CHAR	INPUT	PRED	SQRT
CHR	INTEGER	PUT	SUCC
COS	LN	READ	TEXT
DISPOSE	MAXINT	READLN	TRUE
EOF	NEW	REAL	TRUNC
EOLN	ODD	RESET	UNPACK
EXP	ORD	REWRITE	WRITE
			WRITELN

Le caratteristiche al livello Esteso e di Sistema aggiungono i seguenti alla lista di identificatori predichiarati del Pascal.

1. Identificatori predichiarati propri di stringa

CONCAT	INSERT
COPYLIST	POSITN
COPYSTR	SCANEQ
DELETE	SCANNE

2. Identificatori predichiarati propri di livello Esteso

ABORT	HIBYTE
BYWORD	LOBYTE
DECODE	LOWER
ENCODE	RESULT
EVAL	SIZEOFF
	UPPER

3. Identificatori predichiarati propri di livello di Sistema

FILLC	MOVESL
FILLSC	MOVESR
MOVEL	RETYPE
MOVER	

4. Identificatori predichiarati di I/O di livello Esteso

ASSIGN	READFN
CLOSE	READSET
DIRECT	SEEK
DISCARD	SEQUENTIAL
FCBFQQ	TERMINAL
FILEMODES	

5. Identificatori predichiarati di tipo INTEGER4

BYLONG	LOWORD
FLOATLONG	MAXINT4
HIWORD	ROUNDLONG
INTEGER4	TRUNCLONG

6. Identificatori predichiarati di tipo super array

LSTRING
NULL
STRING

7. Identificatori predichiarati di tipo WORD

MAXWORD
WORD
WRD

8. Identificatori predichiarati misti

ADRMEM	INTEGER2
ADSMEM	REAL4
BYTE	REAL8
INTEGER1	SINT

6. COSTANTI

SOMMARIO

Questo capitolo prende in considerazione come il linguaggio Pascal tratta le costanti, come esse vengono dichiarate e quali sono le varie categorie di costanti.

INDICE

COS' E' UNA COSTANTE ?

Una costante e' un valore conosciuto prima dell'inizio di un programma, che non cambia con il procedere del programma. Come esempi di costanti si possono considerare il numero di giorni in una settimana, la propria data di nascita, il nome del proprio cane o le fasi lunari.

Ad una costante puo' essere assegnato un identificatore, ma non si puo' variare il valore associato a quell'identificatore durante l'esecuzione del programma. Quando si dichiara una costante, il suo identificatore diventa un sinonimo della costante stessa.

Ogni costante appartiene in modo implicito a qualche categoria di dati:

- Le costanti numeriche (descritte in "Costanti Numeriche") appartengono ad uno dei diversi tipi: REAL, INTEGER, WORD o INTEGER4

- Le costanti con caratteri (descritte in "Stringhe di Caratteri") sono stringhe di caratteri racchiuse tra apici e in Pascal sono chiamate "stringhe letterali"

- Disponibili al livello Esteso, le costanti strutturate (descritte piu' avanti in "Costanti Strutturate") comprendono array costanti, record e insiemi di tipi.

Sono anche disponibili al livello Esteso le espressioni costanti (descritte oltre, in "Espressioni Costanti") che consentono il calcolo di una costante basandosi sui valori delle costanti dichiarate precedentemente nelle espressioni.

Gli identificatori definiti in un tipo enumerato sono costanti di quel tipo e non possono essere usate direttamente con espressioni costanti numeriche (o stringa). Questi identificatori possono essere usati con le funzioni ORD, WRD o CHR (per esempio ORD (BLUE)). Al livello Esteso e' consentito inoltre leggere e scrivere direttamente gli identificatori di costante di un tipo enumerato come stringhe di caratteri.

TRUE e FALSE sono costanti predichiarate del tipo BOOLEAN e possono essere ridichiarate. NIL e' la costante di un qualunque tipo puntatore; pero', poiche' si tratta di una parola riservata, non puo' essere ridefinita. Inoltre, l'insieme nullo e' una costante di un qualunque tipo set.

Le etichette numeriche delle istruzioni non hanno nulla a che fare con le costanti numeriche; non si puo' usare un identificatore di costante o un'espressione come un'etichetta. Internamente, tutte le costanti hanno una lunghezza limitata per un massimo di 255 byte.

DICHIARAZIONE DI IDENTIFICATORI DI COSTANTE

La dichiarazione di un identificatore di costante presenta l'identificatore come sinonimo della costante. Questa dichiarazione deve essere inserita nella sezione CONST di un'unita' compilativa, procedura o funzione.

La forma generale di una dichiarazione di identificatore di costante e' costituita dall'identificatore seguito da un segno di uguale e dal valore costante. Il seguente frammento di programma comprende tre istruzioni che identificano costanti che iniziano dopo la parola "CONST":

```
PROGRAM DEMO (INPUT, OUTPUT);
CONST DAYSINYEAR = 365;
      DAYSINWEEK = 7;
      NAMEOFPLANET = 'EARTH';
```

In questo esempio i numeri 365 e 7 sono costanti numeriche; 'EARTH' e' una costante letterale di tipo stringa e deve essere racchiusa fra apici.

Quando si compila un programma, gli identificatori di costante non sono realmente definiti finche' le dichiarazioni non sono elaborate. Percio' una dichiarazione di costante come la seguente non ha alcun significato:

```
N = -N
```

Lo standard ISO definisce un ordine preciso da seguire per le dichiarazioni di un programma:

```
CONST MAX = 10;
TYPE NAME = PACKED ARRAY [1..MAX] OF CHAR;
VAR FIRST : NAME;
```

Il linguaggio Pascal non rispetta rigorosamente questo ordine, e consente piu' di un'istanza per ciascun tipo di dichiarazione:

```
TYPE  COMPLEX = RECORD R, I : REAL END;
CONST PII = COMPLEX (3.1416, 00);
VAR   PIX : COMPLEX;
TYPE  IVEC = ARRAY [1..3] OF COMPLEX;
CONST PIVEC = IVEC (PII, PII, COMPLEX (0.0, 1.0));
```

COSTANTI NUMERICHE

Le costanti numeriche sono numeri irriducibili come 45, 12.3 e 9E12. La notazione di una costante numerica indica generalmente il suo tipo: REAL, INTEGER, WORD o INTEGER4.

I numeri possono avere un segno iniziale piu' (+) o meno (-), tranne quando i numeri sono all'interno di espressioni. Quindi si ha che:

```
ALPHA := +10        {E' consentito}

ALPHA + -10         {Non e' consentito}
```

Inoltre non sono consentiti spazi vuoti all'interno delle costanti.

Il compilatore tronca qualunque numero che supera un certo numero massimo di caratteri, e da' un segnale di avvertimento quando cio' si verifica. La lunghezza massima di una costante (19 o 31) e' uguale alla lunghezza massima degli identificatori. Per quanto riguarda la lunghezza massima delle costanti e degli identificatori nella specifica versione del linguaggio, si puo' vedere l'Appendice A, "Caratteristiche di Implementazione" nel manuale "Linguaggio Pascal Guida Utente".

La sintassi delle costanti numeriche si applica non soltanto al testo corrente dei programmi, ma anche al contenuto dei file testo (textfile) letto da un programma.

Esempi di costanti numeriche:

```
 123                    0.17
+12.345                 007
-1.7E-10               -26.0
 17E+3                  26.0E12
-17E3                   1E1
```

Le costanti numeriche possono trovarsi in qualsiasi delle seguenti istanze:

- sezioni CONST
- espressioni
- clausole di tipo
- costanti set
- costanti strutturate
- costanti CASE di istruzioni CASE
- valori di indicatori di record varianti.

I differenti tipi di costanti numeriche sono descritti in dettaglio nelle sezioni che seguono.

COSTANTI REAL

Il tipo di un numero e' REAL se il numero include un punto decimale o un esponente. Il campo di variabilita' del valore REAL dipende dall'unita' numerica REAL della macchina. Generalmente il formato usato per il numero REAL e' quello IEEE oppure Microsoft. Questo dispone di circa sette cifre di precisione, con un valore massimo di circa 1.701411E38.

Vi e', tuttavia, una distinzione fra i valori REAL e le costanti REAL. Il campo di variabilita' delle costanti REAL puo' essere un sottoinsieme del campo di variabilita' dei valori REAL. Nel formato Microsoft le costanti numeriche REAL devono essere maggiori o uguali a 1.0E-38, e minori di 1.0E+38. Nel formato IEEE le costanti numeriche REAL sono in doppia precisione e percio' possono variare da circa 1E-306 a 1E306.

Il compilatore emette un segnale di avvertimento se non vi e' almeno una cifra su ogni lato di un punto decimale. Un numero REAL che inizia o termina con un punto decimale puo' essere fuorviante. Per esempio, poiche' la parentesi con punto sinistra sostituisce la parententesi quadra sinistra e la parentesi con punto destra sostituisce la parentesi quadra destra, la seguente costante:

```
(.1+2.)
```

e' interpretata come

```
[1+2]
```

La notazione scientifica nei numeri REAL (come in 1.23E-6 oppure 4E7) e' ammessa. Il punto decimale e il segno di esponente sono opzionali quando e' dato un esponente. Sia la lettera maiuscola "E" che la minuscola "e" sono consentite nei numeri REAL. "D" e "d" servono ad indicare un esponente; cio' fornisce compatibilita' con altri linguaggi.

Quando sono usati i formati IEEE REAL4 e REAL8, tutte le costanti reali sono memorizzate nel formato REAL8 (doppia precisione). Se si richiede una costante REAL4 in singola precisione, occorre dichiarare una variabile REAL4 e assegnarle il valore della costante reale nella sezione VALUE. (Si puo' anche assegnare a questa variabile l' attributo READONLY).

Le versioni del compilatore che operano su una macchina, ma generano codice per un'altra, possono perdere qualcosa di significativo riguardo le costanti REAL.

COSTANTI INTEGER, WORD e INTEGER4

Il tipo di una costante numerica non REAL puo' essere INTEGER, WORD, o INTEGER4. La tabella 6-1 indica il campo di variabilita' che le costanti di ciascuno di questi tipi possono assumere.

Tipo	Campo di variabilita' dei valori (minimo/massimo)	Identificatore di costante predichiarato
INTEGER	da -MAXINT a MAXINT	MAXINT=32767
WORD	da 0 a MAXWORD	MAXWORD=65535
INTEGER4	da -MAXINT a MAXINT4	MAXINT=2147483647

Tabella 6-1 Costanti Integer, Word e Integer4

MAXINT, MAXWORD e MAXINT4 sono tutti identificatori di costanti predichiarati.

Quando si dichiara un identificatore di costante numerica si verifica una delle tre cose:

1. Un identificatore di costante da -MAXINT a MAXINT diventa un INTEGER.

2. Un identificatore di costante da MAXINT+1 a MAXWORD diventa un WORD.

3. Un identificatore di costante da -MAXINT4 a -MAXINT-1 oppure da MAXWORD+1 a MAXINT4 diventa un INTEGER4.

Tuttavia qualunque costante di tipo INTEGER (comprese le espressioni e i valori costanti da -32767 a -1) se necessario si converte automaticamente al tipo WORD; se il valore INTEGER e' negativo, ad esso viene aggiunto il numero 65536 e il valore fondamentale di 16 bit non e' trasformato.

Per esempio, si puo' dichiarare un subrange di tipo WORD come WRD(0)..127; il limite superiore di 127 e' automaticamente assegnato al tipo WORD. Non e' vero il contrario: le costanti di tipo WORD non sono automaticamente convertite nel tipo INTEGER.

Le funzioni ORD e WRD trasformano inoltre il tipo di una costante ordinale in INTEGER o WORD. Inoltre qualunque costante INTEGER o WORD si converte automaticamente nel tipo INTEGER4 se necessario, ma non e' vero il contrario.

Esempi di conversioni significative sono forniti nella Tabella 6-2.

Costanti	Tipo assunto
0	INTEGER puo' diventare WORD o INTEGER4
-32768	Solo INTEGER4
32768	WORD puo' diventare INTEGER4
0..20000	Subrange di tipo INTEGER
0..50000	Subrange di tipo WORD
0..80000	Non consentito: nessun subrange di tipo INTEGER4
-1..50000	Non consentito: diventa 65535..50000 (cioe' -1 e' trattato come 65536)

Tabella 6-2 Conversioni di Costanti

Al livello Standard qualunque costante numerica (cioe' letterale o identificatore) puo' avere il segno piu' (+) o meno (-).

NUMERAZIONE NON DECIMALE

Al livello Esteso il Pascal ammette non soltanto la notazione di numero decimale ma anche numeri in esadecimale, ottale, binaria o di altra base (dove la base puo' variare da 2 a 36). Il segno di numero (#) si comporta come un separatore di radice.

Esempi di numeri in notazione non decimale:

```
16#FF02
10#987
 8#776
 2#111100
```

Gli zeri iniziali sono riconosciuti nell'ambito della radice, quindi un numero come 008#147 e' consentito. Nella notazione esadecimale sono consentite le lettere, maiuscole o minuscole, da A ad F. Una costante non decimale senza la radice (come #44) e' assunta come esadecimale. La notazione non decimale non richiede una costante WORD e puo' essere usata per costanti INTEGER, WORD o INTEGER4. Non si deve usare la notazione non decimale per le costanti REAL o per le etichette di istruzioni numeriche.

STRINGHE DI CARATTERI

La maggior parte dei manuali Pascal indica sequenze di caratteri racchiudendole tra apici come "stringhe". In questo Pascal esse sono chiamate "stringhe letterali" per distinguerle dalle stringhe di costanti, che possono essere espressioni o valori del tipo STRING.

Una stringa costante contiene da 1 a 255 caratteri. Se essa e' piu' lunga di un carattere e' di tipo PACKED ARRAY [1..n] OF CHAR, noto anche in Pascal come il tipo STRING (n). Una stringa costante che contiene solo un carattere e' di tipo CHAR. Tuttavia, se necessario, il tipo si converte da CHAR a PACKED ARRAY [1..1] OF CHAR (per esempio, STRING (1)). Per esempio, una costante ('A') di tipo CHAR puo' essere assegnata a una variabile di tipo STRING (1).

Un apostrofo letterale (apice) e' rappresentato da due apici adiacenti (per esempio, 'DONT''T GO'). La stringa nulla ('') non e' consentita. Una stringa letterale deve essere posizionata su di una riga. Il compilatore riconosce tali stringhe se racchiuse tra doppi apici (") o tra accenti (') invece che fra apici, ma emette un messaggio di avvertimento quando li incontra.

La caratteristica di espressione costante (descritta piu' avanti in "Espressioni Costanti") consente di costruire stringhe costanti con concatenazioni di altre stringhe costanti, compresi gli identificatori di stringhe costanti, la funzione CHR () e le costanti strutturate di tipo STRING. Cio' e' utile per rappresentare stringhe costanti che sono piu' lunghe di una riga e contengono caratteri non stampabili. Per esempio:

```
'THIS IS UNDERLINED' * CHR(13) * STRING (DO 18 OF '_')
```

La caratteristica LSTRING del Pascal aggiunge il tipo super array LSTRING. LSTRING e' simile a PACKED ARRAY [0..n] OF CHAR, ma l'elemento 0 contiene la lunghezza della stringa, che puo' variare da 0 ad un massimo di 255. (Per una descrizione dei tipi LSTRING si puo' vedere la parte su "LSTRING", nel Capitolo 9). Per ora e' sufficiente osservare che, se necessario, una costante di tipo STRING (n) o CHAR si converte automaticamente nel tipo LSTRING.

NULL e' una costante predichiarata per il tipo LSTRING nullo, con l'elemento 0 (l'unico) uguale a CHR (0). NULL non puo' essere concatenato poiche' non e' di tipo STRING, e costituisce la sola costante di tipo LSTRING.

Esempi di dichiarazioni di stringhe letterali:

```
NAME = 'John Jacob';          {stringa letterale consentita}
LETTER = 'Z';                 {LETTER e' di tipo CHAR}
QUOTED_QUOTE = '''';          {apici}
NULL_STRING = NULL;           {consentito}
NULL_STRING = '';             {non consentito}
DOUBLE = "OK";                {genera messaggio di avvertimento}
```

COSTANTI STRUTTURATE

Il Pascal Standard consente solo costanti numeriche e stringa gia' indicate, il valore costante NIL di puntatore ed insiemi di costanti non appartenenti a tipi.

Al livello Esteso del Pascal, si possono usare costanti di tipo array, record e set. Le costanti strutturate possono essere usate dove e' consentito un valore strutturato, sia nelle espressioni che nelle sezioni CONST e VALUE.

- Una costante di tipo array e' costituita da un identificatore di tipo seguito da una lista di valori costanti fra parentesi separati da virgole.

  Esempio di costante di tipo array:

```
TYPE   VECT_TYPE = ARRAY [-2..2] OF INTEGER;
CONST  VECT = VECT_TYPE (5, 4, 3, 2, 1);
VAR    A : VECT_TYPE;
VALUE  A := VECT;
```

- Una costante di tipo record e' costituita da un identificatore di tipo seguito da una lista di valori costanti fra parentesi separati da virgole.

  Esempio di costante di tipo record:

```
TYPE REC_TYPE = RECORD
                A, B: BYTE;
                C, D: CHAR
                END;
CONST RECR = REC_TYPE (#20, 0, 'A', CHR (20));
VAR   FOO : REC_TYPE;
VALUE FOO := RECR;
```

- Una costante di tipo set e' costituita da un identificatore opzionale di tipo set seguito dagli elementi della costante set racchiusi tra parentesi quadre. Gli elementi della costante set sono separati da virgole. Un elemento di una costante set e' una costante ordinale oppure due costanti ordinali separate da due punti per indicare un intervallo di valori costanti.

  Esempio di costante di tipo set:

```
TYPE  COLOR_TYPE = SET OF (RED, BLUE, WHITE, GREY, GOLD);
CONST SETC = COLOR_TYPE [RED, WHITE..GOLD];
VAR   RAINBOW : COLOR_TYPE;
VALUE RAINBOW := SETC;
```

Una costante all'interno di una costante strutturata di tipo array o valore.

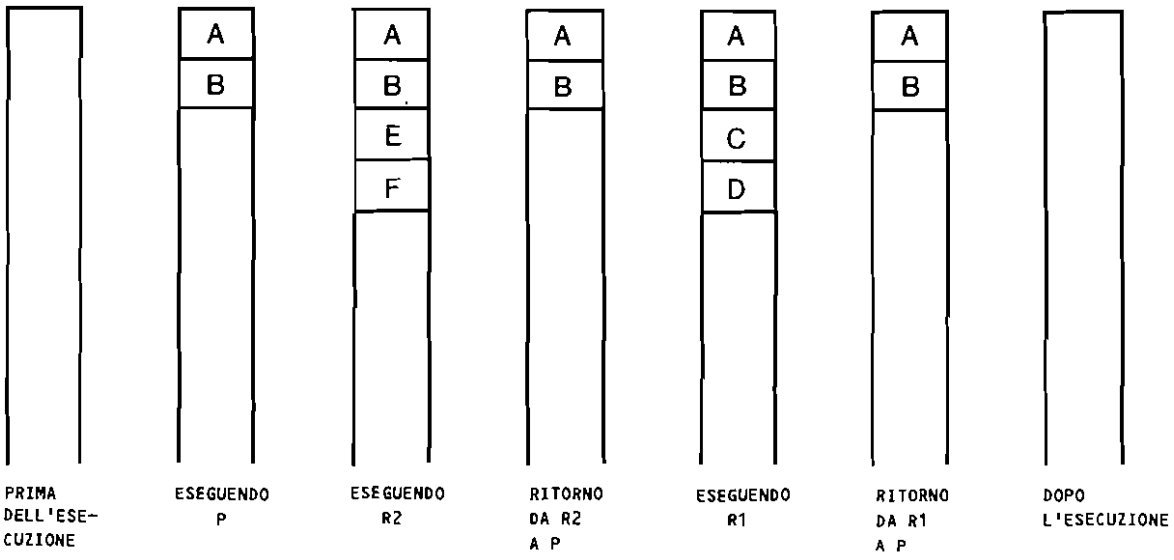


Fig. 1-16 Evoluzione Temporale del Run Time Stack

Nei due esempi precedenti si puo' osservare che le variabili globali (A e B) sono sempre allocate durante l'esecuzione del programma, mentre prima e dopo l'esecuzione non e' allocata alcuna variabile.
Cio' significa che un programma Pascal non puo' comunicare con l'ambiente esterno attraverso le sue variabili.
In Pascal, le tecniche di input/output sono basate su un particolare tipo di variabile chiamata "file", che non viene allocata nell'area RTS.

Infine si consideri il meccanismo di allocazione nel caso di routine ricorsive, ad esempio:

```
PROGRAM P;

    PROCEDURE R1;
      VAR A.....
      BEGIN
        .
        .
        .
        chiamata alla routine R1 (*RICORSIONE*)
        .
        .
        .
    END;

  BEGIN
    .
    .
    .
    chiamata alla routine R1 (*PRIMA CHIAMATA*)
    .
    .
    .
END.
```

Fig. 1-17 Routine Ricorsiva

In tal caso viene allocata una nuova copia di A ogni volta che viene eseguita una chiamata a R1 e deallocata dopo ogni ritorno da essa. Se R1 chiama se stessa due volte ricorsivamente, l'evoluzione dell'area RTS e' la seguente:

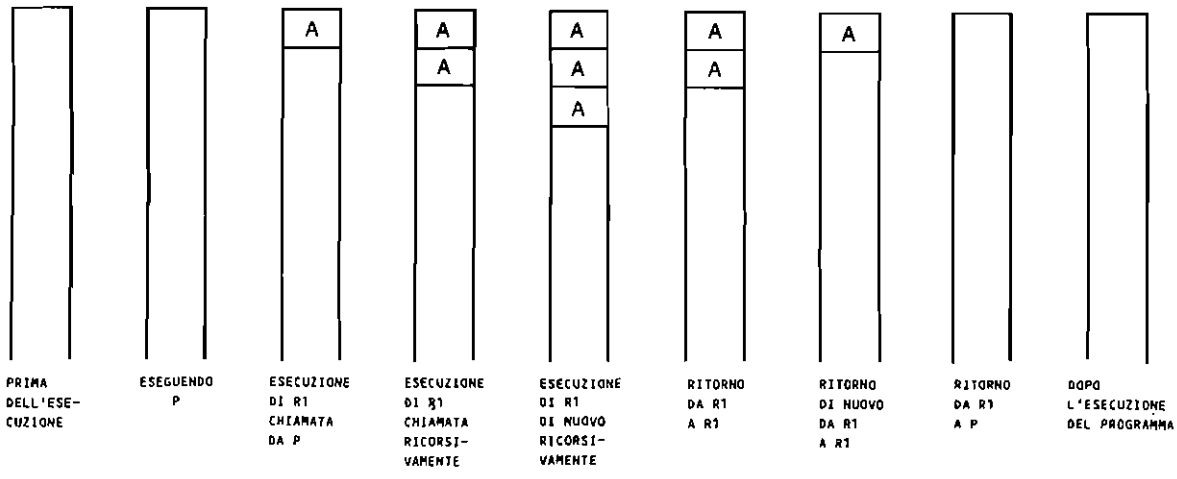


Fig. 1-18 Evoluzione Temporale dell'area RTS Durante l'Esecuzione di una Routine Ricorsiva

Si deve osservare che la stessa variabile A e' allocata piu' volte. Cio'

non crea ambiguita' poiche' ogni chiamata di una routine ricorsiva implica l'allocazione di una copia privata delle proprie variabili.

L'AREA HEAP

La tecnica di allocazione vista precedentemente, sebbene dinamica, non e' direttamente controllata dalle istruzioni del programma, poiche' le variabili sono allocate e deallocate automaticamente a causa delle chiamate di routine.

Il linguaggio Pascal fornisce un altro tipo di variabili dinamiche, allocate indipendentemente dalle chiamate di routine tramite l'esecuzione di istruzioni specifiche.

Tali variabili non possono essere allocate nell'area RTS, poiche' non devono avvertire l'effetto delle chiamate di routine; quindi vengono allocate in una diversa area di memoria chiamata HEAP.

Inoltre non ci si puo' riferire ad esse direttamente, ma solo attraverso un particolare tipo di variabile, allocata nell'area RTS, chiamato "puntatore".

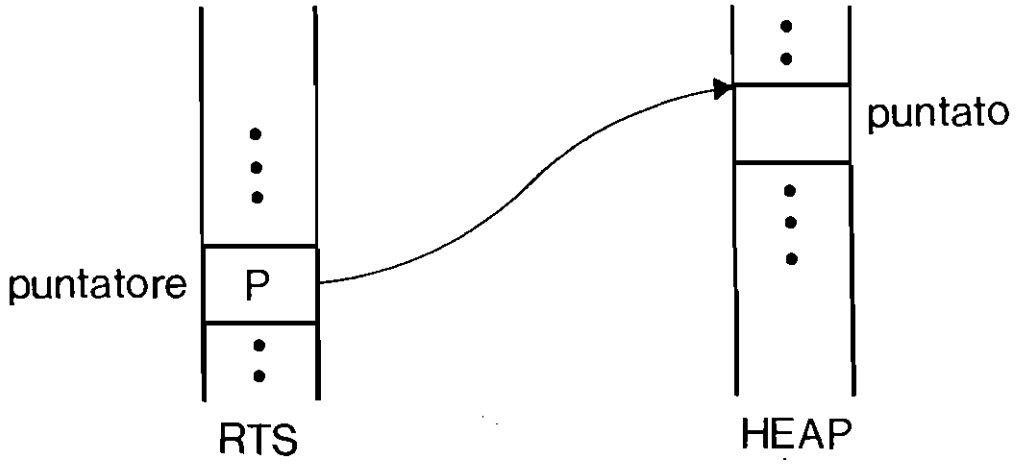


Fig. 1-19 Run Time Stack e Heap

Durante l'esecuzione del programma, e' possibile allocare nuove variabili nello heap usando un' apposita istruzione ("new"). Tale istruzione puo' essere eseguita indifferentemente all'interno o all'esterno di una routine.

Questo tipo di variabili dinamiche consente la costruzione di strutture di dati piu' complesse, come liste, alberi, code ecc.

Sebbene le variabili dinamiche possono rimanere nello heap anche dopo la fine dell'esecuzione del programma, non possono essere usate per

l'input/output poiche', dopo la deallocazione del puntatore dall'area RTS, non sono piu' accessibili.

COMPILAZIONI SEPARATE E LIBRERIE DI RISORSE

Come precedentemente descritto, si e' sempre considerato un programma come un' unita' compilabile ed eseguibile in modo autonomo. Tuttavia la manutenzione di un grande programma risulta facilitata da compilazioni separate di alcune delle sue risorse; cosi' la modifica di una di tali "risorse esterne" non richiede la ricompilazione dell'intero programma.

Inoltre tali risorse esterne possono essere utilizzate da diversi programmi semplicemente "importandole" attraverso particolari parole chiave, senza richiedere l'inclusione del loro testo.

Cos' e' una Libreria di Risorse

Un programma Pascal puo' essere costruito separando fisicamente la descrizione di alcune delle sue risorse.
Le risorse descritte separatamente costituiscono la "libreria di risorse".

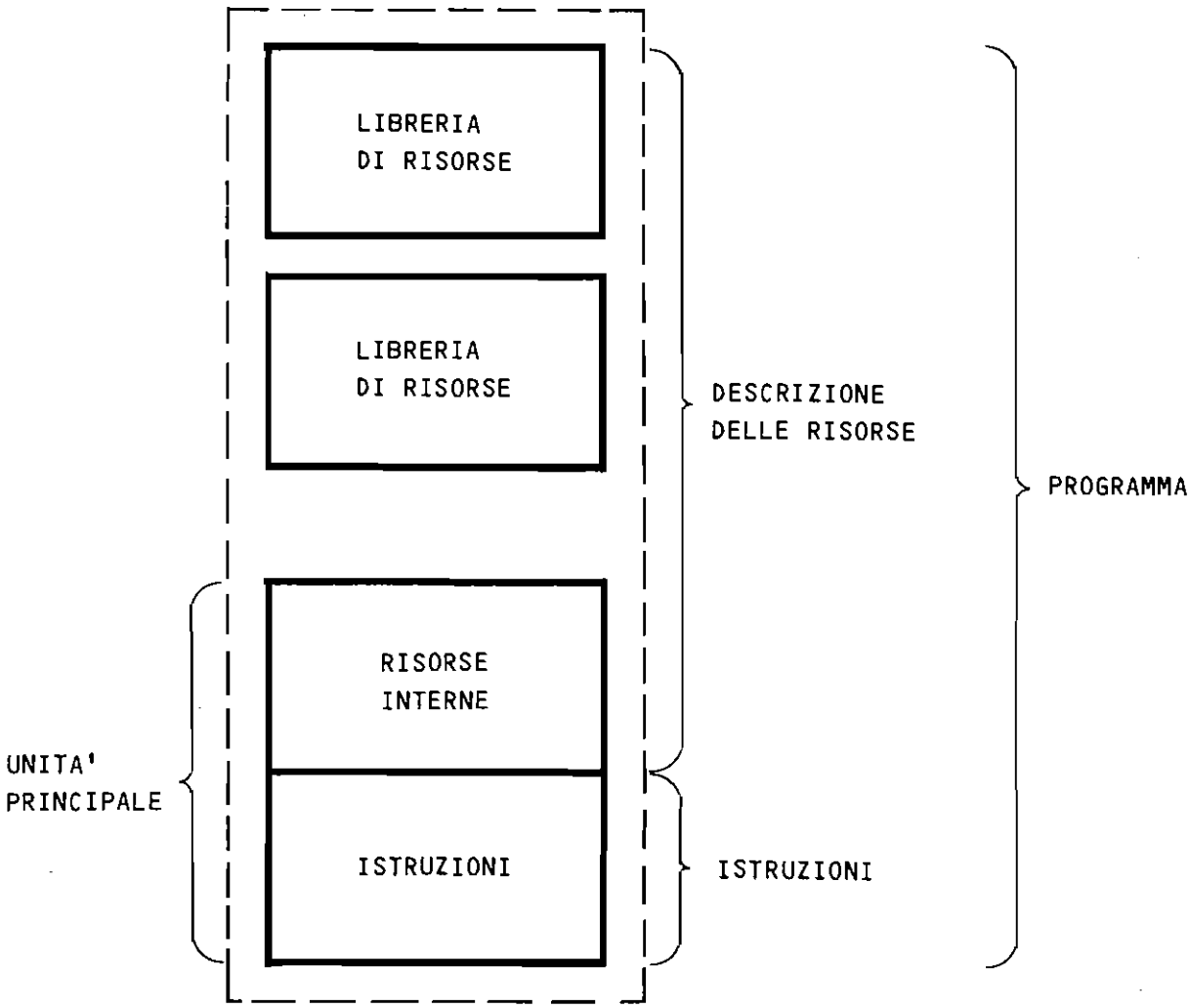


Fig. 1-20 Un Programma Pascal

Una libreria di risorse e' strutturalmente identica alla parte descrizione di risorse di un programma Pascal, quindi puo' contenere la descrizione di alcune costanti, tipi, variabili, procedure e funzioni.

L'unica differenza e' che essa inizia con la parola chiave MODULE e non contiene la parte istruzioni.

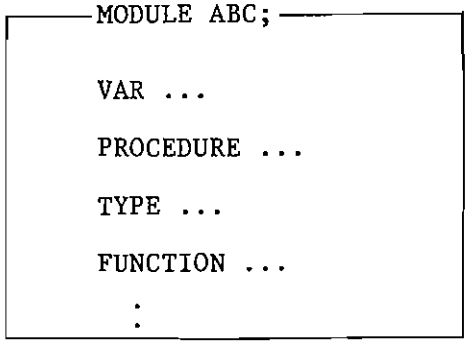


Fig. 1-21 Una Libreria di Risorse

E' opportuno indicare come "unita' principale" l'unita' contenente le istruzioni.

Quindi un programma Pascal e' costituito da una raccolta di unita' di compilazione che contengono alcune "librerie di risorse" ed una e una sola "unita' principale".

Le librerie di risorse possono essere compilate separatamente ma, naturalmente, non sono eseguibili; devono essere concatenate (operazione di linking) ad un' unita' principale che le "importi".

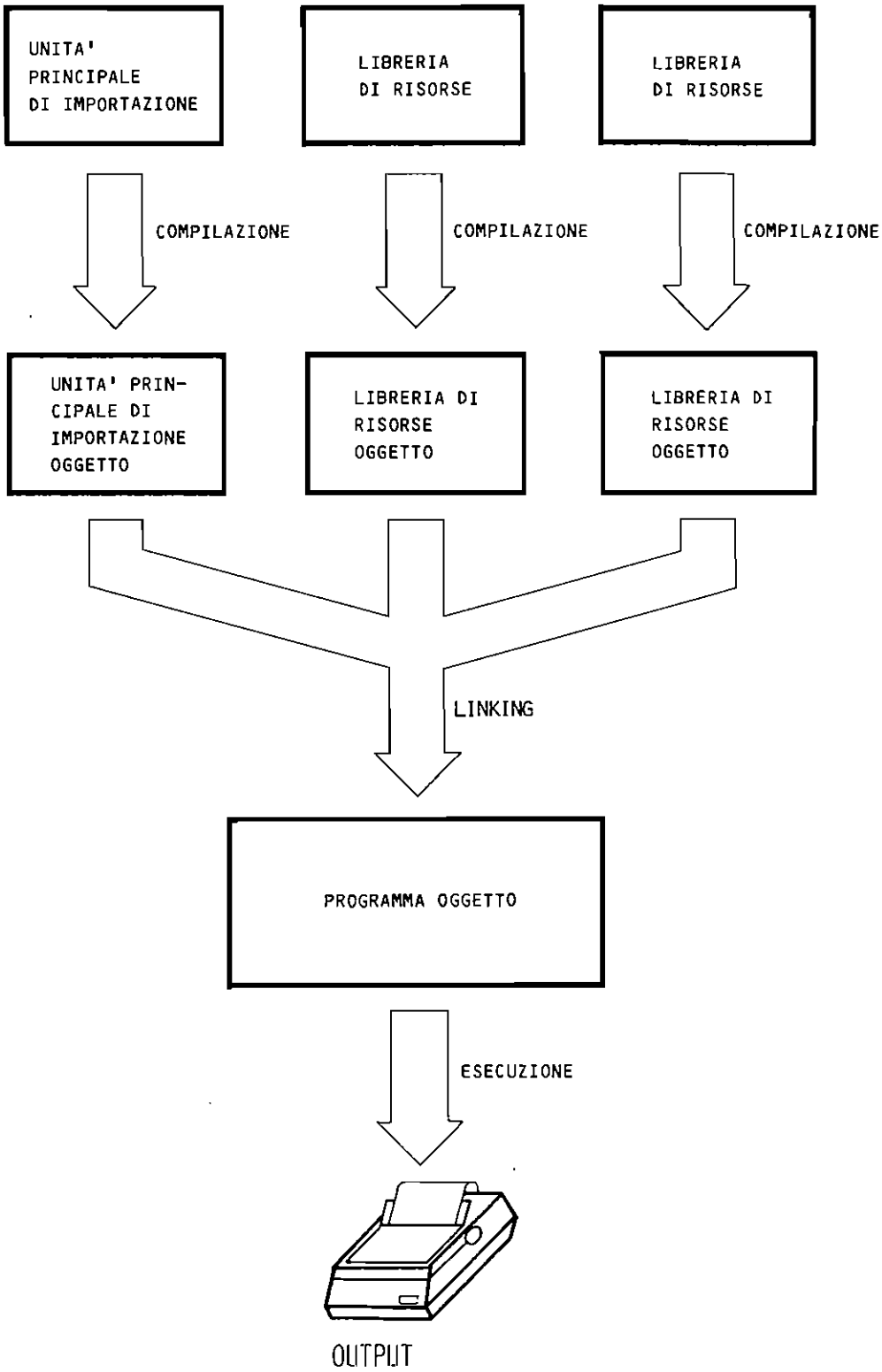


Fig. 1-22 Compilazione, Linking ed Esecuzione di un Programma Pascal

La stessa libreria di risorse puo' essere concatenata a piu' unita' principali di importazione.

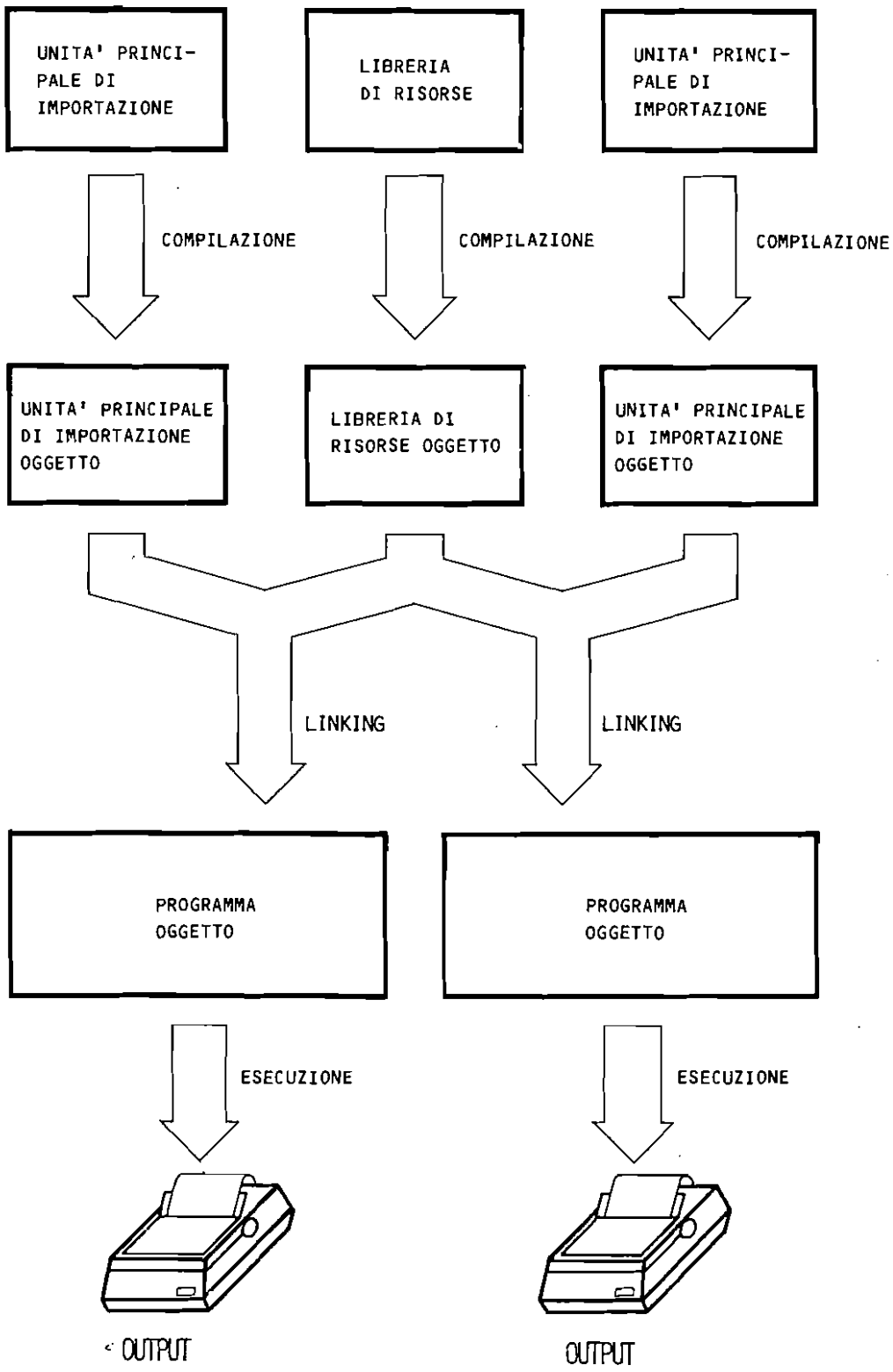


Fig. 1-23 Linking di una Libreria di Risorse con due Unita' Principali

Si considerano ora le regole di visibilita' per le librerie di risorse: un' unita' principale ed una libreria di risorse sono "fratelli", quindi l'una non puo' usare le risorse interne dell'altra.

Tuttavia alcune risorse della libreria possono essere viste dall'unita' principale se sono esplicitamente "esportate" dalla libreria ed "importate" nell'unita' principale, e viceversa.

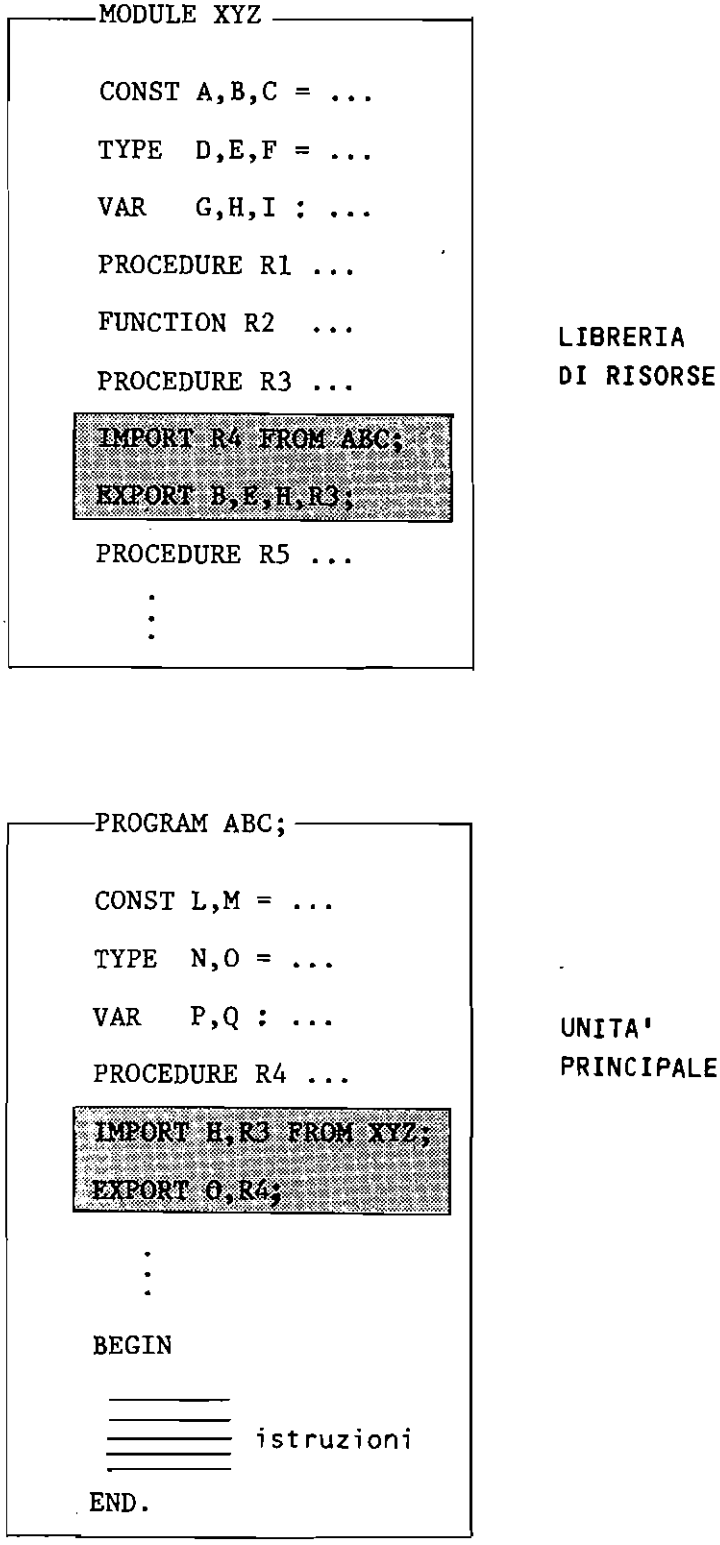


Fig. 1-24 Esempio di Meccanismo di Import/Export

Nell'esempio precedente, le istruzioni dell'unita' principale possono chiamare la routine R3 oppure utilizzare la variabile H poiche', sebbene descritte nella libreria di risorse, esse sono esportate da XYZ ed importate in ABC. La procedura R3 della libreria di risorse puo' chiamare R4 poiche' e' importata da ABC.

L'associazione tra risorse importate ed esportate e' basata sulla corrispondenza tra i loro nomi.

Tutte le risorse non esportate sono completamente nascoste all'ambiente esterno e si possono considerare protette da accessi illegali.

Esistono due usi principali delle librerie di risorse:

- routine separate;
- tipi di dati protetti;

Routine Separate

Una libreria di risorse puo' essere utilizzata per contenere un insieme di routine logicamente correlate.

Ad esempio, e' possibile costruire una libreria matematica che include alcune funzioni o procedure non predefinite.

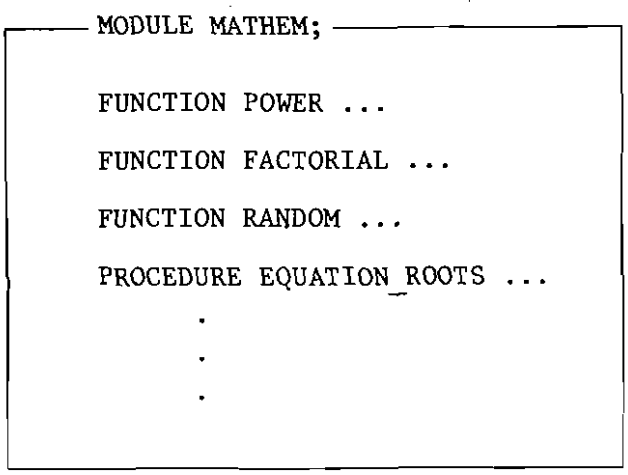


Fig. 1-25 Esempio di Routine Separate

Un altro esempio puo' essere una libreria per la gestione di periferiche.

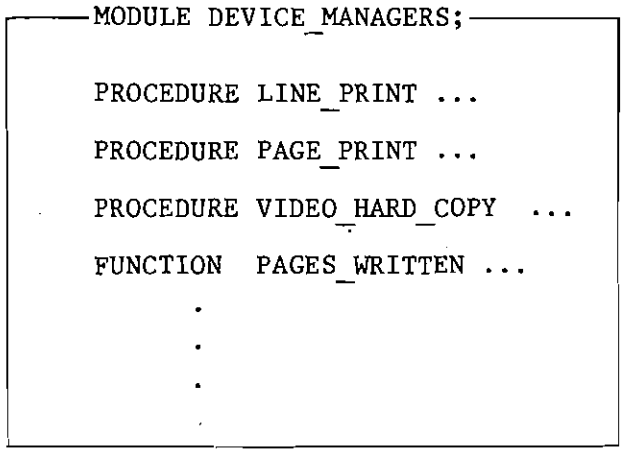


Fig. 1-26 Esempio di Routine Separate

Infine, per grandi progetti che coinvolgono parecchi programmatori, le librerie di risorse consentono una separazione fisica dei testi dei programmi in modo tale da riflettere la divisione del lavoro e delle responsabilita'.
Una volta stabilita e memorizzata l'interfaccia comune tra due qualsiasi parti, queste possono essere sviluppate e compilate separatamente.
L'interfaccia comune e' costituita da un testo fisicamente separato, e cio' assicura che ricompilazioni distinte di ciascuna parte non invalidino tale interfaccia.

Tipi di Dati Protetti

Una libreria di risorse puo' essere usata come una descrizione generalizzata di un tipo strutturato.

Un tipo strutturato e' un insieme di valori ciascuno dei quali costituito da alcuni valori di un tipo base; l'operazione di selezione permette l'accesso a valori del tipo base, talvolta chiamati "elementi".

Una libreria di risorse puo' definire un tipo strutturato descrivendo semplicemente alcune variabili che possono contenere un insieme di valori, ed alcune routine che consentono l'accesso agli elementi, o altre operazioni associate.

```
MODULE BYNARY_TREE;

VAR  A: ...

PROCEDURE INSERT ...

   :
   :
   :
PROCEDURE SEARCH ...

   :
   :
   :

EXPORT INSERT, SEARCH;
```

Fig. 1-27 Esempio di Tipo di Dati Protetti: un Albero Binario

Nell'esempio precedente, la libreria di risorse definisce il tipo strutturato "albero binario di...", le cui operazioni di selezione sono "insert" e "search".

Ovviamente vengono esportate solo le operazioni e non le variabili, ci si garantisce cosi' contro possibili accessi illegali agli elementi dell'albero.

Altri possibili esempi sono i tipi strutturati "matrix of...", "stack of...".

```
MODULE MATRIX;

CONST C = ...

VAR   A : ...

PROCEDURE SUM ...
   :
   :
   .

PROCEDURE TIMES ...
   :
   :
   .

EXPORT ...
```

```
MODULE STACK;

TYPE T = ...

VAR  A : ...

PROCEDURE PUSH ...
   :
   :
   .

PROCEDURE POP ...
   :
   :
   .

EXPORT ...
```

Fig. 1-28 Esempi di Tipi di Dati Protetti

Si puo' aggiungere infine che una libreria di risorse e' utilizzabile come una definizione di costruttore, cioe' come un tipo strutturato il cui tipo base e' parametrizzato.

Nell'esempio che segue, la libreria di risorse e' usata sia come "coda di caratteri" che come "coda di colori":

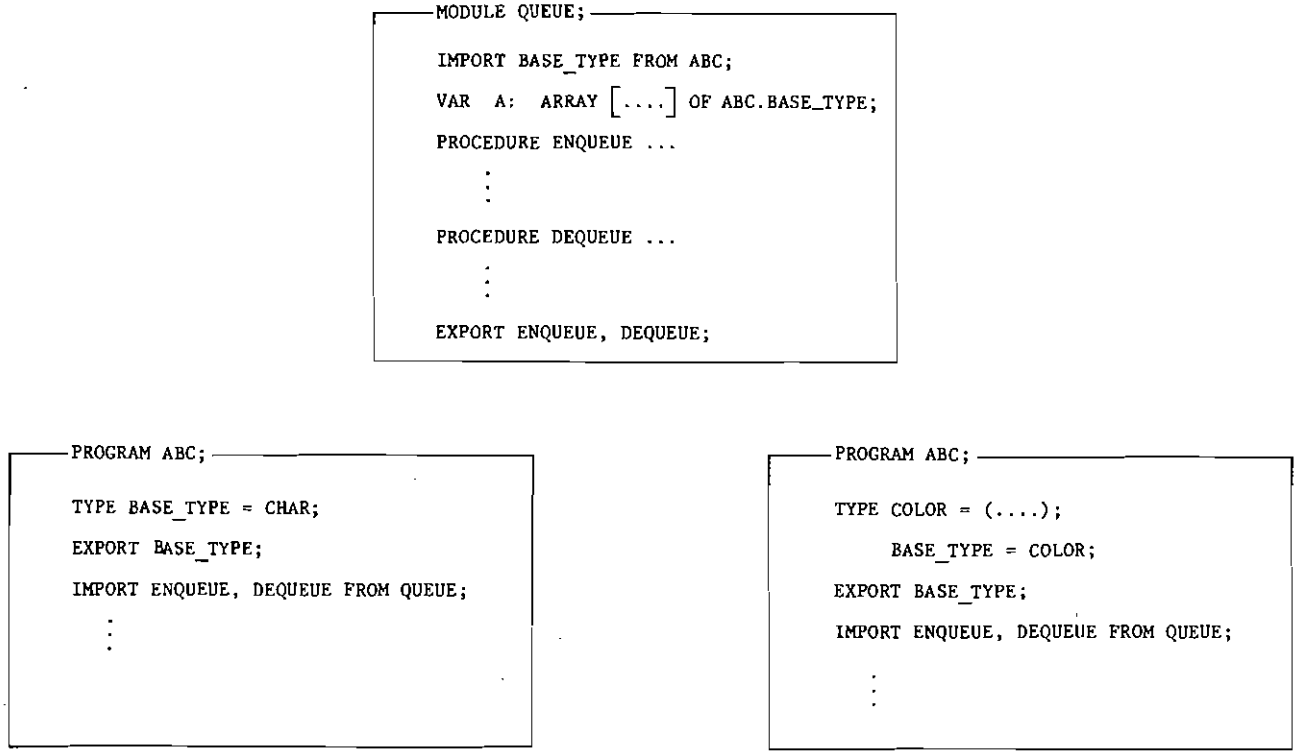


Fig. 1-29 Libreria di Risorse vista come un Costruttore

La libreria di risorse importa il tipo base e lo usa nella descrizione della coda.

UNITA' PRINCIPALE DEL PROGRAMMA

L'unita' principale e' la sola unita' che contiene una sezione istruzioni in un programma. L'unita' principale ha la seguente struttura:

- la parola chiave PROGRAM ;
- l'identificatore, che costituisce il nome dell'unita' principale;
- la lista dei parametri del programma, se ce n'e' una, che definisce le risorse disponibili prima e dopo l'esecuzione del programma;
- un carattere di punto e virgola ";" ;
- la descrizione delle risorse e' costituita dalla seguente lista delle risorse usate nelle istruzioni del programma:
  - . etichette
  - . costanti

. tipi

. variabili

. procedure

. funzioni.

La descrizione delle risorse puo' contenere qualsiasi numero di descrizioni, anche nessuna.
Si deve osservare che alcune delle risorse sopra elencate possono essere esportate o importate nelle librerie di risorse, per cui esse devono essere elencate nelle due liste corrispondenti:

. lista di export

. lista di import.

- la sezione istruzioni dell'unita' principale, preceduta dalla parola chiave BEGIN e terminante con la parola chiave END. Questa parte della struttura e' chiamata istruzione composta.

MODULI

Modulo e' il nome che indica una libreria di risorse; la sua struttura e' simile a quella dell'unita' principale, senza pero' contenere la sezione istruzioni.

Esso e' costituito dai seguenti elementi:

- la parola chiave MODULE
- l'identificatore, che e' il nome del modulo
- il carattere di punto e virgola ";"
- la descrizione delle risorse, che e' la stessa dell'unita' principale
- un punto "."

record deve avere un tipo da assegnare al corrispondente tipo componente. Per i record con varianti, il valore di un elemento costante corrispondente ad un indicatore di campo seleziona una variante, anche se l'indicatore di campo e' vuoto. Il numero degli elementi costanti deve essere uguale al numero dei componenti nella struttura, tranne per le costanti strutturate di tipo super array; sono consentite costanti strutturate nidificate.

Una costante array o record nidificata entro un'altra costante strutturata deve avere sempre l'identificatore del tipo precedente. Percio' una costante super array puo' avere soltanto una dimensione (per una descrizione dei super array si puo' vedere "Super array", nel Capitolo 9). La capacita' di rappresentazione di una costante strutturata deve essere da 1 a 255 byte. Se questo limite di 255 byte costituisce un problema, si puo' dichiarare una variabile strutturata con l'attributo READONLY e inizializzare i suoi componenti nella sezione VALUE.

Esempio di una costante strutturata complessa:

```
TYPE R3 = ARRAY [1..3] OF REAL;
TYPE SAMPLE = RECORD I: INTEGER;
                     A: R3;
                     CASE BOOLEAN OF
                     TRUE: (S: SET OF 'A'..'Z';
                            P: ^SAMPLE);
                     FALSE: (X: INTEGER)
              END;
CONST SAMP_CONST= SAMPLE (27, R3 (1.4, 1.4, 1.4),
                  TRUE, ['A', 'E', 'I'], NIL);
```

Gli elementi costanti possono essere ripetuti con la frase DO <n> OF <costante>, quindi l' esempio precedente puo' includere "DO 3 OF 1.4" invece di "1.4, 1.4, 1.4".

Il Pascal non ammette espressioni costanti di tipo set come [' '] + LETTERS, o espressioni costanti di file. La costante 'ABC' di tipo STRING (3) e' equivalente alla costante strutturata STRING ('A', 'B', 'C'). Le costanti srutturate LSTRING non sono ammesse; al loro posto si devono usare le corrispondenti costanti STRING.

Le costanti strutturate (e altri valori strutturati come le variabili e i valori ritornati dalle funzioni) possono essere passate per riferimento usando parametri CONST. Per ulteriori informazioni si puo' vedere nel Capitolo 15 la parte sui "Parametri Procedurali e Funzionali".

Vi sono due varieta' di costanti di tipo set: una di tipo esplicito, come ad esempio CHARSET ['A'..'Z'], e una di tipo non conosciuto, come ad esempio [20..40]. Si possono usare in un'espressione oppure per definire il valore di un identificatore di costante. Le costanti set di tipo esplicito possono inoltre essere passate come parametri di riferimento (CONST). Gli insiemi di tipo non conosciuto non sono impaccati ma il tipo, se necessario, si converte in PACKED. Passare insiemi per riferimento e' generalmente piu' efficace che passarli come parametri

ESPRESSIONI COSTANTI

Le espressioni costanti consentono il calcolo di costanti basate sui valori di costanti dichiarate precedentemente all'interno di espressioni. Le espressioni costanti possono anche essere all'interno di istruzioni di un programma.

Esempio di dichiarazione di espressione costante:

```
CONST HEIGHT_OF_LADDER = 6;
      HEIGHT_OF_MAN    = 6;
      REACH = HEIGHT_OF_LADDER + HEIGHT_OF_MAN;
```

Poiche' un'espressione costante puo' contenere soltanto costanti dichiarate precedentemente, la seguente dichiarazione non e' consentita:

```
CONST MAX = A + B;
      A   = 10;
      B   = 20;
```

Alcune funzioni possono essere usate all'interno di espressioni costanti. Per esempio:

```
CONST A = LOBYTE (-23) DIV 23;
      B = HIBYTE (-A);
```

La tabella 6-3 indica le funzioni e gli operatori che si possono usare con le costanti REAL, INTEGER, WORD e altre costanti ordinali, come le costanti enumerate e subrange.

Tipo di operando	Funzioni e operatori
REAL, INTEGER	Unario piu' (+) Unario meno (-)
INTEGER, WORD	+ DIV OR HIBYTE() - MOD NOT LOBYTE() * AND XOR BYWORD()
Tipi ordinali	< <= CHR() LOWER() > >= ORD() UPPER() = <> WRD()
Boolean	AND NOT OR
ARRAY	LOWER() UPPER()
Qualunque tipo	SIZEOF() RETYPE()

Tabella 6-3 Operatori e Funzioni Costanti

Esempi di espressioni costanti:

```
CONST FOO = (100 + ORD('X')) * 8#100 + ORD('Y');
      MAXSIZE = 80;
      X = (MAXSIZE > 80) OR (IN_TYPE = PAPERTAPE);
      {X e' una costante BOOLEANA}
```

Oltre agli operatori indicati nella Tabella 6-3 per le costanti numeriche, si puo' anche usare l'operatore di concatenazione di stringa (*) con stringhe costanti, nel modo seguente:

```
CONST A = 'abcdef';
      M = CHR (109);   {CHR e' consentito}
      ATOM = A * 'ghijkl' * M;
      {ATOM = 'abcdefghijlm'}
```

Queste costanti possono occupare piu' di una riga, ma sono sempre limitate ad un massimo di 255 caratteri. Queste espressioni di stringhe costanti sono consentite ogni volta che e' consentita una stringa letterale, tranne che nei metacomandi.

7. INTRODUZIONE AI TIPI DI DATI

SOMMARIO

Questo capitolo contiene un'introduzione ai tipi di dati, descrive i gruppi di categorie cui appartengono e come essi vengono dichiarati. Tale argomento viene trattato in modo piu' approfondito nei capitoli successivi.

INDICE

COS'E' UN TIPO ?

Un tipo e' l'insieme di valori che una variabile o un valore possono avere all'interno di un programma. I tipi sono predichiarati o dichiarati esplicitamente.

Per esempio i tipi INTEGER e REAL sono predichiarati, mentre il tipo ARRAY [1..10] OF INTEGER e' dichiarato esplicitamente. Ad un tipo dichiarato esplicitamente puo' anche essere assegnato un identificatore di tipo, pero' per svolgere quest'ultimo compito e' richiesta una dichiarazione di tipo.

Nel linguaggio Pascal i tipi rientrano in tre grandi categorie: semplici, strutturati e di riferimento. La tabella 7-1 fornisce una classificazione dei tipi in ciascuno di questi gruppi. Il resto del capitolo descrive i tipi in generale; dal Capitolo 8 al Capitolo 11 sono trattati in dettaglio i diversi gruppi.

Categoria	Include	Commenti/Esempi
Tipi semplici	Tipi ordinali	
	INTEGER	-MAXINT..MAXINT
	WORD	0..MAXWORD
	CHAR	CHR(0)..CHR(255)
	BOOLEAN	(FALSE,TRUE)
	tipi enumerati	es., (RED,BLUE)
	tipi subrange	es., 100..5000
	REAL4, REALB	
	INTEGER4	-MAXINT4..MAXINT4
Tipi strutturati	ARRAY OF tipo	
	generico (OF di ogni tipo)	
	SUPER ARRAY (OF tipo)	
	STRING (n)	[1..n] of CHAR
	LSTRING (n)	[0..n] of CHAR
	RECORD	
	SET OF tipo	
	FILE OF	
	file (binary) generici	
	TEXT	Come FILE OF CHAR
Tipi di riferimento	Tipi puntatore	es., ^TREETIP
	ADR OF tipo	Indirizzo relativo
	ADS OF tipo	Indirizzo segmentato
Tipi procedurali e funzionali		Solo come tipo di parametro

Tabella 7-1 Categorie di Tipi nel Pascal

DICHIARAZIONE DEI TIPI DI DATI

La dichiarazione di tipo associa un identificatore con un tipo di valore. I tipi si dichiarano nella sezione TYPE di un programma, di una procedura, funzione, modulo, interfaccia o implementazione (non nell'intestazione di una procedura o funzione).

Una dichiarazione di tipo e' costituita da un identificatore seguito da un segno di uguale e da una clausola di tipo.

Esempi di definizioni di tipo:

```
TYPE LINE = STRING (80);
     PAGE = RECORD
              PAGENUM : 1..499;
              LINES : ARRAY [1..60] OF LINE;
              FACE : (LEFT, RIGHT);
              NEXTPAGE : ^PAGE
            END;
```

Dopo la dichiarazione dei tipi di dati, si dichiarano le variabili dei tipi gia' definiti nella sezione VAR di un programma, procedura, funzione, modulo o interfaccia, o nell'intestazione di una procedura o funzione. Nell'esempio seguente la sezione VAR dichiara le variabili dei tipi definiti nella sezione TYPE che precede:

```
VAR PARAGRAPH : LINE;
    BOOK : PAGE;
```

Poiche' un identificatore di tipo non e' definito finche' la sua dichiarazione non e' elaborata dal compilatore, una dichiarazione ricorsiva di tipo come la seguente non e' consentita:

```
T = ARRAY [0..9] OF T;
```

Si ha un'eccezione a tale regola ed e' trattata nel Capitolo 11, "Tipi di Riferimento e Altri Tipi", insieme ad ulteriori dettagli sui tipi di riferimento.

Una caratteristica particolare del Pascal e' la categoria chiamata "tipi super". Una dichiarazione di tipo super determina l'insieme dei tipi che gli indicatori di quel tipo super possono assumere; essa inoltre associa un identificatore con il tipo super. Le dichiarazioni di tipo super si hanno anche nella sezione TYPE. I soli tipi super attualmente disponibili nel linguaggio Pascal sono i super array.

COMPATIBILITA' DI TIPO

Per quanto riguarda la compatibilita' di tipo, il linguaggio Pascal segue lo standard ISO, con qualche regola in piu' per i tipi super array, i tipi LSTRING e le limitazioni sulle costanti (cioe' i cambiamenti forzati nel tipo di una costante). Le funzioni di trasferimento di tipo, che annullano le regole sui tipi, sono disponibili tramite alcune prestazioni del Pascal.

Due tipi possono essere "identici", "compatibili" o "incompatibili". Un'espressione puo', o non puo', essere "compatibile in assegnazione" con una variabile, con un parametro di valore o con un indice di array.

IDENTITA' DI TIPI E PARAMETRI DI RIFERIMENTO

Due tipi sono identici se hanno identici identificatori oppure se gli identificatori sono dichiarati equivalenti per mezzo di una definizione di tipo come la seguente:

```
TYPE T1 = T2;
```

Nel Pascal i tipi "identici" sono realmente identici: nell'esempio appena visto non c'e' differenza tra i tipi T1 e T2 . L'identita' di tipo e' basata sul nome dei tipi, e non sul modo in cui essi sono dichiarati o strutturati. Quindi, ad esempio, T1 e T2 non sono identici nelle dichiarazioni che seguono:

```
TYPE T1 = ARRAY [1..10] OF CHAR;
     T2 = ARRAY [1..10] OF CHAR;
```

I parametri di riferimento attuali e formali devono appartenere allo stesso tipo. Se un parametro formale di riferimento e' di tipo super array, il parametro attuale deve essere dello stesso tipo super array oppure di un tipo derivato da esso. I due tipi record o array devono essere identici per l'assegnazione.

La sola eccezione riguarda le stringhe, dove i parametri attuali di tipo CHAR, STRING, STRING (n), LSTRING e LSTRING (n) sono compatibili con un parametro formale del tipo super array STRING. Inoltre il tipo di una costante stringa si trasforma in qualsiasi tipo LSTRING con un limite sufficientemente grande. Per esempio il tipo di

Inoltre un parametro attuale di qualunque tipo FILE puo' essere passato a un parametro formale di uno speciale tipo RECORD FCBFQQ. Allo stesso modo, un parametro attuale di tipo FCBFQQ puo' essere passato ad un parametro formale di qualunque tipo file. Per una descrizione del tipo FCBFQQ si puo' vedere "Livello di sistema I/O", nel Capitolo 10.

STRING (n) e' una notazione abbreviata per:

```
PACKED ARRAY [1..n] OF CHAR
```

I due tipi sono identici. Comunque, poiche' le variabili di tipo LSTRING sono trattate in modo particolare nelle assegnazioni, nei confronti,

nelle procedure READ e WRITE, il tipo LSTRING (n) non e' una notazione abbreviata per PACKED ARRAY [0..n] OF CHAR. I due tipi non sono identici, compatibili, o compatibili nell'assegnazione. Per ulteriori informazioni sui tipi stringa, si puo' vedere "Uso dei tipi STRING e LSTRING", nel Capitolo 9.

COMPATIBILITA' NEI TIPI E NELLE ESPRESSIONI

Due tipi semplici o di riferimento sono compatibili se si verifica una delle seguenti condizioni:

- sono identici
- sono entrambi tipi ADR
- sono entrambi tipi ADS
- uno e' un subrange dell'altro
- sono subrange di tipi compatibili

Due tipi strutturati sono compatibili se si verifica una delle seguenti condizioni:

- sono identici
- sono tipi SET di tipi base compatibili
- sono tipi derivati da STRING aventi uguale lunghezza
- sono tipi derivati da LSTRING

Tuttavia, due tipi strutturati non sono compatibili se si verifica una delle seguenti condizioni:

- uno dei tipi e' un FILE o contiene un FILE
- uno dei tipi e' un super array
- un tipo e' PACKED e l'altro non lo e'

Due valori devono essere di tipi compatibili se combinati con un operatore all'interno di un'espressione. (La maggior parte degli operatori presenta delle limitazioni aggiuntive sul tipo dei loro operandi. Per maggiori dettagli si puo' vedere il Capitolo 13, "Espressioni").
Un'istruzione di tipo CASE con espressioni indice deve essere compatibile con tutti i valori delle costanti CASE. Si deve osservare che due insiemi non sono mai compatibili se solo uno di essi e' PACKED.

COMPATIBILITA' NELL'ASSEGNAZIONE

Alcuni tipi sono implicitamente compatibili. Cio' consente di fare assegnazioni tra tipi superando le normali restrizioni. Per esempio, si puo' supporre di dichiarare le seguenti variabili:

```
VAR DESTINATION : T_DEST;
    SOURCE      : T_SOURCE;
```

SOURCE e' un'assegnazione compatibile con DESTINATION (cioe', DESTINATION := SOURCE e' consentito), se una delle seguenti condizioni risulta verificata:

- T_SOURCE e T_DEST sono tipi identici
- T_SOURCE e T_DEST sono compatibili e SOURCE ha un valore compreso nel campo di variabilita' del tipo subrange T_DEST
- T_DEST e' di tipo REAL e T_SOURCE e' compatibile con il tipo INTEGER o INTEGER4
- T_DEST e' di tipo INTEGER4 e T_SOURCE e' compatibile con il tipo INTEGER o WORD.

Inoltre, se T_DEST e T_SOURCE sono tipi strutturati compatibili, allora SOURCE e' compatibile nell'assegnazione con DESTINATION se si verifica una delle seguenti condizioni:

- Per i tipi SET, ogni elemento di SOURCE appartiene al tipo base di T_DEST
- per i tipi LSTRING, UPPER (DESTINATION) >= SOURCE.LEN

Oltre che nella stessa istruzione di assegnazione, la compatibilita' nell'assegnazione e' richiesta nei seguenti casi di assegnazione implicita:

- passaggi di parametri per valore
- procedure READ e READLN
- variabile di controllo e limiti in un'istruzione FOR
- limiti di array di un tipo super array, e indici di array

La compatibilita' nell'assegnazione e' generalmente riconosciuta al momento della compilazione, e un'assegnazione genera semplici istruzioni. Tuttavia, alcune assegnazioni di subrange, set e LSTRING dipendono dal valore dell'espressione che deve essere assegnata, percio' non si ha alcun controllo fino al tempo di esecuzione. Se viene fatto il controllo sul campo di variabilita', la compatibilita' nell'assegnazione e' controllata durante il runtime; altrimenti non viene fatto alcun controllo.

8. TIPI SEMPLICI

SOMMARIO

I tipi di dati si possono suddividere principalmente in semplici e strutturati. Questo capitolo considera i tipi di dati semplici, che a loro volta si distinguono in tipi ordinali, REAL e INTEGER4.

INDICE

INTRODUZIONE

La distinzione fondamentale tra tipi di dati semplici e strutturati si basa sul fatto che i tipi semplici non possono essere divisi in altri tipi, mentre i tipi strutturati (trattati nel Capitolo 9, "Array, Record e Set", e nel Capitolo 10, "File"), sono composti di altri tipi. I tipi di dati semplici rientrano in tre categorie:

1. tipi ordinali
2. REAL
3. INTEGER

TIPI ORDINALI

I tipi ordinali sono tutti finiti e numerabili, e comprendono i seguenti tipi semplici:

- INTEGER
- WORD
- CHAR
- BOOLEAN
- tipi enumerati
- tipi subrange

INTEGER4, anche se e' finito e numerabile, non e' un tipo ordinale.

INTEGER

I valori INTEGER sono un sottoinsieme dei numeri interi e variano da -MAXINT a MAXINT compreso lo 0. MAXINT e' la costante predichiarata 32767 (cioe' 2^15 - 1) sulle macchine oggetto dell'attuale Pascal. (Il valore -32768 non e' un INTEGER valido; il compilatore lo usa per controllare variabili di tipo INTEGER non inizializzate e variabili di tipo subrange di INTEGER).

INTEGER non e' un subrange di INTEGER4 (gia' trattato in "INTEGER4", nel Capitolo 5). Se lo fosse, le espressioni con segno dovrebbero essere calcolate usando il tipo INTEGER4, e il risultato dovrebbe essere convertito in INTEGER.

Le espressioni sono sempre calcolate usando un tipo base, non un tipo subrange. Le costanti di tipo INTEGER possono essere trasformate al loro interno, se necessario, in un tipo WORD, mentre questo non accade per le variabili INTEGER. In un'espressione, se necessario, i valori INTEGER si trasformano in REAL o INTEGER4. La funzione ORD converte un valore di un qualunque tipo ordinale in un tipo INTEGER.

Il tipo predichiarato INTEGER2 e' identico a INTEGER.

WORD

I tipi WORD e INTEGER sono simili; differiscono principalmente nel campo di variabilita' dei loro valori. Entrambi sono tipi ordinali. I valori WORD possono essere considerati come un gruppo di 16 bit o come un sottoinsieme dei numeri interi da 0 a MAXWORD (65535, cioe' 2^16 - 1). Il tipo WORD e' una particolare caratteristica del Pascal, utile in diversi modi:

- per esprimere valori che variano da 32768 a 65535
- per operare su indirizzi di macchina
- per eseguire operazioni primitive, come quella di AND tra word e di shift tra word, senza usare il tipo INTEGER

A differenza degli INTEGER, tutti i tipi WORD sono valori non negativi. La funzione WRD trasforma qualsiasi valore di tipo ordinale in un tipo WORD. Come i valori INTEGER, in un'espressione i valori WORD sono convertiti nel tipo INTEGER4, se necessario.

Avendo sia un tipo INTEGER che un tipo WORD, si puo' stabilire una corrispondenza tra quantita' di 16 bit in uno dei seguenti due modi:

1. come un valore, con segno, variabile da -32767 a +32767
2. come un valore positivo variabile da 0 a 65535

Comunque non si devono usare contemporaneamente nella stessa espressione valori WORD e INTEGER (anche se questo genera un segnale di avvertimento invece che un errore). I valori WORD e INTEGER non sono neppure compatibili con l'assegnazione.

CHAR

Nel Pascal i valori CHAR sono valori ASCII di 8 bit. CHAR e' un tipo ordinale, e in esso sono inclusi tutti i valori di 256 byte. Inoltre e' consentito l'uso di SET OF CHAR. I confronti relazionali avvengono utilizzando la sequenza ASCII.

Poiche' nello standard ISO, il carattere di cursore usato nei file TEXT non fa parte del tipo CHAR, alcuni sistemi operativi possono richiedere per il Pascal l'inclusione del cursore nel tipo CHAR (per esempio, carattere di ritorno carrello).

La funzione CHR trasforma qualsiasi valore di tipo ordinale in uno di tipo CHAR, finche' l'ORD del valore varia da 0 a 255. Per un elenco completo dei caratteri ASCII si puo' vedere l'Appendice D, "Codici dei Caratteri ASCII".

BOOLEAN

BOOLEAN e' un tipo ordinale con due soli valori (predichiarati): FALSE e TRUE. Il tipo BOOLEAN e' un caso speciale di tipo enumerato, dove ORD (FALSE) e' 0 e ORD (TRUE) e' 1. Cio' significa che FALSE < TRUE.

E' possibile ridefinire gli identificatori di tipo BOOLEAN, FALSE e TRUE, ma il compilatore, nelle espressioni booleane e nelle istruzioni IF, REPEAT e WHILE, usa implicitamente il tipo precedentemente definito.

Non esiste una funzione che trasforma un valore di tipo ordinale in uno di tipo BOOLEAN. Comunque si puo' ottenere lo stesso risultato con la funzione ODD per i valori INTEGER e WORD, o con l'espressione:

```
ORD (valore)  < >  0
```

TIPI ENUMERATI

Un tipo enumerato definisce un insieme ordinato di valori. Tali valori sono costanti ed enumerati dagli identificatori che li denotano.

Esempi di dichiarazioni di tipo enumerato:

```
FLAGCOLOR = (RED, WHITE, BLUE)
SUITS     = (CLUB, DIAMOND, HEARTH, SPADE)
DOGS      = (MAUDE, EMILY, BRENDAN)
```

Ogni tipo enumerato e' anche un tipo ordinale. Gli identificatori per tutte le costanti di tipo enumerato devono essere univoci all'interno del loro livello di dichiarazione.

Al livello Esteso, le procedure READ e WRITE e le funzioni ENCODE e DECODE operano sui valori di tipo enumerato trattando l'effettivo identificatore di costante come una stringa. Cio' significa che i valori enumerati possono essere letti direttamente.

La funzione ORD, al livello Standard, puo' essere usata per trasformare valori enumerati in valori di tipo INTEGER; la funzione WRD trasforma valori enumerati in valori di tipo WORD.

La funzione RETYPE, al livello di Sistema, puo' essere usata per trasformare valori INTEGER o WORD in valori di tipo enumerato.
Per esempio:

```
IF RETYPE (COLOR, I) = BLUE THEN WRITELN ('TRUE BLUE')
```

I valori ottenuti applicando la funzione ORD alle costanti di un tipo enumerato iniziano sempre con zero. Cosi' i valori ottenuti per il tipo FLAGCOLOR dell'esempio precedente sono i seguenti:

```
ORD (RED)   = 0
ORD (WHITE) = 1
ORD (BLUE)  = 2
```

I tipi enumerati sono particolarmente utili per rappresentare un insieme di nomi, come nomi di operazioni o di comandi. E' piu' sicuro modificare un programma aggiungendo un nuovo valore a un tipo enumerato piuttosto che usando semplicemente dei numeri, poiche' ogni array indicizzato di quel tipo, o set basato su quel tipo, viene automaticamente trasformato.

Per esempio, l'input interattivo di un comando puo' essere realizzato leggendo l'identificatore di tipo enumerato che corrisponde a un comando. Poiche' i tipi enumerati sono ordinati, possono anche essere utili confronti come RED < GREEN. A volte e' utile accedere ai valori piu' bassi e piu' alti del tipo enumerato; cio' lo si puo' fare usando le funzioni LOWER e UPPER, come nell'esempio che segue:

```
VAR TINT :  COLOUR;
FOR TINT := LOWER (TINT) TO UPPER (TINT)
            DO PAINT (TINT)
```

TIPI SUBRANGE

Un tipo subrange e' un sottoinsieme di un tipo ordinale. Il tipo da cui e' preso il sottoinsieme viene chiamato il tipo "ospite". Percio' tutti i tipi subrange sono anche tipi ordinali.

Si puo' definire un tipo subrange specificando il limite inferiore e superiore del subrange (in questo ordine). Il limite inferiore non deve essere maggiore del limite superiore, pero' possono essere uguali. Il tipo subrange e' usato frequentemente come tipo indice di un limite di array o come tipo base di un set. (Per una trattazione degli array e dei set si puo' vedere "Array, Record e Set" nel Capitolo 9).

Esempi di tipi subrange insieme al loro tipo ordinale ospite:

Subrange	Ordinale ospite
INTEGER	100..200
WORD	WRD (1)..9
CHAR	'A'..'Z'
tipo enumerato	RED..YELLOW

Inoltre e' possibile sostituire una clausola subrange con una lista di valori nelle seguenti circostanze:

- costanti set
- costruttori set
- costanti di istruzioni CASE ed etichette di record con varianti (al livello Esteso)

Oltre che in dichiarazioni di array e set, e' possibile usare il tipo subrange per assicurare che il valore di una variabile sia compreso entro limiti accettabili. Se il controllo del campo di variabilita' e' abilitato durante la compilazione, questi limiti sono controllati durante il runtime.

Per esempio, se la logica di un programma implica che una variabile ha sempre un valore compreso tra 100 e 999, e la variabile viene dichiarata di tipo subrange, allora il compilatore controlla che il valore ad essa assegnato non superi questo campo di variabilita'.

Inoltre la dichiarazione di un tipo subrange puo' consentire al compilatore di allocare meno spazio e di compiere operazioni piu' semplici. Per esempio, dichiarare che BOTTLES e' di tipo subrange INTEGER 1..100 significa che il tipo puo' essere allocato in otto bit invece che in sedici.

I seguenti tre tipi subrange sono predichiarati:

1. BYTE = WRD (0)..255; {subrange WORD di 8 bit}

2. SINT = -127..127; {subrange INTEGER di 8 bit}

3. INTEGER1 = SINT

Il tipo BYTE e' particolarmente utile nelle applicazioni orientate su macchina. Per esempio i tipi ADRMEM e ADSMEM (per dettagli vedere "Tipi Indirizzo" nel Capitolo 11) considerano generalmente la memoria come un array di byte. Tuttavia, poiche' il tipo BYTE e' realmente un subrange del tipo WORD, se necessario le espressioni aritmetiche con valori BYTE sono calcolate usando 16 bit invece di 8.

In alcuni casi (per esempio, l'assegnazione di un'espressione BYTE ad una variabile BYTE quando il controllo aritmetico e' disabilitato), il compilatore puo' ottimizzare operazioni aritmetiche su 16 bit in operazioni su 8 bit. In generale, l'uso di BYTE invece che di WORD evita alla memoria conversioni da BYTE a WORD nei calcoli delle espressioni.

Al livello Esteso, i limiti del subrange possono essere espressioni costanti. Dato che il compilatore assume che le parentesi sinistre diano sempre inizio ad una dichiarazione di tipo enumerato, la prima espressione in una dichiarazione di subrange non deve iniziare con una parentesi sinistra.

Per esempio:

```
TYPE {I primi due sono consentiti.}
     FEE = (A, B, C);
     FIE = M + 2 * N .. (P-2) * N;
     {FOO non e' valido cosi' come dichiarato.}
     FOO = (M+2) * N .. P - 2 * N;
```

REAL

I valori REAL sono valori non ordinali, a cui e' assegnato un campo di variabilita' ed una precisione; il campo di variabilita' dei valori disponibili dipende dal sistema su cui si opera. Per informazioni piu' specifiche sul sistema usato si puo' far riferimento al manuale "Linguaggio Pascal Guida Utente".

La maggior parte delle implementazioni del Pascal usa numeri reali in singola precisione nel formato Microsoft oppure IEEE. Questi formati hanno una mantissa di 24 bit ed un esponente di 8 bit, fornendo circa sette cifre di precisione ed un valore massimo di 1.701411E38. Le costanti REAL del formato Microsoft sono limitate all'intervallo compreso tra 1.0E-38 e 1.0E+38.

La versione attuale del Pascal comprende estensioni di tipi di dati numerici per l'elaborazione di numeri reali (e interi) di piu' alta precisione. Per quanto riguarda i numeri reali, questo include il supporto per numeri reali in singola e doppia precisione, secondo lo standard IEEE sul punto mobile.

Mentre il Pascal standard fornisce un tipo REAL, il Pascal fornisce tre tipi reali: REAL, REAL4 e REAL8. Comunque il tipo REAL e' sempre identico a REAL4 oppure a REAL8. La scelta e' fatta con il metacomando, $REAL:n, dove n e' 4 oppure 8. {$REAL:8} ha lo stesso effetto di TYPE REAL = REAL8. Il tipo REAL per default e' generalmente REAL4.

Qualsiasi (o tutti) di questi numeri reali puo' essere usato in un singolo programma. Comunque i programmi che usano REAL4 e REAL8 non sono trasferibili.

Nel formato IEEE il tipo REAL4 e' di 32 bit, mentre il tipo REAL8 e' di 64 bit. Il formato standard IEEE e' il seguente:

REAL4 Bit con segno, esponente binario a 8 bit con massimo valore raggiungibile di 127, mantissa di 23 bit.

REAL8 Bit con segno, esponente binario a 11 bit con massimo valore raggiungibile di 1023, mantissa di 52 bit.

In entrambi i casi la mantissa ha un bit "nascosto" piu' significativo (sempre uno) e rappresenta un numero maggiore o uguale a 1.0 ma minore di 2.0. Un esponente di zero da' un valore zero, e il massimo esponente e' un valore chiamato NaN ("not a number"). I byte sono in ordine "inverso"; il piu' basso byte indirizzato e' quello meno significativo della mantissa.

Il campo di variabilita' numerico di REAL4 e' di appena sette cifre significative (24 bit), con un esponente che varia da E-38 a E+38. Il campo di variabilita' numerico di REAL8 comprende oltre quindici cifre significative (53 bit), con un esponente che varia da E-306 a E+306.

Il carattere di esponente puo' essere "D" oppure "d", ma anche "E" oppure "e", quindi un numero come 12.34d56 e' consentito. Questa estensione minore fornisce la compatibilita' con altri linguaggi Microsoft.

Comunque il carattere di esponente D, oppure d, non indica doppia precisione (come nel FORTRAN), poiche' cio' comporta che i numeri con i caratteri di esponente E, oppure e, siano in singola precisione.

Nel Pascal le costanti letterali di tipo REAL sono convertite prima in formato REAL8 e poi, se necessario, in REAL4 (per esempio, per essere passate come un parametro CONST o per inizializzare una variabile in una sezione VALUE). Se si ha la necessita' di usare costanti REAL4, esse devono essere dichiarate come variabili REAL4 (magari aggiungendo l'attributo READONLY) ed assegnare loro una costante in una sezione VALUE.

I valori REAL4 e REAL8 vengono passati alle funzioni intrinseche come parametri di riferimento (CONSTS), piuttosto che come parametri di valore. Il compilatore accetta le espressioni REAL come parametri CONSTS; esso valuta l'espressione, assegna il risultato ad uno stack temporaneo, e ne trasmette l'indirizzo, il che e' generalmente piu' efficiente che passare il valore stesso (specialmente nel caso REAL8).

Le funzioni che ritornano valori REAL usano il metodo "long return", cioe' la routine di chiamata passa un indirizzo, addizionale, nascosto e relativo di uno stack temporaneo che riceve il risultato. Questo si applica a tutte le funzioni che ritornano valori REAL4 o REAL8, sia definiti dall'utente che intrinseci. Per una descrizione dei confronti REAL che producono un risultato non ordinato, si puo' vedere "Espressioni Booleane" nel Capitolo 13.

La libreria di runtime del Pascal fornisce funzioni addizionali REAL per supportare il FORTRAN Microsoft. Queste funzioni sono disponibili in Pascal, ma non sono predichiarate (per ulteriori informazioni sulle funzioni disponibili e su come usarle, si puo' vedere il Capitolo 16 "Procedure e Funzioni Disponibili").

INTEGER4

Come i valori INTEGER e WORD, i valori INTEGER4 sono un sottoinsieme dei numeri interi. I valori INTEGER4 variano da -MAXLONG a MAXLONG. MAXLONG e' una costante predichiarata con il valore di 2,147,483,647 (cioe' 2^31 - 1). Il valore -2,147,487,648 (cioe' -2^31) non e' un INTEGER4 valido.

A differenza di INTEGER e WORD, il tipo INTEGER4 non e' considerato un tipo ordinale. Non vi sono subrange INTEGER4, e INTEGER4 non puo' essere un indice di array o il tipo base di un set. Inoltre i valori INTEGER4 non possono essere usati per controllare istruzioni FOR e CASE.

INTEGER4 e' generalmente un tipo numerico esteso, come REAL. In un'espressione i valori di tipo INTEGER o WORD si trasformano automaticamente in INTEGER4 se l'espressione richiede un valore intermedio esterno al campo di variabilita' di INTEGER o WORD. In un'espressione i valori di tipo INTEGER4 non si trasformano in REAL; per fare la conversione si deve usare esplicitamente la funzione FLOATLONG.

9. ARRAY, RECORD E SET

SOMMARIO

Questo capitolo descrive i tipi di dati strutturati, prendendo in esame altri tipi di cui essi sono costituiti, come array, record e set.

La trattazione dei file avviene nel capitolo successivo.

INDICE

INTRODUZIONE

Un tipo strutturato e' composto di altri tipi. I componenti di tipi strutturati sono tipi semplici oppure altri tipi strutturati. Un tipo strutturato e' caratterizzato dai tipi dei suoi componenti e dal modo in cui e' strutturato. In Pascal un tipo strutturato puo' occupare fino a 65534 byte di memoria.

I tipi strutturati del Pascal sono i seguenti:

```
ARRAY <campo di variabilita'> OF <tipo>
SUPER ARRAY <campo di variabilita'> OF <tipo>
  STRING   (n)
  LSTRING  (n)
RECORD
SET OF <tipo-base>
FILE OF <tipo>
```

Poiche' i componenti delle strutture possono essere esse stesse tipi strutturati, si puo' avere, per esempio, un array di array, un file di record contenenti set, o un record che contiene un file e un altro record. Questo e' un esempio della flessibilita' nell'assegnare dei tipi ai dati: essa fornisce al Pascal gran parte della sua potenza di linguaggio di programmazione.

Nel resto di questo capitolo sono trattati array, record e set. Per una descrizione dei file si puo' vedere il Capitolo 10, "File".

ARRAY

Un tipo array e' una struttura costituita da un numero fisso di componenti, i quali sono tutti dello stesso tipo (chiamato "tipo componente").

Gli elementi dell'array sono indicati tramite indici, che sono valori del "tipo indice" dell' array. Il tipo indice deve essere un tipo ordinale: BOOLEAN, CHAR, INTEGER, WORD, subrange oppure enumerato.

Nel Pascal gli array sono ad una dimensione, ma poiche' il tipo componente puo' anche essere un array, sono ammessi anche array ad n-dimensioni.

Esempi di dichiarazioni di tipo per array:

```
TYPE
INT_ARRAY : ARRAY [1..10] OF INTEGER;
ARRAY_2D  : ARRAY [0..7] OF ARRAY [0..8] OF 0..9;
MORAL_RAY : ARRAY [PEOPLE] OF (GOOD, EVIL);
```

Nell'ultima dichiarazione PEOPLE e' un tipo subrange, mentre GOOD e EVIL sono costanti enumerate.

Esiste una notazione abbreviata disponibile per array ad n-dimensioni, che rende l'istruzione seguente uguale alla seconda dell'esempio visto precedentemente:

```
ARRAY_2D : ARRAY [0..7, 0..8]  OF 0..9;
```

Dopo aver dichiarato questi array, si possono assegnare ai loro componenti le seguenti istruzioni:

```
INT_ARRAY [10]          := 1234;
ARRAY_2D  [0,8]         := 9;
MORAL_RAY [Machiavelli] := EVIL;
```

Tutti gli array PACKED ad n-dimensioni sono impaccati; quindi le istruzioni che seguono sono equivalenti:

```
PACKED ARRAY [1..2, 3..4] OF REAL

PACKED ARRAY [1..2] OF PACKED ARRAY [3..4] OF REAL
```

Per una descrizione dei tipi impaccati si puo' consultare il Capitolo 11, "Tipi di Riferimento e Altri Tipi".

SUPER ARRAY

Un super array e' un esempio di "tipo super" del Pascal. Un tipo super e' simile ad un insieme di tipi o ad una funzione che ritorna un tipo. I tipi super in generale, ed i super array in particolare, costituiscono delle particolari caratteristiche del Pascal.

Il tipo super array ha diversi importanti usi, e puo' essere usato per qualunque di questi scopi:

- Per elaborare stringhe.

  Sia STRING che LSTRING sono tipi super array predichiarati. Il tipo LSTRING tratta stringhe di lunghezza variabile. Il tipo STRING tratta stringhe di lunghezza fissa e stringhe lunghe piu' di 255 caratteri.

- Per allocare dinamicamente array di dimensioni variabili.

  In caso contrario tali array richiedono la massima allocazione di spazio possibile.

- Come tipo di un parametro formale in una procedura o funzione.

Tale dichiarazione rende la procedura o funzione utilizzabile per un insieme o classe di tipi, piuttosto che soltanto per un tipo di lunghezza fissa.

Un identificatore di tipo super specifica l'insieme dei tipi rappresentati dal tipo super. Una dichiarazione successiva di tipo puo' dichiarare un normale identificatore di tipo come un tipo "derivato" da quella classe di tipi. Tale tipo derivato e' simile a qualunque altro tipo.

Una dichiarazione di tipo super array e' una dichiarazione di tipo array avente come prefisso la parola chiave SUPER. Il limite superiore di ogni array e' sostituito da un asterisco, come nell'esempio che segue:

```
TYPE VECTOR = SUPER ARRAY [1..*] OF REAL;
```

Secondo la precedente dichiarazione di tipo, si possono dichiarare le seguenti variabili:

```
VAR  ROW : VECTOR (10);
     COL : VECTOR (30);
     ROWP: ^VECTOR;
```

In questo esempio, VECTOR e' un identificatore di tipo super array. VECTOR (10) e VECTOR (30) sono indicatori di tipo che denotano "tipi derivati". ROW e COL sono variabili di tipi derivati da VECTOR. ROWP e' un puntatore al tipo super array VECTOR.

Anche se il concetto generale di tipi super consente altri "tipi di tipi", come super subrange e super set (oltre ai super array), i tipi super generalmente consentono soltanto un tipo array con limiti superiori parametrici. Un tipo super e' una classe di tipi e non un tipo specifico. Percio', nella sezione VAR di un programma, procedura o funzione, non si possono dichiarare variabili di tipo super; esse devono essere dichiarate come variabili di un tipo derivato dal tipo super.

Tuttavia in una procedura o in una funzione si puo' assegnare un tipo super a un parametro formale di riferimento: cio' consente alla routine di operare su qualunque dei possibili tipi derivati. (In altri Pascal questo tipo di parametro e' chiamato "conformant array").

Ad un tipo puntatore di riferimento puo' anche essere assegnato un tipo super. Cio' permette a un puntatore di riferirsi a qualunque dei possibili tipi derivati. Un puntatore di riferimento a un tipo super consente "array dinamici". Questi array vengono allocati nell'area heap trasmettendo il loro limite superiore alla procedura NEW. Una descrizione dei tipi puntatori e dell'allocazione dinamica e' trattata nel Capitolo 11, "Tipi di Riferimento e Altri Tipi". Per avere invece maggiori informazioni sulla procedura NEW si puo' consultare il Capitolo 16, "Procedure e Funzioni Disponibili".

Il seguente e' un esempio sull'uso della procedura NEW per l'allocazione dinamica:

```
VAR STR_PNT: ^SUPER PACKED ARRAY [1..*] OF CHAR;
    VEC_PNT: ^SUPER ARRAY [0..*, 0..*] OF REAL;
       .
       .
    NEW (STR_PNT, 12);
    NEW (VEC_PNT, 9, 99);
```

Un parametro attuale, in una procedura o funzione, puo' essere di un tipo super piuttosto che di un tipo derivato, ma solo se il parametro e' un parametro di riferimento o un puntatore di riferimento. (Questi sono i soli generi di variabili che possono essere di un tipo super piuttosto che derivato).

Esempio di super array:

```
TYPE VECTOR = SUPER ARRAY [1..*] OF REAL;
{"VECTOR" e' l'identificatore del tipo super array.}

VAR X: VECTOR (12); Y: VECTOR (24); Z: VECTOR (36);
{X, Y, e Z sono tipi derivati da VECTOR.}

{Qui sotto, SUM accetta variabili di ogni tipo derivato}
{dal tipo super VECTOR.}
FUNCTION SUM (VAR V: VECTOR) : REAL;
{V e' il parametro formale di riferimento del tipo super VECTOR.}

VAR S: REAL; I: INTEGER;
BEGIN
  S := 0;
  FOR I := 1 TO UPPER (V) DO S := S + V [I];
  SUM := S
END;

BEGIN
  .
  .
  TOTAL := SUM (X) + SUM (Y) + SUM (Z);
  .
  .
END
```

Le normali regole di tipo per i componenti di un tipo super array e per gli indicatori di tipo che utilizzano un tipo super array, consentono l'assegnazione, il confronto e il passaggio come parametri dei componenti.

La funzione UPPER ritorna l'attuale limite superiore di un parametro super array o di riferimento. Il massimo estremo superiore di un tipo derivato da un tipo super array, e' limitato al valore massimo del tipo indice implicato dall'estremo inferiore (per esempio, MAXINT, MAXWORD).
Esistono due tipi super array predichiarati: STRING e LSTRING.
Il compilatore supporta direttamente i tipi STRING e LSTRING nei seguenti modi:

- assegnazione di LSTRING e STRING
- confronto di LSTRING e STRING
- procedure di READ su LSTRING e STRING
- accesso alla lunghezza di un tipo STRING tramite la funzione UPPER
- accesso alla lunghezza massima di un tipo LSTRING tramite la funzione UPPER
- accesso alla lunghezza del tipo LSTRING tramite STR.LEN e STR[0].

Questi argomenti sono trattati nel paragrafo "Uso dei Tipi STRING e LSTRING" in questo capitolo.

TIPI STRING

I tipi STRING sono super array predichiarati di caratteri:

```
TYPE STRING = SUPER PACKED ARRAY [1..*] OF CHAR;
```

Una stringa letterale come 'abcdefg' e' automaticamente di tipo STRING (n). La dimensione dell'array 'abcdefg' e' 7; quindi la costante e' di tipo derivato STRING: STRING (7).

Il Pascal Standard richiama qualunque packed array di caratteri avente l'estremo inferiore della stringa uguale ad uno e consente poche operazioni speciali su questo tipo (come confronto e scrittura) che non si possono eseguire con altri array.

Nel Pascal, invece, la notazione STRING (n) del super array e' identica a PACKED ARRAY [1..n] OF CHAR (n puo' variare da 1 a MAXINT). Non c'e' alcun valore di default per n, come in altri Pascal, poiche' STRING indica il tipo super array stesso e non una stringa con una lunghezza di default.

L'identificatore STRING e' relativo a un super array, e quindi puo' essere usato solo come parametro formale di riferimento o come tipo puntatore di riferimento. Si hanno altre limitazioni applicate ai super array: non e' possibile confrontare un tale parametro, un puntatore deferenziato oppure definirlo come un intero.

Qualsiasi variabile (o costante) di tipo super array STRING, oppure CHAR o STRING (n), oppure PACKED ARRAY [1..n] OF CHAR, puo' essere passata ad un parametro formale di riferimento del tipo super array STRING. Inoltre una variabile di tipo LSTRING o LSTRING (n) puo' anche essere passata ad un parametro formale di riferimento del tipo STRING. Per una descrizione del tipo STRING come parametro formale di riferimento, si puo' vedere il paragrafo "Uso dei Tipi STRING e LSTRING" in questo capitolo.

Il Pascal Standard ammette l'assegnazione, il confronto e la scrittura dei tipi STRING. Il livello Esteso consente la lettura dei tipi STRING, compreso il tipo super array STRING e il tipo derivato STRING (n). La

lettura di un tipo STRING provoca l'input di caratteri fino a che non si raggiunge la fine della riga o del tipo STRING. Se si raggiunge prima la fine della riga, il resto del tipo STRING e' riempito di spazi (blank). Nello scrivere una stringa si scrivono tutti i suoi caratteri.

Le normali regole di compatibilita' di tipo del Pascal sono, per i tipi STRING, flessibili. Se le loro lunghezze sono uguali, due qualsiasi variabili o costanti di tipo PACKED ARRAY [1..n] OF CHAR o di tipo STRING (n) possono essere confrontate o assegnate. Tuttavia, poiche' la lunghezza di un tipo super array STRING puo' variare, confronti e assegnazioni non sono consentiti.

Esempio di un'assegnazione STRING non consentita:

```
PROCEDURE CANNOT_DO (VAR S : STRING);
VAR STR : STRING (10);
BEGIN
  STR := S
  {Questa assegnazione non e' consentita perche'}
  {la lunghezza di S puo' variare.}
END;
```

Il prefisso PACKED nella dichiarazione PACKED ARRAY [1..n] OF CHAR, cosi' come e' definito nello standard ISO, implica generalmente che un componente non puo' essere passato come un parametro di riferimento. Nel Pascal questa limitazione non viene applicata.

Per essere conformi con lo standard ISO, il passaggio di un componente CHAR di un tipo STRING come un parametro di riferimento viene definito come "errore non rilevato". Inoltre il tipo indice di una stringa e' ufficialmente un INTEGER, ma i valori di tipo WORD possono anche essere usati come indici di un tipo STRING. Molte applicazioni sull'elaborazione di stringhe possono certamente trarre vantaggio dal tipo LSTRING descritto in "Tipi LSTRING", in questo capitolo.

Nel Capitolo 16, "Procedure e Funzioni Disponibili", sono descritte alcune procedure e funzioni intrinseche riguardanti le stringhe. Molte di queste procedure e funzioni si applicano ai tipi STRING, alcune soltanto ai tipi LSTRING.

TIPI LSTRING

In Pascal la caratteristica LSTRING consente stringhe di lunghezza variabile. LSTRING (n) e'un tipo predichiarato come segue:

```
TYPE LSTRING = SUPER PACKED ARRAY [0..*] OF CHAR
```

Tuttavia una variabile di tipo esplicito PACKED ARRAY [0..n] OF CHAR non e' "identica" al tipo LSTRING (n), anche se essi sono strutturalmente la stessa cosa. Non c'e' valore di default per n: esso varia da 0 a 255. In un tipo LSTRING si puo' accedere ai caratteri con la solita notazione di array.

Al loro interno i tipi LSTRING contengono una lunghezza, (L), seguita da una stringa di caratteri. La lunghezza e' contenuta nell'elemento zero del tipo LSTRING e puo' variare da 0 all'estremo superiore. Alla lunghezza di una variabile T di tipo LSTRING si puo' accedere come a T[0] con il tipo CHAR, o come a T.LEN con il tipo BYTE. Le costanti stringa del tipo CHAR o STRING (n) sono automaticamente trasformate nel tipo LSTRING.

La costante predichiarata NULL e' la stringa vuota LSTRING (0). NULL e' l'unica costante di tipo LSTRING; non c'e' alcun modo per definire altre costanti LSTRING. Come per i tipi STRING, un componente CHAR di tipo LSTRING puo' essere passato come parametro di riferimento, e i valori WORD e INTEGER possono essere usati per indicizzare un tipo LSTRING.

Numerose operazioni funzionano su tipi LSTRING in modo diverso che su tipi STRING. Qualunque tipo LSTRING puo' essere assegnato a qualunque altro tipo LSTRING, a condizione che la lunghezza corrente dell'estremo destro non superi la lunghezza massima dell'estremo sinistro. Allo stesso modo, un tipo LSTRING puo' essere passato come parametro valore ad una procedura o funzione, ma in questo caso la lunghezza corrente del parametro attuale non deve superare la lunghezza massima specificata dal parametro formale. Se e' attivo il controllo sul campo di variabilita', il compilatore controlla l'assegnazione dei tipi LSTRING e il passaggio dei parametri LSTRING (n). Il numero attuale di byte assegnati o passati e' quello minimo degli estremi superiori dei tipi LSTRING.

Nessuno dei due lati in un'assegnazione LSTRING puo' essere un parametro del tipo super array LSTRING; entrambi devono essere tipi derivati da esso.

Esempi di assegnazioni LSTRING:

```
{Dichiarazione delle variabili}
VAR A : LSTRING (19);
    B : LSTRING (14);
    C : LSTRING (6);
    .
    .
{Assegnazione delle variabili}
A := '19 caratteri stringa';
B := '14 caratteri';
C := 'abbreviazione';
A := B;
{Questo e' consentito, poiche' la lunghezza di B}
{e' minore della lunghezza massima di A.}
C := A;
{Questo non e' consentito, poiche' la lunghezza di A}
{e' maggiore della lunghezza massima di C.}
```

Si possono confrontare due qualsiasi tipi LSTRING, compresi i tipi LSTRING di tipo super array (l'unico confronto consentito di tipo super array). La lettura di una variabile LSTRING provoca l'input di caratteri fino alla fine della riga o del tipo LSTRING, e stabilisce la lunghezza in base al numero dei caratteri letti. La scrittura da un tipo LSTRING scrive la stringa di lunghezza corrente.

USO DEI TIPI STRING E LSTRING

Vengono ora descritte le operazioni STRING e LSTRING direttamente supportate dal compilatore. Alla fine di questo argomento viene commentato un programma che illustra l'uso dei tipi STRING e LSTRING in un contesto.

Per la descrizione delle seguenti procedure e funzioni di stringa, si puo' vedere il Capitolo 16, "Procedure e Funzioni Disponibili".

CONCAT	INSERT
COPYLST	POSITN
COPYSTR	SCANEQ
DELETE	SCANNE

Al livello di Sistema del Pascal, le procedure FILLC, FILLSC, MOVEL, MOVESL, MOVER e MOVESR operano anche su stringhe.

Il Pascal supporta direttamente tipi STRING e LSTRING nei seguenti modi:

- Assegnazione

  Si puo' assegnare qualunque valore LSTRING a qualunque variabile LSTRING, pero' la lunghezza massima della variabile oggetto deve essere maggiore o uguale alla lunghezza corrente del valore sorgente e non deve essere di tipo super array LSTRING. Se la lunghezza massima dell'oggetto e' minore della lunghezza corrente del sorgente, viene assegnata soltanto la lunghezza dell'oggetto e, se il controllo di variabilita' e' attivo, si verifica un errore di runtime. Si puo' assegnare un valore STRING ad una variabile STRING: in questo caso la lunghezza di entrambi gli estremi e' la stessa e nessun estremo e' di tipo super array STRING. Passare STRING o LSTRING come parametro valore e' molto simile ad un'operazione di assegnazione.

- Confronto

  Gli operatori LSTRING < < = > > = < > = utilizzano la lunghezza del byte per i confronti di stringa; gli operandi possono essere di lunghezze differenti. Per essere considerate uguali, due stringhe devono essere della stessa lunghezza. Se due stringhe di lunghezze differenti sono uguali fino alla lunghezza della piu' breve, quest'ultima e' considerata minore della piu' lunga. Gli operandi possono essere del tipo super array LSTRING. Per i tipi STRING sono disponibili gli stessi operatori relazionali, ma le lunghezze devono essere le stesse e non sono consentiti operandi del tipo super array STRING.

- READ e WRITE

  READ LSTRING legge fino a che LSTRING e' riempito, o finche' non viene raggiunta la fine della riga. La lunghezza corrente e' stabilita in base al numero dei caratteri letti. WRITE LSTRING utilizza la lunghezza corrente. Si puo' inoltre vedere READSET (descritto nel Capitolo 17, "Procedure e Funzioni Orientate su

File"), che fa la lettura in un tipo LSTRING, se pero' i caratteri in input sono in un dato SET OF CHAR. READ STRING inserisce spazi se la riga e' piu' corta del tipo STRING. WRITE STRING scrive tutti i caratteri nella stringa. Sia READ che WRITE ammettono tipi super array STRING e LSTRING, cosi' come i loro tipi derivati.

- Accesso alla lunghezza

  Si puo' accedere alla lunghezza corrente di una variabile T di LSTRING con T.LEN, che e' di tipo BYTE, oppure con T[0], di tipo CHAR. Questa notazione puo' assegnare una nuova lunghezza e determinare la lunghezza corrente. La funzione UPPER trova la lunghezza massima di un tipo LSTRING o la lunghezza di un tipo STRING; cio' e' particolarmente utile per trovare l'estremo superiore di un parametro di riferimento super array o di riferimento a un puntatore.

Non si possono assegnare o confrontare tipi STRING e LSTRING mescolati; lo si puo' fare solo se il tipo STRING e' costante. Si possono assegnare tipi STRING a tipi LSTRING, o viceversa, tramite una delle routine di trasferimento o le procedure COPYSTR e COPYLST. Poiche' le costanti del tipo STRING o CHAR, se necessario, si trasformano automaticamente nel tipo LSTRING, le costanti LSTRING sono considerate normali costanti STRING. NULL (il tipo LSTRING di lunghezza zero) e' l'unica costante LSTRING esplicita.

Nell'esempio di programma posto alla fine di questa sezione, tutti i parametri STRING (CONST oppure VAR) possono essere di tipo STRING o LSTRING; tutti i parametri LSTRING sono VAR LSTRING e devono contenere una variabile di tipo LSTRING.

Una "trasformazione speciale" consente di passare un parametro LSTRING attuale ad un parametro formale di riferimento di tipo STRING. La lunghezza di un tipo STRING formale e' la lunghezza attuale del tipo LSTRING. Percio', se LSTR (nell'esempio che segue) e' di tipo LSTRING (n) o LSTRING, esso puo' essere passato ad una procedura o funzione con un parametro formale di riferimento di tipo STRING:

```
VAR LSTR : LSTRING (10);
  .
  .
PROCEDURE TIE_STRING (VAR STR : STRING);
  .
  .
TIE_STRING (LSTR);
  .
  .
```

In questo caso UPPER (STR) e' equivalente a LSTR.LEN.

Le procedure e funzioni con parametri di riferimento di tipo super STRING possono operare ugualmente bene su tipi STRING e LSTRING. Si dichiara un parametro di tipo LSTRING solo quando la lunghezza deve essere cambiata. Generalmente un tipo LSTRING, in una procedura o funzione, e' un parametro VAR o VARS, poiche' un parametro CONST o CONSTS di tipo LSTRING

non possono essere trasformati.

Esempio di un programma che usa tipi STRING e LSTRING:

```
PROGRAM STRING_SAMPLE;

PROCEDURE STRING_PROC   (CONST S:  STRING); BEGIN  END;
PROCEDURE LSTRING_PROC  (CONST S: LSTRING); BEGIN  END;

VAR
  CHR1VAR: CHAR;
  STR5VAR: STRING (5);
  LST5VAR: LSTRING (5);
  LST9VAR: LSTRING (9);
  STR4VAR: PACKED ARRAY [1..4] OF CHAR;
  STR6VAR: PACKED ARRAY [1..6] OF CHAR;

BEGIN

  {Si devono osservare tutti i tipi di stringhe che un parametro}
  {CONST STRING prende.}
STRING_PROC ('A');
  {La costante di tipo carattere  e' corretta.}
STRING_PROC (CHR1VAR);
  {La variabile di tipo carattere e' corretta.}
STRING_PROC ('STRING');
  {La costante STRING e' corretta.}
STRING_PROC (STR5VAR);
  {La variabile STRING e' corretta.}
STRING_PROC (LST5VAR);
  {La variabile LSTRING e' corretta.}
```

```
  {Tuttavia un parametro CONST LSTRING non puo' prendere}
  {variabili non-LSTRING.}
LSTRING_PROC ('A');
  {La costante di tipo carattere e' corretta.}
LSTRING_PROC (CHR1VAR);
  {La variabile di tipo carattere non e' corretta.}
LSTRING_PROC ('STRING');
  {La costante STRING e' corretta.}
LSTRING_PROC (STR5VAR};
  {La variabile STRING non e' corretta.}
LSTRING_PROC (LST5VAR);
  {La variabile LSTRING e' corretta. Le assegnazioni ad una}
  {variabile STRING sono limitate allo stesso tipo.}
STR5VAR := 'A';
  {La costante di carattere non e' corretta.}
STR5VAR := CHR1VAR;
  {La variabile di carattere non e' corretta.}
STR5VAR := 'TINY';
  {La costante STRING e' troppo piccola.}
STR5VAR := 'RIGHT';
  {Entrambi i lati hanno cinque caratteri: e' corretto.}
STR5VAR := 'LONGER';
  {Non e' corretto; la costante STRING e' troppo grande.}
STR5VAR := LST5VAR;
  {Non e' corretto; non si possono assegnare tipi LSTRING}
  {a tipi STRING.}
COPYSTR (LST5VAR, STR5VAR);
  {COPYSTR e' una procedura intrinseca.}
STR5VAR := STR4VAR;
  {Non e' corretto; la variabile STRING e' troppo piccola.}
COPYSTR (STR4VAR, STR5VAR);
  {COPYSTR e' corretto; in STR5VAR[5] c'e' riempimento degli spazi.}
STR5VAR := STR5VAR;
  {Corretto; entrambi i lati hanno cinque caratteri.}
STR5VAR := STR6VAR;
  {Non e' corretto; la variabile STRING e' troppo grande.}
```

```
  {Tuttavia le assegnazioni ad una variabile LSTRING}
  {sono piu' flessibili.}
LST5VAR := 'A';
  {La costante di carattere e' corretta.}
LST5VAR := CHR1VAR;
  {La variabile di carattere non e' corretta.}
LST5VAR := 'TINY';
  {La costante STRING piu' piccola e' corretta.}
LST5VAR := 'RIGHT';
  {La costante STRING della stessa lunghezza e' corretta.}
LST5VAR := 'LONGER';
  {Cio' causa un errore solo durante il runtime; per ora e' corretto.}
LST5VAR := LST9VAR;
  {Cio' puo' dare errore durante il runtime; per ora e' corretto.}
LST9VAR := LST5VAR;
  {Cio' non e' neppure controllato durante il runtime; e' sempre corretto.}
LST5VAR := STR5VAR;
  {Non e' corretto; non si puo' assegnare una variabile STRING}
  {ad una variabile LSTRING.}
COPYLST (STR5VAR, LST5VAR)
  {Questo e' il modo di copiare una variabile STRING da una LSTRING.}

END.
```

RECORD

Una struttura record si comporta come un modello per dati di tipi differenti concettualmente correlati. Il tipo record in se stesso e' una struttura costituita da un numero fisso di componenti, generalmente di tipi differenti.

Ogni componente di un tipo record e' detto "campo". La definizione di un tipo record specifica il tipo ed un identificatore per ciascun campo all'interno del record. Poiche' il raggio d'azione di questi "identificatori di campo" e' la definizione stessa di record, essi devono essere univoci all'interno della dichiarazione. I valori dei campi associati agli identificatori dei campi sono accessibili tramite la notazione di record o l'istruzione WITH.

Per esempio si puo' dichiarare il seguente tipo record:

```
TYPE LP = RECORD
          TITLE   : LSTRING (100);
          ARTIST  : LSTRING (100);
          PLASTIC : ARRAY [1..SONG_NUMBER] OF SONG_TITLE
          END
```

Si puo' quindi dichiarare una variabile di tipo LP, come segue:

```
VAR BEATLES_1 : LP;
```

Infine, si puo' accedere ad un componente del record con una notazione di campo o con l'istruzione WITH (si deve osservare il punto che separa gli identificatori di campo):

```
BEATLES_1.TITLE := 'Meet The Beatles';
WITH BEATLES_1 DO
     PLASTIC[1] := 'I Wanna Hold Your Hand'
```

RECORD CON VARIANTI

Un record puo' avere numerose "varianti", e in tal caso un determinato campo, chiamato "campo etichettato", indica quale variante usare. Il campo etichettato puo' avere o no un identificatore e memorizzarlo nel record. Alcune operazioni, come le procedure NEW e DISPOSE e la funzione SIZEOF, possono specificare un valore di etichetta anche se questa non e' memorizzata come parte del record.

Esempi di record con varianti:

```
TYPE OBJECT = RECORD
                X, Y: REAL;
                CASE S: SHAPE OF
                  SQUARE: (SIZE, ANGLE: REAL);
                  CIRCLE: (DIAMETER: REAL)
              END;

     FOO = RECORD
             CASE BOOLEAN OF
               TRUE: (I, J: INTEGER);
               FALSE: (CASE COLOR OF
                         BLUE: (X: REAL);
                         RED: (Y: INTEGER4))
           END;
```

E' consentita soltanto una parte variante per record; essa deve essere l'ultimo campo del record. Tuttavia questa parte puo' anche avere a sua volta una variante (e cosi' via, a qualunque livello). Tutti gli identificatori di campo in un dato record devono essere univoci, anche in varianti differenti. Per esempio, dopo aver dichiarato i suddetti tipi record, si possono creare delle variabili ed assegnare loro dei valori dei precedenti tipi nel seguente modo:

```
VAR O, P : OBJECT;
    F, G : FOO;

BEGIN
  O.DIAMETER := 12.34;          {CASE di CIRCLE}
  P.SIZE := 1.2;                {CASE di SQUARE}
  F.I := 1; F.J := 2;           {CASE di TRUE}
  G.X := 123.45;                {CASE di FALSE e BLUE}
  G.Y := 678999                 {CASE di FALSE e RED; questo}
                                {sostituisce il valore di G.X.}
END;
```

L'ultimo standard ISO richiede tutti i possibili valori di campo etichettato per scegliere alcune varianti. Percio' non e' consentito includere CASE INTEGER OF e omettere una variante per ogni possibile valore INTEGER. Tuttavia in Pascal una tale omissione non e' rilevata come errore.

Il Pascal ammette l'uso di ampie opzioni di costante CASE nella clausola con variante; cioe' una lista di costanti puo' definire un caso. Al livello Esteso, i subrange e l'istruzione OTHERWISE possono anche definire un caso. Quando usato, OTHERWISE si applica all'ultima variante della lista e non e' seguito dai due punti. Si puo' anche dichiarare una variante vuota, come POINT:() oppure OTHERWISE (). Si puo' anche dichiarare un tipo record interamente vuoto, anche se il compilatore emette un avvertimento ogni volta che il record viene utilizzato.

Lo standard ISO definisce un certo numero di errori che si riferiscono ai record con varianti; in Pascal questi errori non sono rilevati, anche se e' attivo il controllo sull'etichetta. (La correttezza del valore dell'etichetta e' verificata tramite un controllo sull'etichetta che genera un codice ogni volta che viene utilizzato un campo con variante). Nella dichiarazione del tipo record OBJECT (nell'esempio precedente), ogni uso di SIZE genera un controllo su S = SQUARE. Tuttavia, nel caso di FOO, gli usi di "I" non possono essere controllati poiche' il Pascal non alloca il campo etichettato BOOLEAN.

Inoltre lo standard ISO stabilisce che quando si verifica un "cambio di variante" (come quando e' assegnato un nuovo valore all'etichetta), tutti i campi con varianti diventano non definiti. Tuttavia il Pascal, nel caso analogo, non posiziona i campi ad un valore non inizializzato; quindi l'uso di un campo con variante che ha un valore non definito e' un errore non rilevato.

Il Pascal non impone molte limitazioni ad una variabile di tipo record allocata nell'area heap con la forma lunga della procedura NEW (per ulteriori dettagli si puo' vedere il Capitolo 16, "Procedure e Funzioni Disponibili"). Tuttavia il Pascal non controlla l'assegnazione a un simile "record corto" per verificare che solo il record corto stesso e' stato modificato nell'area heap.

Un record allocato con la forma lunga di NEW puo' essere rilasciato usando la forma breve di DISPOSE senza alcun effetto negativo (si tratta di un errore ISO non rilevato nel Pascal). E' anche un errore non rilevato nel Pascal l'uso di DISPOSE di un record passato come parametro

di riferimento oppure usato da un'istruzione attiva WITH.

I record con varianti interagiscono con le caratteristiche del Pascal in due modi:

1. Non e' un metodo sicuro dichiarare una variante che contiene un file; qualunque cambiamento nei dati del file che usano un campo in un'altra variante puo' portare ad errori di I/O, anche se il file e' chiuso. Nell'esempio che segue, qualunque uso di R puo' condurre ad errori in F:

```
RECORD CASE BOOLEAN OF
            TRUE  : (F: FILE OF REAL);
            FALSE : (R: ARRAY [1..100] OF REAL)
       END;
```

2. Dare dati iniziali a diverse varianti che si sovrappongono in una variabile di una sezione VALUE, puo' provocare risultati imprevedibili. Nell'esempio che segue, il valore iniziale di LAP e' incerto:

```
VAR LAP : RECORD CASE BOOLEAN OF
            TRUE : (I: INTEGER4);
            FALSE : (R: REAL)
          END;
VALUE LAP.I := 10; LAP.R := 1.5;
```

Se si tenta di applicare una di queste due operazioni, il Pascal genera un messaggio di avvertimento.

SPIAZZAMENTO ESPLICITO DI CAMPO

Il Pascal consente di assegnare spiazzamenti (offset) espliciti di byte ai campi di un record. Questa caratteristica a livello di Sistema puo' essere utile per interfacciare il software in altri linguaggi, poiche' i formati dei blocchi di controllo possono non essere conformi al metodo comune di allocazione di campo del Pascal. Tuttavia, poiche' essa consente di eseguire anche operazioni non sicure, come sovrapposizione di campi e di valori di word agli estremi dispari del byte, e' consigliabile solo se l'interfaccia risulta necessaria.

Esempio di assegnazione di spiazzamento esplicito di byte:

```
TYPE CPM = RECORD
             NDRIVE [00]: BYTE;
             FILENM [01]: STRING (8);
             FILEXT [09]: STRING (3);
             EXTENT [12]: BYTE;
             CPMRES [13]: STRING (20);
             RECNUM [33]: WORD;
             RECOVF [35]: BYTE
           END;

   OVERLAP = RECORD
               BYTEAR [00]: ARRAY [0..7] OF BYTE;
               WORDAR [00]: ARRAY [0..3] OF WORD;
               BITSAR [00]: SET OF 0..63
             END;
```

Come si puo' vedere nell'esempio, lo spiazzamento e' compreso tra parentesi quadre, come avviene per la notazione di attributo. Il numero indica lo spiazzamento del byte all'inizio del campo. Alcune macchine oggetto non consentono di accedere ad un valore di 16 bit di un indirizzo dispari, ma il compilatore non lo rileva come errore.

Se si da' un qualsiasi spiazzamento di un campo, si danno spiazzamenti a tutti i campi. Per ogni spiazzamento che si omette, il compilatore prende un valore arbitrario. Anche se il compilatore elabora una dichiarazione che include sia gli spiazzamenti che i campi con varianti, si devono usare solo gli uni o gli altri in un certo programma.

Sebbene si possa controllare completamente la sovrapposizione di un campo con spiazzamenti espliciti, le varianti forniscono le forme estese delle procedure NEW, DISPOSE e SIZEOF. Volendo allocare record di differenti lunghezze, si devono usare le procedure RETYPE e GETHQQ, invece che le varianti e la forma estesa di NEW. Per esempio:

```
CPMPV := RETYPE (CPMP, GETHQQ (36));
```

Il compilatore supporta costanti strutturate per i tipi record con spiazzamenti espliciti. Al loro interno, i campi di lunghezza dispari maggiori di uno sono arrotondati alla lunghezza pari successiva. Per esempio:

```
ODDR = RECORD
         F1[00] : STRING (3);
         F2[03] : CHAR
       END;
```

In questo esempio il campo F1 e' lungo quattro byte, quindi un'assegnazione a F1 si sovrappone ad F2. In un tale record devono essere assegnati per primi tutti i campi di lunghezza dispari.

SET

Un tipo set definisce il campo di variabilita' dei valori che un insieme puo' assumere. Questo campo di variabilita' costituisce la "potenza dell'insieme" del tipo base specificato nella definizione di tipo. La potenza dell'insieme e' l'insieme di tutti i possibili insiemi che possono essere composti da un tipo base ordinale. L'insieme nullo, [], e' un elemento di ogni insieme.

Supponendo di dichiarare i seguenti tipi set:

```
TYPE HUES = SET OF COLOR;
     CAPS = SET OF 'A'..'Z';
     MATTER = SET OF (ANIMAL, VEGETABLE, MINERAL);
```

Allora si dichiarano variabili come queste:

```
VAR FLAG : HUES;
    VOWELS : CAPS;
    LIVE : MATTER;
```

Infine, si possono fare assegnazioni a queste variabili di tipo set:

```
FLAG := [RED, WHITE, BLUE];
VOWELS := ['A', 'E', 'I', 'O', 'U'];
LIVE := [ANIMAL, VEGETABLE];
```

Gli elementi di tipo set devono essere inclusi fra parentesi quadre: si deve notare la differenza rispetto all'uso delle parentesi tonde per racchiudere i tipi base enumerati in una dichiarazione di tipo set.

Le operazioni sui tipi set sono implementate direttamente da codice generato su linea o dalle routine dell'unita' set. Per una descrizione completa sulle operazioni sui tipi set, si puo' vedere il Capitolo 13, "Espressioni".

Il valore ORD del tipo base puo' variare da 0 a 255. Quindi SET OF CHAR e' consentito, mentre non lo e' SET OF 1942..1984.

I tipi set il cui valore ORD massimo e' 15 (cioe' i set che stanno in un tipo WORD) sono di solito piu' efficienti di quelli piu' grandi. Inoltre, se il controllo del campo di variabilita' e' abilitato, il passare un insieme come un parametro di valore richiede un controllo di compatibilita' di runtime, a meno che gli insiemi formali ed attuali abbiano lo stesso tipo.

I tipi set forniscono un modo chiaro ed efficiente di dare diverse qualita' o attributi ad un oggetto.
In un altro linguaggio e' possibile assegnare ad ogni qualita' una potenza di due:

```
READY  = 1
GETSET = 2
ACTIVE = 4
DONE   = 8
```

Si possono dunque assegnare le qualita' con un'istruzione come questa:

```
X := READY + ACTIVE
```

e poi controllarle usando OR e AND come operatori tra bit con un'istruzione come:

```
IF ((X AND ACTIVE) < > 0) THEN WRITELN ('GO FISH')
```

In Pascal la dichiarazione equivalente puo' essere:

```
QUALITIES = SET OF (READY, GETSET, ACTIVE, DONE);
```

Si possono dunque assegnare le qualita' tramite X := [GETSET, ACTIVE] e controllarle con le seguenti operazioni:

IN	controlla un bit
+	posiziona un bit
-	cancella un bit

Per esempio, una costruzione appropriata puo' essere:

```
IF ACTIVE IN X THEN WRITELN ('GO FISH')
```

Si puo' anche usare SET OF 0..15 per controllare e posizionare i bit in un tipo WORD. Per usare i tipi WORD sia come un insieme di bit che come il tipo WORD, occorre assegnare due tipi alla word, con un record con variante, la funzione RETYPE, o con un tipo indirizzo.

I bit di un insieme sono assegnati a partire da quello piu' significativo nel byte con indirizzo inferiore. Quindi, su una macchina "byte-swapped", l'insieme [0, 7, 8, 15] ha il valore WORD di #80 + #01 + #8000 + #0100. Per maggiori dettagli si puo' consultare il manuale "Linguaggio Pascal Guida Utente".

10. FILE

SOMMARIO

In questo capitolo è descritta la struttura dei file. Si tratta di uno degli argomenti più importanti: infatti tramite i file il linguaggio Pascal interfaccia con il sistema operativo. Quindi è necessaria una buona comprensione del tipo file per poter eseguire 1/0 da un programma.

INDICE

INTRODUZIONE

Un file e' una struttura costituita da una sequenza di componenti, tutti dello stesso tipo. Attraverso i file il Pascal interfaccia con un dato sistema operativo. Quindi e' necessario capire il tipo FILE per poter eseguire l'input/output in/da un programma.

DICHIARAZIONE DI FILE

Come con qualunque altro tipo, si deve dichiarare una variabile di tipo file prima di poterla usare. Tuttavia il numero di componenti di un file non si stabilisce dichiarando un tipo FILE.

Esempi di dichiarazioni di FILE:

```
TYPE F1 = FILE OF COLOR;
     F2 = FILE OF CHAR;
     F3 = TEXT;
```

Un file e' semplicemente un altro tipo di dati, simile ad un array, ma privo di limiti ed e' possibile accedere solo ad un componente per volta. Tuttavia un file e' associato generalmente ad uno dei seguenti elementi:

- file su dischi
- terminali
- stampanti
- altre unita' di input e output.

Cio' implica le seguenti limitazioni in Pascal: un FILE OF FILE non e' consentito, ne' direttamente ne' indirettamente. Altre strutture, come i FILE OF ARRAY oppure un ARRAY OF FILE, sono consentite.

La maggior parte delle implementazioni del Pascal collegano variabili file ai file di dati del sistema operativo. Questo Pascal utilizza sempre il sistema operativo per accedere ai file, ma non impone ulteriori formattazioni o strutture sui file del sistema operativo.

Il Pascal supporta normali file allocati in modo statico, file come variabili locali (allocate nello stack) e file come puntatori di riferimento (allocati nell'area heap). Ad eccezione che per i file nei super array, il compilatore genera codice per inizializzare un file quando esso e' allocato e per chiudere (con la procedura CLOSE) un file quando e' deallocato.

Questa inizializzazione avviene, nella maggior parte dei casi, automaticamente. Tuttavia, un file dichiarato in un modulo o in un'interfaccia di unita' non inizializzata, ha la sua inizializzazione

solo se l'identificatore di modulo o di unita' viene chiamato come una procedura. Le dichiarazioni di file in questi casi generano il seguente avvertimento del compilatore:

Contains file initialize module

Soltanto un file in un'interfaccia di un'unita' non inizializzata non genera questo avvertimento.

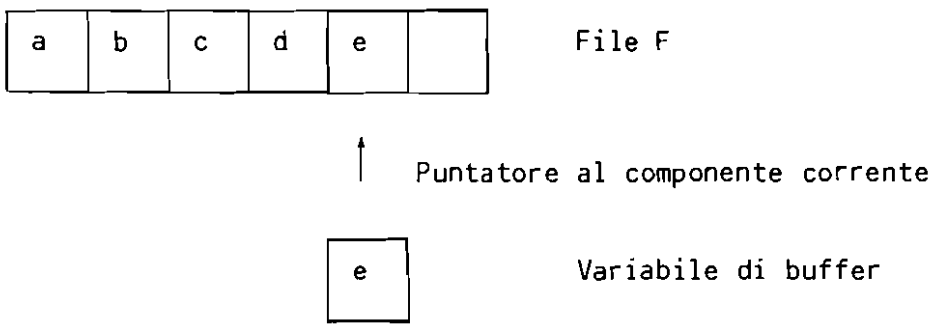
Il Pascal si avvale di file standard, INPUT e OUTPUT (trattati in "I File Predichiarati INPUT e OUTPUT", in questo capitolo). Nel Pascal Standard, i file devono essere dati nell'intestazione del programma, e quando si esegue il programma, il sistema runtime sollecita per avere i nomi dei file. Al livello Esteso, si possono usare le procedure ASSIGN e READFN per dare i nomi di file esplicitamente al sistema operativo, per i file non compresi nell'intestazione del programma.

I file nei record con varianti o nei tipi super array non sono consigliabili; se si usano, il compilatore emette un segnale di avvertimento. Una variabile di tipo file non puo' essere assegnata, confrontata o passata per valore: puo' solo essere dichiarata e passata come parametro di riferimento.

Al livello Esteso, si puo' indicare un metodo di accesso ad un file o altre caratteristiche, specificando il modo del file. Il modo e' un valore di tipo enumerato predichiarato FILEMODES. I modi generalmente disponibili includono i tre modi base di accesso: SEQUENTIAL, TERMINAL e DIRECT. A tutti i file, tranne INPUT e OUTPUT, sono assegnati per default modi di accesso SEQUENTIAL. A INPUT e OUTPUT sono assegnati per default modi di accesso TERMINAL.

LA VARIABILE DI BUFFER

Ogni file F ha una variabile di buffer F^ associata ad esso. Una variabile di buffer e il suo file associato possono presentarsi cosi':



Le procedure GET e PUT utilizzano questa variabile di buffer per leggere (READ) e scrivere (WRITE) sui file. La procedura GET copia il componente corrente del file nella variabile di buffer; la procedura PUT fa il contrario, cioe' copia il valore della variabile di buffer nel componente corrente.

La variabile di buffer puo' essere indicata (cioe' il suo valore puo' essere prelevato o memorizzato) come qualunque altra variabile del Pascal. Cio' consente esecuzioni di assegnazioni come questa:

```
F^ := 'z'
C  := F^
```

Una variabile di buffer puo' essere passata come parametro di riferimento ad una procedura o funzione, oppure utilizzata in un'istruzione WITH di un record. Tuttavia la variabile buffer di file non puo' essere aggiornata correttamente se la posizione del file cambia all'interno della procedura, della funzione o dell'istruzione WITH. Il compilatore emette un messaggio di avvertimento per segnalare questa possibilita'.

Per esempio, il seguente uso di una variabile buffer di file genera un segnale di avvertimento durante la compilazione:

```
VAR A : TEXT;
PROCEDURE CHAR_PROC (VAR X : CHAR);
  .
  .
CHAR_PROC (A^);
{Qui viene emesso un segnale di avvertimento}
```

In Pascal si hanno due speciali meccanismi interni: la valutazione pigra ("lazy evaluation") e l'I/O simultaneo, i quali consentono rispettivamente in modo naturale l'input interattivo da terminale e l'I/O sovrapposto all'esecuzione del programma. La valutazione pigra e' applicata a tutti i file strutturati in modo ASCII ed e' necessaria per il normale input da terminale. L'I/O simultaneo e' applicato a tutti i file strutturati in modo BINARY ed e' necessario per alcuni sistemi operativi che supportano sovrapposizioni di input e di output.

Entrambi i meccanismi generano una chiamata di runtime che viene eseguita prima di qualsiasi uso della variabile buffer. Per avere ulteriori dettagli si possono vedere le sezioni su "Valutazione Pigra" e "I/O Simultaneo" nel Capitolo 17.

STRUTTURE DI FILE

I file del Pascal hanno due strutture fondamentali: BINARY e ASCII. Esse corrispondono rispettivamente a file di dati grezzi e a file-testo leggibili dall'utente.

FILE A STRUTTURA BINARY

Il tipo di dati FILE OF <tipo> del Pascal standard corrisponde al tipo file di struttura BINARY del Pascal. Questi corrispondono, a loro volta, ai file non formattati del sistema operativo.

Con sistemi operativi che dividono i file in record, ogni record costituisce un componente del tipo file (da non confondersi con il tipo record). Procedure primitive come GET e PUT operano sulla base di un record. Con sistemi operativi che non hanno proprie strutture di record, le procedure primitive GET e PUT trasferiscono un numero fisso di byte per chiamata, uguale alla lunghezza di un componente. Per un'ulteriore descrizione dei file BINARY si puo' vedere la parte sui "Modi di Accesso ai File" in questo capitolo.

FILE A STRUTTURA ASCII

Il tipo di dati TEXT del Pascal standard corrisponde ai file di struttura ASCII del Pascal. Questi, a loro volta, corrispondono ai file testuali del sistema operativo (chiamati "file-testo" in questo manuale).

Il tipo TEXT del Pascal e' simile ad un FILE OF CHAR, tranne per il fatto che i gruppi di caratteri sono organizzati in "righe" e in "pagine". Le procedure primitive di file, come GET e PUT, operano sempre su una base di caratteri.

Tuttavia con sistemi operativi che dividono i file in record, ogni record e' una riga (non un carattere). Anche in sistemi operativi che non hanno una struttura di record propria, vi sono comunque altri linguaggi e prestazioni che prevedono qualche metodo per organizzare i byte in righe di caratteri.

Il Pascal fornisce un certo numero di funzioni e procedure speciali che utilizzano questa caratteristica di divisione in righe. Poiche' il Pascal non impone alcuna formattazione aggiuntiva per i file di sistema operativo aventi diverse modalita' di accesso (comprese SEQUENTIAL, TERMINAL e DIRECT), i programmi in altri linguaggi possono generare e utilizzare questi file.

I file-testo (file del tipo TEXT) del Pascal sono suddivisi in righe tramite un "marcatore di riga", che e' un carattere non di tipo CHAR. In teoria un file-testo puo' contenere qualunque valore di tipo CHAR. Tuttavia con alcuni sistemi operativi la scrittura di un carattere particolare (per esempio CHR (13), cioe' carattere di ritorno a capo, oppure CHR (10), carattere di avanzamento riga) puo' concludere la riga corrente (il record). Questo valore di carattere in questo caso e' il marcatore di riga e, alla lettura, si presenta sempre come uno spazio

vuoto (blank).

Con altri sistemi operativi puo' mancare un carattere di chiusura. Tuttavia ogni riga e' seguita da un marcatore di riga che si legge come uno spazio vuoto.

Al livello Esteso, una dichiarazione di un file-testo puo' includere una lunghezza opzionale di riga. E' necessario stabilire la lunghezza di riga, che stabilisce a sua volta la lunghezza del record, soltanto nei file-testo che hanno DIRECT come modo di accesso. Si puo' specificare la lunghezza di riga anche per altri modi di accesso, ma non si ottiene alcun effetto.

Si deve specificare la lunghezza di riga di un file-testo come una costante tra parentesi dopo la parola TEXT:

```
TYPE NAMEADDR = TEXT (128);
     DEFAULTX = TEXT;
     SMALLBUF = TEXT (2);
```

MODI DI ACCESSO AI FILE

Nel Pascal i modi di accesso ai file sono SEQUENTIAL, TERMINAL e DIRECT. I modi SEQUENTIAL e TERMINAL sono disponibili al livello Standard; tutti e tre, compreso DIRECT, sono disponibili al livello Esteso. I modi di accesso SEQUENTIAL e TERMINAL ai file di struttura ASCII possono avere lunghezze di record (di righe) variabili; i file con modo di accesso DIRECT devono avere record o righe di lunghezza fissa.

La dichiarazione di un file in Pascal implica la sua struttura, ma non il suo modo di accesso. Per esempio FILE OF STRING (80) indica una struttura BINARY; TEXT indica una struttura ASCII. Un'assegnazione come F.MODE := DIRECT stabilisce il modo di accesso; cio' funziona soltanto al livello Esteso e generalmente e' necessario solo per stabilire il modo di accesso DIRECT.

FILE CON MODO DI ACCESSO TERMINAL

I file con modo di accesso TERMINAL corrispondono sempre ad un terminale interattivo o ad una stampante. I file con modo di accesso TERMINAL, come quelli con accesso SEQUENTIAL, sono aperti con posizionamento all'inizio del file per la lettura o per la scrittura. Si accede ai record uno dopo l'altro, finche' non si raggiunge la fine del file.

L'operazione di input del modo di accesso TERMINAL per i terminali, dipende dalla struttura del file (ASCII o BINARY). Per la struttura ASCII (tipo TEXT) sono lette righe intere in una sola volta. Cio' permette la normale editazione infralinea del sistema operativo, che comprende il carattere di ritorno indietro, l'avanzamento del cursore e il carattere

di annullamento. I caratteri sono ripetuti sul video del terminale mentre la riga e' digitata.

Pero' se il sistema operativo non supporta l'editazione infralinea o ripetizione, allora e' l'interfaccia del file system del Pascal che fornisce queste funzionalita'. Tuttavia, poiche' si legge un'intera riga alla volta, non si possono leggere i caratteri mentre sono digitati, o chiamare diversi messaggi e risposte sulla stessa riga, ecc. Con il modo di accesso TERMINAL della struttura BINARY (di solito di tipo FILE OF CHAR), si possono leggere i caratteri mentre sono digitati; inoltre non viene fatta alcuna editazione infralinea o ripetizione. Questo metodo consente editazione su video, selezione di menu e altra programmazione interattiva sulla base di una battuta piuttosto che di una riga.

I file con modo di accesso TERMINAL usano la valutazione pigra per trattare correttamente la normale lettura interattiva del terminale con tastiera. Per maggiori dettagli si puo' vedere "Valutazione Pigra" nel Capitolo 17.

FILE CON MODO DI ACCESSO SEQUENTIAL

I file con modo di accesso SEQUENTIAL sono generalmente file su disco o altre unita' di accesso sequenziale. Come i file con modo di accesso TERMINAL, i file con acceso SEQUENTIAL sono aperti con posizionamento all'inizio del file per la lettura o per la scrittura, e si accede ai record uno dopo l'altro fino alla fine del file. I file del Pascal Standard hanno modo di accesso SEQUENTIAL per default (tranne INPUT e OUTPUT).

FILE CON MODO DI ACCESSO DIRECT

I file con modo di accesso DIRECT sono generalmente file su disco o altri dispositivi con accesso diretto. Essi, insieme alle altre capacita' di accesso ad un file, sono disponibili al livello Esteso del Pascal.

I file di struttura ASCII con modo di accesso DIRECT, come i file di struttura BINARY, hanno record di lunghezza fissa, dove un record puo' essere una riga oppure un componente di file. (Qui il termine "record" non si riferisce al normale tipo record del Pascal, ma ad un'unita' disco strutturata). I file con accesso DIRECT sono sempre aperti sia per la lettura che per la scrittura, e si puo' accedere direttamente ai record tramite il numero di record. Non c'e' il numero zero di record; i record iniziano con il record numero uno.

I FILE PREDICHIARATI INPUT E OUTPUT

Due file, INPUT e OUTPUT, sono predichiarati in ogni programma Pascal. A questi file e' riservato uno speciale trattamento come parametri di programma, e sono normalmente richiesti come parametri nell'intestazione del programma:

```
PROGRAM ACTION (INPUT, OUTPUT);
```

Se non ci sono parametri di programma e il programma non usa i file INPUT e OUTPUT, l'intestazione si presenta cosi':

```
PROGRAM ACTION;
```

Tuttavia si possono includere INPUT e OUTPUT come parametri di programma se usati, esplicitamente o implicitamente, nel programma stesso:

```
WRITE (OUTPUT, 'Prompt: ')      {Uso esplicito}
WRITE ('Prompt: ')              {Uso implicito}
```

Questi esempi generano un segnale di avvertimento se il file di OUTPUT non e' dichiarato nell'intestazione del programma. Il solo effetto dei file di INPUT e OUTPUT come parametri di programma, e' quello di annullare questo segnale.

Anche se e' possibile ridefinire gli identificatori INPUT e OUTPUT, il file assunto dalle procedure e dalle funzioni (per esempio READ, EOLN) input e output del file-testo, costituisce una definizione predichiarata. Le procedure RESET (INPUT) e REWRITE (OUTPUT) sono generate automaticamente, con INPUT e OUTPUT presenti come parametri di programma, oppure no; (queste procedure possono essere anche usate in modo esplicito).

I file INPUT e OUTPUT hanno struttura ASCII e modo di accesso TERMINAL. Sono connessi fin dall'inizio al terminale e aperti automaticamente. Al livello Esteso del Pascal e' possibile, se si vuole, cambiare queste caratteristiche.

I/O AL LIVELLO ESTESO

Un file variabile nel Pascal e' in realta' un record, di tipo FCBFQQ, chiamato blocco di controllo file (file control block). Al livello Esteso, qualche campo standard in questo record permette facilmente di trattare i modi di accesso ai file e di far notare gli errori.

I campi addizionali e lo stesso record di tipo FCBFQQ possono essere usati al livello di Sistema, descritto in "I/O al Livello di Sistema" in questo capitolo. Oltre che l'accesso a determinati campi FCB, l'I/O al livello Esteso comprende anche le seguenti procedure:

ASSIGN	READFN
CLOSE	READSET
DISCARD	SEEK

Per una descrizione di queste procedure si puo' vedere la parte su "I/O al Livello Esteso" nel Capitolo 17.

Per accedere ai campi FCB si deve usare la normale sintassi di campo di record. Per un file F, i campi sono chiamati F.MODE, F.TRAP e F.ERRS. E' possibile cambiare o esaminare tali campi in qualunque momento.

- F.MODE: FILEMODES

  Questo campo contiene il modo di accesso al file: SEQUENTIAL, TERMINAL o DIRECT. Questi valori sono costanti del tipo predichiarato enumerato FILEMODES. Il file system usa il campo MODE solo durante le procedure RESET e REWRITE. Percio' cambiare il campo MODE di un file aperto non ha alcun effetto. Tranne che per l'INPUT e l'OUTPUT, che hanno modi di accesso TERMINAL, il modo di accesso ad un file e' SEQUENTIAL per default.

  Le procedure RESET e REWRITE cambiano il modo di accesso da SEQUENTIAL a TERMINAL, se scoprono che il dispositivo che viene aperto e' un terminale o una stampante, e se il sistema operativo lo consente. Cio' e' utile nei programmi destinati a girare in modo interattivo o in modo a lotti (batch). Bisogna stabilire il modo di accesso DIRECT prima delle procedure RESET o REWRITE se si vuole usare la procedura SEEK su un file.

- F.TRAP: BOOLEAN

  Se questo campo e' TRUE, viene attivato la ricerca di errori per il file F. Quindi, se si verifica un errore di input/output, il programma non si arresta e si puo' esaminare il codice di errore. Inizialmente a F.TRAP e' dato il valore FALSE. Se si verifica FALSE e un errore di I/O, il programma si arresta.

- F.ERRS: WRD(0)..15

  Questo campo contiene il codice di errore per il file F. Un codice di errore zero significa che non vi e' alcun errore; i valori da 1 a 15 implicano una condizione di errore. Se si prova a fare un'operazione su file diversa da CLOSE o DISCARD, e F.ERRS non e' zero, il programma si arresta immediatamente se F.TRAP e' FALSE. Comunque, se F.TRAP e' TRUE, l'operazione sul file viene ignorata e il programma continua.

  CLOSE e DISCARD non esaminano il valore iniziale di F.ERRS, quindi non sono mai ignorati e non causano un arresto immediato. Nonostante cio', se gli stessi CLOSE o DISCARD generano una condizione di errore, F.TRAP e' usato per determinare se individuare l'errore o se arrestarsi.

  Un'operazione ignorata a causa di una condizione di errore non cambia il file stesso, ma puo' cambiare la variabile buffer o le variabili

di input della procedura READ. Per un elenco completo dei messaggi di errore e dei segnali di avvertimento, si puo' consultare l'Appendice H, "Messaggi di Errore".

Anche al livello Esteso si puo' stabilire la lunghezza di riga per un file-testo, nel modo seguente:

```
TYPE SMALLBUF = TEXT (16);
VAR RANDOMTEXT: TEXT (132);
```

La dichiarazione di lunghezza di riga si applica soltanto ai file di struttura ASCII con accesso DIRECT, dove la lunghezza di riga e' la lunghezza di record usata per leggere e scrivere. Il fatto di stabilire la lunghezza di riga non ha alcun effetto su altri file ASCII.

I/O AL LIVELLO DI SISTEMA

Al livello di Sistema del Pascal, si possono chiamare procedure e funzioni che hanno un parametro formale di riferimento di tipo FCBFQQ con un parametro attuale di tipo FILE OF <tipo> o TEXT, oppure il tipo identico FCBFQQ.

Il tipo FCBFQQ e' il tipo record fondamentale usato per implementare il tipo file in Pascal. L'interfaccia per il tipo FCBFQQ del sistema oggetto (e per qualunque altro tipo necessario), fa di solito parte del file system interno. Percio' le procedure e funzioni che indicano parametri FCBFQQ possono essere chiamate con ogni tipo di file, comprese le procedure e funzioni predichiarate come CLOSE e READ.

Una variabile di tipo FCBFQQ puo' essere passata a procedure come READLN e WRITELN che richiedono un file-testo. Cio' consente, per esempio, di chiamare direttamente le routine di interfaccia sul sistema operativo oggetto, mescolando Pascal e MS-FORTRAN (che condividono l'interfaccia di file system ma hanno campi speciali FCBFQQ), e altre attivita' speciali del file system.
Tali attivita' richiedono una grande conoscenza del file system.

11. TIPI DI RIFERIMENTO E ALTRI TIPI

SOMMARIO

I tipi array, record e set discussi nel Capitolo 9 consentono di rappresentare strutture di dati la cui forma e dimensione sono predeterminate ed ai cui componenti si può accedere in modo standard. Il tipo file, descritto nel Capitolo 10, è una struttura che varia in dimensione ma la cui forma e modo di accesso sono predeterminati.

In questo capitolo si descrivono i tipi di riferimento, che consentono strutture di dati che variano in dimensione e forma ed il cui modo di accesso è specifico del problema implicito nella programmazione. Inoltre sono incluse note sui tipi PACKED e sui tipi procedurali e funzionali.

INDICE

TIPI DI RIFERIMENTO

Un riferimento ad una variabile o costante e' un modo indiretto di accedere ad essa. Il tipo puntatore e' un tipo astratto usato per creare, utilizzare e cancellare variabili allocate da un'area chiamata heap. Lo heap e' un'area di memoria che aumenta e diminuisce in modo dinamico ed e' allocata per variabili di tipo puntatore.

Il Pascal fornisce inoltre due tipi indirizzo orientati su macchina: uno per gli indirizzi che si possono rappresentare in 16 bit, l'altro per gli indirizzi che richiedono 32 bit.

I puntatori sono generalmente usati per eseguire operazioni su alberi, grafi e liste. L'uso dei puntatori e' portabile, strutturato e relativamente sicuro.

I tipi indirizzo forniscono un'interfaccia per l'hardware e per il sistema operativo; il loro uso e' spesso non strutturato, specifico per la macchina, di basso livello e non sicuro. Sia i puntatori che i tipi indirizzo sono descritti piu' oltre.

TIPI PUNTATORE

Un tipo puntatore e' un insieme di valori che indicano le variabili di un tipo dato. Il tipo delle variabili puntate e' chiamato il "tipo di riferimento". Le variabili di riferimento sono tutte allocate in modo dinamico dall'area heap tramite la procedura NEW. Le variabili del Pascal sono di solito allocate nello stack o in locazioni fisse.

Sui puntatori si possono eseguire soltanto queste operazioni:

- assegnazione
- controllo della loro uguaglianza e disuguaglianza tramite i due operatori = e < >
- passaggio come valori o come parametri di riferimento
- prelievo del contenuto puntato, mediante la freccia in alto (^)

Ogni tipo puntatore include il valore NIL di puntatore. I puntatori sono spesso usati per creare strutture di liste di record, come mostrato nell'esempio che segue:

```
TYPE
  TREETIP = ^ TREE;
  TREE = RECORD
           VAL : INTEGER;
                 {Valore della casella TREE.}
           LEFT, RIGHT: TREETIP
                 {Puntatori alle altre caselle TREETIP.}
                 {Si deve osservare la definizione ricorsiva.}
         END;
```

A differenza della maggior parte delle dichiarazioni di tipo, la dichiarazione di un tipo puntatore si puo' riferire a un tipo di cui e' esso stesso un componente. La dichiarazione si puo' anche riferire ad un tipo dichiarato successivamente nella stessa sezione TYPE, come TREE e TREETlP nell'esempio precedente.

Tale dichiarazione e' denominata una dichiarazione anteriore ("forward") di puntatore, e consente strutture ricorsive e mutuamente ricorsive. Poiche' i puntatori sono usati cosi' spesso nelle strutture di liste, le dichiarazioni di puntatore anteriore si verificano frequentemente.

Il compilatore controlla se una dichiarazione di puntatore puo' essere ambigua. Supponendo che l'esempio precedente sia in una procedura nidificata in un'altra, che ha dichiarato a sua volta un tipo TREE, allora il tipo di riferimento di TREETlP puo' essere la definizione esterna o quella seguente nella stessa sezione TYPE. Il Pascal assume che il tipo TREE ideato sia quello successivo nella stessa sezione TYPE e dia questo segnale di avvertimento:

```
Pointer Type Assumed Forward
```

Al livello Esteso un puntatore puo' avere un tipo super array come tipo di riferimento. Gli estremi superiori attuali dell'array sono passati alla procedura NEW per creare una variabile heap della corretta dimensione. Le dichiarazioni di puntatore anteriore del tipo super array non sono consentite.

Il Pascal fa riferimento ai requisiti ISO per mantenere una stretta compatibilita' tra i puntatori. Per esempio non si possono dichiarare due puntatori con tipi differenti e poi assegnarli o confrontarli, anche se essi puntano lo stesso tipo fondamentale. Per esempio:

```
VAR PRA : ^ REAL;
    PRE : ^ REAL;
BEGIN PRA := PRE END;    {Non e' consentito.}
```

Di solito i programmi contengono soltanto una dichiarazione di tipo per un puntatore a un tipo dato. Nell'esempio TREETlP, il tipo di LEFT e RIGHT puo' essere ^TREE invece che TREETlP, ma allora non si possono assegnare variabili di tipo TREETlP a questi campi. Comunque, a volte e' utile assicurarsi che due classi di puntatori non siano usate insieme, anche se puntano lo stesso tipo.

Per esempio si puo' supporre di avere un tipo RESOURCE compreso in una lista e di dichiarare due tipi, OWNER e USER, del tipo ^RESOURCE. Il compilatore prende l'assegnazione dei valori OWNER alle variabili USER, e viceversa, ed emette un messaggio di avvertimento.

In teoria i puntatori non hanno niente a che fare con gli indirizzi di macchina attuali. Infatti un puntatore puo' essere implementato in modi differenti su differenti macchine. Tra le altre possibilita', un puntatore puo' essere implementato come un normale indirizzo, come un indirizzo espresso in spiazzamento di segmento, come uno spiazzamento da una o piu' locazioni fisse, oppure come un indirizzo indiretto.

Se il controllo di inizializzazione e' attivo, un puntatore appena creato ha un valore non inizializzato. Se il controllo su NIL e' attivo, i valori del puntatore sono controllati con alcuni valori non validi. I valori non validi includono NIL, i valori non inizializzati, il riferimento ad un elemento dello heap che e' stato rilasciato con la procedura DISPOSE, o un valore non valido come riferimento per lo heap.

TIPI INDIRIZZO

Come linguaggio di implementazione di sistema, il Pascal richiede un metodo per creare, trattare e prelevare il contenuto degli indirizzi attuali di macchina. Il tipo puntatore si puo' applicare soltanto a variabili che si trovano nell'area heap.

Esistono due generi di indirizzi: relativi e segmentati. Le parole chiave ADR e ADS indicano rispettivamente i tipi a indirizzo relativo e i tipi a indirizzo segmentato. Come mostrano i seguenti esempi, le parole chiave si usano sia come prefissi di clausole di tipo che come prefissi di operatori:

```
VAR INT_VAR  :  INTEGER;
    REAL_VAR :  REAL;
    A_INT    :  ADR OF INTEGER;
    {Dichiarazione di variabile ADR}
    AS_REAL  :  ADS OF REAL;
    {Dichiarazione di variabile ADS}

BEGIN
   INT_VAR   :=  1;
   {Normale variabile intera}
   REAL_VAR  := 3.1415;
   {Normale variabile reale}
   A_INT     := ADR INT_VAR;
   {ADR e' usato come operatore}
   AS_REAL   := ADS REAL_VAR;
   {ADS e' usato come operatore}
   WRITELN (A_INT^, AS_REAL^)
   {Si deve notare l'uso della freccia in alto}
   {per prelevare il contenuto dei tipi indirizzo.}
   {Come output si ha 1 e 3.1415.}
END.
```

Nella tabella 11-1 sono mostrate le caratteristiche dei tipi a indirizzo relativo e segmentato, e come sono implementate su differenti macchine.

Macchina	ADR	ADS
8080	16 bit assoluti	Come per ADR
8086	Valore di default dello spiazzamento del segmento di dati di 16 bit	Spiazzamento di 16 bit, segmento di 16 bit
Z8000 (non segment.)	Dati assoluti di 16 bit	Come per ADR
Z8000 (segmentato)	Come per ADS	Segmento di 16 bit, spiazzamento di 16 bit

Tabella 11-1 Indirizzi di Macchina Relativi e Segmentati

Nel Pascal si puo' dichiarare una variabile che e' un indirizzo:

```
VAR X :  ADR OF BYTE;
```

Poi, con la seguente notazione di record, si possono assegnare valori numerici alla variabile attuale:

```
X.R := 16#FFFF
```

In un ambiente non segmentato, il .R (indirizzo relativo) e' l'unico campo di record disponibile per gli indirizzi ADR e ADS.

Poiche' il Pascal consente la numerazione non decimale, si puo' specificare il valore assegnato nella notazione esadecimale. Lo si puo' inoltre assegnare ad un campo segmento con il tipo ADS in un ambiente segmentato, usando la notazione di campo .S (indirizzo di segmento). Si puo' cosi' dichiarare una variabile di tipo ADS e poi assegnare i valori ai suoi due campi.

```
VAR Y : ADS OF WORD;
  .
  .
Y.S := 16#0001
Y.R := 16#FFFF
```

Come mostrato sopra, qualunque valore di 16 bit puo' essere assegnato direttamente per variabili di tipo indirizzo, usando i campi .R e .S. Con gli operatori ADR e ADS si ottengono questi indirizzi direttamente.

L'esempio seguente assegna degli indirizzi alle variabili X e Y:

```
VAR X : ADR OF BYTE;
    Y : ADS OF WORD;
    W : WORD;
    B : BYTE;
      .
      .
X := ADR B;
Y := ADS W;
```

Il Pascal ammette questi due tipi indirizzo predichiarati:

```
ADRMEM = ADR OF ARRAY [0..32766] OF BYTE;
ADSMEM = ADS OF ARRAY [0..32766] OF BYTE;
```

Quando il tipo indicato dall'indirizzo e' un array di byte, e' possibile indicizzare il byte. Per esempio se A e' di tipo ADRMEM, allora A^[15] e' il byte all'indirizzo A.R + 15, dove .R specifica un indirizzo attuale di 16 bit.

Si possono usare i tipi indirizzo per un indirizzo costante (una forma di costante strutturata); si puo' anche prendere l'indirizzo di una costante o di un'espressione. Per esempio:

```
TYPE ADRWORD = ADR OF WORD;
     ADSWORD = ADS OF WORD;
VAR W: WORD;
    R: ADRWORD;
CONST CONADR = ADRWORD (1234);
BEGIN
  W := CONADR^;
  {Prende la word all'indirizzo 1234}
  W := ADSWORD (0, 32)^;
  {Prende la word all'indirizzo 0:32}
  W := (ADS W).S;
  {Prende il valore del registro segmento DS}
  R := ADR '123';
  {Prende l'indirizzo di un valore costante}
  R := ADR (W DIV 2 + 1)
  {Prende l'indirizzo del valore di un'espressione}
END;
```

Tuttavia le costanti o le espressioni che producono indirizzi generalmente non possono essere usate come oggetto di un'assegnazione (o come un parametro di riferimento o come un record WITH):

```
CONST ADSCON = ADSWORD (256, 64);      {corretto}
FUNCTION SOME_ADDRESS: ADSWORD;        {corretto}
BEGIN
  ADSWORD (0, 32)^ := W;               {non consentito}
  ADSCON^ := 12;                       {non consentito}
  SOME_ADDRESS^ := 100                 {non consentito}
END;
```

PARAMETRI SEGMENTO PER I TIPI INDIRIZZO

VARS e CONSTS sono due parole chiave disponibili, come VAR e CONST, come prefissi di parametro per passare gli indirizzi segmentati di una variabile. Se P e' di tipo ADS FOO, allora P^ puo' essere passato ad un parametro formale VARS, come VARS X: FOO, ma non puo' essere passato ad un parametro formale VAR.

In un ambiente di macchina segmentato, si assume per default un segmento di dati, e in tal caso un parametro VAR e' passato per default come spiazzamento di segmento di dati per default di una variabile. Un parametro VARS e' passato sia come valore del segmento che come valore dello spiazzamento.

Nell'ambiente 8086, sia i parametri VARS che le variabili ADS hanno il valore (.R) dello spiazzamento nel tipo WORD con l'indirizzo piu' basso e il valore (.S) di segmento nell'indirizzo piu' due.

Nell'ambiente segmentato Z8000, il valore (.S) di segmento e' nell'indirizzo piu' basso e il valore (.R) di spiazzamento e' nell'indirizzo piu' due. Inoltre il tipo ADR e' identico al tipo ADS.

Nell'ambiente non segmentato (per esempio, 8080), VAR e CONST sono identiche a VARS e CONSTS. Poiche' in un ambiente non segmentato ADS e ADR sono identiche, il tipo ADS e' utile in situazioni in cui l'ambiente puo' cambiare. Per esempio nel Pascal alcune chiamate primitive di file system sono dichiarate con parametri ADS.

In dichiarazioni di tipo puntatore, la freccia in alto (^) fa da prefisso al tipo indicato; nelle istruzioni di programma essa preleva il contenuto di un puntatore, in modo che si possano fare assegnazioni o operazioni sul valore puntato. La freccia in alto prende anche il contenuto dei tipi ADR e ADS nelle istruzioni di programma.

La selezione dei componenti con la freccia in alto (^) e' eseguita prima degli operatori unari ADR o ADS. Poiche' il selettore di freccia in alto (^) puo' apparire dopo qualunque variabile indirizzo per produrre una nuova variabile, vi puo' essere per esempio un parametro di riferimento nell'oggetto di un'assegnazione, cosi' come nelle espressioni. Poiche' ADS e ADR sono prefissi di operatori, essi sono usati soltanto nelle espressioni, dove si applicano solo ad una variabile, costante o espressione.

Il Pascal e' un linguaggio fortemente tipizzato; due variabili puntatore sono compatibili solo se hanno lo stesso tipo (non e' sufficiente che essi puntano allo stesso tipo). Tuttavia due tipi indirizzo sono considerati dello stesso tipo se sono sia tipi ADR che tipi ADS. Cio' permette di assegnare un ADR OF WORD a un ADR OF STRING (200). Tale assegnazione facilita la cancellazione di una parte della memoria, tramite l'assegnazione di una variabile di tipo STRING (200) ai 200 byte che aprono l'indirizzo di una variabile WORD.

Se P1 e' di tipo ADR OF STRING (200) e P2 e' di qualsiasi tipo ADR OF, l'assegnazione P1^ := P2^ genera codice veloce con nessun controllo del campo di variabilita'. Anche se questa capacita' non e' sicura, qualche

volta i sistemi operativi e altri elementi di software la richiedono.

ADR e ADS non sono compatibili tra di loro, ma la notazione .R puo' superare, o almeno ridurre, il problema.

USO DEI TIPI INDIRIZZO

Entro certi limiti, i due tipi indirizzo si possono combinare e mescolare. L'esempio seguente illustra le regole che si applicano in un ambiente segmentato:

```
VAR
  P: ADS OF DATA;
  {P e' l'indirizzo segmentato del tipo DATA.}
  Q: ADR OF DATA;
  {Q e' l'indirizzo relativo del tipo DATA.}
  X: DATA;
  {X e' una variabile del tipo DATA.}

BEGIN
  P := ADS X;
  {Assegna l'indirizzo di X a P.}
  X := P^;
  {Assegna a X il valore puntato da P.}
  P := ADS P^;
  {Assegna a P l'indirizzo del valore il cui indirizzo e'}
  {puntato da P.  P non viene cambiato da questa assegnazione.}
  Q := ADR X;
  {Assegna l'indirizzo relativo di X a Q.}
  Q.R := (ADR X). R;
  {Assegna l'indirizzo relativo di X a Q, usando il tipo WORD.}
  P := ADS Q^;
  {Assegna l'indirizzo della variabile in Q a P.}
  {Si puo' sempre applicare ADS a ADR^.}
  Q := ADR P^;
  {Non consentito; non si puo' applicare ADR ad ADS^.}
  P.R := 16#8000;
  {Assegna 32768 allo spiazzamento del campo di P.}
  P.S := 16;
  {Assegna 16 al campo segmento di P.}
  Q.R := P.R + 4;
  {Assegna lo spiazzamento di P piu' 4 per ottenere il valore di Q.}
END;
```

Si possono anche vedere gli esempi gia' forniti in "Tipi Indirizzo" in questo capitolo.

NOTE SUI TIPI DI RIFERIMENTO

II tipo indirizzo e il tipo puntatore devono essere trattati come due tipi distinti. Il tipo puntatore, in teoria, e' una corrispondenza non definita da una variabile a un'altra. Il metodo di implementazione non e' definito; tuttavia il tipo indirizzo riguarda gli indirizzi attuali di macchina.

Percio' il tipo puntatore e' un tipo astratto di dati che si comporta allo stesso modo in tutte le implementazioni; generalmente il tipo indirizzo, se non usato con cautela, non e' portabile. l tipi indirizzo sono portabili soltanto se ci si limita ad usare ADS e non li si assegna mai a campi. Nonostante queste limitazioni, comunque, essi possono essere piuttosto utili.

Le seguenti prestazioni speciali che fanno uso di variabili puntatore non sono consentite con variabili indirizzo.

- Le procedure NEW e DISPOSE sono consentite soltanto con puntatori. NlL non si applica al tipo indirizzo. Non ci sono speciali valori indirizzo per indirizzi vuoti, non inizializzati o non validi.

- Il tipo "indirizzo del tipo super array" non e' supportato allo stesso modo del "puntatore al tipo super array". Tuttavia e' consentito ottenere con ADR e ADS l'indirizzo di una variabile super array. Per esempio, se un parametro formale di una procedura o funzione e' dichiarato come VAR S: STRlNG, allora l'espressione ADS S e' consentita all'interno della procedura o funzione. A differenza del puntatore, l'indirizzo non contiene estremi superiori.

TIPI PACKED

Qualunque tipo strutturato puo' essere PACKED. Cio' consente di evitare alla memoria di sprecare tempo di accesso o spazio di accesso al codice. Tuttavia in Pascal sono di solito applicate alcune limitazioni sull'uso di strutture PACKED:

- Il prefisso PACKED e' sempre ignorato, tranne per il controllo di tipo, nei tipi set, nei file e negli array di caratteri; nella maggior parte delle versioni Pascal esso non ha un reale effetto sulla rappresentazione dei record e di altri array. lnoltre PACKED puo' solo precedere uno dei nomi di strutture ARRAY, RECORD, SET o FILE; non puo' precedere un identificatore di tipo. Per esempio, se COLORMAP e' l'identificatore di un tipo array non impaccato, "PACKED COLORMAP" non e' accettato.

- Un componente di una struttura PACKED puo' essere passato come un parametro di riferimento o usato come record di un'istruzione WITH, soltanto se la struttura e' di tipo stringa. lnoltre, non e'

consentito ottenere l'indirizzo di un componente PACKED con ADR o ADS.

- Un prefisso PACKED si applica solo alla struttura che e' stata definita: tutte i componenti di quella struttura, anch'esse strutture, sono impaccati soltanto se si include esplicitamente la parola riservata PACKED nella loro definizione. La sola eccezione a questa regola, gli array ad n-dimensioni, e' descritta in "Array" nel Capitolo 9.

TIPI PROCEDURALI E FUNZIONALI

l tipi procedurali e funzionali sono diversi dagli altri tipi Pascal; (d'ora in poi, l'uso del termine "procedurale" implica sia procedurale che funzionale). ln una sezione TYPE non si puo' dichiarare un identificatore, e neppure una variabile, per un tipo procedurale. Tuttavia si possono usare tipi procedurali per dichiarare il tipo di un parametro procedurale, e in questo senso essi risultano conformi al concetto di tipo del Pascal.

Un tipo procedurale definisce una procedura o funzione di un'intestazione e da' qualsiasi parametro. Per una funzione esso definisce anche il tipo risultante. La sintassi di un tipo procedurale e' la stessa di una procedura o funzione dell'intestazione, compresi tutti gli attributi. ln Pascal non esistono variabili procedurali, ma solo parametri procedurali.

Esempio di una dichiarazione di tipo procedurale:

```
PROCEDURE ZERO (FUNCTION FUN (X, Y: REAL): REAL)
```

Gli identificatori di parametro in un tipo procedurale (X e Y nell'esempio precedente) sono ignorati; e' importante solo il loro tipo.

Per ulteriori informazioni sui tipi procedurali in Pascal, si puo' leggere la sezione sui "Parametri Procedurali e Funzionali" nel Capitolo 15.

12. VARIABILI E VALORI

SOMMARIO

L'argomento trattato in questo capitolo riguarda le variabili e la sezione VALUE. Le variabili sono istanze di certi tipi di dati; tramite la sezione VALUE si possono assegnare alle variabili valori iniziali nell'ambito di un programma, modulo, procedura o funzione.

INDICE

COS'E' UNA VARIABILE ?

Una variabile e' un valore che generalmente cambia durante il corso di un programma. Ogni variabile deve essere di un tipo di dati specifico. Una variabile puo' avere un identificatore.

Se A e' una variabile di tipo INTEGER, allora l'uso di A in un programma si riferisce effettivamente ai dati indicati da A.

Per esempio:

```
VAR A: INTEGER;
BEGIN
   A := 1;
   A := A + 1
END;
```

Queste istruzioni prima assegnano un valore 1 ai dati indicati da A, e successivamente assegnano ad esso un valore 2.

Le variabili sono trattate mediante certe notazioni adatte a indicare la variabile, nel caso piu' semplice l'identificatore di una variabile. In altri casi le variabili possono essere indicate con indici di array, campi di record, prelievo del contenuto del puntatore o, infine, con variabili indirizzo.

Il compilatore stesso puo' a volte creare variabili "nascoste", allocate nello stack, in circostanze come queste:

- Quando e' chiamata una funzione che ritorna un risultato strutturato, il compilatore alloca una variabile nella routine di chiamata per il risultato.

- Quando e' necessario l'indirizzo di un'espressione (per esempio per passarlo come parametro di riferimento o per usarlo come istruzione WITH in un record o con ADR o ADS), il compilatore alloca una variabile per il valore dell'espressione.

- I valori iniziali e finali di un ciclo FOR possono richiedere l'allocazione di una variabile.

- Quando il compilatore valuta un'espressione, puo' allocare una variabile per memorizzare i risultati intermedi.

- Ogni istruzione WITH richiede una variabile da allocare per l'indirizzo di un record di WITH.

DICHIARAZIONE DI UNA VARIABILE

Una dichiarazione di variabile e' costituita dall'identificatore della nuova variabile seguito da due punti e da un tipo. Si possono dichiarare variabili dello stesso tipo fornendo una lista degli identificatori di variabile, seguiti dal loro tipo comune. Per esempio:

```
VAR XCOORD, YCOORD: REAL
```

Si puo' dichiarare una variabile in una qualsiasi delle seguenti locazioni:

- nella sezione VAR di un programma, procedura, funzione, modulo, interfaccia o implementazione.

- in una lista di parametri formali di una procedura, funzione, o parametro procedurale.

In una sezione VAR si puo' dichiarare una variabile di un qualunque tipo consentito; in una lista di parametri formali si puo' includere soltanto un identificatore di tipo (cioe' non si puo' dichiarare un tipo nell'intestazione di una procedura o funzione). Per esempio:

```
PROCEDURE NAME (GEORGE:  ARRAY [1..10] OF COLOR)
{Non consentito; GEORGE e' di un tipo nuovo.}

VAR VECTOR_A:  VECTOR (10)
{Consentito; VECTOR (10) e' un tipo derivato da un tipo super.}
```

Ogni dichiarazione di una variabile F di file del tipo FILE OF T implica la dichiarazione di una variabile di buffer di tipo T, indicata da F^. Al livello Esteso, una dichiarazione di file implica anche la dichiarazione di una variabile record di tipo FCBFQQ, i cui campi sono indicati come F.TRAP, F.ERRS, F.MODE e cosi' via. Per ulteriori informazioni sulle variabili di buffer e sui campi FCBFQQ si possono vedere, rispettivamente, "La Variabile di Buffer" e "I/O al Livello Esteso", nel Capitolo 10.

LA SEZIONE VALUE

In Pascal la sezione VALUE consente di dare valori iniziali alle variabili, in un programma, modulo, procedura o funzione. Si puo' anche inizializzare una variabile in un'implementazione, ma non in un'interfaccia. La sezione VALUE puo' includere solo variabili allocate staticamente, cioe' qualsiasi variabile dichiarata a livello di programma, modulo o implementazione, o una variabile con l'attributo STATIC e PUBLIC. Le variabili con attributo EXTERN o ORIGIN non possono essere presenti in una sezione VALUE, poiche' non sono allocate dal compilatore.

La sezione VALUE puo' contenere assegnazioni di costanti a intere variabili o a componenti di variabili.
Per esempio:

```
VAR  ALPHA : REAL;
     1D    : STRING (7);
     1     : INTEGER;

VALUE
     ALPHA := 2.23;
     1D[1] := 'J';
     1     := 1;
```

Tuttavia, all'interno di una sezione VALUE non si puo' assegnare una variabile ad un'altra variabile. L'ultima riga nel seguente esempio non e' consentita, poiche' "1" deve essere una costante:

```
CONST MAX = 10;
VAR 1,J : INTEGER;
VALUE 1 := MAX;
      J := 1;
```

Se il metacomando $ROM non e' attivo, le variabili sono inizializzate caricando il segmento statico di dati. Se il metacomando $ROM e' attivo, la sezione VALUE genera un messaggio di errore, poiche' i sistemi basati su ROM in genere non possono inizializzare i dati in modo statico.

USO DI VARIABILI E VALORI

Al livello Standard del Pascal, l'indicazione di una variabile puo' designare una delle tre cose:

1. una variabile intera

2. un componente di una variabile

3. una variabile indicata da un puntatore

Un valore puo' essere:

- una variabile

- una costante

- un indicatore di funzione

- un componente di un valore

- una variabile indicata da un valore di riferimento

Al livello Esteso una funzione puo' anche ritornare un array, un record o un set. La stessa sintassi usata per le variabili puo' essere usata per indicare i componenti delle strutture ritornate da queste funzioni.

Questa caratteristica consente anche di prelevare il contenuto di un tipo di riferimento ritornato da una funzione. Tuttavia si puo' soltanto usare l'indicatore di funzione come un valore, non come una variabile. Per esempio, la seguente funzione non e' consentita:

```
F (X, Y)^ := 42;
```

Anche al livello Esteso si possono dichiarare costanti di un tipo strutturato. I componenti di una costante strutturata usano la stessa sintassi delle variabili dello stesso tipo (per un'ulteriore descrizione di questo argomento si puo' vedere "Espressioni Costanti", nel Capitolo 6).

Esempi di componenti di costanti strutturate:

```
TYPE REAL3 = ARRAY [1..3] OF REAL;
{Un tipo array}
CONST PIES = REAL3 (3.14, 6.28, 9.42);
{Una costante array}
  .
  .
 X := PIES [1] * PIES [3];
 {Cioe', 3.14 * 9.42}
 Y := REAL3 (1.1, 2.2, 3.3) [2];
 {Cioe', 2.2}
```

COMPONENTI DI VARIABILI E VALORI INTERI

Al livello Standard un identificatore di variabile indica una variabile intera. Una variabile, un indicatore di funzione o una costante indica un valore intero.

Un componente di una variabile o valore e' indicato dall'identificatore seguito da un selettore che specifica la componente. Il formato di un selettore dipende dal tipo di struttura (array, record, file o riferimento).

Variabili e Valori Indicizzati

Un componente di un array e' indicato dalla variabile o dal valore dell'array, seguito da un'espressione indice. L'espressione indice deve essere compatibile in assegnazione con il tipo indice della dichiarazione di tipo array. Un tipo indice deve sempre essere un tipo ordinale. L'indice stesso deve essere racchiuso tra parentesi quadre e seguire l'identificatore di array.

Esempi di variabili e valori indicizzati:

```
ARRAY_OF_CHAR ['C']
{Indica l'elemento C.}

'STRING CONSTANT' [6]
{Indica il sesto elemento, la lettera 'G'.}

BETAMAX [12] [-3]
BETAMAX [12,-3]
{Questi due dicono la stessa cosa.}

ARRAY_FUNCTION (A, B) [C, D]
{Indica un componente di un array bidimensionale}
{restituito da ARRAY_FUNCTION (A, B). A e B sono}
{parametri attuali.}
```

Si puo' specificare la lunghezza corrente di una variabile LSTRING, LSTR, in uno dei due modi:

1. con la notazione LSTR [0], per accedere alla lunghezza come un componente CHAR

2. con la notazione LSTR.LEN, per accedere alla lunghezza come un valore BYTE

Variabili e Valori di Campo

Un componente di un record e' indicato dalla variabile o dal valore del record seguiti dall'identificatore di campo del componente. I campi sono separati dal punto (.). In un'istruzione WITH si assegna la variabile o il valore del record soltanto una volta. All'interno dell'istruzione WITH si puo' usare direttamente l'identificatore di campo di una variabile record.

Esempi di variabili e di valori di campo:

```
PERSON.NAME := 'PETE'

PEOPLE.DRIVERS.NAME := 'JOAN'

WITH PEOPLE.DRIVERS DO NAME := 'GERI'

RECURSING_FUNC ('XYZ').BETA
{Sceglie il campo BETA del record, ritornato}
{dalla funzione denominata RECURSIVE_FYNC.}

COMPLEX_TYPE (1.2, 3.14).REAL_PART
```

La notazione di campo di record si applica anche ai file per i campi FCBFQQ, per indirizzare valori di tipi per le rappresentazioni numeriche, e ai tipi LSTRING per la lunghezza corrente.

Buffer e Campi di File

Si puo' accedere soltanto ad un componente di file alla volta. Il componente a cui si puo' accedere e' determinato dalla posizione corrente del file ed e' rappresentato dalla variabile di buffer. A seconda dello stato della variabile di buffer, estraendo il suo valore si puo' prima leggere il valore dal file; (cio' viene definita "valutazione pigra"; per maggiori dettagli si puo' leggere la sezione su "Valutazione Pigra", nel Capitolo 17).

Se una variabile di buffer di file e' passata come un parametro di riferimento o e' usata come un record di un'istruzione WITH, il compilatore emette un segnale per avvertire che il valore della variabile di buffer non puo' essere corretto dopo che la posizione del file e' stata cambiata con una procedura GET o PUT.

Esempi di variabili di riferimento di un file:

```
INPUT^
ACCOUNTS_PAYABLE.FILE^
```

VARIABILI DI RIFERIMENTO

Le variabili o i valori di riferimento indicano i dati che si riferiscono a qualche tipo di dati. Esistono tre varieta' di variabili e valori di riferimento:

1. variabili e valori di puntatore

2. variabili e valori ADR

3. variabili e valori ADS

In generale una variabile o un valore di riferimento "punta" a dei dati in oggetto. In questo modo il valore di una variabile o valore di riferimento e' un riferimento a quei dati.
Per ottenere gli attuali dati effettivamente indicati, si deve "prelevare il contenuto" della variabile di riferimento aggiungendo una freccia in alto (^) alla variabile o al valore.

Esempio di uso di valori puntatore:

```
VAR  P,Q : ^INTEGER;
{P e Q sono puntatori a interi.}

NEW (P); NEW (Q);
{P e Q sono assegnati a valori di riferimento di zone}
{della memoria corrispondenti a dati di tipo INTEGER.}

P := Q;
{P e Q ora puntano la stessa zona di memoria.}

P^ := 123;
{Assegna il valore 123 al valore INTEGER puntato da P.}
{Poiche' anche Q punta a questa locazione, anche ad esso}
{viene assegnato 123.}
```

Usare NIL^ e' un errore (poiche' un puntatore a NIL non si riferisce a nulla). Al livello Esteso, si puo' anche aggiungere una freccia in alto (^) ad un indicatore di funzione per una funzione che ritorna un puntatore o un tipo indirizzo. In questo caso la freccia in alto indica il valore cui si riferisce il valore ritornato. Questa variabile non puo' essere assegnata o passata come parametro di riferimento.

Esempi di funzioni che ritornano valori di riferimento:

```
DATA1 := FUNK1 (I, J)^
{FUNK1 ritorna un valore di riferimento. La freccia in alto}
{preleva il contenuto del valore di riferimento ritornato, assegnando}
{a DATA1 i dati indicati.}

DATA2 := FUNK2 (K, L)^.FOO [2]
{FUNK2 ritorna un valore di riferimento. La freccia in alto preleva}
{il contenuto del valore di riferimento ritornato. In questo caso}
{il valore del contenuto e' un record.}
{Il componente di array FOO [2] di quel record e' assegnato alla}
{variabile DATA2.}
```

Se P e' di tipo ADR OF qualche tipo, P.R indica il valore indirizzo del tipo WORD. Se P e' di tipo ADS OF qualche tipo, P.R indica la porzione di spiazzamento (offset) dell'indirizzo e P.S la porzione segmento dell'indirizzo. Entrambe le porzioni sono di tipo WORD.

Esempi di variabili indirizzo:

```
BUFF_ADR.R
DATA_AREA.S
```

ATTRIBUTI

Al livello Esteso del Pascal, una dichiarazione di variabile o l'intestazione di una procedura o funzione puo' includere uno o piu' attributi. L'attributo di una variabile fornisce al compilatore speciali informazioni sulla variabile.

La tabella 12-1 visualizza gli attributi forniti dal Pascal per le variabili.

Attributo	Variabile
STATIC	Allocata su una locazione fissa, non sullo stack
PUBLIC	Accessibile da altri moduli tramite EXTERN, implica STATIC
EXTERN	Dichiarata come PUBLIC in un altro modulo, implica STATIC
ORIGIN	Collocata su indirizzi specifici, implica STATIC
PORT	Indirizzo di I/O, implica STATIC
READONLY	Non puo' essere alterata o scritta

Tabella 12-1 Attributi di Variabili

L'attributo EXTERN e' anche una direttiva di procedura e funzione; come PUBLIC e ORIGIN sono anche attributi di procedure e funzioni. Per una descrizione degli attributi e delle direttive di procedure e funzioni si puo' vedere "Attributi e Direttive", nel Capitolo 15. Le sezioni seguenti descrivono in dettaglio ogni attributo di variabili.

Si possono assegnare attributi alle variabili solo in una sezione VAR. Non e' consentito specificare gli attributi delle variabili in una sezione TYPE o in una lista di parametri di procedure o funzioni.

In una dichiarazione di variabili si assegnano uno o piu' attributi, racchiusi tra parentesi quadre e separati da virgole (se viene specificato piu' di un attributo).

La posizione delle parentesi quadre specifica uno dei due casi:

1. In una sezione VAR, un attributo tra parentesi quadre dopo un identificatore di variabile, si applica solo a quella variabile.

2. Un attributo tra parentesi quadre, dopo la parola riservata VAR, si applica a tutte le variabili della sezione.

Esempi che specificano gli attributi delle variabili:

```
VAR A, B, C [EXTERN] : INTEGER;
{Si applica solo a C.}

VAR [PUBLIC] A, B, C : INTEGER;
{Si applica ad A, B e C.}

VAR [PUBLIC] A, B, C [ORIGIN 16#1000] : INTEGER;
{A, B e C sono tutti PUBLIC.  ORIGIN di C e'}
{l'indirizzo assoluto esadecimale 1000.}
```

L'ATTRIBUTO STATIC

L'attributo STATIC da' ad una variabile una locazione univoca e fissa in memoria. Cio' e' in contrasto con una variabile di procedura o funzione allocata nello stack oppure allocata in modo dinamico nell'area heap. L'uso di variabili STATIC puo' evitare perdite di tempo e di spazio per il codice, ma aumenta lo spazio dati.

A tutte le variabili a livello di programma, modulo o unita' e' automaticamente assegnata una locazione fissa di memoria oltre l'attributo STATIC.

Le funzioni e le procedure che usano variabili STATIC possono essere eseguite ricorsivamente, pero' le variabili STATIC devono essere usate solo per dati comuni a tutte le chiamate. Poiche' la maggior parte degli altri attributi di variabili implica l'attributo STATIC, cio' che consente di evitare perdita di tempo e di spazio di codice oppure riduzione dello spazio dati e' il ricorso agli attributi PUBLIC, EXTERN, ORIGIN e PORT.

I file dichiarati in una procedura o funzione con l'attributo STATIC sono inizializzati quando e' lanciata la routine, sono chiusi quando la routine termina come altri file. Tuttavia le altre variabili STATIC sono inizializzate soltanto prima dell'esecuzione del programma. Cio' significa che, tranne che per le variabili di FILE aperto, le variabili STATIC possono essere usate per memorizzare i valori tra le chiamate di una procedura o funzione.

Esempi di dichiarazioni di variabile STATIC:

```
VAR VECTOR [STATIC]: ARRAY [0..MAXVEC] OF INTEGER;
VAR [STATIC] I, J, K : 0..MAXVEC;
```

L'attributo STATIC non si applica a procedure o funzioni, come fanno altri attributi.

GLI ATTRIBUTI PUBLIC E EXTERN

L'attributo PUBLIC indica una variabile a cui si puo' accedere da altri moduli caricati; l'attributo EXTERN identifica una variabile che risiede in qualche altro modulo caricato. L'identificatore e' passato al linker nel file di codice oggetto generato (dove puo' essere troncato se il linker impone una limitazione di lunghezza).

Le variabili con attributo PUBLIC o EXTERN sono implicitamente STATIC.

Esempi di dichiarazioni di variabili PUBLIC e EXTERN:

```
VAR [EXTERN] GLOBE1, GLOBE2: INTEGER;
{Le variabili GLOBE1 e GLOBE2 sono dichiarate EXTERN, cio' significa}
{che devono essere dichiarate PUBLIC in qualche altro modulo caricato.}

VAR BASE_PAGE [PUBLIC, ORIGIN #12FE]: BYTE;
{La variabile BASE_PAGE e' collocata a 12FE, esadecimale. Poiche'}
{essa e' anche PUBLIC, vi si puo' accedere da altri moduli caricati}
{che dichiarano BASE_PAGE con l'attributo EXTERN.}
```

Le variabili PUBLIC sono di solito allocate dal compilatore, se non si assegna loro l'attributo ORIGIN. Assegnare ad una variabile gli attributi PUBLIC e ORIGIN, e' un modo per dire al caricatore che un nome globale ha un indirizzo assoluto. PUBLIC non puo' essere combinato con PORT.

Se PUBLIC e ORIGIN sono entrambi presenti, il compilatore non ha bisogno del caricatore per tradurre l'indirizzo. Tuttavia, l'identificatore e' sempre passato al linker per poter essere usato da altri moduli.

Le variabili EXTERN non sono allocate dal compilatore. Non hanno neppure un ORIGIN, poiche' assegnare sia EXTERN che ORIGIN implica due modi differenti di accedere alla variabile. La parola riservata EXTERNAL e' sinonimo di EXTERN. Cio' accresce la portabilita' da altri Pascal, poiche' questi di solito utilizzano una delle due.

Alle variabili nell'interfaccia di un'unita' e' automaticamente assegnato l'attributo PUBLIC oppure EXTERN. Se un programma, modulo o unita' utilizza tramite la clausola USE un'interfaccia, le sue variabili sono rese EXTERN; se si compila la IMPLEMENTATION dell'interfaccia, le sue variabili sono rese PUBLIC.

GLI ATTRIBUTI ORIGIN E PORT

L'attributo ORIGIN ordina al compilatore di collocare una variabile in un determinato indirizzo di memoria; l'attributo PORT specifica un certo tipo di indirizzo I/O. In entrambi i casi l'indirizzo deve essere una costante di un qualunque tipo ordinale. Ai punti di accesso I/O, ai vettori di interruzione, ai dati di sistema operativo e ad altri dati relativi si puo' accedere tramite le variabili con ORIGIN o PORT.

Esempi di dichiarazioni di variabili con attributi ORIGIN e STATIC:

```
VAR KEYBOARDP [PORT 16#FFF2]:  CHAR;
VAR INTRVECT [ORIGIN 8#200]:  WORD;
```

Le variabili con attributi ORIGIN o PORT sono implicitamente STATIC. Inoltre esse impediscono l'ottimizzazione di comuni sottoespressioni. Per esempio, se GATE ha l'attributo ORIGIN, le due istruzioni X := GATE; Y := GATE accedono a GATE due volte nell'ordine dato, invece di usare il primo valore per entrambe le assegnazioni. Questo assicura una corretta operazione, se GATE rappresenta un punto di accesso input in memoria. Tuttavia, se GATE e' passato come un parametro di riferimento, i riferimenti al parametro possono essere ottimizzati altrove. Per questo motivo le variabili PORT non possono essere passate come parametri di riferimento.

Le variabili con ORIGIN e PORT non sono mai allocate o inizializzate dal compilatore. L'indirizzo associato indica soltanto dove viene trovata la variabile. ORIGIN implica sempre un indirizzo di memoria, ma il significato di PORT varia al variare dell'implementazione.

Nella maggior parte delle implementazioni, si assume che l'I/O sia in memoria, percio' PORT e' sinonimo di ORIGIN. Altre implementazioni usano le istruzioni originarie di macchina per l'input e l'output. Altre ancora chiamano l'input del punto di accesso e le routine di output per ogni accesso.

Per ulteriori informazioni sull'attributo PORT si puo' vedere l'Appendice A, "Caratteristiche di Implementazione", nel manuale "Linguaggio Pascal Guida Utente".

Assegnare gli attributi PORT e ORIGIN tra parentesi quadre immediatamente dopo la parola chiave VAR, e' ambiguo e genera un errore durante la compilazione. (E' poco chiaro per il compilatore se tutte le variabili seguenti sono allo stesso indirizzo o se gli indirizzi sono assegnati sequenzialmente.)

```
VAR [ORIGIN 0] FIRST, SECOND:  BYTE;     {Non valido}
```

ORIGIN (ma non PORT) consente un indirizzo segmentato usando la notazione "segmento: spiazzamento".

```
VAR SEGVECT [ORIGIN 16#0001 : 16#FFFE]:  WORD;
```

Di solito una variabile con l'attributo ORIGIN segmentato non puo' essere usata come variabile di controllo in un'istruzione FOR.

L'ATTRIBUTO READONLY

L'attributo READONLY impedisce assegnazioni ad una variabile e che questa possa essere passata come un parametro VAR o VARS. Inoltre una variabile READONLY non puo' essere letta con un'istruzione READ o usata come una variabile di controllo FOR. Si puo' usare READONLY con qualunque degli altri attributi.
Esempi di dichiarazioni di variabili con attributo READONLY:

```
VAR INPORT [PORT 12, READONLY]: BYTE;
{INPORT e' una variabile con attributi READONLY e PORT.}

VAR [READONLY] I, J [PUBLIC], K [EXTERN]: INTEGER;
{I, J e K sono tutte READONLY;}
{J e' anche PUBLIC; K e' anche EXTERN.}
```

Ai parametri CONST e CONSTS, come alle variabili di controllo del ciclo FOR (mentre sono nel corpo del ciclo), e' automaticamente assegnato l'attributo READONLY. READONLY e' il solo attributo di variabile che non implica l'allocazione STATIC.

Una variabile sia READONLY che PUBLIC o EXTERN in un file sorgente non e' necessariamente READONLY quando viene usata in un altro file sorgente. L'attributo READONLY non si applica a procedure e funzioni.

COMBINAZIONE DI ATTRIBUTI

Si possono dare ad una variabile vari attributi, separati da virgole, e si deve racchiudere la lista tra parentesi quadre:

```
VAR [STATIC]
X, Y, Z [ORIGIN #FFFE, READONLY]: INTEGER;
```

In questo esempio Z e' una variabile con gli attributi STATIC, READONLY con un ORIGIN all'indirizzo FFFE esadecimale. Nella combinazione di attributi si applicano queste regole:

- se si assegna ad una variabile l'attributo EXTERN, non le si possono assegnare gli attributi PORT, ORIGIN o PUBLIC nella compilazione corrente.

- se si assegna ad una variabile l'attributo PORT, non le si possono assegnare gli attributi ORIGIN, PUBLIC o EXTERN.

- se si assegna ad una variabile l'attributo ORIGIN, non le si possono anche assegnare gli attributi PORT o EXTERN. Tuttavia si puo' combinare ORIGIN con PUBLIC.

- se si assegna ad una variabile l'attributo PUBLIC, non le si possono anche assegnare gli attributi PORT o EXTERN. Tuttavia si puo' combinare PUBLIC con ORIGIN.

- si possono usare STATIC e READONLY con qualunque altro attributo.

13. ESPRESSIONI

SOMMARIO

Questo capitolo definisce le espressioni usate in Pascal, classificandole in semplici, booleane e set. Descrive poi gli indicatori di funzione, le regole che si devono rispettare per formulare e valutare un'espressione ed altre caratteristiche relative alle espressioni.

INDICE

INTRODUZIONE

Le espressioni sono costruzioni che danno come risultato dei valori. La tabella 13-1 illustra una varieta' di espressioni che, se A = 1 e B = 2, danno come risultato il valore indicato.

Espressione	Valore
2	2
A	1
A + 2	3
(A + 2)	3
(A + 2) * (B - 3)	-3

Tabella 13-1 Espressioni

In un'espressione gli operandi possono essere un valore o qualunque altra espressione. Quando un qualunque operatore e' applicato ad un'espressione, questa e' chiamata operando. Tramite le parentesi che servono a raggruppare e tramite gli operatori che usano altre espressioni, si possono costruire espressioni della lunghezza e complessita' desiderate.

Questi sono gli operatori disponibili, nell'ordine in cui sono applicati:

- Unario

  NOT [ADR ADS]

- Moltiplicatore

  * / DIV MOD AND (ISR SHL SHR)

- Addizionatore

  + - OR (XOR)

- Relazionale

  = <> <= >= < > IN

Gli operatori indicati in parentesi tonde sono disponibili solo al livello Esteso del Pascal; quelli in parentesi quadre solo al livello di Sistema.

Un'espressione Pascal e' un valore oppure il risultato dell'applicazione di un operatore ad uno o due valori. Anche se un valore puo' essere di quasi tutti i tipi, la maggior parte degli operatori Pascal si applica solto ai tipi seguenti:

INTEGER
WORD
REAL
lNTEGER4
BOOLEAN
SET

Gli operatori relazionali si applicano anche ai tipi CHAR, enumerati, stringa e di riferimento. Per tutti gli operatori (tranne l'operatore su insiemi lN), gli operandi devono avere tipi compatibili.

ESPRESSIONI DI TIPI SEMPLICI

Come regola, gli operandi e il valore risultante da un'operazione sono tutti dello stesso tipo. A volte, tuttavia, il tipo di un operando e' convertito nel tipo richiesto da un operatore.

Questa conversione avviene a due livelli: uno soltanto per gli operandi di costante, e uno per tutti gli operandi. La conversione da lNTEGER a WORD avviene soltanto per gli operandi costanti; quella da INTEGER a REAL e da lNTEGER o WORD a INTEGER4 avviene per tutti gli operandi.

Se necessario, nelle espressioni costanti i valori lNTEGER si convertono nel tipo WORD. Occorre fare attenzione quando si mescolano costanti lNTEGER e WORD nelle espressioni. Per esempio, se CBASE e' la costante 16#C000 e DELTA e' la costante -1, l'espressione seguente segnala un superamento di capacita' (overflow) WORD:

```
WRD (CBASE) + DELTA
```

Il superamento di capacita' si verifica perche' DELTA e' convertito nel valore WORD 16#FFFF, e 16#C000 piu' 16#FFFF e' maggiore di MAXWORD. Invece, la seguente espressione e' corretta:

```
WRD (ORD (CBASE) + DELTA)
```

Questa espressione da' il valore lNTEGER -16385, che si converte in WORD 16#BFFF. Se la conversione e' richiesta da un operatore o per un'assegnazione, il compilatore esegue le seguenti conversioni:

- da INTEGER a REAL o lNTEGER4
- da WORD a lNTEGER4

Le seguenti regole determinano il tipo del risultato di un'espressione che comprende questi tipi semplici:

1. + - *

   Questi operatori operano su tipi INTEGER, REAL, WORD e INTEGER4, come indicato nei seguenti esempi:

```
+123
A + 123
-23.4
A - 8
A * B * 3
```

   E' consentito mescolare tipi REAL con INTEGER, e INTEGER4 con INTEGER o WORD. Dove entrambi gli operandi sono dello stesso tipo, il tipo del risultato e' quello degli operandi. Se ciascun operando e' REAL, il tipo del risultato e' REAL; altrimenti, se ciascun operando e' INTEGER4, il tipo del risultato e' INTEGER4.

   Sono ammessi gli operatori unari piu' (+) e meno (-), insieme alle forme binarie. Il meno unario su un tipo WORD e' il complemento a 2 (NOT e' il complemento a 1); poiche' non ci sono valori WORD negativi, cio' genera sempre un segnale di avvertimento.

   Il meno unario ha lo stesso livello di precedenza degli operatori addizionali:

```
(X * -1)            {Non e' consentito}

(-256 AND X)        {E' interpretato come -(256 AND X)}
```

2. /

   Questo e' il "vero" operatore della divisione. Il risultato e' sempre REAL. Gli operandi possono essere INTEGER o REAL (non WORD o INTEGER4).

   Esempi di divisione:

```
34 / 26.4 = 1.28787...
18 / 6    = 3.00000...
```

3. DIV MOD

   Questi sono gli operatori rispettivamente per quoziente e resto di divisione intera. L'operando di sinistra (dividendo) e' diviso dall'operando di destra (divisore).

   Esempi di divisione intera:

```
 123 MOD  5 =  3
-123 MOD  5 = -3   {Il segno del risultato e' il segno del dividendo}
 123 MOD -5 =  3
 1.3 MOD  5        {Non consentito con operandi REAL}
```

```
 123 DIV  5 =  24
 1.3 DIV  5        {Non consentito con operandi REAL}
```

Entrambi gli operandi devono essere dello stesso tipo: INTEGER, WORD o INTEGER4 (non REAL). Il segno del resto (MOD) e' sempre quello del dividendo.

Il Pascal si differenzia dall'attuale versione dello standard ISO riguardo la semantica di DIV e MOD con operandi negativi, ma il codice risultante e' piu' efficiente. I programmi che si vogliono trasferire devono usare DIV e MOD solo se entrambi gli operandi sono positivi.

4. AND OR XOR NOT

Questi operatori del livello Esteso sono funzioni logiche a livello di bit. Gli operandi devono essere INTEGER o WORD o INTEGER4 (mai un misto), e non possono essere REAL. Il risultato e' del tipo degli operandi.

NOT e' un'operazione di complemento a uno a livello di bit sul singolo operando. Se una variabile V di tipo INTEGER ha il valore MAXINT, NOT V da' il valore INTEGER non consentito -32768. Cio' genera un errore se il controllo di inizializzazione e' attivato e il valore e' usato successivamente nel programma.

Dati i seguenti valori iniziali INTEGER,

```
X = 2#1111000011110000
Y = 2#1111111100000000
```

AND, OR, XOR e NOT eseguono le seguenti funzioni:

```
X AND Y      1111000011110000
             1111111100000000
             ----------------
             1111000000000000

X OR Y       1111000011110000
             1111111100000000
             ----------------
             1111111111110000

X XOR Y      1111000011110000
             1111111100000000
             ----------------
             0000111111110000

NOT X        1111000011110000
             ----------------
             0000111100001111
```

5. SHL SHR 1SR

Questi operatori di livello Esteso forniscono funzioni di shift a livello di bit.

SHL e SHR sono shift logici destri e sinistri. 1SR e' uno shift aritmetico a destra di un intero con segno: il bit con segno e' sempre trasmesso, anche su un operando di tipo WORD. Poiche' il compilatore non puo' generare un semplice shift a destra per le divisioni di 1NTEGER (-1 D1V 2 e' scorretto) e la divisione e' un'operazione che richiede molto tempo, allora si possono usare, appropriatamente, SHR o 1SR invece di DIV.

Gli operandi devono essere 1NTEGER, WORD o 1NTEGER4; non possono essere REAL. 11 risultato e' dello stesso tipo degli operandi.

L'operando sinistro viene "shiftato", e il destro contiene il calcolo dello shift in bit. Un calcolo dello shift minore di 0 o maggiore di 32 produce risultati non definiti e genera un messaggio di errore se e' attivo il controllo di variabilita'. Gli shift non causano mai errori di overflow; i bit shiftati sono semplicemente perduti.

Dato che X = 2#1111111100000000, le funzioni di shift eseguono le seguenti operazioni:

X	1111111100000000	
X SHL 1	1111111000000000	
X SHR 1	0111111110000000	
X 1SR 1	1111111110000000	{estensione di segno}

ESPRESSIONI BOOLEANE

Gli operatori booleani al livello Standard del Pascal sono:

NOT	AND	OR
=	<	>
<>	<=	>=

XOR e' disponibile ai livelli Esteso e di Sistema.

Si puo' anche usare P <> Q come una funzione di OR esclusivo. Poiche' FALSE < TRUE, P <= Q indica l'operazione booleana "P implica Q". 1noltre gli operatori booleani AND e OR non sono gli stessi degli operatori WORD e 1NTEGER con lo stesso nome, che sono funzioni logiche tra bit. Gli operatori booleani AND e OR possono o no calcolare le proprie operazioni.

L'esempio seguente illustra come e' pericoloso supporre che non lo facciano:

```
WHILE (I <= MAX) AND (V [I] <> T) DO I := I + 1;
```

Se l'array V ha un estremo superiore MAX, allora la valutazione di V [I] per I > MAX e' un errore di runtime. Questa valutazione puo' o no avvenire. Qualche volta entrambi gli operandi sono valutati durante l'ottimizzazione, e qualche volta la valutazione di uno puo' alterare quella dell'altro. In quest'ultimo caso, l'uno o l'altro operando puo' essere valutato per primo.

Si puo' invece usare la seguente costruzione:

```
WHILE I <= MAX DO
   IF V [I] <> T THEN I := I + 1 ELSE BREAK;
```

Per avere informazioni sull'uso di AND THEN e di OR ELSE nelle situazioni in cui, come nell'esempio precedente, i testi sono esaminati sequenzialmente, si puo' vedere la sezione sul "Controllo Sequenziale", nel Capitolo 14.

Gli operatori relazionali producono un risultato booleano. I tipi degli operandi di un operatore relazionale (tranne IN) devono essere compatibili. Se non lo sono, uno deve essere REAL e l'altro compatibile con INTEGER.

I tipi di riferimento possono soltanto essere confrontati con = e <>. Per confrontare un tipo indirizzo con uno degli altri operatori relazionali, si deve usare la notazione del campo indirizzo:

```
IF (A.R < B.R) THEN <istruzione>;
```

Ad eccezione dei tipi stringa STRING e LSTRING, non si possono confrontare file, array e record come per gli interi. Per essere confrontati, due tipi STRING devono avere lo stesso estremo superiore; due tipi LSTRING possono avere estremi superiori differenti.

Nei confronti di tipi LSTRING, vengono ignorati i caratteri che superano la lunghezza corrente. Se tale lunghezza di un LSTRING e' minore della lunghezza dell'altro e tutti i caratteri fino alla lunghezza del piu' corto sono uguali, il compilatore assume che il piu' corto e' "minore" del piu' lungo. Tuttavia due LSTRING sono considerati uguali solo se tutti i caratteri correnti e le loro lunghezze correnti sono uguali.

I sei operatori relazionali =, <>, <=, >=, < e > assumono il loro significato normale quando applicati a operandi numerici, enumerati, CHAR o stringa. La sezione "Espressioni di Insiemi" in questo capitolo descrive il significato di questi operatori relazionali (compreso l'operatore relazionale IN) quando applicati a insiemi. Poiche' gli operatori relazionali nelle espressioni booleane hanno una minore priorita' di AND e OR, la seguente espressione non e' corretta:

```
IF I < 10 AND J = K THEN
```

Si deve invece scrivere:

```
IF (I < 10) AND (J = K) THEN
```

Inoltre non si possono usare i tipi numerici dove e' richiesto un operando booleano. Per un intero I, la clausola IF I THEN non e' consentita; al suo posto si deve usare la seguente:

```
IF I <> 0 THEN
```

Si deve tuttavia osservare che i metacomandi Pascal consentono di scrivere:

```
$IF I $THEN
```

L'inclusione di valori speciali "not-a-number" (NaN) significa che un confronto tra due numeri reali puo' dare un risultato diverso da minore di, uguale a, oppure maggiore di. I numeri possono essere non ordinati, cioe' uno o entrambi sono NaN. Un risultato non ordinato e' lo stesso che "non uguale a, non minore di, e non maggiore di".

Per esempio, se le variabili A o B sono valori NaN:

- A < B e' falso
- A <= B e' falso
- A > B e' falso
- A >= B e' falso
- A = B e' falso
- A <> B e', comunque, vero

I confronti REAL non seguono le stesse regole degli altri confronti in molti modi. A < B non e' sempre lo stesso che NOT (B <= A); cio' previene alcune ottimizzazioni altrimenti eseguite dal compilatore. Se A e' un NaN, allora A <> A e' vero; infatti e' un buon modo per controllare un valore NaN.

ESPRESSIONI DI INSIEMI

La tabella 13-2 indica gli operatori Pascal che si applicano in modo diverso ad insiemi che ad altri tipi di espressioni.

Operatore	Significato delle operazioni su insiemi
+	Unione di insiemi
-	Differenza tra insiemi
*	Intersezione tra insiemi
=	Controllo sull'uguaglianza tra insiemi
<>	Controllo sulla disuguaglianza tra insiemi
<= e >=	Controllo su sottoinsiemi e soprainsiemi
< e >	Controllo su sottoinsiemi e soprainsiemi propri
IN	Controllo sugli elementi dell'insieme

Tabella 13-2 Operatori su Insiemi

Qualunque operatore il cui tipo e' SET OF S, dove S e' un subrange di T, e' trattato come SET OF T, (T e' limitato ad un campo di variabilita' compreso tra 0 e 255 o ai valori ORD equivalenti). Entrambi gli operandi devono essere PACKED, oppure nessuno deve essere PACKED, a meno che un operando non sia una costante o un insieme costruito.

Con l'operatore IN, l'operando sinistro (un ordinale) deve essere compatibile con il tipo base dell'operando destro (un insieme). Il valore dell'espressione X IN B e' TRUE se X e' un elemento dell'insieme B, altrimenti e' FALSE. X puo' essere, correttamente, esterno al campo di variabilita' del tipo base di B. Per esempio, X IN B e' sempre falso se le seguenti istruzioni sono vere:

```
X = 1
B = SET OF 2..9
```

(1 e' compatibile, ma non compatibile per l'assegnazione, con 2..9).

Le parentesi ad angolo sono operatori di insiemi solo al livello Esteso del Pascal, poiche' lo standard ISO non li ammette per gli insiemi. Esse controllano che un insieme sia un sottoinsieme proprio oppure un soprainsieme di un altro insieme. La formazione di sottoinsiemi propri non prevede l'uso di un insieme come sottoinsieme, se i due insiemi sono uguali.

Le espressioni che comprendono insiemi possono usare il "costruttore set", che presenta gli elementi di un insieme racchiusi tra parentesi quadre. Ciascun elemento puo' essere un'espressione il cui tipo e' nel tipo base dell'insieme oppure e' delimitato dagli estremi inferiori e superiori del campo di variabilita' degli elementi nel tipo base. Gli elementi non possono essere insiemi essi stessi.

Esempi di insiemi che comprendono costruttori set:

```
SET_COLOR := [RED, BLUE.. PURPLE] - [YELLOW]

SET_NUMBER := [12, J+K, TRUNC (EXP (X))..TRUNC (EXP (X+1))]
```

La sintassi del costruttore set e' simile a quella delle costanti CASE. Se X > Y allora [X..Y] indica l'insieme vuoto. Le parentesi vuote indicano anche l'insieme vuoto e sono compatibili con tutti gli insiemi. Inoltre, se tutti gli elementi sono costanti, un costruttore set e' come una costante set.

Come altre costanti strutturate, l'identificatore di tipo di una costante set puo' essere incluso in una costante set, come in COLORSET [RED..BLUE]. Cio' non significa che ad un costruttore set con elementi variabili non puo' essere assegnato un tipo in un'espressione: NUMBERSET [I..J] non e' consentito se I o J e' una variabile.

Un costruttore set come [I, J..K], oppure un insieme non tipizzato come [1, 5..7], e' compatibile con un insieme PACKED o non impaccato. Una costante di insieme tipizzato, come DIGITS [1, 5..7], e' compatibile soltanto con insiemi PACKED o non impaccati, rispettivamente come il tipo esplicito della costante.

INDICATORI DI FUNZIONE

Un indicatore di funzione specifica l'attivazione di una funzione. Esso e' costituito dall'identificatore di funzione, seguito da una lista (possibilmente vuota) di "parametri attuali" in parentesi tonde:

```
{Dichiarazione della funzione ADD.}
FUNCTION ADD (A, B: INTEGER): INTEGER;
   .
   .
{Uso della funzione ADD in un'espressione.}
X := ADD (7, X * 4) + 123;
{ADD e' un indicatore di funzione.}
```

Questi parametri attuali sostituiscono, posizione per posizione, i loro corrispondenti "parametri formali" definiti nella dichiarazione di funzione.

I parametri possono essere variabili, espressioni, procedure o funzioni. Se la lista di parametri e' vuota, le parentesi devono essere omesse. (Per ulteriori informazioni sui parametri si puo' vedere la sezione sui "Parametri di Procedure e Funzioni", nel Capitolo 15).

L'ordine di valutazione ed associazione dei parametri attuali varia a seconda delle ottimizzazioni usate. Se il metacomando $SIMPLE e' attivato, l'ordine e' da sinistra a destra.

Nella maggior parte dei linguaggi per calcolatori, le funzioni hano due usi differenti:

1. In senso matematico, esse prendono uno o piu' valori da un dominio per produrre un valore risultante in un campo di variabilita'. In questo caso, se la funzione non fa mai altre cose (come assegnare una variabile globale o eseguire un input/output), e' chiamata funzione "pura".

2. Il secondo tipo di funzione puo' avere effetti collaterali, come la conversione di una variabile statica o di un file. Le funzioni di questo genere sono dette "impure".

Al livello Standard, una funzione puo' ritornare un tipo semplice o un puntatore. Al livello Esteso, una funzione puo' ritornare qualunque tipo assegnabile (cioe' qualunque tipo tranne un file o un super array).

Al livello Standard un puntatore ritornato da una funzione puo' soltanto essere confrontato, assegnato o passato come un parametro valore. Al livello Esteso, tuttavia, e' prevista la normale sintassi di selezione per tipi di riferimento, array e record, preceduta dall'indicatore di funzione. Per ulteriori informazioni si puo' vedere "Uso di Variabili e Valori", nel Capitolo 12.

Esempi di indicatori di funzione:

```
SIN (X+Y)

NEXTCHAR

NEXTREC (17) ^
{Qui la funzione ritorna un tipo puntatore, ed e' prelevato}
{il contenuto del valore del puntatore ritornato.}

NAD.NAME [1]
{Qui la funzione non ha parametri. Il tipo di ritorno e' un record,}
{e un campo di esso e' un array. L'identificatore per quel campo e'}
{NAME. L'esempio suddetto seleziona il primo componente dell'array}
{del record ritornato.}
```

E' piu' efficiente ritornare un componente di una struttura piuttosto che ritornare una struttura e poi usare soltanto un componente di essa. Il compilatore tratta una funzione che ritorna una struttura come una procedura, con un parametro VAR extra che rappresenta il risultato della funzione. La routine chiamante della funzione alloca una variabile

nascosta (nello stack) per ricevere il valore di ritorno, ma questa "variabile" e' allocata soltanto durante l'esecuzione dell'istruzione che contiene la chiamata di funzione.

ESPRESSIONI DI VALUTAZIONE

Nei casi di ambiguita', un operatore a piu' alto livello viene applicato prima di uno a piu' basso livello. Per esempio, la seguente espressione da' come risultato 7 e non 9:

```
1 + 2 * 3
```

Si devono usare le parentesi per cambiare la priorita' degli operatori. Quindi la seguente espressione da' come risultato 9 piuttosto che 7:

```
(1 + 2) * 3
```

Se il metacomando $SIMPLE e' attivato, le sequenze di operatori che hanno la stessa priorita' sono eseguite da sinistra a destra. Se e' disattivato, il compilatore puo' riordinare le espressioni e valutare le comuni sottoespressioni solo una volta, per generare un codice ottimizzato. La semantica delle relazioni di priorita' e' conservata, ma vengono usate le normali leggi associative e distributive.
Per esempio:

```
X * 3 + 12
```

e' un'ottimizzazione di:

```
3 * (6 + (X - 2))
```

Queste ottimizzazioni possono generare occasionalmente errori di overflow inaspettati.
Per esempio:

```
(I - 100) + (J - 100)
```

sara' ottimizzato in questo modo:

```
(I + J) - 200
```

Cio' puo' generare un errore di overflow, anche se l'espressione di origine non lo generava (per esempio, se "I" e "J" sono ciascuno 16400).

Un'espressione nel file sorgente puo' o no effettivamente essere valutata quando il programma e' in esecuzione. Per esempio l'espressione F (X + Y) * 0 e' sempre zero, percio' non occorre eseguire la sottoespressione (X + Y) e la chiamata di funzione.

Il compilatore non ottimizza le espressioni reali (come le espressioni

intere), questo serve ad assicurare che il risultato di un'espression reale corrisponda sempre alla semplice valutazione dell'espression data.
Per esempio, l'espressione intera

```
((1 + 1) - 1) * J
```

e' ottimizzata a:

```
1 * J
```

ma la stessa espressione con variabili reali non e' ottimizzata, poiche' i risultati possono essere differenti a causa della perdita di precisione. Le normali sottoespressioni, come 2 * X in SIN (2 * X) * COS (2 * X) , possono essere calcolate solo una volta e, se necessario, ricaricate, pero' sono salvate in una speciale precisione intermedia di 80 bit.

L'ordine della valutazione puo' essere fissato dalle parentesi:

```
(A + B) + C
```

e' valutata addizionando prima A e B, ma

```
A + B + C
```

puo' essere valutata addizionando prima A e B, B e C oppure anche A e C.

Qualunque espressione puo' essere passata come un parametro CONST o CONSTS o avere l'"indirizzo" rilevato. L'espressione e' calcolata e memorizzata in una variabile temporanea nello stack, e il suo indirizzo puo' essere utilizzato come un parametro di riferimento o in qualche altro contesto di indirizzo.

Per evitare ambiguita' si deve racchiudere questa espressione, con gli operatori o le chiamate di funzione, tra parentesi. Per esempio, per richiamare una procedura si deve usare FOO (CONST X, Y: INTEGER), FOO (1, (J+14)) invece di FOO (1, J+14).

Cio' comporta una piccola distinzione nel caso delle funzioni.
Per esempio:

```
FUNCTION SUM (CONST A, B: INTEGER): INTEGER;
BEGIN
  SUM := A;
  IF B <> 0 THEN
     SUM := SUM (SUM, (SUM (B, 0) - 1)) + 1
END;
```

In questo esempio SUM e' chiamata ricorsivamente, sottraendo uno da B finche' B e' zero.

L'uso dell'identificatore di funzione in un'istruzione WITH segue una regola simile. Per esempio, data una funzione,COMPLEX, senza parametri, che ritorna un record, "WITH COMPLEX" significa "WITH il valore corrente

della funzione". Cio' puo' verificarsi soltanto all'interno della funzione COMPLEX stessa. Comunque "WITH (COMPLEX)" provoca la chiamata della funzione e l'assegnazione del risultato ad una variabile locale temporanea.

Un altro modo per descrivere cio' e' nella distinzione tra "indirizzo" e "valore". La parte sinistra di un'assegnazione, un parametro di riferimento, gli operatori ADR e ADS e l'istruzione WITH richiedono tutti un indirizzo. La parte destra di un'assegnazione e un parametro di valore richiedono tutti un valore.

Se e' richiesto un indirizzo ma e' disponibile solo un valore (come una costante o un'espressione tra parentesi), il valore deve essere memorizzato in modo da avere un indirizzo. Per le costanti il valore va nella memoria statica; per le espressioni il valore va nella memoria stack (locale). Un identificatore di funzione rinvia al valore corrente della funzione come a un indirizzo, ma provoca la chiamata della funzione come un valore.

Infine, nel dominio di una funzione la procedura intrinseca RESULT consente di riferirsi al valore corrente di una funzione invece di chiamarlo ricorsivamente. Per una funzione F cio' significa che ADR F e ADR RESULT (F) sono la stessa cosa: l'indirizzo del valore corrente di F. RESULT forza l'uso del valore corrente in un modo che equivale a mettere la funzione tra parentesi, come in (F(X)).

ALTRE CARATTERISTICHE DELLE ESPRESSIONI

EVAL e RESULT sono due procedure, disponibili al livello Esteso, da usare con le espressioni. EVAL ottiene il risultato di una procedura da una funzione; RESULT produce il valore corrente di una funzione all'interno di una funzione o di una procedura o funzione nidificata.

Al livello di Sistema, la funzione RETYPE consente di cambiare il tipo di un valore.

LA PROCEDURA EVAL

EVAL valuta i suoi parametri senza chiamare realmente nulla. Generalmente si usa EVAL per ottenere il risultato di una procedura da una funzione. In tali casi, i valori ritornati dalle funzioni non interessano, quindi EVAL e' utile soltanto per le funzioni con effetti laterali. Per esempio, una funzione che fa avanzare all'elemento successivo e inoltre ritorna l'elemento, puo' essere richiamata in EVAL solo per far avanzare all'elemento successivo, poiche' non occorre ottenere un valore di ritorno da una funzione.

Esempi di procedura EVAL:

```
EVAL (NEXTLABEL (TRUE))
EVAL (SIDEFUNC (X, Y), INDEX (4), COUNT)
```

LA FUNZIONE RESULT

Nel dominio di una funzione, la procedura intrinseca RESULT consente di rinviare al valore corrente di una funzione invece di richiamarlo ricorsivamente. Per una funzione F, cio' significa che ADR F e ADR RESULT (F) sono la stessa cosa, cioe' l'indirizzo del valore corrente di F.
RESULT forza l'uso del valore corrente in un modo tale che, mettendo la funzione tra parentesi, come in (F (X)), forza la valutazione della funzione.

Esempi di funzione RESULT:

```
FUNCTION FACTORIAL (I: INTEGER): INTEGER;
BEGIN
  FACTORIAL := 1;
  WHILE I > DO
  BEGIN
    FACTORIAL := I * RESULT (FACTORIAL);
    I := I - 1
  END
END;

FUNCTION ABSVAL (I: INTEGER): INTEGER;
BEGIN
  ABSVAL := I;
  IF I < 0 THEN ABSVAL := -RESULT (ABSVAL)
END;
```

LA FUNZIONE RETYPE

A volte occorre trasformare il tipo di un valore. Cio' puo' essere eseguito con la funzione RETYPE, disponibile al livello di Sistema del Pascal. Se il nuovo tipo e' una struttura, RETYPE puo' essere seguita dalla normale sintassi di selezione. Si deve usare RETYPE con cautela: essa opera sulla memoria a livello di byte e ignora se l'ordine inferiore dei byte di un numero a due byte venga prima o dopo in memoria.

Esempi di funzione RETYPE:

```
RETYPE (COLOR, 3)                 {inverso di ORD}
RETYPE (STRING2, 1 * J + K) [2]   {il risultato puo' variare}
```

14. ISTRUZIONI

SOMMARIO

Le istruzioni indicano le azioni che il programma puo' eseguire.
Questo capitolo prima illustra la sintassi delle istruzioni Pascal, poi distingue e descrive le due categorie di istruzioni: semplici e strutturate.

INDICE

INTRODUZIONE

Il corpo di un programma, procedura o funzione contiene delle istruzioni. Le istruzioni indicano le azioni che il programma puo' eseguire. Esistono due categorie di istruzioni: semplici e strutturate. Un'istruzione semplice non contiene parti che sono esse stesse altre istruzioni; un'istruzione strutturata e' costituita da due o piu' istruzioni.
La tabella 14-1 elenca le istruzioni in ciascuna categoria del Pascal.

Semplice	Strutturata
Assegnazione (:=)	Composta
Procedura	IF/THEN/ELSE
GOTO	CASE
BREAK	FOR
CYCLE	WHILE
RETURN	REPEAT
Vuota	WITH

Tabella 14-1 Istruzioni del Pascal

LA SINTASSI DELLE ISTRUZIONI PASCAL

Le istruzioni Pascal sono separate da un punto e virgola (;) e comprese tra parole riservate come BEGIN e END. Un'istruzione puo' iniziare con una etichetta. Ciascuno di questi tre elementi della sintassi di istruzioni vengono descritti nelle sezioni che seguono.

ETICHETTE

Qualunque istruzione indicata da un'istruzione GOTO deve avere una etichetta. Al livello Standard, una etichetta e' costituita da una o piu' cifre; gli zeri iniziali sono ignorati. Identificatori di costante, espressioni e notazione non decimale non sono considerati etichette.

Tutte le etichette devono essere dichiarate in una sezione LABEL. Al livello Esteso un'etichetta puo' anche essere un identificatore.

Esempio di uso di etichette e di istruzioni GOTO:

```
PROGRAM LOOPS (INPUT, OUTPUT);
LABEL 1, HAWAII, MAINLAND;

BEGIN
  MAINLAND: GOTO 1;
  HAWAII: WRITELN ('Qui io sono in Hawaii');
  1: GOTO HAWAII
END.
```

Si definisce etichetta di ciclo (di loop) qualunque etichetta che precede immediatamente un'istruzione ciclica: WHILE, REPEAT o FOR. Al livello Esteso, un'istruzione BREAK o CYCLE puo' anche indicare un'etichetta di ciclo.

Un'istruzione puo' essere preceduta sia da una lista di costanti CASE che da un'etichetta GOTO; in questo caso vengono prima le costanti CASE e poi l'etichetta GOTO. Nel seguente esempio, 321 e' un valore CASE, 123 e' un'etichetta:

```
321:  123:  IF LOOP THEN GOTO 123
```

SEPARAZIONE DI ISTRUZIONI

Le istruzioni sono separate da punti e virgola. I punti e virgola non fanno terminare le istruzioni. Comunque, poiche' il Pascal consente l'istruzione vuota, non e' quasi mai sbagliato usare il punto e virgola come carattere finale di un'istruzione.

Esempio di punto e virgola per separare istruzioni:

```
BEGIN
  10: WRITELN;
  A := 2 + 3;
  GOTO 10
END
```

Un errore comune e' quello di far terminare la clausola THEN in un'istruzione IF/THEN/ELSE con un punto e virgola. Il seguente esempio genera un messaggio di avvertimento:

```
IF A = 2 THEN WRITELN;
ELSE A := 3
```

Un altro errore comune e' quello di mettere un punto e virgola dopo il DO in un'istruzione WHILE o FOR:

```
FOR I := 1 TO 10 DO;
BEGIN
  A[1] := I;
  B[I] := 10 - 1
END;
```

L'esempio precedente, cosi' come scritto, "esegue" per dieci volte un'istruzione vuota, poi esegue le assegnazioni dell'array per una volta. Poiche' vi sono facoltativi usi consentiti per ripetere un'istruzione vuota, quando cio' si verifica non viene dato alcun segnale di avvertimento.

Il punto e virgola segue anche la parola riservata END in chiusura di un blocco di istruzioni di programma.

LE PAROLE RISERVATE BEGIN E END

Ogni volta che si vuole far eseguire ad un programma un gruppo di istruzioni, invece di una singola istruzione semplice si puo' racchiudere il blocco tra le parole riservate BEGIN e END.

Per esempio, il seguente gruppo di istruzioni fra BEGIN e END viene tutto eseguito se la condizione nell'istruzione IF e' TRUE:

```
IF (MAX > 10) THEN
BEGIN
  MAX = 10;
  MIN = 0;
  WRITELN (MAX, MIN)
END;
WRITELN ('eseguito')
```

Al livello Esteso si puo' sostituire una coppia di parentesi quadre con la coppia di parole chiave BEGIN e END.

ISTRUZIONI SEMPLICI

Un'istruzione si dice semplice se nessuna sua parte costituisce un'altra istruzione. Le istruzioni semplici nel Pascal Standard sono:

- l'istruzione di assegnazione
- l'istruzione di procedura
- l'istruzione GOTO
- l'istruzione vuota

L'istruzione vuota non contiene alcun simbolo e non indica alcuna azione. Essa e' inclusa nella definizione del linguaggio principalmente per permettere di usare un punto e virgola dopo l'ultima di un gruppo di istruzioni comprese tra BEGIN e END.

Il livello Esteso del Pascal aggiunge tre istruzioni semplici: BREAK, CYCLE e RETURN.

ISTRUZIONI DI ASSEGNAZIONE

L'istruzione di assegnazione sostituisce il valore corrente di una variabile con un nuovo valore, che viene specificato come un'espressione. Un'assegnazione si distingue per i caratteri di due punti e uguale adiacenti (:=).

Esempi di istruzioni di assegnazione:

```
A := B

A[1] := 12 * 4 + (B * C)

X : = Y
{Non consentito. I segni di due punti (:) ed uguale (=)}
{devono essere adiacenti.}

A + 2 := B
{Non consentito. A + 2 non e' una variabile.}

A := ADD (1,1)
```

Il valore di un'espressione deve essere compatibile per l'assegnazione con il tipo della variabile. La selezione della variabile puo' comportare l'indicizzazione di un array o il prelevamento del contenuto di un indirizzo o puntatore: in questo caso il compilatore puo', a seconda dell'ottimizzazione eseguita, mescolare queste azioni con la valutazione dell'espressione. Se il metacomando $SIMPLE e' attivato, l'espressione e' valutata per prima.

L'assegnazione ad una variabile non locale (comprendendo quella di ritorno da una funzione) mette un segno di uguale (=) o di percento (%) nella colonna G del listing. (Per ulteriori informazioni su questi e altri simboli usati nel listing, si puo' vedere "Formato del Listing", nel Capitolo 19).

All'interno del blocco di una funzione, un'assegnazione all'identificatore di funzione stabilisce il valore ritornato dalla funzione. L'assegnazione ad un identificatore di funzione puo' avvenire all'interno del corpo effettivo della funzione o nel corpo di una procedura o funzione nidificata in essa.

Se il controllo di variabilita' e' attivato, un'assegnazione ad una variabile set, subrange o LSTRING puo' comportare una chiamata di runtime al codice di controllo dell'errore.

Conformemente all'ottimizzazione del Pascal, ciascuna sezione di codice senza etichetta o altro punto che puo' ricevere controllo, e' utile per riordinare e per eliminare sottoespressioni comuni. Naturalmente l'ordine di esecuzione, quando e' necessario, viene conservato.

Date queste istruzioni,

```
X := A + C + B;
Y := A + B;
Z := A
```

il compilatore puo' generare codice per eseguire le seguenti operazioni:

- prendere il valore di A e salvarlo
- aggiungere il valore di B e salvare il risultato
- aggiungere il valore di C ed assegnarlo a X
- assegnare a Y il valore salvato A + B
- assegnare a Z il valore salvato A

Questa ottimizzazione avviene soltanto se le assegnazioni a X e a Y e il raggiungimento dei valori di A, B o C sono indipendenti. Se C e' una funzione senza l'attributo PURE, e A e' una variabile globale, la valutazione di C puo' trasformare A. Quindi, poiche' l'ordine di valutazione all'interno di un'espressione in questo caso non e' fissato, il valore di A nella prima assegnazione puo' essere sia il valore vecchio che quello nuovo. Tuttavia, poiche' l'ordine di valutazione delle istruzioni e' fissato, il valore di A nella seconda e terza assegnazione e' quello nuovo.

Le seguenti azioni possono limitare la capacita' dell'ottimizzatore di trovare comuni sottoespressioni:

- assegnazione ad una variabile non locale
- assegnazione ad un parametro di riferimento
- assegnazione al referente di un puntatore
- assegnazione al referente di una variabile indirizzo
- chiamata di una procedura
- chiamata di una funzione senza l'attributo PURE

L'ottimizzatore consente degli "pseudonimi", cioe' una singola variabile con due identificatori, ad esempio uno come variabile globale e uno come parametro di riferimento.

ISTRUZIONI DI PROCEDURA

Un'istruzione di procedura esegue la procedura indicata dall'identificatore di procedura.
Per esempio, se si e' definita la procedura DO_IT:

```
PROCEDURE DO_IT;
BEGIN
  WRITELN ('Did it')
END;
```

DO_IT e' ora un'istruzione che puo' essere eseguita semplicemente chiamando il suo nome:

```
DO_IT
```

Se si dichiara la procedura con una lista di parametri formali, l'istruzione di procedura deve includere i parametri attuali.
Il Pascal include un grande numero di procedure predichiarate. Per una completa informazione si puo' vedere il Capitolo 16, "Procedure e Funzioni Disponibili". Una delle procedure predichiarate e' ASSIGN; non occorre dichiararla prima di poterla usare.

```
ASSIGN (INFILE, 'MYFILE')
```

Si deve osservare che la procedura ASSIGN contiene una lista di parametri. Questi parametri sono i parametri attuali che sono legati a quelli formali nella dichiarazione di procedura. Per una descrizione dei parametri formali e di riferimento si puo' vedere la sezione "Parametri di Procedure e Funzioni", nel Capitolo 15.

L'ISTRUZIONE GOTO

Un'istruzione GOTO indica che la successiva elaborazione continua in un'altra parte del testo di programma, cioe' nel punto in cui e' l'etichetta. Si deve dichiarare una LABEL nella sezione dichiarativa LABEL, prima di usarla in un'istruzione GOTO.

Vi sono numerose limitazioni per l'uso delle istruzioni GOTO:

- Una GOTO non deve saltare in un'istruzione nidificata piu' internamente, cioe' in un'istruzione IF, CASE, WHILE, REPEAT, FOR o WITH. Sono consentite istruzioni GOTO da un settore all'altro di un'istruzione IF o CASE.

- E' consentita una GOTO da una procedura o funzione ad un'etichetta nella parte principale di un programma o nel livello superiore di una procedura o funzione. Una GOTO puo' saltare da una di queste istruzioni, ma in questo caso l'istruzione sta direttamente entro il corpo della procedura o funzione. Tuttavia un tale salto genera un codice supplementare nella locazione della GOTO e nella locazione dell'etichetta. La GOTO e l'etichetta devono essere nella stessa unita' compilativa poiche' alle etichette, a differenza delle variabili, non puo' essere assegnato l'attributo PUBLIC.

I seguenti sono esempi di istruzioni GOTO, consentite e non:

```
PROGRAM LABEL_EXAMPLES;
LABEL 1, 2, 3, 4;

  PROCEDURE ONE;
  LABEL 11, 12, 13;

    PROCEDURE IN_ONE;
    LABEL 21;
    {Le GOTO di livello esterno non possono saltare a 21.}

    BEGIN
      IF TUESDAY THEN GOTO 1
      ELSE GOTO 11;
      {1 e 11 sono entrambe etichette valide di livello esterno.}
      21: WRITE ('IN_ONE')
    END;

  BEGIN   {Procedura uno}
    IF RAINING THEN GOTO 1 ELSE GOTO 11;
    {Consentito}
    11: GOTO 21
    {Non consentito. Non si puo' saltare in procedure di}
    {livello piu' interno.}
  END;

  PROCEDURE TWO;
  BEGIN
    GOTO 11
    {Non consentito. Non si puo' saltare in procedure differenti}
    {allo stesso livello.}
  END;

BEGIN   {Livello principale}
  IF SEATTLE
  THEN
    BEGIN
      GOTO 2;
      {E' corretto andare a 2 a livello di programma.}
      4: WRITE ('here')
    END
  ELSE GOTO 4;
  {E' corretto saltare nella clausola THEN.}
  2: GOTO 3;
  {Non consentito. Non si puo' saltare in un'istruzione REPEAT.}
  REPEAT
     WHILE MS_BYRON DO
        3: GOTO 2
        {E' corretto saltare fuori dai cicli.}
  UNTIL DATE;
  1:GOTO 11
  {Non consentito. Non si puo' saltare nella procedura da}
  {programma.}

END.
```

Se il metacomando $GOTO e' attivo, ogni istruzione GOTO e' identificata con un segnale di avvertimento che ricorda che "le istruzioni GOTO sono considerate dannose". Cio' puo' essere utile a scopo didattico o per trovare tutte le istruzioni GOTO in un programma al fine di localizzare un errore. La colonna J (jumps) del listing contiene:

- Un segno piu' (+) o un asterisco (*), che segnalano un'istruzione GOTO a un'etichetta in una parte successiva del listing.

- Un segno meno (-) o un asterisco (*), che segnalano un'istruzione GOTO a un'etichetta gia' incontrata nel listing.

Per maggiori dettagli sui listing si puo' vedere "Formato del Listing", nel Capitolo 19.

LE ISTRUZIONI BREAK, CYCLE E RETURN

Al livello Esteso sono consentite le istruzioni BREAK, CYCLE e RETURN, oltre alle istruzioni semplici gia' descritte. Queste istruzioni compiono le seguenti funzioni:

1. BREAK permette di uscire dal ciclo di esecuzione in corso.

2. CYCLE permette di uscire dall'iterazione corrente di un ciclo e da' inizio all'iterazione successiva.

3. RETURN permette di uscire dalla procedura, funzione, programma o implementazione in corso.

Tutte e tre queste istruzioni sono funzionalmente equivalenti ad un'istruzione GOTO.

1. Un'istruzione BREAK e' una GOTO alla prima istruzione che segue un'istruzione ripetitiva.

2. Un'istruzione CYCLE e' una GOTO ad un'implicita istruzione vuota seguente il corpo di un'istruzione ripetitiva. Questo salto da' inizio alla successiva iterazione di un ciclo. In un'istruzione WHILE o REPEAT, CYCLE esegue il controllo booleano nella clausola WHILE o UNTIL prima di eseguire nuovamente l'istruzione; in un'istruzione FOR, CYCLE va al valore successivo della variabile di controllo.

3. Un'istruzione RETURN e' una GOTO ad un'implicita istruzione vuota seguente l'ultima istruzione nella procedura o funzione in corso, o nel corpo di un programma o implementazione.

La colonna J (jumps) del listing contiene un segno piu' (+) o un asterisco (*) per un'istruzione BREAK, un segno meno (-) o un asterisco (*) per un'istruzione CYCLE, e un asterisco (*) per un'istruzione RETURN. (Per ulteriori informazioni sui listing si puo' vedere "Formato del Listing", nel Capitolo 19).

BREAK e CYCLE hanno due forme, una con un'etichetta di ciclo e una senza. Se si da' un'etichetta di ciclo, l'etichetta identifica il ciclo per uscire o reiniziare. Se non si da' un'etichetta, si considera il ciclo piu' interno, come indicato nel seguente esempio:

```
OUTER: FOR I := 1 TO N1 DO
  INNER: FOR J := 1 TO N2 DO
    IF A [I, J] = TARGET THEN BREAK OUTER;
```

ISTRUZIONI STRUTTURATE

Le istruzioni strutturate sono composte esse stesse da altre istruzioni. Vi sono quattro tipi di istruzioni strutturate:

1. istruzioni composte
2. istruzioni condizioanli
3. istruzioni ripetitive
4. istruzioni WITH

Il livello di controllo e' mostrato nella colonna C (control) del listing. Il valore nella colonna C e' incrementato ogni volta che il controllo passa a un'istruzione nidificata; al contrario, questo valore e' diminuito ogni volta che il controllo ritorna all'istruzione nidificata. Cio' e' utile per individuare un END mancante o supplementare nel programma.

ISTRUZIONI COMPOSTE

L'istruzione composta e' una sequenza di istruzioni semplici, compresa tra le parole riservate BEGIN e END. I componenti di un'istruzione composta eseguono nella stessa sequenza in cui appaiono nel file sorgente.

Esempi di istruzioni composte:

```
BEGIN
  TEMP := A [I];
  A[I] := A [J];
  A[J] := TEMP
  {Il punto e virgola qui non e' necessario.}
END

BEGIN
  OPEN_DOOR;
  LET_EM_IN;
  CLOSE_DOOR;
  {Il punto e virgola significa istruzione vuota.}
END
```

Tutte le strutture di controllo condizionali e ripetitive del Pascal (tranne REPEAT) operano su una singola istruzione, non su istruzioni multiple con separatori finali. In questo contesto, BEGIN e END servono da punteggiatura, come il punto e virgola, i due punti o le parentesi. Se si preferisce, si puo' sostituire una coppia di parentesi quadre alla coppia di parole riservate BEGIN e END. Si deve osservare che una parentesi quadra destra (]) corrisponde solo ad una parentesi quadra sinistra ([), non un BEGIN, CASE o RECORD. In altre parole, la parentesi destra non e' sinonimo di END.

Le parentesi quadre non devono essere usate come sinonimi di BEGIN e END per racchiudere il corpo di un programma, implementazione, procedura o funzione; solo BEGIN e END possono essere usati per questo scopo.

Esempi di parentesi quadre che sostituiscono BEGIN e END:

```
IF FLAG THEN [X := 1; Y := -1]
ELSE [X := -1; Y := 0];

WHILE P.N <> NIL DO
   [Q := P;  P := P.N;  DISPOSE (Q)];

FUNCTION R2 (R: REAL): REAL;
   [R2 := R * 2]
   {Non consentito.}
```

ISTRUZIONI CONDIZIONALI

Un'istruzione condizionale seleziona, per l'esecuzione, solo una delle sue istruzioni componenti. Le istruzioni condizionali sono IF e CASE. Si usa l'istruzione IF per una o due condizioni, l'istruzione CASE per piu' condizioni.

L'Istruzione IF

L'istruzione IF consente l'esecuzione condizionale di un'istruzione. Se l'espressione booleana seguente l'IF e' vera, viene eseguita l'istruzione seguente il THEN. Se l'espressione booleana seguente l'IF e' falsa, viene eseguita l'istruzione seguente l'ELSE, se questa e' presente.

Esempi di istruzioni IF:

```
IF 1 > 0 THEN 1 := 1 - 1
{Il punto e virgola qui non occorre.}
ELSE I := I + 1

IF (1 <= TOP) AND (ARRI [I] <> TARGET) THEN I := I + 1

IF I <=  TOP THEN
   IF ARRI [I] <> TARGET THEN I := 1 + 1

IF 1 = 1 THEN
   IF J = 1 THEN WRITELN ('I equals J')
            ELSE WRITELN ('DONE only if I = 1 and J <> 1')
   {Questo ELSE corrisponde all'IF nidificato piu' internamente.}
   {Quindi il secondo WRITELN e' eseguito solo se 1 = 1 e J <> 1.}

IF 1 = 1 THEN BEGIN
   IF J = 1 THEN WRITELN ('I equals J')
   END
ELSE  WRITELN ('DONE only if I <> 1')
{Ora l'ELSE e' accoppiato con il primo IF, poiche' la seconda}
{istruzione IF e' messa tra parentesi dalla coppia BEGIN/END.}
{Quindi il secondo WRITELN e' eseguito se I <> 1.}
```

Un punto e virgola (;) seguito dall'ELSE e' sempre scorretto. Il compilatore lo salta durante la compilazione ed emette un messaggio di avvertiemnto.

L'espressione booleana seguente un IF puo' includere gli operatori di controllo sequenziale descritti in "Controllo Sequenziale", piu' oltre in questo stesso capitolo.

L'Istruzione CASE

L'istruzione CASE e' costituita da un'espressione (chiamata indice di CASE) e da una lista di istruzioni. Ciascuna istruzione e' preceduta da una lista di costanti, denominata lista di costanti CASE. L'istruzione eseguita e' quella la cui lista di costanti CASE contiene il valore corrente dell'indice CASE. L'indice CASE e tutte le costanti devono essere di tipi ordinali compatibili.

Esempi di istruzioni CASE:

```
CASE OPERATOR OF
  PLUS:  X := X + Y;
  MINUS: X := X - Y;
  TIMES: X := X * Y
END;
{OPERATOR e' l'indice CASE. PLUS, MINUS e TIMES sono costanti CASE.}
{In questo esempio, essi sono tutti i valori che possono essere}
{assunti dalla variabile enumerata OPERATOR.}

CASE NEXTCH OF
  'A'..'Z', '_'          :  IDENTIFIER;
  '+', '-', '*', '/'     :  OPERATOR;
  {Le virgole separano le costanti CASE e i campi di variabilita'}
  {delle costanti CASE.}
  OTHERWISE  WRITE ('Unknown Character')
  {Cioe', se si tratta di qualunque altro carattere}
END;
```

La sintassi della costante CASE e' la stessa delle dichiarazioni di RECORD con varianti. Nel Pascal Standard una costante CASE e' formata da una o piu' costanti separate da virgole. Al livello Esteso, si puo' sostituire un campo di costanti, come 'A'..'Z' con una costante. Nessun valore costante si puo' applicare a piu' di un'istruzione. Il livello Esteso consente anche all'istruzione CASE di terminare con una clausola OTHERWISE. La clausola OTHERWISE contiene istruzioni addizionali da eseguire se il valore dell'indice CASE non sta nell'insieme dato di valori costanti di CASE. Se il valore dell'indice CASE non sta nell'insieme e se non e' presente alcuna clausola OTHERWISE, si verifica una delle due cose:

- Se il controllo di variabilita' e' attivo, si genera un errore di runtime.

- Se il controllo di variabilita' non e' attivo, il risultato non e' definito (e puo' essere catastrofico).

Nel Pascal il controllo non passa automaticamente alla successiva istruzione eseguibile, come nel Pascal UCSD e in qualche altro linguaggio. Se si vuole ottenere questo effetto, si deve includere una clausola OTHERWISE vuota.

Un punto e virgola (;) puo' essere presente dopo l'istruzione finale della lista, ma non e' richiesto. Il compilatore salta un segno di due punti (:) dopo un OTHERWISE ed emette un segnale di avvertimento.

A seconda dell'ottimizzazione, il codice generato dal compilatore per un'istruzione CASE puo' essere costituito da una "tabella di salto" o da serie di confronti (o da entrambi). Se si tratta di una tabella di salto, si puo' verificare un salto ad una locazione arbitraria in memoria, se la variabile di controllo e' esterna al campo di variabilita' e il controllo di variabilita' non e' attivo.

ISTRUZIONI DI RIPETIZIONE

Le istruzioni di ripetizione specificano la ripetuta esecuzione di un'istruzione. Nel Pascal Standard esse includono le istruzioni WHILE, REPEAT e FOR.

A livello Esteso del Pascal, vi sono due istruzioni addizionali, BREAK e CYCLE, per permettere o reiniziare la ripetizione delle istruzioni. Queste istruzioni sono funzionalmente equivalenti ad una GOTO, ma sono piu' facili da usare.

L'Istruzione WHILE

L'istruzione WHILE ripete un'istruzione zero o piu' volte, finche' un'espressione booleana diventa falsa.

Esempi di istruzioni WHILE:

```
WHILE P <> NIL DO P := NEXT (P)

WHILE NOT MICKEY DO
  BEGIN
    NEXTMOUSE;
    MICE := MICE + 1
  END
```

L'espressione booleana in un'istruzione WHILE puo' includere gli operatori di controllo sequenziale descritti in "Controllo Sequenziale", piu' avanti in questo capitolo.

Si usa WHILE se non e' necessaria alcuna iterazione del ciclo; si usa REPEAT dove ci si aspetta la richiesta di almeno una iterazione del ciclo.

L'Istruzione REPEAT

L'istruzione REPEAT ripete una sequenza di istruzioni una o piu' volte, finche' un'espressione booleana diventa vera.

Esempi di istruzioni REPEAT:

```
REPEAT
  READ (LINEBUFF);
  COUNT := COUNT + 1
UNTIL EOF;

REPEAT GAME UNTIL TIRED;
```

L'espressione booleana in un'istruzione REPEAT puo' includere gli operatori di controllo sequenziale descritti in "Controllo Sequenziale", piu' avanti in questo capitolo.

Si usa l'istruzione REPEAT per eseguire istruzioni, non soltanto una volta ma una o piu' volte, finche' una condizione diventa vera. Si puo' osservare la differenza rispetto all'istruzione WHILE, in cui una singola istruzione puo' anche non essere eseguita affatto.

L'Istruzione FOR

L'istruzione FOR ordina al compilatore di eseguire un'istruzione ripetutamente mentre ad una variabile, denominata variabile di controllo dell'istruzione FOR, e' assegnata una progressione di valori. I valori assegnati iniziano e terminano con un valore chiamato rispettivamente iniziale e finale.

L'istruzione FOR ha due forme, una in cui la variabile di controllo aumenta di valore ed una in cui questa diminuisce:

```
FOR I := 1 TO 10 DO
{I e' la variabile di controllo.}
   SUM := SUM + VICTORVECTOR [I]

FOR CH := 'Z' DOWNTO 'A' DO
{CH e' la variabile di controllo.}
   WRITE (CH)
```

Si puo' usare anche un'istruzione FOR per eseguire passo per passo la valutazione di un insieme, come qui descritto:

```
FOR TINT :=
   LOWER (SHADES) TO UPPER (SHADES) DO
      IF TINT IN SHADES
      THEN PAINT_AREA (TINT);
```

Lo standard ISO fornisce regole esplicite riguardo la variabile di controllo nelle istruzioni FOR:

1. Deve essere di un tipo ordinale

2. Deve anche essere una variabile intera, non un componente di una struttura.

3. Deve essere locale al programma, procedura o funzione che la comprendono direttamente, e non puo' essere un parametro di riferimento della procedura o funzione.

   Comunque, al livello Esteso del Pascal, la variabile di controllo puo' anche essere una variabile STATIC, come una variabile dichiarata a livello di programma, a meno che la variabile non abbia l'attributo segmentato ORIGIN. L'uso di una variabile a livello di programma e' un errore ISO non rilevato.

4. Nell'istruzione ripetuta non e' consentita alcuna assegnazione alla variabile di controllo. Questo errore e' rilevato facendo il READONLY della variabile di controllo entro l'istruzione FOR; non e' rilevato quando una procedura o funzione richiamata dall'istruzione

ripetuta altera la variabile di controllo. La variabile di controllo non puo' essere passata come un parametro VAR (o VARS) ad una procedura o funzione.

5. I valori iniziali e finali della variabile di controllo devono essere compatibili con il tipo della variabile di controllo. Se l'istruzione e' eseguita, i valori iniziali e finali devono anche essere compatibili per l'assegnazione con la variabile di controllo. Dapprima e' sempre valutato il valore iniziale, poi quello finale. Entrambi sono valutati soltanto una volta prima che l'istruzione sia eseguita.

L'istruzione seguente il DO non e' affatto eseguita se:

- il valore iniziale e' maggiore del valore finale nel caso TO.

- il valore iniziale e' minore del valore finale nel caso DOWNTO.

La sequenza di valori, data la variabile di controllo, parte con il valore iniziale. Tale sequenza e' definita con la funzione SUCC per il caso TO o con la funzione PRED per il caso DOWNTO fino all'ultima esecuzione dell'istruzione, quando la variabile di controllo assume il suo valore finale. Il valore della variabile di controllo, dopo che un'istruzione FOR termina naturalmente, non e' definito (anche se non e' eseguito il corpo). Esso puo' variare a causa dell'ottimizzazione e, se il metacomando $INITCK e' attivo, puo' essere posizionato ad un valore non inizializzato. Tuttavia il valore della variabile di controllo, dopo aver interrotto un'istruzione FOR con GOTO o BREAK, e' definito come il valore da essa raggiunto al momento dell'uscita dal ciclo.

Nel Pascal Standard il corpo di un'istruzione FOR puo' essere o no eseguito; quindi e' necessaria un controllo per vedere se esso viene eseguito. Tuttavia, se la variabile di controllo e' di tipo WORD (o subrange) e il suo valore iniziale e' una costante zero, il corpo deve essere eseguito qualunque sia il valore finale. In questo caso, non occorre eseguire alcun controllo supplementare e non e' generato alcun codice per eseguire un tale controllo. Inoltre una variabile di controllo con attributo STATIC puo' essere piu' efficiente di una senza tale attributo.

Al livello Esteso del Pascal si possono usare variabili di controllo temporanee:

```
FOR VAR <variabile di controllo>
```

Con il prefisso VAR la variabile di controllo e' dichiarata locale all'istruzione FOR (cioe', ad una visibilita' di piu' basso livello) e non occorre dichiararla in una sezione VAR. Tale variabile di controllo non e' disponibile al di fuori dell'istruzione FOR, e qualunque altra variabile con lo stesso identificatore non e' disponibile all'interno dell'istruzione FOR stessa. Altre variabili sinonime sono, comunque, disponibili alle procedure o funzioni richiamate all'interno dell'istruzione FOR.

Esempi di variabili di controllo temporanee:

```
FOR VAR I := 1 TO 100 DO
    SUM := SUM + VICTOR [1]

FOR VAR COUNTDOWN := 10 DOWNTO LIFT_OFF DO
    MONITOR_ROCKET
```

Le Istruzioni BREAK e CYCLE

In teoria, un programma che usa le istruzioni BREAK e CYCLE del Pascal di livello Esteso puo' non usare alcuna istruzione GOTO.

Ciascuna di queste due istruzioni ha due forme, una con una etichetta di ciclo e una senza. Un'etichetta di ciclo e' una normale etichetta GOTO come prefisso ad un'istruzione FOR, WHILE o REPEAT. Poiche', al livello Esteso, e' possibile usare etichette di identificatori, si consiglia di usare interi per le etichette indicate dalle istruzioni GOTO e dagli identificatori per le etichette di ciclo.

Esempi di istruzioni CYCLE e BREAK:

```
LABEL SEARCH, CLIMB;
  .
  .
SEARCH : WHILE 1 <= I_TOP DO
  IF PILE [I] = TARGET THEN BREAK SEARCH
  ELSE I := I + 1;
  .
  .
FOR I := 1 TO N DO
  IF NEXT [1] = NIL THEN BREAK;
  .
  .
CLIMB : WHILE NOT ITEM^.LEAF DO
  BEGIN
    IF ITEM^.LEFT <> NIL
       THEN [ITEM := ITEM^.LEFT; CYCLE CLIMB];
    IF ITEM^.RIGHT <> NIL
       THEN [ITEM := ITEM^.RIGHT; CYCLE CLIMB];
    WRITELN ('Very strange node');
    BREAK CLIMB
  END;
```

L'Istruzione WITH

L'istruzione WITH apre il dominio di un'istruzione fino ad includere i campi di uno o piu' record, in modo che ci si possa riferire direttamente ai campi. Per esempio le seguenti istruzioni sono equivalenti:

```
WITH PERSON DO WRITE (NAME, ADDRESS, PHONE)
WRITE (PERSON.NAME, PERSON.ADDRESS, PERSON.PHONE)
```

Il record dato puo' essere una variabile, un identificatore di costante, una costante strutturata o un identificatore di funzione; non puo' essere un componente di una struttura PACKED. Se si usa un identificatore di funzione, esso si riferisce alla variabile del risultato della funzione locale. Se il record dato in un'istruzione WITH e' una variabile di buffer di file, il compilatore emette un segnale di avvertimento, poiche' il cambiamento della posizione in un'istruzione WITH puo' causare un errore.

Il record dato puo' anche essere una qualunque espressione tra parentesi, nel qual caso l'espressione e' valutata e il risultato e' assegnato ad una variabile temporanea (nascosta). Se si vuole valutare un indicatore di funzione, lo si deve racchiudere tra parentesi.

Dopo il WITH si puo' dare una lista di record separati da virgole. Ciascun record della lista deve essere di tipo differente da tutti gli altri, poiche' gli identificatori di campo si riferiscono soltanto all'ultima istanza del record tramite il tipo. Le seguenti istruzioni sono equivalenti:

```
WITH PMODE, QMODE DO istruzione
WITH PMODE DO WITH QMODE DO istruzione
```

Ogni variabile record di un'istruzione WITH che e' componente di un'altra variabile, e' selezionata prima dell'esecuzione dell'istruzione. Le variabili attive WITH non devono essere passate come parametri VAR o VARS, e i loro puntatori non possono essere passati alla procedura DISPOSE. Comunque questi errori non sono rilevati dal compilatore. Sono consentite le assegnazioni a qualunque variabile record nella lista WITH o alle componenti di queste variabili, ma in questo caso il record WITH deve essere una variabile.

In Pascal ogni istruzione WITH alloca una variabile indirizzo che possiede l'indirizzo del record. Se la variabile record e' nello heap, il puntatore ad essa non deve essere "DISPOSE" entro l'istruzione WITH. Se la variabile record e' un buffer di file, non deve essere fatto alcun I/O nel file all'interno dell'istruzione WITH. Occorre evitare assegnazioni all'istruzione WITH nel record stesso nei programmi destinati ad essere portabili.

CONTROLLO SEQUENZIALE

Per aumentare la velocita' di esecuzione o assicurare una corretta valutazione e' spesso utile, nelle istruzioni IF, WHILE e REPEAT considerare le espressioni booleane come una serie di prove. Se una prova fallisce, le altre prove non sono eseguite. In Pascal sono previsti due operatori di livello Esteso per tali prove:

1. AND THEN

   X AND THEN Y e' falso se X e' falso; Y e' valutato solto se X e' vero.

2. OR ELSE

X OR ELSE Y e' vero se X e' vero; Y e' valutato solto se X e' falso.

Se si usano diversi operatori di controllo sequenziale, il compilatore li valuta esattamente da sinistra a destra.

Si possono solo includere questi operatori nell'espressione booleana di una clausola IF, WHILE o UNTIL; essi non possono essere usati in altre espressioni booleane. Inoltre non possono essere inclusi tra parentesi e sono valutati dopo tutti gli altri operatori.

Esempi di operatori di controllo sequenziale:

```
IF SYM <> NIL AND THEN SYM^.VAL < 0 THEN
   NEXT_SYMBOL

WHILE I <= MAX AND THEN VECT [I] <> KEY DO
   I := I + 1;

REPEAT GEN (VAL)
UNTIL VAL = 0 OR ELSE (QU DIV VAL) = 0;

WHILE POOR AND THEN GETTING_POORER
   OR ELSE BROKE AND THEN BANKRUPT DO
   GET_RICH
```

PARTE SECONDA

15. INTRODUZIONE A PROCEDURE E FUNZIONI

SOMMARIO

Questo capitolo introduce, in generale, le procedure e le funzioni e descrive il loro uso e la loro costruzione.

INDICE

INTRODUZIONE

Le procedure e le funzioni agiscono come sottoprogrammi che vengono eseguiti sotto il controllo di un programma principale. Tuttavia, a differenza dei programmi, le procedure e le funzioni possono essere nidificate tra loro e possono addirittura richiamare se stesse. Inoltre esse presentano una sofisticata capacita' di passare i parametri, che manca ai programmi. Le procedure vengono richiamate come istruzioni di programma; le funzioni possono essere richiamate da istruzioni di programma ogni volta che e' richiesto un valore.

Il formato generale per le procedure e funzioni e' simile a quello per i programmi. La struttura, formata da tre parti, comprende un'intestazione, una sezione dichiarativa e un corpo.

Esempio di dichiarazione di una procedura:

```
{Intestazione}
PROCEDURE MODEL (I:INTEGER; R:REAL);

{Inizio della sezione dichiarativa}
LABEL 123;
CONST ATOP = 199;
TYPE INDEX = 0..ATOP;
VAR ARAY : ARRAY [INDEX] OF REAL;  J : INDEX;

{Dichiarazione di funzione}
FUNCTION FONE (RX:REAL): REAL;
BEGIN
  FONE := RX * I
END;

{Dichiarazione di procedura}
PROCEDURE FOUT (RY:REAL);
BEGIN
  WRITE ('Output is ',RY)
END;

{Corpo della procedura MODEL}
BEGIN
  FOR J := 0 TO ATOP DO
    IF GLOBALVAR THEN
    {Attivazione della procedura FOUT con il}
    {valore ritornato dalla funzione FONE.}
       FOUT (FONE (R + ARAY [J]))
     ELSE GOTO 123;

  123: WRITELN ('Done');
END;
```

La parte dichiarativa ed il corpo, insieme, sono chiamati 'blocco'.
La dichiarazione di una procedura o funzione associa un identificatore ad una porzione di programma. Questa porzione di programma puo' essere, poi, attivata mediante un'appropriata istruzione della procedura o un indicatore della funzione.

PROCEDURE

L'esempio seguente illustra il formato generale della dichiarazione di procedura. L'intestazione e' seguita da:

1. dichiarazioni di etichette, costanti, tipi, variabili e valori

2. procedure e funzioni locali

3. il corpo della procedura, racchiuso tra le parole riservate BEGIN e END.

Quando il corpo della procedura termina l'esecuzione, il controllo ritorna all'elemento di programma che ha effettuato la chiamata.

A livello Standard, l'ordine delle dichiarazioni deve essere il seguente:

1. LABEL

2. CONST

3. TYPE

4. VAR

5. Procedure e funzioni

A livello Esteso si puo' avere un numero qualsiasi di LABEL, CONST, TYPE, VAR e sezioni VALUE, come pure di dichiarazioni di procedure e di funzioni, in ordine qualsiasi. Anche se le dichiarazioni di dati (CONST, TYPE, VAR, VALUE) possono essere mescolate con quelle di procedure e funzioni, risulta piu' chiaro, generalmente, porre per prime tutte le dichiarazioni di dati.

Comunque, il porre le dichiarazioni di variabili dopo le dichiarazioni di procedure e funzioni assicura che queste variabili non vengano usate dalle procedure o funzioni.

In generale, il valore iniziale di una variabile non e' definito. La sezione VALUE, che deve seguire la sezione VAR, e' un'estensione del Pascal che consente di inizializzare esplicitamente programmi, moduli, implementazioni, variabili STATIC e PUBLIC. Se l'indicatore di inizializzazione ($INITCK) e' attivo, tutte le variabili INTEGER, subrange di INTEGER, REAL e le variabili puntatore non vengono inizializzate. Le variabili di file vengono sempre inizializzate, indipendentemente dalla posizione del controllo di inizializzazione.

FUNZIONI

Le funzioni sono simili alle procedure, con la differenza che esse vengono richiamate in un'espressione al posto di un'istruzione e ritornano un valore.

La dichiarazione di funzione definisce le parti del programma che calcolano un valore. Le funzioni sono attivate quando viene incontrato, in un'espressione, un indicatore di funzione.

La dichiarazione di una funzione ha lo stesso formato della dichiarazione di procedura, con la differenza che l'intestazione contiene anche il tipo del valore ritornato dalla funzione.

Esempio di intestazione di funzione:

```
FUNCTION MAXIMUM (I, J : INTEGER) : INTEGER;
```

All'interno del blocco di una funzione, nel corpo stesso della funzione o in una procedura o funzione nidificata in esso, deve essere eseguita almeno un'assegnazione all'identificatore di funzione per poterne ritornare il valore. Se il controllo di inizializzazione non e' attivo e il tipo ritornato non e' INTEGER, REAL o puntatore, il compilatore non effettua controlli runtime su questa assegnazione. Comunque, se non esistono assegnazioni all'identificatore di funzione, il compilatore emette un messaggio di errore.

A livello Standard, le funzioni possono ritornare qualsiasi tipo semplice (ordinale, REAL o INTEGER4) oppure un puntatore. A livello Esteso, le funzioni possono ritornare qualunque tipo semplice, strutturato o di riferimento. Tuttavia non possono ritornare tipi che non possono essere assegnati (cioe', un tipo super array oppure una struttura che contiene un file, anche se e' permesso un tipo derivato super array).

Un identificatore di funzione in un'espressione richiama le funzione in modo ricorsivo invece di fornire il valore corrente della funzione.

Per ottenere il valore corrente, occorre usare la funzione RESULT la quale prende l'identificatore della funzione come parametro ed e' disponibile a livello Esteso.

L'esempio seguente illustra come usare la funzione RESULT per ottenere il valore corrente di una funzione all'interno di un'espressione:

```
FUNCTION FACT (F: REAL): REAL;
BEGIN
  FACT := 1;
  WHILE F > 1 DO
    BEGIN
      FACT := RESULT (FACT) * F;  F := F - 1
    END
END
```

L'uso della funzione RESULT e' piu' efficiente di quello di una singola variabile locale per il valore della funzione, e dell'assegnazione di

questa variabile locale all'identificatore di funzione. Se la funzione ha un valore strutturato, la funzione RESULT puo' essere seguita dalla normale sintassi di selezione dei componenti.

Un identificatore di funzione alla sinistra di un'assegnazione fa riferimento alla variablie locale della funzione che contiene il suo valore corrente invece di richiamare ricorsivamente la funzione stessa. Gli altri posti in cui, usando un identificatore di funzione, si fa riferimento a questa variabile locale sono:

- un parametro di riferimento

- il record di un'istruzione WITH

- l'operando di uno degli operatori ADR e ADS

Ognuno di questi casi prevede l'uso dell'indirizzo (non del valore) di una variabile.

Invece di usare la variabile locale della funzione, si puo' richiamare la funzione ed usare il suo valore di ritorno. Come descritto in "Cos'e' una Variabile ?", nel Capitolo 12, ottenere l'indirizzo di un'espressione significa valutare l'espressione, memorizzarne il risultato in una variabile temporanea (nascosta) ed usare l'indirizzo di questa variabile.

Per ottenere cio' con una funzione occorre forzare la valutazione racchiudendo il nome della funzione tra parentesi, nel modo seguente:

```
TYPE IREC= RECORD I: INTEGER END;

FUNCTION SUM (A, B: INTEGER): IREC;
                              {Ritorna la somma di A e B}
BEGIN
  IF TUESDAY THEN
  BEGIN
    IF B = 0 THEN BEGIN SUM := A; RETURN END;
    WITH (SUM (A, B-1))       {Richiama ricorsivamente SUM.}
      DO SUM.I := I + 1       {I e' il risultato della chiamata.}
  END
  ELSE
    WITH SUM                  {Usa la variabile locale della funzione.}
      DO I := A + B;          {I e' la variabile locale.}
END;
```

ATTRIBUTI E DIRETTIVE

Un attributo fornisce ulteriori informazioni su una procedura o su una funzione. Gli attributi sono disponibili anche al livello Esteso del Pascal. Essi vengono posti dopo l'intestazione, racchiusi tra parentesi quadre e separati da virgole. Gli attributi disponibili comprendono ORIGIN, PUBLIC, FORTRAN, PURE e INTERRUPT.

Una direttiva fornisce informazioni su una procedura o funzione, pero' indica anche che solo l'intestazione della procedura o funzione e' presente, sostituendo cosi' il blocco (parte dichiarativa e corpo) incluso normalmente dopo l'intestazione. Le direttive sono valide nel Pascal Standard, e le uniche disponibili sono EXTERN e FORWARD. EXTERN puo' essere usata solamente con procedure o funzioni nidificate direttamente in un programma, modulo, implementazione o interfaccia. Questa restrizione evita l'accesso a variabili non locali presenti nello stack.

La tabella 15-1 contiene gli attributi e le direttive per le procedure e le funzioni. Le sezioni seguenti descrivono in dettaglio questi attributi e direttive.

NOME	USO
FORWARD	Direttiva. Permette il richiamo di una procedura o di una funzione prima di fornire la sua definizione nel file sorgente.
EXTERN	Direttiva. Indica che una procedura o una funzione risiede in un altro modulo.
PUBLIC	Attributo. Indica che una procedura o una funzione puo' essere richiamata da altri moduli.
ORIGIN	Attributo. Dice al compilatore dove risiede il codice di una procedura o funzione EXTERN.
FORTRAN	Attributo. Specifica una sequenza di chiamata compatibile con il FORTRAN Microsoft (solo nel caso in cui e' disponibile l'MS-FORTRAN).
INTERRUPT	Attributo. Fornisce alle procedure una speciale sequenza di chiamata, che salva lo stato del programma nello stack.
PURE	Attributo. Indica che la funzione non puo' modificare variabili globali.

Tabella 15-1 Attributi e Direttive per Procedure e Funzioni

Le seguenti regole valgono quando si combinano attributi nella dichiarazione di procedure e funzioni:

- ad ogni funzione puo' essere associato l'attributo PURE

- le procedure e le funzioni con attributi devono essere nidificate direttamente all'interno di un programma, modulo o unita'; l'unica eccezione a questa regola e' costituita dall'attributo PURE

- in una procedura o funzione puo' essere specificato soltanto un attributo di sequenza di chiamata (cioe' FORTRAN o INTERRUPT, ma non entrambi)

- l'attributo PUBLIC e la direttiva EXTERN si escludono l'uno con l'altro, cosi' come gli attributi PUBLIC e ORIGIN

Le direttive EXTERN o FORWARD vengono automaticamente associate a tutti i costituenti dell'interfaccia di un'unita'; nell'implementazione, l'attributo PUBLIC e' automaticamente associato a tutti i costituenti che non sono EXTERN.

Dato che i costituenti di un'unita' vengono dichiarati soltanto nell'interfaccia (non nell'implementazione), l'interfaccia risiede dove sono presenti gli attributi. Si puo' fornire la direttiva EXTERN in un'implementazione dichiarando per prime tutte le funzioni e procedure EXTERN; l'attributo ORIGIN non puo' essere usato ne' nell'interfaccia ne' nell'implementazione di un'unita'.

In un modulo si puo' fornire un gruppo di attributi nell'intestazione; essi vengono applicati a tutte le funzioni e procedure nidificate direttamente. L'unica eccezione a questa regola e' costituita dall'attributo ORIGIN, che puo' essere associato soltanto ad una singola procedura o funzione.

Se l'attributo PUBLIC e' uno degli attributi specificati nell'intestazione di un modulo, un attributo EXTERN assegnato ad un procedura o funzione all'interno del modulo prevale sull'attributo globale PUBLIC. Se l'intestazione del modulo non ha attributi, l'attributo PUBLIC viene assunto per tutte le procedure e funzioni direttamente nidificate.

L'attributo PUBLIC permette il richiamo di una funzione o procedura da parte di altro codice presente in memoria e non puo' essere usato con la direttiva EXTERN. La direttiva EXTERN permette il richiamo di altro codice in memoria usando ORIGIN o il linker. PUBLIC, EXTERN ed ORIGIN forniscono un mezzo a basso livello per effettuare il concatenamento (link) di routine Pascal con altre routine Pascal o con routine scritte in altri linguaggi.

Una dichiarazione di una procedura o funzione con la direttiva EXTERN o FORWARD e' costituita solo dall'intestazione, senza il blocco. Le routine EXTERN hanno il blocco fuori del programma. Le routine FORWARD sono completamente dichiarate (cioe' hanno il loro blocco) in una zona successiva all'interno della parte compilativa. Entrambi queste direttive sono disponibili al livello Standard del Pascal. La parola chiave

EXTERNAL e' sinonimo di EXTERN.

L'attributo PURE puo' essere usato soltanto per le funzioni, non per le procedure. Al contrario, l'attributo INTERRUPT puo' esssere usato soltanto per le procedure, ma non per le funzioni. PURE e' l'unico attributo che puo' essere usato per funzioni nidificate.

LA DIRETTIVA FORWARD

La dichiarazione FORWARD permette di chiamare una procedura o una funzione prima della loro dichiarazione nel testo sorgente. Questo permette una ricorsione indiretta, cioe' A chiama B e B chiama A. Una dichiarazione FORWARD avviene facendo seguire l'intestazione della procedura o funzione specificata dalla direttiva FORWARD. In seguito si deve dichiarare effettivamente la procedura o funzione, senza ripetere la lista dei parametri formali, oppure gli attributi , oppure il tipo del valore di ritorno di una funzione.

Esempio di dichiarazione FORWARD:

```
{Dichiarazione di ALPHA, con lista di parametri ed attributi}
FUNCTION ALPHA (Q, R: REAL): REAL [PUBLIC];
FORWARD;

{Chiamata di ALPHA}
PROCEDURE BETA (VAR S, T: REAL);
BEGIN
  T := ALPHA (S, 3.14)
END;

{Dichiarazione di ALPHA, senza lista di parametri}
FUNCTION ALPHA;
BEGIN
  ALPHA := (Q + R);
  IF R < 0.0 THEN BETA (3.14, ALPHA);
END;
```

LA DIRETTIVA EXTERN

La direttiva EXTERN identifica una procedura o una funzione che risiede in un altro modulo. Occorre solo fornire l'intestazione della procedura o della funzione seguita dalla parola EXTERN. Si presuppone che l'implementazione della procedura o della funzione sia presente in un altro modulo.

EXTERN e' un attributo se usato con una variabile, mentre costituisce una direttiva quando riferito ad una procedura o funzione. Come per le variabili, EXTERNAL e' sinonimo di EXTERN.

La direttiva EXTERN, associata ad una procedura o funzione all'interno di un modulo, annulla l'attributo PUBLIC del modulo stesso. La direttiva EXTERN e' pure ammessa, nell'implementazione di un'unita', per un costituente di una procedura o funzione. Tutti questi costituenti esterni

devono essere dichiarati all'inizio dell'implementazione, prima delle altre procedure e funzioni.

Ogni procedura o funzione avente direttiva EXTERN deve essere direttamente nidificata in un programma. Si possono concatenare routine Pascal concatenando unita' compilate separatamente (a tal proposito si puo' vedere il Capitolo 18, "Unita' Compilabili di un Programma").

Esempi di intestazioni di procedure e funzioni dichiarate con la direttiva EXTERN:

```
FUNCTION POWER (X, Y: REAL): REAL; EXTERN;

PROCEDURE ACCESS (KEY: KTYP) [ORIGIN SYSB+4]; EXTERN;
```

In questo esempio la funzione POWER e' dichiarata EXTERN, come pure la procedura ACCESS. Entrambe sono implementate in unita' compilative esterne. La procedura ACCESS ha inoltre associato l'attributo ORIGIN, che viene descritto oltre in "L'Attributo ORIGIN". Non e' possibile dichiarare una procedura o funzione come EXTERN se e' gia' stata dichiarata precedentemente come FORWARD.

L'ATTRIBUTO PUBLIC

L'attributo PUBLIC indica che ad una procedura o funzione si puo' accedere da altri moduli, dichiarandola EXTERN nei moduli che la chiamano. Quindi e' sufficiente dichiarare una procedura PUBLIC e definirla in un modulo, e poi usarla in un altro modulo semplicemente dichiarandola EXTERN.

In modo analogo alle variabili, l'identificatore della procedura o funzione viene passato al concatenatore (linker) e puo' essere troncato, se il concatenatore lo richiede. Per informazioni specifiche sulle restrizioni imposte dal compilatore e dal linker, riguardo gli identificatori, si puo' consultare l'Appendice A, "Caratteristiche d'Implementazione" nel manuale "Linguaggio Pascal Guida Utente". PUBLIC e ORIGIN si escludono l'uno con l'altro; le routine PUBLIC richiedono un blocco e le routine ORIGIN devono essere EXTERN.

Qalsiasi procedura o funzione con attributo PUBLIC deve essere direttamente nidificata in un programma o in un'implementazione. Un modo ad alto livello per effettuare il concatenamento di routine Pascal e' quello di concatenare separatamente le unita' compilate. Per ulteriori dettagli si puo' vedere il Capitolo 18, "Unita' Compilabili di un Programma".

Esempi di procedure e funzioni dichiarate PUBLIC:

```
FUNCTION POWER (X, Y: REAL) : REAL [PUBLIC];
{La funzione POWER e' disponibile per altri moduli}
{dato che e' stata dichiarata PUBLIC}
BEGIN
  .
  .
END;

PROCEDURE ACCESS (KEY : KTYP) [ORIGIN SYSB+4, PUBLIC];
BEGIN
  .
  .
END;      {Illegale, poiche' ORIGIN deve essere pure EXTERN}
```

L'ATTRIBUTO ORIGIN

L'attributo ORIGIN deve essere usato con la direttiva EXTERN; ORIGIN dice al compilatore dove la procedura o funzione puo' essere direttamente trovata; in questo modo il linker non richiede il corrispondente identificatore PUBLIC.
Esempi di procedure e funzioni dati con attributo ORIGIN:

```
PROCEDURE OPSYS [ORIGIN 8, FORTRAN];
EXTERN;

FUNCTION A_TO_D (C: SINT): SINT [ORIGIN #100];
EXTERN;
```

Nel primo esempio, la procedura OPSYS inizia dall'indirizzo decimale assoluto 8, ha la sequenza di chiamata FORTRAN (descritta in seguito, in "L'Attributo FORTRAN") ed e' dichiarata EXTERN. Nel secondo esempio, la funzione A_TO_D accetta il valore SINT come parametro (SINT e' un valore subrange intero predichiarato che varia da -127 a +127). La funzione si trova all'indirizzo esadecimale 100.

I modo analogo alle variabili ORIGIN, il compilatore usa l'indirizzo per trovare il codice e non fornisce direttive al linker. Questo permette, ad esempio, la chiamata di routine che si trovano in ROM ad indirizzi gia' determinati. In casi semplici puo' essere usato per sostituire un caricatore di linker.

Occorre ricordare che ORIGIN implica sempre EXTERN. Quindi, procedure o funzioni dichiarate precedentemente FORWARD non possono essere dichiarate con l'attributo ORIGIN. Non e' neppure consentito applicare l'attributo ORIGIN dopo l'intestazione di un modulo.
Generalmente non e' consentito usare ORIGIN con un costituente di un'unita', ne' in un'interfaccia ne' in un'implementazione.

In modo analogo alle variabili, l'origine puo' essere un indirizzo segmentato. Su macchine segmentate, un'origine procedurale non segmentata assume il segmento di codice corrente con lo spiazzamento fornito dall'attributo; questa forma non ha usi evidenti.

L'ATTRIBUTO FORTRAN

Se sulla specifica macchina e' disponibile l'MS-FORTRAN, allora si puo' usare l'attributo FORTRAN. Questo attributo puo' essere applicato sia a procedure che a funzioni (ma non a variabili). Invece dell'usuale sequenza di chiamate Pascal, viene generata una sequenza di chiamate compatibile con il compilatore FORTRAN. Questo permette la chiamata di procedure o funzioni Pascal da parte di programmi FORTRAN e, viceversa, l'attivazione di routine FORTRAN da parte di un programma Pascal.

Esempio di procedura con l'attributo FORTRAN:

```
PROCEDURE DELTA (I, J: INTEGER) [FORTRAN];
FORWARD;
```

Ogni procedura o funzione con l'attributo FORTRAN deve essere nidificata direttamente in un programma o in un'implementazione.

In un ambiente a 16 bit, il Pascal usa la stessa sequenza di chiamata del compilatore per il FORTRAN Microsoft, BASIC Microsoft e COBOL Microsoft. Quindi non e' necessario specificare l'attributo FORTRAN; questo attributo, quando specificato, viene ignorato dal compilatore Pascal.

L'ATTRIBUTO INTERRUPT

L'attributo INTERRUPT si applica solo a procedure, non a funzioni o variabili. Esso permette la generazione di una speciale sequenza di chiamata che memorizza lo stato del programma nello stack; che a sua volta permette di gestire un'interruzione hardware, ripristinare lo stato del programma, restituire il controllo al programma, senza pero' modificare lo stato attuale del programma.

Esempio di procedura con l'attributo INTERRUPT:

```
PROCEDURE INCHAR [INTERRUPT];
```

Poiche' le procedure con attributo INTERRUPT vengono richiamate per la gestione delle interruzioni hardware, esse non devono essere richiamate da un'istruzione di procedura. Una procedura INTERRUPT puo' essere soltanto richiamata quando viene emessa l'interruzione a cui e' associata. Inoltre, le procedure con INTERRUPT non hanno parametri.

La dichiarazione di una procedura con l'attributo INTERRUPT assicura la conformita' della procedura con la gestione delle interruzioni, cioe':

- una speciale sequenza di chiamata memorizza completamente lo stato del programma nello stack
- lo stato del programma comprende i registri e gli indicatori di macchina, piu' tutti i dati globali del compilatore come il puntatore dell'elemento
- lo stato del programma viene ripristinato all'uscita della procedura

Tutte le procedure INTERRUPT devono essere direttamente nidificate in un'unita' compilativa.

Le interruzioni non vengono rinviate automaticamente alle procedure INTERRUPT; inoltre, per quanto possibile sulla macchina oggetto, le interruzioni non vengono ne' abilitate ne' disabilitate da una procedura INTERRUPT. L'invio e l'abilitazione di interruzioni sono operazioni troppo dipendenti dalla macchina per essere incluse in un linguaggio, indipendente dalla macchina, come il Pascal.

Tuttavia il Pascal fornisce la procedura di libreria VECTIN, la quale accetta come parametri il livello ed una procedura di interruzione; questa procedura agisce sul vettore di interruzione in modo dipendente dalla macchina.

In modo analogo, le procedure di libreria ENABIN e DISBIN abilitano e disabilitano, ripettivamente, le interruzioni in modo dipendente dalla macchina. Vedere il Capitolo 16, "Procedure e Funzioni Disponibili", per ulteriori informazioni su queste routine, e l'Appendice A, "Caratteristiche d'Implementazione" nel manuale "Linguaggio Pascal Guida Utente", per informazioni sull'implementazione di queste routine.

Una procedura INTERRUPT effettua un ritorno normale alla routine interrotta per poterne continuare l'elaborazione. Cio' significa che:

- non si deve eseguire un'istruzione GOTO che permette di uscire dalla procedura INTERRUPT

- tutto il controllo di debug deve essere tralasciato ($DEBUG-, $ENTRY-, e $RUNTIME-)

- l'overflow dello stack puo' non essere controllato anche se $STACKCK e' attivo

L'uso di procedure INTERRUPT permette, in Pascal, la rientranza del codice: il codice generato e' rientrante come il sistema runtime (eccetto l'unita' heap e, nella maggior parte dei sistemi operativi, le porzioni di unita' di file).

Alcune sezioni critiche nel sistema runtime sono protette da semafori che generano un errore runtime, se queste sezioni sono gia' in uso e quindi bloccate. Ad esempio, se il gestore dello heap e' in esecuzione quando viene riconosciuta un'interruzione, e la procedura INTERRUPT cerca di allocare un blocco dallo heap, allora la struttura dello heap potrebbe diventare non valida. Questa condizione causa l'emissione di un errore runtime.

Comunque, nella maggior parte dei casi, il file system non e' protetto da un semaforo. E' quindi consigliabile non effettuare operazioni di I/O all'interno di una procedura INTERRUPT. Oppure si possono eliminare molti problemi di I/O in una procedura INTERRUPT evitando di aprire o chiudere file (cioe' non dichiarando variabili locali di file o creando file nello heap) e non eseguendo input/output su file che possono essere in procinto di I/O quando viene emessa l'interruzione.

L'ATTRIBUTO PURE

L'attributo PURE puo' essere applicato soltanto alle funzioni, non alle procedure o variabili. PURE indica all'ottimizzatore del compilatore che la funzione non modifica variabili globali ne' direttamente ne' chiamando altre procedure o funzioni.

Esempio di dichiarazione PURE:

```
FUNCTION AVERAGE (CONST TABLE: RVECTOR) : REAL [PURE];
```

Per ulteriori dettagli si possono esaminare le seguenti istruzioni:

```
A := VEC [I * 10 + 7];
B := FOO;
C := VEC [I * 10 + 9];
```

Se la funzione FOO ha l'attributo PURE, l'ottimizzatore genera codice per calcolare I*10 una volta sola. Tuttavia la funzione FOO, nel caso che non sia stata dichiarata PURE, puo' modificare I. In questo caso I*10 deve essere ricalcolato dopo la chiamata a FOO.

Le funzioni vengono considerate PURE solo se viene specificato esplicitamente l'attributo. Il compilatore controlla che l'attributo PURE non presenti alcuna di queste condizioni:

- assegnazioni a variabili non locali
- parametri VAR o VARS (i parametri CONST e CONSTS sono permessi)
- chiamate a funzioni che non hanno attributo PURE.

Anche se le seguenti restrizioni non vengono controllate dal compilatore, una funzione PURE non deve:

- usare il valore di una variabile globale
- modificare i riferimenti passati per valore (ad esempio, riferimenti di puntatore o di tipo indirizzo)
- effettuare input o output

Poiche' il risultato di una funzione PURE con gli stessi parametri deve essere sempre lo stesso, si puo' ottimizzare la chiamata di funzione.

Ad esempio, se nelle seguenti istruzioni DSIN e' PURE, il compilatore effettua una sola chiamata a DSIN:

```
HX := A * DSIN (P[I, J] * 2);
HY := B * DSIN (P[I, J] * 2);
```

PARAMETRI DI PROCEDURE E FUNZIONI

Le procedure e le funzioni ammettono tre diversi tipi di parametri:

1. parametri valore

2. parametri di riferimento

3. parametri procedurali e funzionali

Ognuno di questi tipi di parametri viene descritto separatamente, nell'ordine indicato, nei paragrafi che seguono.

La descrizione riguarda sia i parametri formali che quelli attuali. Un parametro formale e' il parametro fornito quando una procedura o funzione viene dichiarata, con un identificatore, nella relativa intestazione. Quando una funzione o una procedura viene chiamata, il parametro attuale sostituisce quello formale; in questo caso il parametro assume la forma di una variabile, di un valore o di un'espressione.

Il Pascal ha diverse funzionalita' di parametri a livello Esteso:

- un tipo super array puo' essere passato come parametro di riferimento

- un parametro di riferimento puo' essere dichiarato READONLY

- possono essere dichiarati espliciti parametri di referimento segmentati.

PARAMETRI VALORE

Quando viene passato un parametro per valore, il parametro attuale e' un'espressione. Questa espressione viene valutata nell'ambiente di chiamata di una procedura o funzione ed il suo valore viene assegnato al parametro formale. Il parametro formale e' una variabile locale alla procedura o alla funzione chiamata.

Quindi i parametri valore formali sono sempre locali ad una procedura o funzionè.

Esempio di parametri valore:

```
{Dichiarazione di funzione}
FUNCTION ADD (A, B, C : REAL) : REAL;
{A, B e C sono parametri formali}
   .
   .
X := ADD (Y, ADD (1.111, 2.222, 3.333), (Z * 4))
```

In questa particolare chiamata di funzione, Y, ADD(...) e (Z * 4) sono espressioni che rappresentano i parametri attuali. In questo esempio, i valori forniti dalle espressioni devono essere di tipo REAL. (L'esempio inoltre effettua una chiamata ricorsiva alla funzione ADD).

Il valore dell'espressione (parametro attuale) deve essere compatibile con il tipo del parametro formale.

Il passaggio per valore di tipi strutturati e' consentito, tuttavia non e' efficiente, poiche' l'intera struttura deve essere ricopiata. Un parametro valore di tipo SET, LSTRING o subrange puo' anche richiedere un controllo di errore runtime, se il controllo di variabilita' e' attivo. Inoltre, i parametri valore di tipo SET e LSTRING possono richiedere un codice addizionale generato per adattamento di dimensioni.

Una variabile file o super array non puo' essere passata come parametro valore, dato che non puo' essere assegnata. Comunque puo' essere passata una variabile di tipo derivato da super array o di buffer di file. Passare una variabile di buffer di file come parametro valore implica una valutazione della variabile stessa.

PARAMETRI DI RIFERIMENTO

Al livello Standard del Pascal, quando viene passato un parametro di riferimento, il parametro formale e' preceduto dalla parola chiave VAR. Inoltre, il parametro attuale deve essere una variabile, non un'espressione. Il parametro formale indica la variabile attuale durante l'esecuzione della procedura.

Tutte le operazioni effettuate sul parametro formale sono eseguite immediatamente sul parametro attuale, passando l'indirizzo macchina della variabile attuale alla procedura. Per processori con supporto segmentato, questo indirizzo e' uno spiazzamento all'interno del segmento dati di default.

Esempio di parametri variabili:

```
PROCEDURE CHANGE_VARS (VAR A, B, C : INTEGER);
{A, B e C sono parametri formali di riferimento.}
{Essi indicano variabili, non valori.}
   .
   .
CHANGE_VARS (X, Y, Z);
```

In questo esempio X, Y e Z devono essere variabili, non espressioni. Inoltre le variabili X, Y e Z vengono alterate ogni volta che vengono alterati i parametri formali A, B e C nella procedura dichiarata. Questo differisce dalla gestione dei parametri valore, in quanto puo' effettuare solo delle copie di variabili. Se la selezione di una variabile prevede il calcolo di un indice di un array oppure la valutazione di un puntatore o di un indirizzo, queste operazioni vengono eseguite prima della procedura stessa. Il tipo del parametro attuale deve coincidere con il tipo del parametro formale.

Il passaggio di una variabile non locale come parametro VAR e' indicato dal carattere '/' o dal segno percento '%' nella colonna G (globale) del file listing (vedere "Formato del File Listing", nel Capitolo 19, per informazioni sul significato di questi caratteri nella colonna G).

I seguenti elementi non possono essere passati come parametri VAR:

- un componente di una struttura PACKED (eccetto CHAR di una STRING o LSTRING)

- qualsiasi variabile con attributo READONLY o PORT (inclusi i parametri CONST e CONSTS e la variabile di controllo FOR).

Il passaggio di una variabile di buffer di file come parametro di riferimento genera un messaggio di avvertimento, poiche' non viene effettuata la chiamata normale a file system come accade in genere usando una variabile buffer. Queste chiamate non sono generate quando una variabile file e' passata per riferimento.

Su una macchina segmentata, un parametero VAR passa un indirizzo che e' in realta' uno spiazzamento all'interno del segmento dati di default. In alcuni casi occorre accedere ad oggetti che risiedono in altri segmenti. Per passare il riferimento a questi oggetti si deve usare un indirizzo segmentato contenente sia il registro di segmento che il valore dello spiazzamento. Il livello Esteso prevede il prefisso VARS al posto di VAR:

```
PROCEDURE CONCATS (VARS T, S: STRING);
```

VARS puo' essere usato come parametro dati in procedure e funzioni, non nella sezione dichiarativa di programmi, procedure e funzioni. I parametri VARS e CONSTS vengono forniti principalmente per mantenere la compatibilita' con macchine che hanno due diversi spazi di indirizzi. Questi parametri non sono necessari per macchine con un singolo spazio indirizzi. Su queste macchine le parole chiave VARS e CONSTS sono equivalenti a VAR e CONST.

PARAMETRI SUPER ARRAY

I parametri super array possono comparire come parametri formali di riferimento. Questo permette ad una procedura o funzione di operare su un array con un particolare tipo super array (od anche tipo componente o tipo indice), senza un estremo superiore prefissato. Il parametro formale e' un parametro di riferimento dello stesso tipo super array.

Il tipo del parametro attuale deve essere un tipo derivato da quello super array oppure di tipo super array stesso (cioe' un altro parametro di riferimento oppure un puntatore a cui e' stato tolto il riferimento). Eccetto per il confronto di LSTRING, i parametri di tipo super array non possono essere assegnati o confrontati direttamente.

I valori attuali degli estremi superiori e inferiori sono disponibili mediante le funzioni UPPER and LOWER; questo permette alle routine di operare su array di qualsiasi dimensione. Un parametro attuale LSTRING puo' essere passato ad un parametro di riferimento di tipo super array STRING. Quindi il parametro super array STRING puo' essere usato per procedure e funzioni che operano su stringhe di entrambi i tipi STRING e LSTRING.

Esempio di parametri super array:

```
TYPE REALS = ARRAY [0..*] OF REAL;
PROCEDURE SUMRS (VARS X: REALS; CONST X: REALS);
BEGIN
  .
  .
END;
```

Per ulteriori informazioni, si possono vedere, nel Capitolo 9 le seguenti sezioni: "Super Array", "Tipi STRING", "Tipi LSTRING".

Parametri Costante e Segmento

A livello Esteso, un parametro formale preceduto dalla parola riservata CONST implica che il parametro attuale e' un parametro di riferimento READONLY. Questo e' utile specialmente per parametri di tipo strutturato, che possono essere costanti, dato che viene eliminata la necessita' di copiare un parametro valore, per evitare un consumo di tempo. Il parametro attuale puo' essere una variabile, il risultato di una funzione oppure un valore costante.

Non puo' essere fatta alcuna assegnazione a parametri CONST e neanche a qualche loro componente. Sono permessi parametri CONST di tipo super array. Un parametro CONST in una procedura non puo' essere passato come parametro VAR in un'altra procedura. E' comunque permesso passare un parametro VAR in una procedura come parametro CONST in un'altra procedura.

Esempio di parametro CONST:

```
PROCEDURE ERROR (CONST ERRMSG : STRING);
```

Su una macchina segmentata, un parametro CONST viene passato come un indirizzo che in realta' e' uno spiazzamento all'interno del segmento dati corrente. In alcuni casi occorre accedere ad oggetti che risiedono in altri segmenti. Per passare il riferimento a questi oggetti occorre far generare al compilatore un indirizzo segmentato contenente sia il registro di segmento che il valore dello spiazzamento. Il livello Esteso prevede il prefisso CONSTS al posto di CONST. L'uso di CONSTS e' analogo all'uso di VARS per parametri formali di riferimento.

Esempio di parametro CONSTS:

```
PROCEDURE CAT (VARS T: STRING; CONSTS S: STRING);
```

Un parametro CONSTS puo' essere usato solamente come parametro dati in procedure e funzioni, non nella sezione dichiarativa di programmi, procedure e funzioni.

E' possibile passare il valore di un'espressione come parametro CONST o CONSTS. L'espressione viene valutata ed il valore e' assegnato ad una variabile temporanea (nascosta) nell'ambiente della funzione o procedura di chiamata. L'espressione deve essere racchiusa in parentesi allo scopo

di forzarne la valutazione.

Un identificatore di funzione puo' essere passato come riferimento ad un parametro VAR, VARS, CONST o CONSTS. In questo caso viene passata la variabile locale della funzione, e occorre quindi che la chiamata venga effettuata nel corpo della funzione stessa oppure in una procedura o funzione dichiarata con la funzione.

Il valore ritornato da un indicatore di funzione puo' essere passato, come qualsiasi espressione, ad un parametro CONST o CONSTS. In modo analogo alle espressioni passate per riferimento, l'indicatore di funzione deve essere racchiuso in parentesi, come segue:

```
PROCEDURE WRITE_ANSWER (CONSTS A: INTEGER);
BEGIN
  WRITELN ('THE ANSWER IS ,' A)
END;

FUNCTION ANSWER : INTEGER;
BEGIN
  ANSWER := 42;
  WRITE_ANSWER (ANSWER);
  {Viene passato il riferimento alla variabile locale.}
END;

PROCEDURE HITCH_HIKE;
BEGIN
  WRITE_ANSWER ((ANSWER))
  {Richiamare ANSWER, assegnarla alla variabile temporanea,}
  {passare il riferimento alla variabile temporanea}
END;
```

PARAMETRI PROCEDURALI E FUNZIONALI

I parametri procedurali possono essere usati nei modi seguenti:

- in analisi numerica
- chiamando alcune routine di libreria
- in applicazioni speciali.

In analisi numerica, puo' essere utile passare una funzione ad una procedura o ad un'altra funzione che calcola un integrale tra due limiti oppure un valore minimo o massimo, e cosi' via. Alcuni interessanti algoritmi fanno uso di parametri procedurali in campi come il riconoscimento di testi e l'intelligenza artificiale.

Quando viene passato un parametro funzionale o procedurale, l'identificatore attuale e' il nome di una procedura o di una funzione. Il parametro formale e' costituito dall'intestazione di una procedura o funzione, compresi tutti gli attributi, preceduta dalla parola riservata PROCEDURE o FUNCTION.

Si esamini, ad esempio, queste dichiarazioni:

```
TYPE DOOR  = (FRONT, BARN, CELL, DOG_HOUSE);
     SPEED = (FAST, SLOW, NORMAL);
     DIRECTION = (OPEN, SHUT);

PROCEDURE OPEN_DOOR_WIDE
  (VAR A: DOOR; B: SPEED; C: DIRECTION);
     .
     .
PROCEDURE SLAM_DOOR
  (VAR DR: DOOR; SP: SPEED; DIR: DIRECTION);
     .
     .
PROCEDURE LEAVE_AJAR
  (VAR DD: DOOR; SS: SPEED; DD: DIRECTION);
```

Tutte le procedure nell'esempio hanno un numero uguale di parametri. I tipi dei parametri sono non solo compatibili, ma anche uguali. I parametri formali non devono avere necessariamente lo stesso nome.

Un parametro procedurale o funzionale puo' accettare una di queste procedure, se la procedura o la funzione sono state impostate correttamente, cioe':

```
FUNCTION DOOR_STATUS (PROCEDURE MOVE_DOOR
  (VAR X: DOOR; Y: SPEED; Z: DIRECTION);
  VAR XX: DOOR; YY: SPEED; ZZ: DIRECTION) : INTEGER;
  {"PROCEDURE MOVE_DOOR" e' il parametro procedurale formale;}
  {le due linee seguenti sono gli altri parametri formali.}

BEGIN {door_status}
  DOOR_STATUS := 0;
  MOVE_DOOR (XX, YY, ZZ);
  {Viene eseguita una delle tre procedure dichiarate}
  {precedentemente.}

  IF XX = BARN AND ZZ = SHUT
  THEN DOOR_STATUS := 1;

  IF XX = CELL AND ZZ = OPEN
  THEN DOOR_STATUS := 2;

  IF XX = DOG_HOUSE AND ZZ = SHUT
  THEN DOOR_STATUS := 3
END;
```

L'uso del parametro procedurale MOVE_DOOR puo' essere il seguente:

```
IF DOOR_STATUS
   (SLAM_DOOR, CELL, FAST, SHUT) = 0
THEN SOCIETY := SAFE;

IF DOOR_STATUS
   (OPEN_DOOR_WIDE, BARN, SLOW, OPEN) = 0
THEN COWS_ARE_OUT := TRUE;

IF DOOR_STATUS
    (LEAVE_AJAR, DOG_HOUSE, SLOW, OPEN) = 0
THEN DOG_CAN_GET_IN := TRUE;
```

In ognuno dei casi sopra descritti, la lista dei parametri attuali e' compatibile con quella formale, sia nel numero che nel tipo dei parametri. Se il parametro passato e' di tipo funzionale, allora anche il valore di ritorno della funzione deve essere congruente nel tipo.

L'insieme degli attributi definiti per i parametri procedurali formali deve coincidere con quello definito per quelli attuali corrispondenti, tranne gli attributi PUBLIC e ORIGIN e la direttiva EXTERN che vengono ignorati.
Una procedura PUBLIC o EXTERN, o qualsiasi procedura locale a qualsiasi livello di nidificazione, puo' essere passata ad un parametro formale dello stesso tipo.
Devono comunque coincidere l'eventuale attributo PURE e tutti gli altri attributi presenti nella sequenza di chiamata. Inoltre in sistemi che hanno un codice con indirizzi segmentati, le procedure o le funzioni passate come parametri ad una procedura o funzione EXTERN, devono a loro volta essere PUBLIC o EXTERN.

In Pascal non e' ammesso il passaggio di procedure predichiarate e funzioni compilate come codice in linea; esse possono essere passate soltanto da subroutine richiamate. Le famiglie READ, WRITE, ENCODE e DECODE hanno codice generato che varia a seconda dei tipi degli argomenti; esse non possono quindi essere passate come parametri funzionali. Le corrispondenti routine usate su file o per encode/decode possono invece essere passate. Ad esempio, READ di INTEGER diventa una chiamata RTIFQQ e questa procedura puo' essere passata come parametro.

Le seguenti procedure e funzioni intrinseche non possono essere passate come parametri ad una funzione o procedura:

1. Livello Standard di Pascal:

ABS	EOLN	PACK	SQR
ARCTAN	EXP	PAGE	SQRT
CHR	LN	PRED	SUCC
COS	NEW	READ	UNPACK
DISPOSE	ODD	READLN	WRITE
EOF	ORD	SIN	WRITELN

2. Livello Esteso e livello Sistema di Pascal:

BYLONG	FLOAT4	READFN	SIZEOF
BYWORD	HIBYTE	READSET	TRUNC
DECODE	HIWORD	RESULT	TRUNC4
ENCODE	LOBYTE	RETYPE	UPPER
EVAL	LOWER	ROUND	WRD
FLOAT	LOWORD	ROUND4	

Quando viene attivata una procedura o funzione passata come parametro, le variabili non locali fanno riferimento a quelle presenti quando la procedura o la funzione e' stata passata come parametro, e non a quelle presenti quando essa viene attivata. Internamente, sia l'indirizzo della routine che l'indirizzo dell'ambiente chiamante vengono memorizzati nello stack.

Esempio di uso di procedura formale:

```
PROCEDURE ALPHA;
  VAR I : INTEGER;

  PROCEDURE DELTA;
    BEGIN
      WRITELN ('Delta done')
    END;

  PROCEDURE BETA (PROCEDURE XPR);
    VAR GLOB : INTEGER;

    PROCEDURE GAMMA;
      BEGIN GLOB := GLOB + 1 END;

  BEGIN   {Inizio di BETA}
    GLOB := 0;
    IF I = 0
      THEN BEGIN
           I := 1; XPR; BETA (GAMMA)
           END
      ELSE BEGIN
           GLOB := GLOB + 1; XPR
           END
  END;

BEGIN     {Inizio di ALPHA}
  I := 0;
  BETA (DELTA)
END;
```

Nell'esempio appena descritto avvengono le seguenti cose:

1. viene chiamata ALPHA

2. viene chiamata BETA, passando la procedura DELTA

3. quest'ultima chiamata crea un'istanza di GLOB nello stack (GLOB1)

4. innanzitutto BETA cancella GLOB1 posizionandola a zero. Poi, dato che I vale 0, viene eseguita la clausola THEN che posiziona I a uno ed esegue XPR, che e' legata a DELTA

5. viene quindi emesso in output il messaggio 'Delta done'

6. ora BETA viene richiamata in modo ricorsivo. BETA viene passata a GAMMA ed effettua in questo caso un accesso alle variabili non locali usate da GAMMA (cioe' GLOB1)

7. la seconda chiamata a BETA crea una nuova istanza di GLOB (GLOB2). Quando GLOB2 viene azzerata, I vale 1, e quindi GLOB2 viene incrementata di uno

8. viene quindi chiamata XPR, che e' legata a GAMMA, cosi' GAMMA e' eseguita e incrementa l'istanza attiva di GLOB (GLOB1) quando GAMMA viene passata a BETA

9. GAMMA ritorna, cosi' pure la seconda chiamata di BETA; poi torna la prima chiamata di BETA ed infine ALPHA effettua il suo ritorno.

16. PROCEDURE E FUNZIONI DISPONIBILI

SOMMARIO

Questo capitolo descrive otto categorie di procedure e funzioni, disponibili perche' predichiarate o perche' parti della libreria runtime del Pascal. Tutte, tranne quelle che si riferiscono al file di input e di output, sono descritte in dettaglio.

INDICE

INTRODUZIONE

Tutte le versioni di Pascal includono un grande numero di procedure e funzioni comuni, che non devono essere dichiarate nei programmi. Dato che esse sono dichiarate in un ambiente "esterno" al programma, si possono ridichiarare i loro identificatori.

Per favorire la portabilita' dei programmi, il Pascal rende disponibili alcune di queste procedure e funzioni predichiarate soltanto a livello Esteso, inoltre fornisce alcune funzioni e procedure di libreria, che devono pero' essere definite EXTERN per poterle usare.

Il Pascal gestisce tre tipi di funzioni e procedure:

1. alcune sono predichiarate, e il compilatore traduce le loro chiamate in altre chiamate oppure in codice speciale (esse non possono essere passate come parametri)

2. alcune sono predichiarate ma vengono chiamate normalmente (eccetto se il loro nome viene ridefinito)

3. alcune non sono predichiarate ma sono disponibili come parte della libreria runtime del Pascal (devono essere dichiarate esplicitamente).

E' comunque piu' vantaggioso catalogare queste procedure rispetto a quello che fanno, piuttosto che riferirsi alla loro implementazione. La tabella 16-1 contiene questa suddivisione:

CATEGORIA	SCOPO
File system	Operare su file che hanno modi di accesso e strutture diversi.
Allocazione dinamica	Allocare e deallocare dinamicamente strutture dati nello heap durante il runtime.
Conversione dati	Convertire dati da un tipo ad un altro.
Aritmetica	Eseguire funzioni aritmetiche.
Intrinseche di livello Esteso	Fornire procedure e funzioni addizionali al livello Esteso del Pascal.
Intrinseche di livello Sistema	Fornire procedure e funzioni addizionali al livello di Sistema del Pascal.
Intrinseche di stringhe	Operare sui dati di tipo STRING e LSTRING.
Libreria	Disponibilita' nella libreria runtime di Pascal: esse non sono predichiarate e devono quindi essere dichiarate con la direttiva EXTERN.

Tabella 16-1 Categorie delle Procedure e Funzioni Disponibili

CATEGORIE DELLE PROCEDURE E FUNZIONI DISPONIBILI

Il resto di questo capitolo e' suddiviso in due sezioni. La prima sezione descrive ognuna delle categorie contenute nella tabella precedente e contiene la lista delle procedure e funzioni appartenenti a ciascuna categoria. La seconda sezione contiene la lista in ordine alfabetico di tutte le procedure e funzioni disponibili. Per ogni procedura e funzione vengono forniti la sintassi, la descrizione, gli esempi e le annotazioni.

PROCEDURE E FUNZIONI DI FILE SYSTEM

Il file system del Pascal ammette un certo numero di procedure e funzioni che permettono di operare su file con modi di accesso e strutture diversi. Queste procedure e funzioni appartengono a tre categorie, come illustrato nella tabella 16-2.

CATEGORIA	PROCEDURE	FUNZIONI
Primitive	GET PAGE PUT RESET REWRITE	EOF EOLN
I/O di File-testo	READ READLN WRITE WRITELN	
I/O di livello Esteso	ASSIGN CLOSE DISCARD READSET READFN SEEK	

Tabella 16-2 Procedure e Funzioni di File System

Per ulteriori dettagli su ognuna di tali procedure e funzioni si puo' vedere il Capitolo 17, "Procedure e Funzioni Orientate su File".

PROCEDURE DI ALLOCAZIONE DINAMICA

Due procedure, NEW e DISPOSE, permettono l'allocazione e la deallocazione dinamica di strutture dati durante il runtime. La procedura NEW alloca una variabile nello heap, la DISPOSE ne effettua la deallocazione.

PROCEDURE E FUNZIONI DI CONVERSIONE DATI

Le seguenti procedure permettono di convertire i dati da un tipo ad un altro:

CHR	PACK	TRUNC
FLOAT	PRED	TRUNC4
FLOAT4	ROUND	UNPACK
ODD	ROUND4	WRD
ORD	SUCC	

Quattro di queste procedure convertono qualsiasi tipo ordinale in un particolare tipo ordinale:

1. CHR (ordinale) a CHAR

2. ODD (ordinale) a BOOLEAN

3. ORD (ordinale) a INTEGER

4. WRD (ordinale) a WORD

Anche PRED e SUCC operano su tipi ordinali.

Sei routine di conversione effettuano la conversione tra INTEGER o INTEGER4 e REAL:

1. FLOAT converte INTEGER a REAL

2. FLOAT4 converte INTEGER4 a REAL

3. ROUND converte REAL a INTEGER

4. ROUND4 converte REAL a INTEGER4

5. TRUNC converte REAL a INTEGER

6. TRUNC4 converte REAL a INTEGER4

PACK e UNPACK trasferiscono componenti tra array packed e unpacked.

FUNZIONI ARITMETICHE

Tutte le funzioni aritmetiche ammettono un parametro CONSTS di tipo REAL4 o REAL8, oppure un tipo compatibile con INTEGER (detto "numerico" nella lista). ABS e SQR ammettono anche valori di tipo WORD e INTEGER4.

Tutte le funzioni che operano su dati REAL effettuano un controllo di validita' sul dato (dato non inizializzato). Vengono pure controllate particolari condizioni di errore con relativa emissione di messaggio di errore runtime.

Se e' attivo l'indicatore di controllo matematico, l'uso errato delle funzioni ABS e SQR su dati di tipo INTEGER, WORD e INTEGER4 genera un messaggio di errore runtime; se l'indicatore non e' attivo, allora il risultato di un'operazione errata e' indefinito.

La tabella 16-3 contiene tutte le funzioni aritmetiche disponibili e tutte le routine chiamate, a seconda se viene richiesta la singola o la doppia precisione.

NOME	OPERAZIONE	REAL4	REAL8
ABS	Valore assoluto	(codice in linea)	(codice in linea)
ARCTAN	Arco tangente	ATSRQQ	ATDRQQ
COS	Coseno	CNSRQQ	CNDRQQ
EXP	Esponenziale	EXSRQQ	EXDRQQ
LN	Logaritmo naturale	LNSRQQ	LNDRQQ
SIN	Seno	SNSRQQ	SNDRQQ
SQR	Quadrato	(codice in linea)	(codice in linea)
SQRT	Radice quadrata	SRSRQQ	SRDRQQ

Tabella 16-3 Funzioni Aritmetiche Predichiarate

La libreria di runtime MS-FORTRAN contiene molte funzioni addizionali REAL4 e REAL8, come illustrato nella tabella 16-4. Il loro uso prevede che vengano dichiarate con la direttiva EXTERN.

OPERAZIONE	REAL4	REAL8
Arco coseno	ACSRQQ	ACDRQQ
Troncamento	AISRQQ	AIDRQQ
Arrotondamento	ANSRQQ	ANDRQQ
Arco seno	ASSRQQ	ASDRQQ
Arco tangente A/B	A2SRQQ	A2DRQQ
Coseno iperbolico	CHSRQQ	CHDRQQ
Logaritmo decimale	LDSRQQ	LDDRQQ
Modulo	MDSRQQ	MDDRQQ
Minimo	MNSRQQ	MNDRQQ
Massimo	MXSRQQ	MXDRQQ
Potenza (REAL8**INTG4)		PIDRQQ
Potenza (REAL4**INTG4)	PISRQQ	
Potenza (REAL ** REAL)	PRSRQQ	PRDRQQ
Seno iperbolico	SHSRQQ	SHDRQQ
Tangente iperbolica	THSRQQ	THDRQQ
Tangente	TNSRQQ	TNDRQQ

Tabella 16-4 Funzioni REAL della Libreria Runtime MS-FORTRAN

Alcune comuni funzioni matematiche non sono standard nel Pascal, ma sono relativamente semplici da eseguire con istruzioni di programma oppure da definire come funzioni in un programma. Seguono alcuni esempi:

```
SIGN (X)        e'    ORD (X > 0) - ORD (X < 0)
POWER (X, Y)    e'    EXP (Y * LN (X))
```

E' possibile anche scrivere le proprie funzioni in Pascal per ottenere le medesime cose. A funzioni di questo genere e' opportuno associare

I'attributo PURE (per ottenere un codice piu' efficiente).

Ad esempio:

```
FUNCTION POWER (A, B: REAL) : REAL [PURE];
BEGIN
  IF A <= 0 THEN
     ABORT ('Nonplus real to power', 24, 0);
     POWER := EXP (B * LN (A));
END;
```

FUNZIONI E PROCEDURE INTRINSECHE DI LIVELLO ESTESO

Le seguenti funzioni e procedure sono disponibili al livello Esteso di Pascal:

ABORT	EVAL	LOWORD
BYLONG	HIBYTE	RESULT
BYWORD	HIWORD	SIZEOF
DECODE	LOBYTE	UPPER
ENCODE	LOWER	

Molte di queste routine vengono usate per comporre e scomporre elementi di uno, due e quattro byte, cioe' : HIBYTE, LOBYTE, BYWORD, HIWORD, LOWORD, e BYLONG.

ENCODE e DECODE convertono variabili da formato interno a formato stringa e viceversa. ABORT genera un errore runtime.

Le altre routine, EVAL, LOWER, UPPER, RESULT e SIZEOF, vengono usate in situazioni particolari (descritte per ogni funzione nella sezione "Elenco delle Funzioni e Procedure" in questo capitolo).

FUNZIONI E PROCEDURE INTRINSECHE DI LIVELLO DI SISTEMA

Alcune procedure e funzioni addizionali sono disponibili al livello di Sistema del Pascal:

FILLC	MOVESL
FILLSC	MOVESR
MOVEL	RETYPE
MOVER	

Le procedure MOVE e FILL effettuano operazioni a basso livello su stringhe di byte. RETYPE cambia il tipo di un'espressione in modo arbitrario.

ROUTINE INTRINSECHE DI STRINGHE

Le prestazioni intrinseche di stringhe sono fornite da un insieme di procedure e funzioni, alcune delle quali operano sui tipi STRING e LSTRING, ed altre solamente su STRING:

NOME	PARAMETRO
CONCAT	STRING
DELETE	STRING
INSERT	STRING
COPYLST	STRING
COPYSTR	STRING o LSTRING
POSITN	STRING o LSTRING
SCANEQ	STRING o LSTRING
SCANNE	STRING o LSTRING

Tabella 16-5 Procedure e Funzioni Stringa

PROCEDURE E FUNZIONI DI LIBRERIA

Le seguenti routine non sono predichiarate, ma sono disponibili nella libreria runtime del Pascal. Esse devono essere dichiarate, con la direttiva EXTERN, prima di essere usate in un programma.

1. Routine di inizializzazione e terminazione.

   Le routine BEGOQQ e ENDOQQ vengono richiamate rispettivamente durante le fasi di inizializzazione e terminazione. Esse possono essere usate per richiamare il debugger o per emettere, su video, messaggi per l'utente (come la data di esecuzione del programmma). BEGXQQ puo' essere usata per far ripartire un programma e ENDXQQ per farlo terminare.

2. Routine di gestione dello heap.

   Le routine di gestione dello heap fanno da complemento a quelle standard NEW e DISPOSE. Esse includono:

   ALLHQQ
   FREECT
   MARKAS
   MEMAVL
   RELEAS

3. Routine di interruzione.

   Queste routine gestiscono le interruzioni, il loro modo di operare varia da macchina a macchina; esse sono:

   ENABlN
   DlSABlN
   VECTlN

4. Routine di 1/O da terminale.

   Le seguenti routine gestiscono in modo diretto l'input e l'output da terminale:

   GTYUQQ
   PTYUQQ
   PLYUQQ

5. Routine di semaforo.

   Le due procedure LOCKED e UNLOCK permettono l'uso di un semaforo binario. Esse possono essere usate per assicurare l'accesso esclusivo ad una risorsa comune.

6. Funzioni aritmetiche senza overflow.

   Queste funzioni implementano un'aritmetica su 16 e 32 bit. Viene ritornato l'eventuale overflow o riporto senza chiamare le routine di errore.

   LADDOK
   LMULOK
   SADDOK
   SMULOK
   UADDOK
   UMULOK

7. Routine di clock.

   Queste routine gestiscono le informazioni sul clock di sistema:

   TlME
   DATE
   TICS

ELENCO DELLE FUNZIONI E PROCEDURE

Questa sezione contiene un elenco di tutte le funzioni e procedure disponibili, sia quelle predichiarate che quelle di libreria da dichiarare EXTERN. Ogni routine dell'elenco comprende l'intestazione, la categoria cui appartiene l'operazione da effettuare e la descrizione di cosa fa la funzione. Vengono forniti note ed esempi appropriati. Le intestazioni date sono le stesse per gli operandi REAL4 o REAL8, a meno che siano specificate diverse indicazioni.

PROCEDURE ABORT (CONST STRING, WORD, WORD);

E' una procedura intrinseca di livello Esteso. Essa termina l'esecuzione di un programma allo stesso modo di un errore interno runtime. STRING (o LSTRING) e' un messaggio di errore. Il parametro STRING e' di tipo CONST, non CONSTS. La prima WORD contiene un codice di errore (vedi l'Appendice H, "Messaggi di Errore", per i codici di errore); la seconda WORD puo' contenere qualsiasi valore. Essa viene qualche volta usata per ritornare un codice di errore di file da parte del sistema operativo.

I parametri, le informazioni sullo stato della macchina (contatore di programma, puntatore di ambiente, puntatore di stack), e la posizione sorgente della chiamata ABORT (se sono attivi gli indicatori $LINE e/o $ENTRY) vengono resi disponibili in un messaggio di terminazione oppure passati al package di debug.

Se l'indicatore $RUNTIME e' attivo, viene emesso nel messaggio di errore anche l'indirizzo della procedura o funzione nella quale ABORT e' stata richiamata. Se $RUNTIME e' attivo, gli indicatori $LINE e $ENTRY non devono essere presenti e le routine nel file sorgente devono richiamare solamente altre routine $RUNTIME.

FUNCTION ABS (X : NUMERIC) : NUMERIC;

E' una funzione aritmetica. Ritorna il valore assoluto di X. Sia X che il valore di ritorno hanno lo stesso tipo numerico: REAL4, REAL8, INTEGER, WORD o INTEGER4. Dato che i valori WORD non hanno segno, ABS(X) ritorna sempre X se X e' di tipo WORD.

FUNCTION ACSRQQ (CONSTS A : REAL4) : REAL4;
FUNCTION ACDRQQ (CONSTS A : REAL8) : REAL8;

Sono funzioni aritmetiche. Ritornano l'arcocoseno di A. Sia A che il valore di ritorno sono dello stesso tipo : REAL4 o REAL8. Queste funzioni appartengono alla libreria runtime dell'MS-FORTRAN e devono essere dichiarate EXTERN prima dell'uso, se l'MS-FORTRAN e' disponibile sulla macchina specifica.

FUNCTION AISRQQ (CONSTS A : REAL4) : REAL4;
FUNCTION AIDRQQ (CONSTS A : REAL8) : REAL8;

Sono funzioni aritmetiche. Ritornano la parte intera di A, senza arrotondamento. Sia A che il valore di ritorno sono di tipo REAL4 o REAL8. Queste funzioni appartengono alla libreria runtime dell'MS-

FORTRAN e devono essere dichiarate EXTERN prima dell'uso, se l'MS-FORTRAN e' disponibile sulla macchina specifica.

FUNCTION ALLHQQ (SIZE : WORD) : WORD;

E' una routine di libreria che gestisce lo heap. Ritorna il valore zero se lo heap e' pieno, il valore uno se c'e' un errore nella struttura dello heap, oppure MAXWORD se il processo di allocazione e' stato interrotto. Altrimenti, essa ritorna il valore del puntatore alla variabile allocata avente la dimensione (SIZE) richiesta.

Generalmente ALLHQQ viene usata con la funzione RETYPE. Ad esempio:

```
P_VAR := RETYPE (P_TYPE, ALLHQQ (28));
{RETYPE converte il valore ritornato da ALLHQQ (28)}
{nel tipo P_TYPE. Questo valore e' assegnato a P_VAR.}

IF WRD (P_VAR) < 2 THEN GO_ABORT;
{P_VAR e' controllata in caso di heap pieno}
{o di errore di struttura dello heap.}
```

FUNCTION ANSRQQ (CONSTS A : REAL4) : REAL4;
FUNCTION ANDRQQ (CONSTS A : REAL8) : REAL8;

Sono funzioni aritmetiche. Analogamente alle funzioni AISRQQ e AIDRQQ, esse ritornano la parte intera di A, pero' con arrotondamento per eccesso. Sia A che il valore di ritorno sono di tipo REAL4 o REAL8. Queste funzioni appartengono alla libreria runtime dell'MS-FORTRAN e devono essere dichiarate EXTERN nel programma che ne fa uso, se l'MS-FORTRAN e' disponibile sulla macchina specifica.

FUNCTION ARCTAN (X : REAL) : REAL;

E' una funzione aritmetica. Ritorna l'arcotangente di X in radianti. Sia X che il valore di ritorno sono di tipo REAL. Per forzare una precisione particolare, occorre invece usare ATSRQQ (CONSTS REAL4) e/o ATDRQQ (CONSTS REAL8).

FUNCTION ASSRQQ (CONSTS A : REAL4) : REAL4;
FUNCTION ASDRQQ (CONSTS A : REAL8) : REAL8;

Sono funzioni aritmetiche. Ritornano il valore dell'arcoseno di A. Sia A che il valore di ritorno sono di tipo REAL4 o REAL8. Queste funzioni appartengono runtime dell'MS-FORTRAN e devono essere dichiarate EXTERN nel programma che ne fa uso, se l'MS-FORTRAN e' disponibile sulla macchina specifica.

PROCEDURE ASSIGN (VAR F; CONSTS N : STRING);

E' una procedura di file system (I/O di livello Esteso). Assegna il nome contenuto in una STRING (o LSTRING) al file F.

Vedere "Procedure di Livello Esteso", nel Capitolo 17, per una descrizione di ASSIGN.

FUNCTION ATSRQQ (CONSTS A : REAL4) : REAL4;
FUNCTION ATDRQQ (CONSTS A : REAL8) : REAL8;

Vedere FUNCTION ARCTAN.

FUNCTION A2SRQQ (A, B : REAL4) : REAL4;
FUNCTION A2DRQQ (A, B : REAL8) : REAL8;

Sono funzioni aritmetiche. Ritornano l'arcotangente di (A/B). Sia A e B che il valore di ritorno sono di tipo REAL4 o REAL8. Queste funzioni appartengono alla libreria runtime dell'MS-FORTRAN e devono essere dichiarate EXTERN dal programma che ne fa uso, se l'MS-FORTRAN e' disponibile sulla macchina specifica.

PROCEDURE BEGOQQ;

E' una routine di libreria (inizializzazione). BEGOQQ viene chiamata in fase di inizializzazione, ed e' usata in alternativa alla versione di default. Comunque e' possibile scrivere la propria versione di BEGOQQ: cio' puo' essere utile per richiamare un debugger oppure per emettere su video messaggi utente (come ad esempio la data di esecuzione).

Vedere anche PROCEDURE ENDOQQ.

PROCEDURE BEGXQQ;

E' una routine di libreria (inizializzazione). Dopo la fase di concatenamento e di caricamento del programma, BEGXQQ e' il punto d'entrata del programmma. Analogamente alla routine generica di inizializzazione, BEGXQQ effettua le seguenti operazioni:

1. inizializza lo stack e lo heap

2. inizializza il file system

3. chiama BEGOQQ

4. chiama il corpo del programma.

BEGXQQ puo' essere usata per ripartire, dopo qualche errore catastrofico in un sistema ROM. Richiamando questa procedura non si tiene conto pero' di chiudere i file aperti in precedenza. Non vengono reinizializzate le variabili presenti, in origine, nella sezione VALUE oppure quelle con l'indicatore di inizializzazione.

FUNCTION BYLONG (INTEGER-WORD, INTEGER-WORD) : INTEGER4;

E' una funzione intrinseca di livello Esteso. Converte WORD o INTEGER (oppure LOWORD di INTEGER4) in un valore INTEGER4. BYLONG concatena i suoi operandi:

BYLONG (A, B) = ORD (LOWORD (A)) * 65535 + WRD (HIWORD (B))

Se il primo valore e' di tipo WORD, il bit piu' significativo diventa il segno del risultato.

FUNCTION BYWORD (ONE-BYTE, ONE-BYTE) : WORD;

E' una funzione intrinseca di livello Esteso. Converte byte (oppure LOBYTE di INTEGER o WORD) in un valore WORD. Accetta due parametri di qualsiasi tipo ordinale. BYWORD ritorna un valore WORD con il primo byte nella posizione piu' significativa e con il secondo byte nella posizione meno significativa:

```
BYWORD (A, B) = LOBYTE (A) * 256 + LOBYTE (B)
```

Se il primo valore e' di tipo WORD, il bit piu' significativo diventa il segno del risultato.

FUNCTION CHR (X : ORDINAL) : CHAR;

E' una funzione di conversione dati. Converte qualsiasi tipo ordinale in CHR. Il risultato in codice ASCII corrisponde a ORD (X). Questa e' un'estensione dello standard ISO, il quale richiede che X sia di tipo INTEGER. Viene ritornato un errore se ORD(X) > 255 oppure ORD(X) < 0. Questo errore viene rilevato soltanto quando il controllo di variabilita' e' attivo.

FUNCTION CHSRQQ (CONSTS A : REAL4) : REAL4;
FUNCTION CHDRQQ (CONSTS A : REAL8) : REAL8;

Sono funzioni aritmetiche. Ritornano il coseno iperbolico di A. Sia A che il valore di ritorno sono di tipo REAL4 o REAL8. Queste funzioni appartengono alla libreria runtime dell'MS-FORTRAN e devono essere dichiarate EXTERN dal programma che ne fa uso, se l'MS-FORTRAN e' disponibile sulla macchina specifica.

PROCEDURE CLOSE (VAR F);

E' una procedura di file system (I/O di livello Esteso). Provvede alla chiusura di un file, assicurando la corretta terminazione dell'accesso al file.
Vedere "Procedure di Livello Esteso", nel Capitolo 17, per una descrizione di CLOSE.

FUNCTION CNSRQQ (CONSTS A : REAL4) : REAL4;
FUNCTION CNDRQQ (CONSTS A : REAL8) : REAL8;

Vedere FUNCTION COS.

PROCEDURE CONCAT (VARS D : LSTRING; CONSTS S : STRING);

E' una procedura intrinseca su stringhe. Concatena S alla fine di D. La lunghezza di D viene incrementata della lunghezza di S. Viene emesso un errore quando la lunghezza di D e' troppo piccola, cioe' se

```
UPPER (D) < D.LEN + UPPER (S)
```

PROCEDURE COPYLST (CONSTS S : STRING; VARS D : LSTRING);

E' una procedura intrinseca su stringhe. Copia S su D di tipo LSTRING.

La lunghezza di D diventa UPPER (S). Viene emesso un errore se la lunghezza di S e' maggiore della massima lunghezza di D, cioe' se

UPPER (S) > UPPER (D)

PROCEDURE COPYSTR (CONSTS S : STRING; VARS D : STRING);

E' una procedura intrinseca su stringhe. Copia S su D di tipo STRING. Nel caso che UPPER (S) sia minore di UPPER (D), il resto di D viene riempito con spazi vuoti. Viene emesso un errore se la lunghezza di S e' maggiore della massima lunghezza di D, cioe' se

UPPER (S) > UPPER (D)

FUNCTION COS (X : NUMERIC) : REAL;

E' una funzione aritmetica. Ritorna il coseno di X in radianti. Sia X che il valore di ritorno sono di tipo REAL. Per precisioni particolari diverse da REAL occorre usare CNSRQQ (CONSTS REAL4) e/o CNDRQQ (CONSTS REAL8).

PROCEDURE DATE (VAR S : STRING);

E' una procedura di clock. Questa procedura, quando disponibile, memorizza la data di sistema nella variabile S. Nel caso she S sia un tipo LSTRING, occorre impostare la lunghezza desiderata prima di chiamare la procedura. Il formato dipende dal sistema operativo usato.

FUNCTION DECODE (CONSTS LSTR: LSTRING, X:M:N) : BOOLEAN;

E' una funzione intrinseca del livello Esteso. Converte la stringa di caratteri contenuta in LSTRING nella sua rappresentazione interna, e la assegna a X. Se la stringa non contiene codice valido ASCII compatibile con X, DECODE ritorna un valore FALSE ed il valore di X rimane indefinito.

DECODE opera esattamente come la procedura READ, inclusi i parametri M e N (vedere "Formati READ", nel Capitolo 17, per una loro descrizione). Quando X e' un subrange, DECODE ritorna il valore FALSE se il valore di X e' fuori del campo di variabilita' (indipendentemente dalla presenza o meno dell'indicatore di controllo di variabilita'). Nel tipo LSTRING vengono ignorati gli spazi iniziali e di trascinamento. Tutti gli altri caratteri in LSTRING devono far parte della rappresentazione.

X deve essere di tipo INTEGER, WORD, enumerato, uno dei loro subrange, BOOLEAN, REAL4, REAL8, INTEGER4, oppure un puntatore (i tipi indirizzo devono avere il suffisso .R o .S).

In un ambiente a memoria segmentata, il parametro LSTR deve risiedere nel segmento dati di default.
Vedere anche la FUNCTION ENCODE.

PROCEDURE DELETE (VARS D : LSTRING; I, N : INTEGER);

E' una procedura intrinseca di stringa. Cancella N caratteri da D

iniziando con D[1]. Viene emesso un segnale di errore quando si tenta di cancellare piu' caratteri di quanti possibile, cioe' quando:

```
D.LEN < (1 + N - 1)
```

PROCEDURE DISBIN;

E' una routine di libreria (interruzione). Insieme alle procedure ENABIN e VECTIN, DISBIN provvede alla gestione delle interruzioni. DISBIN disabilita le interruzioni; ENABIN le abilita; VECTIN fornisce un vettore di interruzione. Gli effetti di queste procedure variano a seconda della macchina. Si puo' vedere l'Appendice A, "Caratteristiche di Implementazione", nel manuale "Linguaggio Pascal Guida Utente", per avere informazioni sull'implementazione desiderata.

PROCEDURE DISCARD (VAR F);

E' una procedura di file system (I/O di livello Esteso). Chiude e cancella un file aperto.
Si puo' vedere "Procedure di Livello Esteso", nel Capitolo 17, per una descrizione di DISCARD.

PROCEDURE DISPOSE (VARS P : POINTER);

E' una procedura di allocazione dinamica (forma contratta). Libera la memoria usata per contenere la variabile indicata da P. P deve essere un puntatore valido; esso non deve essere NIL o non inizializzato, e non deve indicare un elemento dello heap che e' stato gia' reso disponibile. Questi casi vengono controllati se l'indicatore di controllo NIL e' presente.

P non deve essere un parametro di riferimento o un puntatore di un'istruzione WITH di record, pero' questi errori non vengono segnalati. La DISPOSE di un'istruzione WITH di record puo' essere fatta senza problemi alla fine dell'istruzione WITH stessa.

Se la variabile e' di tipo super array, oppure un record con varianti, si puo' usare senza problemi la forma contratta di DISPOSE per rilasciare la variabile, senza tener conto se questa e' stata allocata usando la forma lunga o contratta di NEW. L'uso della forma contratta di DISPOSE su una variabile dello heap allocata con la forma lunga di NEW e' un errore ISO che il Pascal non rileva.

PROCEDURE DISPOSE (VARS P : POINTER; T1, T2,...TN : TAGS);

E' una procedura di allocazione dinamica (forma lunga). La forma lunga di questa procedura e' esattamente come la forma contratta. La forma lunga controlla pero' che la dimensione della variabile sia congruente con quella del campo etichettato oppure con i valori dei limiti superiori di array (T1, T2, ..., TN). Questi valori etichettati devono essere uguali a quelli definiti nella corrispondente NEW.
Si puo' vedere anche la funzione SIZEOF, la quale fa uso degli stessi valori dei limiti superiori per ritornare il numero di byte di una variabile.

PROCEDURE ENABIN;

E' una routine di libreria (gestione delle interruzioni). Insieme a DISBIN e VECTIN, questa procedura gestisce le interruzioni. ENABIN abilita le interruzioni. Gli effetti di queste procedure possono variare a seconda della macchina. Per avere ulteriori dettagli, si puo' vedere l'Appendice A, "Caratteristiche di Implementazione", nel manuale "Linguaggio Pascal Guida Utente".

FUNCTION ENCODE (VAR LSTR : LSTRING, X:M:N:) : BOOLEAN;

E' una funzione intrinseca del livello Esteso. Converte l'espresione X nella sua rappresentazione ASCII esterna e memorizza questa rappresentazione in LSTR. Essa ritorna il valore TRUE, a meno che LSTRING sia troppo piccola per contenere la stringa generata. In questo caso ENCODE ritorna FALSE ed il valore di LSTR rimane indefinito. ENCODE funziona esattamente come la procedura WRITE, inclusi i parametri M e N (per una loro descrizione si puo' vedere "Formati WRITE", nel Capitolo 17).

X deve essere di uno di questi tipi: INTEGER, WORD, enumerato, oppure uno dei loro subrange, BOOLEAN, REAL4, REAL8, INTEGER4, oppure un puntatore (il tipo indirizzo ha il suffisso .R o .S).

In un ambiente di memoria segmentata, il parametro LSTR deve risiedere nel segmento dati di default.
Vedere anche la FUNCTION DECODE.

PROCEDURE ENDOQQ;

E' una procedura di libreria (terminazione). ENDOQQ viene chiamata in fase di terminazione, e la corrispondente procedura di default non viene richiamata. E' comunque possibile scrivere la propria versione di ENDOQQ allo scopo di richiamare un debugger oppure per emettere su video messaggi utente (come ad esempio la data di esecuzione).

Dato che ENDOQQ viene chiamata dopo la gestione degli errori, se ENDOQQ richiama un errore, il risultato e' un ciclo infinito di terminazione.
Vedere anche PROCEDURE BEGOQQ.

PROCEDURE ENDXQQ;

E' una procedura di terminazione. ENDXQQ e' la procedura generica di terminazione ed esegue le seguenti azioni:

1. richiama ENDOQQ

2. termina il file system (chiudendo tutti i file aperti)

3. ritorna al sistema operativo (oppure a qualunque componente abbia chiamato BEGXQQ).

ENDXQQ puo' essere utile per terminare l'esecuzione di un programma quando si e' all'interno di una procedura o funzione, senza richiamare ABORT. ENDXQQ corrisponde alla procedura HALT presente in altri

compilatori Pascal.

FUNCTION EOF : BOOLEAN;
FUNCTION EOF (VAR F) : BOOLEAN;

E' una funzione di file system. Indica se la posizione corrente del file e' alla fine del file F per i modi di accesso SEQUENTIAL e TERMINAL. EOF senza parametri corrisponde a EOF (INPUT).

Si puo' vedere "EOF e EOLN", nel Capitolo 17, per una descrizione piu' completa di EOF.

FUNCTION EOLN : BOOLEAN;
FUNCTION EOLN (VAR F) : BOOLEAN;

E' una funzione di file system. Indica se la posizione corrente del file e' alla fine di una linea nel textfile F. EOLN senza parametri corrisponde a EOLN (INPUT).

Si puo' vedere "EOF e EOLN", nel Capitolo 17, per una descrizione piu' completa di EOLN.

PROCEDURE EVAL (EXPRESSION, EXPRESSION,...);

E' una procedura intrinseca del livello Esteso. Valuta soltanto parametri di espressioni, ma accetta qualsiasi numero di parametri di qualsiasi tipo. EVAL viene usata per valutare un'espressione come un'istruzione; esso e' comunemente usata per valutare una funzione soltanto nei suoi effetti collaterali, senza far uso del suo valore di ritorno.

FUNCTION EXSRQQ (CONSTS A : REAL4) : REAL4;
FUNCTION EXDRQQ (CONSTS A : REAL8) : REAL8;

Vedere FUNCTION EXP.

FUNCTION EXP (X : NUMERIC) : REAL;

E' una funzione aritmetica. Ritorna il valore esponenziale di X (cioe' e elevato ad X). Sia X che il valore di ritorno sono di tipo REAL. In caso di precisione particolare occorre invece usare le routine EXSRQQ (CONSTS REAL4) e/o EXDRQQ (CONSTS REAL8).

PROCEDURE FILLC (D : ADRMEM; N : WORD; C : CHAR);

E' una procedura intrinseca di livello di Sistema. Riempie D con N copie del carattere C. Non viene effettuato alcun controllo sui limiti di D. Vedere anche PROCEDURE FILLSC, per tipi ad indirizzo segmentato. Le procedure MOVE e FILL accettano parametri di tipo ADRMEM e ADSMEM; dato che tutti i tipi ADR (o ADS) sono compatibili, possono essere passate variabili o costanti di tipo ADR (o ADS). Questo modo di procedere e' pericoloso, ma talvolta utile.

PROCEDURE FILLSC (D : ADSMEM; N : WORD; C : CHAR);

E' una procedura intrinseca di livello di Sistema. Riempie D con N copie del carattere C. Non viene effettuato alcun controllo sui limiti di D.

Per i tipi ad indirizzo relativo si puo' vedere anche PROCEDURE FILLC. Le procedure MOVE e FILL accettano parametri di tipo ADRMEM e ADSMEM; dato che tutti i tipi ADR (o ADS) sono compatibili, possono essere passate variabili o costanti di tipo ADR (o ADS). Questo modo di procedere e' pericoloso, ma talvolta utile.

FUNCTION FLOAT (X : INTEGER) : REAL;

E' una funzione di conversione dati. Converte un valore INTEGER in un valore REAL. Normalmente questa funzione non viene usata, poiche' la conversione INTEGER a REAL viene effettuata anche automaticamente. Tuttavia poiche' FLOAT e' usata dal package runtime, essa e' inclusa nel livello Standard.

FUNCTION FLOAT4 (X : INTEGER4) : REAL;

E' una funzione di conversione dati. Converte un valore INTEGER4 in un valore REAL. Questa conversione viene effettuata anche automaticamente; e' possibile comunque perdere precisione (questo non e' un errore).

FUNCTION FREECT (SIZE : WORD) : WORD;

E' una funzione di libreria. Restituisce una valutazione del numero di volte in cui NEW puo' essere chiamata per allocare nello heap variabili lunghe SIZE byte. FREECT tiene conto di blocchi resi disponibili e di blocchi liberi adiacenti; essa viene usata generalmente in congiunzione con la funzione SIZEOF. Tuttavia questa funzione non considera che nello stack qualsiasi spazio puo' essere necessario; e dato che generalmente lo e', allora il valore ritornato deve essere adeguatamente ridotto.
Esempio:

```
IF FREECT (SIZEOF (REC, TRUE, 5)) > 2
  THEN DO_SOMETHING
```

PROCEDURE GET (VAR F);

E' una procedura di file system. GET riporta la componente corrente di F nella variabile di buffer F^ e fa avanzare il puntatore di file, oppure posiziona lo stato della variabile di buffer al valore vuoto.
Si puo' vedere "GET e PUT", nel Capitolo 17, per una descrizione di GET.

FUNCTION GTYUQQ (LEN : WORD; LOC : ADSMEM) : WORD;

E' una funzione di libreria (I/O da terminale). Legge un massimo di LEN caratteri dalla tastiera di terminale e li memorizza a partire dall'indirizzo LOC. Il valore di ritorno contiene il numero di caratteri effettivamente letti. GTYUQQ legge sempre la linea di caratteri impostata. I caratteri che superano la lunghezza del buffer vengono persi.

Esempio:

```
LSTR.LEN := GTYUQQ (UPPER (LSTR), ADS LSTR (1));
```

In congiunzione a PTYUQQ e PLYUQQ, GTYUQQ e' utile per effettuare I/O da terminale in un ambiente ad alte prestazioni.

FUNCTION HIBYTE (INTEGER-WORD) : BYTE;

E' una funzione intrinseca del livello Esteso. Ritorna il byte piu' significativo di un INTEGER o WORD. A seconda della macchina, il byte piu' significativo puo' essere il primo o il secondo byte indirizzato della word.
Vedere anche FUNCTION LOBYTE.

FUNCTION HIWORD (INTEGER4) : WORD;

E' una funzione intrinseca di livello Esteso. Ritorna la word piu' significativa dei quattro byte di INTEGER4. Il bit del segno di INTEGER4 diventa il bit piu' significativo di WORD.
Vedere anche FUNCTION LOWORD.

PROCEDURE INSERT (CONSTS S:STRING; VARS D:LSTRING; I:INTEGER);

E' una procedura intrinseca di stringa. Inserisce S a partire da D[1]. Viene emesso un errore se D e' troppo piccola, cioe' se:

```
UPPER(D) < UPPER (S) + D.LEN + 1
```

oppure se:

```
D.LEN < 1
```

FUNCTION LADDOK (A, B: INTEGER4; VAR C: INTEGER4) : BOOLEAN;

E' una routine di libreria (senza overflow aritmetico). Posiziona C uguale ad A sommato B. Insieme alla funzione LMULOK effettua l'aritmetica su 32 bit con segno senza causare un errore runtime, anche se l'indicatore di debugging e' attivo.
Sia LADDOK che LMULOK ritornano il valore TRUE se non c'e' stato overflow, e FALSE in caso contrario. Queste routine sono utili in aritmetica a precisione estesa, oppure in aritmetica modulo 2^32, o ancora in aritmetica basata su dati in input dell'utente.

FUNCTION LDSRQQ (CONSTS A : REAL4) : REAL4;
FUNCTION LDDRQQ (CONSTS A : REAL8) : REAL8;

Sono funzioni aritmetiche. Ritornano il logaritmo, in base 10, di A. Sia A che il valore di ritorno sono di tipo REAL4 o REAL8. Queste funzioni sono della libreria runtime del MS-FORTRAN e devono essere dichiarate EXTERN prima di essere usate, se l'MS-FORTRAN e' disponibile sulla macchina specifica.

FUNCTION LMULOK (A, B: INTEGER4; VAR C: INTEGER4) : BOOLEAN;

E' una routine di libreria (senza overflow aritmetico). Posiziona C uguale ad A moltiplicato B. Insieme alla funzione LADDOK, effettua l'aritmetica su 32 bit con segno senza causare un errore runtime su overflow. La normale aritmetica puo' causare un errore di runtime, anche se l'indicatore di debug non e' attivo. Sia LMULOK che LADDOK ritornano TRUE se non c'e' overflow, e FALSE se c'e'. Queste routine sono utili in aritmetica a precisione estesa, o in aritmetica modulo 2^32, o in aritmetica basata su dati in input dell'utente.

FUNCTION LN (X : REAL) : REAL;

E' una funzione aritmetica. Ritorna il logaritmo, in base e, di X. Sia X che il valore di ritorno sono di tipo REAL. Per forzare una particolare precisione, si devono dichiarare ed usare le funzioni LNSRQQ (CONSTS REAL4) e/o LNDRQQ (CONSTS REAL8). Si verifica un errore se X e' minore o uguale a zero.

FUNCTION LNSRQQ (CONSTS A : REAL4) : REAL4;
FUNCTION LNDRQQ (CONSTS A : REAL8) : REAL8;

Vedere FUNCTION LN.

FUNCTION LOBYTE (INTEGER-WORD) : BYTE;

E' una funzione intriseca di livello Esteso. Ritorna il byte meno significativo di un INTEGER o WORD. A seconda dalla macchina, il byte meno significativo puo' essere il primo o il secondo byte indirizzato della word.
Vedere anche FUNCTION HIBYTE.

FUNCTION LOCKED (VARS SEMAPHORE : WORD) : BOOLEAN;

E' una funzione di libreria (gestione dei semafori). Se il semaforo e' disponibile, LOCKED ritorna il valore TRUE e pone il semaforo in stato non-disponibile. Altrimenti, LOCKED ritorna il valore FALSE. UNLOCK rende disponibile il semaforo. Dato che il semaforo e' binario, puo' assumere soltanto due stati.
Vedere anche PROCEDURE UNLOCK.

FUNCTION LOWER (EXPRESSION) : VALUE;

E' una funzione intrinseca del livello Esteso. LOWER accetta un solo parametro di uno dei seguenti tipi: array, set, enumerato oppure subrange. Il valore ritornato da LOWER e' uno dei seguenti:

- il limite inferiore di un array
- il primo elemento disponibile di un insieme
- il primo valore di un tipo enumerato
- il limite inferiore di un subrange

LOWER usa il tipo, non il valore, dell'espressione. Il valore ritornato da LOWER e' sempre una costante.
Vedere anche FUNCTION UPPER.

FUNCTION LOWORD (INTEGER4) : WORD;

E' una funzione intrinseca di livello Esteso. Ritorna la WORD meno significativa dei quattro byte di INTEGER4.
Vedere anche FUNCTION HIWORD.

PROCEDURE MARKAS (VAR HEAPMARK : INTEGER4);

E' una procedura di libreria (gestione dello heap). E' analoga alla procedura MARK presente in altri compilatori Pascal. MARKAS segna i limiti superiori ed inferiori dello heap. La procedura DISPOSE e', in generale, piu' potente, pero' MARKAS puo' essere utile per conversioni da altri linguaggi Pascal.

In altri Pascal il parametro e' di tipo puntatore. Pascal ha bisogno di due parole per memorizzare i limiti dello heap dato che, in alcune implementazioni, lo heap cresce sia verso il limite superiore che verso quello inferiore. La variabile HEAPMARK non deve essere usata come un numero normale INTEGER4; essa deve essere caricata da MARKAS e quindi passata a RELEAS.

Per usare MARKAS e RELEAS si deve passare una variabile INTEGER4, ad esempio M, come parametro VAR a MARKAS. MARKAS memorizza in M i limiti dello heap. Per liberare lo spazio dello heap occorre semplicemente richiamare la procedura RELEAS (M).

MARKAS e RELEAS funzionano come descritto soltanto se non viene mai richiamata la procedura DISPOSE.

FUNCTION MDSRQQ (CONSTS A, B : REAL4) : REAL4;
FUNCTION MDDRQQ (CONSTS A, B : REAL8) : REAL8;

Sono funzioni aritmetiche. La funzione A modulo B e' definita come:

```
MDSRQQ (A, B) = A - AISRQQ (A / B) * B
MDDRQQ (A, B) = A - AIDRQQ (A / B) * B
```

Sia A che B sono di tipo REAL4 o REAL8. Queste funzioni appartengono alla libreria runtime dell'MS-FORTRAN e devono essere dichiarate EXTERN nel programma che ne fa uso, se l'MS-FORTRAN e' disponibile sulla macchina specifica.

FUNCTION MEMAVL : WORD;

E' una funzione di libreria (gestione dello heap). Ritorna il numero di byte disponibili tra lo stack e lo heap. MEMAVL funziona in modo analogo alla funzione MEMAVAIL nel Pascal UCSD. Se e' stata usata la procedura DISPOSE in precendenza, MEMAVL puo' ritornare un valore inferiore al numero effettivo di byte disponibli.

FUNCTION MNSRQQ (CONSTS A, B : REAL4) : REAL4;
FUNCTION MNDRQQ (CONSTS A, B : REAL8) : REAL8;

Sono funzioni aritmetiche. Ritornano il valore piu' piccolo di A e B. Sia A che B sono di tipo REAL4 o REAL8. Queste funzioni appartengono alla libreria runtime dell'MS-FORTRAN e devono essere dichiarate EXTERN dal programma che ne fa uso, se l'MS-FORTRAN e' disponibile sulla specifica macchina.
Vedere anche FUNCTION MXSRQQ e FUNCTION MXDRQQ.

PROCEDURE MOVEL (S, D : ADRMEM; N : WORD);

E' una procedura intrinseca di livello di Sistema. Trasferisce N caratteri (byte) da S^ a D^, a partire dal primo byte di ciascun array. Indipendentemente dal valore del campo di variabilita' e dagli indicatori di controllo indice, non vengono effettuati controlli sui limiti degli array.
Esempio:

```
MOVEL (ADR 'New String Value', ADR V, 16)
```

Per i tipi ad indirizzo segmentato si puo' anche vedere PROCEDURE MOVESL. Per trasferire byte a sinistra, oppure quando i campi di variabilita' degli indirizzi non si sovrappongono, si devono usare MOVEL e MOVESL.

Le procedure MOVE e FILL assumono parametri di tipo ADRMEM e ADSMEM, ma poiche' tutti i tipi ADR (o ADS) sono compatibili, i tipi ADR (o ADS) di qualunque variabile o costante possono essere usati come parametri attuali. Questo modo di procedere e' pericoloso, ma talvolta utile.

PROCEDURE MOVER (S, D : ADRMEM; N : WORD);

E' una procedura intrinseca di livello di Sistema. E' analoga a MOVEL, con la differenza che il trasferimento comincia dall'ultimo byte di ciascun array. Si devono usare MOVER e MOVESR per trasferire byte a destra. Come per MOVEL, non c'e' controllo sui limiti degli array.
Esempio:

```
MOVER (ADR V[0], ADR V[4], 12)
```

Per i tipi ad indirizzo segmentato si puo' vedere anche PROCEDURE MOVESR. Le procedure MOVE e FILL accettano parametri di tipo ADRMEM e ADSMEM; dato che tutti i tipi ADR (o ADS) sono compatibili, possono essere passate variabili o costanti di tipo ADR (o ADS). Questo modo di procedere e' pericoloso, ma talvolta utile.

PROCEDURE MOVESL (S, D : ADSMEM; N : WORD);

E' una procedura intrinseca di livello di Sistema. Trasferisce N caratteri (byte) da S^ a D^ iniziando dal primo byte di ciascun array. Indipendentemente dal valore del campo di variabilita' e dagli indicatori di controllo indice, non viene effettuato alcun controllo sui limiti degli array.

Esempio:

```
MOVESL (ADS 'New String Value', ADS V, 16)
```

Per i tipi ad indirizzo relativo si puo' vedere PROCEDURE MOVEL. Per trasferire byte a sinistra, o quando i campi di variabilita' degli indirizzi non si sovrappongono, si devono usare MOVEL e MOVESL.

Le procedure MOVE a FILL accettano parametri di tipo ADRMEM e ADSMEM; dato che tutti i tipi ADR (o ADS) sono compatibili, possono essere passate variabli o costanti di tipo ADR (o ADS). Questo modo di procedere e' pericoloso ma talvolta utile.

PROCEDURE MOVESR (S, D : ADSMEM; N : WORD);

E' una procedura intrinseca di livello di Sistema. E' analoga a MOVESL, con la differenza che il trasferimento comincia dall'ultimo byte di ciascun array. Si devono usare MOVER e MOVESR per trasferire byte a destra. Analogamente a MOVESL, non c'e' controllo sui limiti degli array.
Esempio:

```
MOVESR (ADR V[0], ADR V[4], 12)
```

Per i tipi ad indirizzo relativo si puo' vedere anche PROCEDURE MOVER.
Le procedure MOVE e FILL accettano parametri di tipo ADRMEM e ADSMEM; dato che tutti i tipi ADR (o ADS) sono compatibili, possono essere passate variabili o costanti di tipo ADR (o ADS). Questo modo di procedere e' pericoloso, ma talvolta utile.

FUNCTION MXSRQQ (CONSTS A, B : REAL4) : REAL4;
FUNCTION MXDRQQ (CONSTS A, B : REAL8) : REAL8;

Sono funzioni aritmetiche. Ritornano il valore massimo tra A e B. Sia A che B sono di tipo REAL4 o REAL8. Queste funzioni appartengono alla libreria MS-FORTRAN e devono essere dichiarate EXTERN dal programma che le utilizza, se l'MS-FORTRAN e' disponibile sulla macchina specifica.

Vedere anche FUNCTION MNSRQQ e MNDRQQ.

PROCEDURE NEW (VARS P : POINTER);

E' una procedura di libreria (gestione dello heap, forma contratta). Alloca una nuova variabile V nello heap ed assegna a P il relativo puntatore (come parametro VARS). Il tipo di V e' determinato dalla dichiarazione del puntatore P. Se V e' di tipo super array occorre usare la forma lunga di NEW. Se V e' un tipo record con varianti, vengono considerate le varianti con lunghezza maggiore: esse permettono cosi' l'utilizzo di ogni variante assegnata a P^.

PROCEDURE NEW (VARS P : POINTER; T1, T2,...TN : TAGS);

E' una procedura di libreria (gestione dello heap, forma lunga). Alloca una nuova variabile con le varianti specificate dai valori etichetta da T1 a TN. I valori etichetta vengono elencati nell'ordine in cui sono dichiarati. Possono essere omessi campi etichettati compresi tra l'iniziale e il finale.

Se tutti i valori etichetta sono costanti,il Pascal alloca soltanto lo spazio richiesto, allineato alla parola. Il valore di ogni campo etichettato omesso viene considerato in modo da allocare la dimensione massima possibile. Se alcuni campi etichettati non sono costanti, il compilatore usa una delle due strategie:

1. Assume che il primo campo etichettato non sia costante e che tutti i seguenti abbiano valori sconosciuti, e quindi alloca la massima dimensione.

2. Genera una chiamata speciale runtime ad una funzione che calcola la dimensione del record dai valori etichetta variabili disponibili. Questo dipende dalla particolare implementazione. Un tale modo di procedere viene usato per DISPOSE e SIZEOF.

I valori dei campi etichettati devono essere impostati dopo la chiamata della NEW e non devono piu' essere cambiati. Il compilatore non esegue alcuna delle seguenti azioni:

- assegnazione di valori etichetta

- controllo della loro corretta inizializzazione

- controllo della non variazione del loro valore durante l'esecuzione.

In accordo allo standard ISO, una variabile creata mediante la forma lunga di NEW ha queste caratteristiche:

- non puo' essere usata come operando in un'espressione

- non puo' essere passata come parametro

- non le si puo' assegnare un valore

Il Pascal non controlla questi errori. I campi all'interno del record possono essere usati normalmente.

L'assegnazione di un record di dimensione maggiore ad un altro di dimensione minore danneggia parte del contenuto dello heap. Questo errore e' difficile da riconoscere in fase di compilazione. Quindi, nel Pascal, ogni assegnazione di un record con varianti nello heap usa la lunghezza del record invece della lunghezza massima.

Un'assegnazione in un campo di un record, se fatta usando le varianti sbagliate, puo' distruggere parte di un'altra variabile nello heap oppure la struttura stessa dello heap. Questo errore non viene segnalato, se i valori di etichetta non sono espliciti e corretti e se e' attivo l'indicatore di controllo di etichetta.

Il livello Esteso permette l'uso di puntatore a super array. La forma lunga di NEW viene usata come descritto in precedenza, a parte il fatto che vengono forniti i limiti superiori dell'array al posto dei valori etichetta. Devono essere forniti tutti i limiti superiori. Essi possono essere costanti o espressioni; in ogni caso viene allocata soltanto la dimensione richiesta.

L'array cui fa riferimento un puntatore non puo' essere assegnato o paragonato come elemento unico; l'unica eccezione e' LSTRING. L'array intero puo' essere passato come parametro di riferimento, se il parametro formale e' dello stesso tipo super array. Le componenti dell'array possono essere usate normalmente.

FUNCTION ODD (X : ORDINAL) : INTEGER;

E' una funzione di conversione dati. Essa controlla il valore ordinale X per vedere se e' dispari. ODD e' TRUE soltanto se ORD (X) e' dispari; altrimenti e' FALSE.

FUNCTION ORD (X : VALUE) : INTEGER;

E' una funzione di conversione dati. Converte in INTEGER ogni valore dei tipi elencati nella tabella 16-6, secondo le regole date.

TIPO DI X	VALORE DI RITORNO
INTEGER	X
WORD <= MAXINT	X
WORD > MAXINT	X - 2 * (MAXINT + 1) (cioe' i 16 bit di partenza)
CHAR	Codice ASCII per X
Enumerato	Posizione di X nella definizione di tipo a partire da 0
INTEGER4	I 16 bit meno significativi (cioe' come ORD (LOWORD (INTEGER4)))
Puntatore	Valore intero del puntatore

Tabella 16-6 Conversione in INTEGER

PROCEDURE PACK (CONSTS A: UNPACKED; I: INDEX; VARS Z: PACKED);

Procedura di conversione dati. Trasferisce elementi da un array non impaccato ad un array impaccato. Se A e' un ARRAY [M..N] OF T e Z e' un PACKED ARRAY [U..V] OF T, allora PACK (A, 1, Z) equivale a:

```
FOR J := U TO V DO Z [J] := A [J - U + 1]
```

In entrambi, PACK e UNPACK, il parametro I e' l'indice iniziale di A. I limiti degli array ed il valore di 1 devono essere ragionevoli, cioe' il numero dei componenti di A da 1 a M deve essere grande almeno come il numero dei componenti di Z. L'indicatore di controllo di variabilita' controlla i limiti degli array.

PROCEDURE PAGE;
PROCEDURE PAGE (VAR F);

E' una procedura di file system. Provoca il salto all'inizio della nuova pagina quando viene stampato il file-testo F. PAGE senza parametri equivale a PAGE (INPUT).
Per una descrizione di PAGE si puo' vedere "PAGE", nel Capitolo 17.

FUNCTION PISRQQ (CONSTS A: REAL4; CONSTS B: INTEGER4) : REAL4;
FUNCTION PIDRQQ (CONSTS A: REAL8; CONSTS B: INTEGER4) : REAL8;

Sono funzioni aritmetiche. Il valore di ritorno e' A**B (A elevato alla parte intera di B). A puo' essere di tipo REAL4 o REAL8. B e' sempre di tipo INTEGER4. Queste funzioni appartengono alla libreria runtime dell'MS-FORTRAN e devono essere dichiarate EXTERN dal programma che ne fa uso, se l'MS-FORTRAN e' disponibile sulla specifica macchina.

PROCEDURE PLYUQQ;

E' una procedura di libreria (I/O da terminale). Scrive un carattere di fine riga sul video.
Insieme a GETYQQ e PTYUGG, PLYUQQ e' utile per effettuare I/O da terminale in un ambiente ad alte prestazioni.

FUNCTION POSITN (CONSTS PAT:STRING; CONSTS S:STRING; I:INTEGER):INTEGER;

E' una funzione intrinseca di stringa. Ritorna la posizione intera della stringa PAT in S; la ricerca parte da S[I]. Se PAT non viene trovata oppure se I > UPPER(S), allora il valore di ritorno e' 0. Se PAT e' una stringa nulla, allora il valore di ritorno e' 1. Non esistono condizioni di errore.

FUNCTION PRED (X : ORDINAL) : ORDINAL;

E' una funzione di conversione dati. Determina il "predecessore" ordinale di X. L'ORD del risultato equivale a ORD(X) - 1. Viene emesso un errore se il predecessore e' fuori del campo di variabilita' oppure se si verifica un overflow. Questi errori vengono segnalati se sono attivi gli appropriati indicatori di debug.

FUNCTION PRSRQQ (A, B : REAL4) : REAL4;
FUNCTION PRDRQQ (A, B : REAL8) : REAL8;

Sono funzioni aritmetiche. Il valore di ritorno e' A**B (A elevato a B). Sia A che B sono di tipo REAL4 o REAL8. Viene emesso un errore se A < 0 (anche se B ha valore intero). Queste funzioni appartengono alla libreria runtime dell'MS-FORTRAN e devono essere dichiarate EXTERN dal programma che ne fa uso, se l'MS-FORTRAN e' disponibile sulla specifica macchina.

PROCEDURE PTYUQQ (LEN : WORD; LOC : ADSMEM);

E' una procedura di libreria (I/O da terminale). Scrive LEN caratteri sul video del terminale. I caratteri iniziano all'indirizzo LOC in memoria.

Esempio:

```
PTYUQQ (8, ADS 'PROMPT: ');
```

Insieme a GETYQQ e PLYUQQ, PTYUQQ e' utile per effettuare I/O da terminale in ambiente ad alte prestazioni.

PROCEDURE PUT (VAR F);

E' una procedura di file system. Scrive il valore della variabile buffer F^ nel componente corrente di F e fa avanzare il puntatore del file.
Per una descrizione di PUT si puo' vedere "GET e PUT", nel Capitolo 17.

PROCEDURE READ (VAR F; P1, P2,..,PN);

E' una procedura di file system. READ legge i dati dai file. Sia READ che READLN sono definite in termini dell'operazione piu' primitiva GET..
Per una descrizione di READ si puo' vedere la sezione "Input e Output con File-Testo", nel Capitolo 17.

PROCEDURE READFN (VAR F; P1, P2,..,PN);

E' una procedura di file system. (1/O di livello Esteso). READFN e' analoga a READ (non READLN), con due eccezioni:

1. il parametro file F deve essere presente (viene assunto INPUT ma viene emessa una segnalazione)

2. se il parametro P e' di tipo FILE, allora viene letta, da F, una sequenza valida di caratteri che viene assegnata a P in modo analogo ad ASSIGN.

I parametri di altri tipi vengono letti nello stesso modo delle procedure READ.
Per una descrizione di READFN si puo' vedere la sezione "Procedure di Livello Esteso", nel Capitolo 17.

PROCEDURE READLN (VAR F; P1, P2,..,PN);

E' una procedura di 1/O su file. A livello primitivo di GET, senza parametri, READLN(F) e' equivalente a :

```
BEGIN
  WHILE NOT EOLN (F) DO GET (F);
  GET (F)
END
```

La procedura READLN e' molto simile a READ, con la differenza che legge anche il carattere di fine riga.
Per una descrizione di READ si puo' vedere "Input e Output con File-Testo", nel Capitolo 17.

PROCEDURE READSET (VAR F; VAR L: LSTRING; CONSTS S: SETOFCHAR);

E' una procedura di file system (1/O di livello Esteso). READSET legge i caratteri e li trasferisce in L, finche' i caratteri sono nell'insieme S e c'e' spazio in L.

Per una descrizione di READSET si puo' vedere "Procedure di Livello Esteso", nel Capitolo 17.

PROCEDURE RELEAS (VAR HEAPMARK : INTEGER4);

E' una routine di libreria (gestione dello heap). E' analoga alla procedura RELEASE presente in altri compilatori Pascal. RELEAS libera l'area heap identificata dalla chiamata precedente a MARKAS. La procedura DISPOSE e' piu' potente, in generale, ma RELEAS puo'essere utile per conversioni da altri linguaggi Pascal. In altri Pascal, il parametro e' di tipo puntatore. Il Pascal ha bisogno di due parole per memorizzare i limiti dello heap, poiche' in alcune implementazioni lo heap cresce sia verso il limite superiore che verso il limite inferiore. La variabile HEAPMARK non deve essere usata come un numero normale INTEGER4; essa deve essere caricata da MARKAS e quindi passata a RELEAS.

Per usare MARKAS e RELEAS occorre passare una variabile INTEGER4, ad esempio M, come parametro VAR a MARKAS. MARKAS memorizza in M i limiti dello heap. Per liberare lo spazio dello heap occorre semplicemente richiamare la procedura RELEAS(M). MARKAS e RELEAS funzionano come descritto soltanto se non viene mai richiamata la procedura DISPOSE.

PROCEDURE RESET (VAR F);

E' una procedura di file system. Posiziona all'inizio il file F ed effettua una GET(F).
Per una descrizione di RESET si puo' vedere "RESET e REWRITE", nel Capitolo 17.

FUNCTION RESULT (FUNCTION-IDENTIFIER) : VALUE;

E' una funzione intrinseca di livello Esteso. Viene usata per ottenere il valore corrente di una funzione; puo' essere usata soltanto all'interno della funzione stessa oppure in una procedura o funzione in essa nidificata.

FUNCTION RETYPE (TYPE-IDENT, EXPRESSION) : TYPE-IDENT;

E' una funzione intrinseca di livello di Sistema. Fornisce una valutazione generica di tipo; ritorna il valore dell'espressione come se essa fosse del tipo specificato. Il tipo identificato e l'espressione dovrebbero avere la stessa lunghezza, ma questo non e' obbligatorio. RETYPE su una struttura puo' essere seguita da selettori di componenti (indice di array, campi, riferimenti etc). RETYPE e' una funzione "pericolosa" e puo' funzionare in modo diverso da quello previsto.

Esempio:

```
TYPE COLOR = (RED, BLUE, GREEN);
     S2    = STRING(2);

VAR C :#CHAR;
    I, J :#INTEGER;
    R :#REAL4;
    TINT :#COLOUR;
    .
    .
    R := RETYPE (REAL4, 'abcd');
    {La stringa letterale di 4 byte viene convertita in un numero REAL4}

    TINT := RETYPE (COLOR, 2)
    {2 viene convertito in un colore, in questo caso GREEN.}
    {Questo e' un uso di RETYPE che non presenta particolari pericoli.}

    C := RETYPE (S2, I) [J]
    {I viene convertito in una stringa di 2 caratteri.}
    {J identifica un carattere della stringa che e' assegnato a C.}
```

Esistono altri due modi di cambiare tipo in Pascal:

1. Si puo' usare un record con varianti, ciascuno del tipo desiderato. Si effettua un'assegnazione ad una variante e si leggono i valori attraverso le altre varianti. (Questo e' un errore non rilevato a livello Standard. Si deve osservare che l'allocazione relativa delle varianti puo' cambiare da compilatore a compilatore.)

2. Si puo' dichiarare una variabile indirizzo del tipo voluto e poi assegnarle l'indirizzo di un'altra variabile (mediante ADR).

Ognuno di questi metodi presenta le sue piccole differenze e deve essere evitato quando possibile.

PROCEDURE REWRITE (F);

E' una procedura di file system. Riporta all'inizio la posizione corrente di un file.
Per una descrizione di REWRITE si puo' vedere "RESET e REWRITE", nel Capitolo 17.

FUNCTION ROUND (X : REAL) : INTEGER;

E' una funzione aritmetica. Arrotonda per eccesso il valore di X. X puo' essere di tipo REAL4 o REAL8; il valore di ritorno e' INTEGER. L'effetto di ROUND su un numero con una parte frazionale di 0.5 varia a seconda dell'implementazione.
Esempio:

```
ROUND (1.6)    e'   2
ROUND (-1.6)   e'  -2
```

Viene emesso un segnale di errore quando ABS (X + 0.5) >= MAXINT.

FUNCTION ROUND4 (X : REAL) : INTEGER4;

E' una funzione aritmetica. Arrotonda per eccesso il valore di X. X puo' essere di tipo REAL4 o REAL8; il valore di ritorno e' di tipo INTEGER4. L'effetto di ROUND4 con una parte frazionale di 0.5 varia a seconda dell'implementazione.
Esempi:

```
ROUND4 (1.6)    e'   2
ROUND4 (-1.6)   e'  -2
```

Viene emesso un segnale di errore quando ABS (X + 0.5)>= MAXINT4.

FUNCTION SADDOK (A, B: INTEGER; VAR C: INTEGER) : BOOLEAN;

E' una routine di libreria (senza overflow aritmetico). Stabilisce che C e' uguale ad A piu' B. Insieme a SMULOK, e' una funzione che effettua l'aritmetica su 16 bit senza causare un errore runtime se avviene l'overflow. L'aritmetica normale emette un segnale di errore runtime su overflow anche se l'indicatore di debugging non e' attivo. Sia SADDOK che SMULOK ritornano il valore TRUE se non c'e' stato overflow, e FALSE in caso contrario. Queste routine sono utili per l'aritmetica a precisione estesa, per l'aritmetica modulo 2^16 oppure per l'aritmetica basata su dati input.

FUNCTION SCANEQ (LEN: INTEGER; PAT: CHAR; CONSTS S: STRING; I: INTEGER): INTEGER;

E' una funzione intrinseca di stringa. Esegue una ricerca su S, a partire da S[I], e ritorna il numero di caratteri esaminati. SCANEQ termina la sua ricerca quando incontra un carattere uguale a PAT oppure dopo aver considerato LEN caratteri. Se LEN < 0, SCANEQ effettua la sua ricerca in senso opposto e ritorna un numero negativo. SCANEQ ritorna il valore LEN se non trova caratteri uguali a PAT oppure se I > UPPER (S). Non vi sono condizioni di errore.

FUNCTION SCANNE (LEN: INTEGER; PAT: CHAR; CONSTS S: STRING; I: INTEGER): INTEGER;

E' una funzione intrinseca di stringa. E' simile a SCANEQ, ma la sua ricerca termina quando incontra un carattere diverso da PAT. La ricerca inizia da S [1] e ritorna il numero di caratteri esaminati. SCANNE termina la sua ricerca quando incontra un carattere diverso da PAT oppure dopo aver considerato LEN caratteri. Se LEN < 0, SCANNE effettua la sua ricerca in senso opposto e ritorna un numero negativo. SCANNE ritorna il valore LEN se non trova caratteri uguali a PAT oppure se 1 > UPPER (S). Non vi sono condizioni di errore.

PROCEDURE SEEK (VAR F; N : INTEGER4);

E' una procedura di file system (I/O di livello Esteso). A differenza dei file normali sequenziali, i file DIRECT hanno strutture ad accesso diretto. SEEK viene usata per accedere direttamente ai componenti di tali file.
Per ulteriori dettagli si puo' vedere "I/O di Livello Esteso", nel Capitolo 17.

FUNCTION SHSRQQ (CONSTS A : REAL4) : REAL4;
FUNCTION SHDRQQ (CONSTS A : REAL8) : REAL8;

Sono funzioni aritmetiche. Ritornano il seno iperbolico di A. A e' di tipo REAL4 o REAL8. Queste funzioni appartengono alla libreria runtime dell'MS-FORTRAN e devono essere dichiarate EXTERN dal programma che ne fa uso, se l'MS-FORTRAN e' disponibile sulla specifica macchina.

FUNCTION SIN (X : NUMERIC) : REAL;

E' una funzione aritmetica. Ritorna il seno di X in radianti. Sia X che il valore di ritorno sono di tipo REAL. In caso di precisioni particolari occorre usare le routine SNSRQQ (CONSTS REAL4) e/o SNDRQQ (CONSTS REAL8).

FUNCTION SIZEOF (VARIABLE) : WORD;
FUNCTION SIZEOF (VARIABLE, TAG1, TAG2,..,TAGN) : WORD;

E' una funzione intrinseca del livello Esteso. Ritorna la dimensione di una variabile in byte. I valori TAG oppure i limiti superiori degli array sono impostati in modo analogo alle funzioni NEW e DISPOSE. Se la variabile e' un record con varianti, e viene usata la prima forma, viene ritornata la dimensione massima possibile. Se la variabile e' di tipo super array, allora deve essere usata la seconda forma, la quale fornisce i limiti superiori.

FUNCTION SMULOK (A, B : INTEGER; VAR C : INTEGER) : BOOLEAN;

E' una routine di libreria (senza overflow aritmetico). Stabilisce che C e' uguale ad A moltiplicato B. Insieme a SADDOK, e' una funzione che effettua l'aritmetica su 16 bit senza causare un errore runtime se avviene l'overflow. L'aritmetica normale emette un errore runtime su overflow anche se l'indicatore di debugging non e' presente. Sia SADDOK che SMULOK ritornano il valore TRUE se non c'e' stato overflow e FALSE in caso contrario. Queste routine sono utili per l'aritmetica a precisione estesa, per l'aritmetica modulo 2^16 oppure per l'aritmetica basata su dati input.

FUNCTION SNSRQQ (CONSTS A : REAL4) : REAL4;
FUNCTION SNDRQQ (CONSTS A : REAL8) : REAL8;

Vedere FUNCTION SIN.

FUNCTION SQR (X : NUMERIC) : NUMERIC;

E' una funzione aritmetica. Ritorna il quadrato di X, dove X puo' essere di tipo REAL, INTEGER, WORD o INTEGER4.

FUNCTION SQRT (X) : REAL;

E' una funzione aritmetica. Ritorna la radice quadrata di X, dove X e' di tipo REAL. In caso di precisione particolare, occorre usare le routine SRSRQQ (CONSTS REAL4) e/o SRDRQQ (CONSTS REAL8). Viene emesso un segnale di errore se X e' minore di zero.

FUNCTION SRSRQQ (CONSTS A : REAL4) : REAL4;
FUNCTION SRDRQQ (CONSTS A : REAL8) : REAL8;

Vedere FUNCTION SRQT.

FUNCTION SUCC (X : ORDINAL) : ORDINAL;

E' una funzione di conversione dati. Determina il "successore" ordinale di X. L'ORD del risultato di ritorno e' uguale a ORD (X) + 1. Viene emesso un segnale di errore se il successore e' fuori del campo di variabilita' oppure se avviene un overflow. Questi errori sono segnalati se l'appropriato indicatore di debug e' presente.

FUNCTION THSRQQ (CONSTS A : REAL4) : REAL4;
FUNCTION THDRQQ (CONSTS A : REAL8) : REAL8;

Sono funzioni aritmetiche. Ritornano la tangente iperbolica di A. Sia A che il valore di ritorno sono di tipo REAL4 o REAL8. Queste funzioni appartengono alla libreria runtime dell'MS-FORTRAN e devono essere dichiarate EXTERN dal programma che ne fa uso, se l'MS-FORTRAN e' disponibile sulla specifica macchina.

FUNCTION TICS : WORD;

E' una routine di libreria (funzione di clock). Se disponibile, TICS ritorna il valore di una locazione di clock del sistema operativo. Il risultato viene fornito in un intervallo di tempo (ad esempio centesimi di secondo) che dipende dal sistema operativo.

PROCEDURE TIME (VAR S : STRING);

E' una routine di libreria (funzione di clock). Se disponibile, questa routine assegna il valore corrente di clock alla variabile STRING (o LSTRING). Se il parametro e' LSTRING, occorre impostare la lunghezza prima di richiamare TIME. Il formato dipende dal sistema operativo.
Vedere anche PROCEDURE DATE.

FUNCTION TNSRQQ (CONSTS A : REAL4) : REAL4;
FUNCTION TNDRQQ (CONSTS A : REAL8) : REAL8;

Sono funzioni aritmetiche. Ritornano il valore della tangente di A. Sia A che il valore di ritorno sono REAL4 o REAL8. Queste funzioni appartengono alla libreria runtime dell'MS-FORTRAN e devono essere dichiarate EXTERN dal programma che ne fa uso, se l'MS-FORTRAN e' disponibile sulla specifica macchina.

FUNCTION TRUNC (X : REAL) : INTEGER;

E' una funzione aritmetica. Tronca il valore di X, dove X e' di tipo REAL4 o REAL8 ed il valore di ritorno e' INTEGER.
Esempi:

```
TRUNC (1.6)    e'   1
TRUNC (-1.6)   e'  -1
```

Viene emesso un segnale di errore se ABS (X - 1.0) >= MAXINT.

FUNCTION TRUNC4 (X : REAL) : INTEGER4;

E' una funzione aritmetica. Tronca il valore di X, dove X e' di tipo REAL4 o REAL8 ed il valore di ritorno e' INTEGER4.
Esempi:

```
TRUNC4 (1.6)   e'   1
TRUNC4 (-1.6)  e'  -1
```

Viene emesso un segnale di errore se ABS (X - 1.0) >= MAXINT4.

FUNCTION UADDOK (A, B: WORD; VAR C: WORD) : BOOLEAN;

E' una routine di libreria (senza overflow aritmetico). Stabilisce che C e' uguale ad A piu' B. Insieme a UMULOK effettua l'aritmetica senza segno su 16 bit senza causare un errore runtime se avviene l'overflow. L'aritmetica normale emette un errore runtime su overflow anche se l'indicatore di debugging non e' attivo. Quello che segue e' il riporto binario risultante dalla somma di A e B:

```
WRD (NOT UADDOK (A, B, C))
```

Sia UADDOK che UMULOK ritornano TRUE se non vi e' stato overflow, e FALSE in caso contrario. Queste routine sono utili per aritmetica a precisione estesa, per aritmetica modulo 2^16 o per aritmetica basata su input utente.

FUNCTION UMULOK (A, B: WORD; VAR C: WORD) : BOOLEAN;

E' una routine di libreria (senza overflow aritmetico). Stabilisce che C e' uguale ad A moltiplicato B. Insieme a UADDOK effettua l'aritmetica senza segno su 16 bit senza causare un errore runtime se avviene l'overflow. L'aritmetica normale puo' causare un errore runtime su overflow anche se l'indicatore di debugging non e' attivo. Entrambe le routine ritornano TRUE se non vi e' stato overflow, e FALSE in caso contrario. Queste routine sono utili per aritmetica a precisione estesa, per aritmetica modulo 2^16 o per aritmetica basata su input utente.

PROCEDURE UNLOCK (VARS SEMAPHORE : WORD);

E' una routine di libreria (gestione dei semafori). UNLOCK rende disponibile un semaforo. Essendo di tipo binario, un semaforo puo' avere soltanto due stati. UNLOCK puo' essere richiamata per un numero qualsiasi di volte e puo' essere usata per inizializzare un semaforo.
Vedere anche FUNCTION LOCKED.

PROCEDURE UNPACK (CONSTS Z: PACKED; VARS A: UNPACKED; I: INDEX);

E' una procedura di conversione dati. Trasferisce elementi da un array impaccato ad un array non impaccato. Se A e' un ARRAY [M..N] OF T, e Z e' un PACKED ARRAY [U..V] OF T, allora la chiamata precedente equivale a:

```
FOR J := U TO V DO A [J - U + 1] := Z [J]
```

Sia in PACK che in UNPACK il parametro I e' l'indice iniziale di A. I limiti degli array ed il valore di I devono essere ragionevoli, cioe' il numero dei componenti di A da I ad M deve essere grande almeno come il numero dei componenti di Z. L'indicatore di controllo di variabilita' controlla i limiti degli array.
Vedere anche PROCEDURE PACK.

FUNCTION UPPER (EXPRESSION) : VALUE;

E' una funzione intrinseca del livello Esteso. UPPER, come LOWER, accetta un solo parametro di uno dei seguenti tipi: array, set, enumerato oppure subrange. Il valore ritornato da UPPER puo' essere uno dei seguenti:

- il limite superiore di un array
- l'ultimo elemento disponibile di un insieme
- l'ultimo valore di un tipo enumerato
- il limite superiore di un subrange.

Se l'espressione non e' di tipo super array, il valore ritornato da UPPER e' sempre una costante Se lo e', viene ritornato il limite superiore del tipo super array. Si deve osservare che, in UPPER, viene usato il tipo e non il valore dell'espressione.
Vedere anche PROCEDURE LOWER.

PROCEDURE VECTIN (V: WORD; PROCEDURE I [INTERRUPT]);

E' una routine di libreria (gestione delle interruzioni). E' una delle tre procedure che gestiscono le interruzioni. VECTIN imposta un vettore di interruzione in modo che le interruzioni di tipo V siano connesse alla procedura I. (ENABIN abilita le interruzioni, DISBIN le disabilita). Gli effetti di queste procedure ed il significato di V variano a seconda dalla macchina. Per maggiori informazioni sull'implementazione si puo' vedere l'Appendice A, "Caratteristiche di Implementazione", nel manuale "Linguaggio Pascal Guida Utente".

FUNCTION WRD (X : VALUE) : WORD;

E' una procedura di conversione dati. Converte in WORD qualsiasi tipo contenuto in tabella 16-7, secondo le regole fornite.

TIPO DI X	VALORE DI RITORNO
WORD	X
INTEGER >= 0	X
INTEGER < 0	X + MAXWORD + 1 (cioe' i 16 bit di partenza)
CHAR	Codice ASCII di X
Enumerato	Posizione di X nella definizione di tipo, a partire da 0
INTEGER4	I 16 bit meno significativi (come LOWORD (INTEGER4))
Puntatore	Valore WORD del puntatore

Tabella 16-7 Conversione in WORD

PROCEDURE WRITE (VAR F; P1, P2,..,PN);
PROCEDURE WRITELN (VAR F; P1,P2,..,PN);

Sono procedure intrinseche a livello di file system. Scrivono dati nei file. WRITE e WRITELN sono definite in termini dell'operazione piu' primitiva PUT. WRITELN e' uguale a WRITE, con la differenza che scrive anche un carattere di fine riga.
Per una descrizione di queste procedure si puo' vedere "WRITE e WRITELN", nel Capitolo 17.

17. PROCEDURE E FUNZIONI ORIENTATE SU FILE

SOMMARIO

Questo capitolo descrive tutte le procedure e funzioni Input/Output, come pure la valutazione pigra (lazy) e l'I/O simultaneo, due caratteristiche speciali del Pascal che facilitano l'uso dei file.

INDICE

INTRODUZIONE

Il file system del Pascal ammette una varieta' di procedure e funzioni che operano su file con modi di accesso e strutture diversi. Queste procedure e funzioni possono essere catalogate come illustrato nella tabella 17-1.

CATEGORIA	PROCEDURE	FUNZIONI
Primitive	GET PAGE PUT RESET REWRITE	EOF EOLN
I/O di file-testo	READ READLN WRITE WRITELN	
I/O di livello Esteso	ASSIGN CLOSE DISCARD READSET READFN SEEK	

Tabella 17-1 Procedure e Funzioni di File System

PROCEDURE E FUNZIONI PRIMITIVE DI FILE SYSTEM

Questa sezione descrive le sette procedure e funzioni primitive di file system che gestiscono l'I/O su file al livello piu' basso. Vengono fornite descrizioni delle procedure READ e WRITE nei termini delle primitive GET e PUT. Vengono anche trattati la valutazione pigra (lazy) e l'I/O simultaneo. Nelle descrizioni che seguono, F e' un parametro file (i parametri file sono sempre parametri di riferimento), ed F^ rappresenta la relativa variabile buffer.

In ambiente segmentato, tutte le variabili file su cui operano queste procedure devono risiedere nel segmento dati di default. Questa limitazione aumenta l'efficienza delle chiamate di file system.

- GET e PUT

  Le procedure primitive GET e PUT leggono e scrivono nella variabile buffer F^. GET trasferisce in F^ il componente successivo del file; PUT effettua l'operazione inversa, cioe' scrive nel componente successivo del file il contenuto della variabile F^.

- RESET e REWRITE

  Le procedure RESET e REWRITE pongono all'inizio la posizione corrente del file. RESET prepara il file per successive operazioni di GET e READ. REWRITE prepara il file per successive operazioni di PUT e WRITE.

- EOF e EOLN

  Le funzioni EOF e EOLN vengono usate per controllare le condizioni di fine file e fine riga. Esse ritornano un risultato booleano. In generale questi valori indicano quando terminare la lettura di una riga o di un file.

- PAGE

  La procedura PAGE contribuisce alla formattazione di file-testo. Non e' una procedura necessaria come GET e PUT.

GET E PUT

Le procedure primitive GET e PUT vengono usate per leggere e scrivere nella variabile buffer F^. GET trasferisce in F^ il componente successivo del file; PUT effettua l'operazione inversa, cioe' scrive nel successivo componente del file il contenuto della variabile F^.

PROCEDURE GET (VAR F);

E' una procedura primitiva intrinseca di file system. Se esiste un componente successivo nel file F, allora:

- viene fatta avanzare la posizione corrente del file sul componente successivo

- il valore di questo componente viene trasferito nella variabile buffer F^

- EOF (F) diventa FALSE.

L'avanzamento e l'assegnazione possono essere ritardate internamente, a seconda del modo di accesso al file.

Se non esiste un componente successivo del file, allora EOF(F) diventa TRUE ed il valore di F^ e' indefinito. EOF (F) deve essere FALSE prima di GET (F), dato che la lettura oltre la fine del file provoca un errore runtime. Comunque, se F ha modo di accesso DIRECT, EOF (F) puo' essere TRUE o FALSE, dato che questo modo di accesso permette ripetute GET alla

fine del file. Se F^ e' un record con varianti, il compilatore legge la variante che ha dimensione maggiore.

PROCEDURE PUT (VAR F);

E' una procedura primitiva intrinseca di file system. Scrive il valore contenuto nella variabile buffer F^ nella posizione del file corrente e poi fa avanzare la posizione al componente successivo.

- Per file con modo di accesso SEQUENTIAL o TERMINAL, PUT e' permessa se l'operazione precedente su F e' stata una REWRITE, una PUT o una WRITE; non una RESET, una GET o un'altra procedura READ.

- Per file con modo di accesso DIRECT, PUT puo' essere effettuata dopo una RESET o una GET.

Le eccezioni a questa regola provocano segnalazione di errori. Il valore di F^ rimane sempre indefinito dopo una PUT.

Nel Pascal il valore di F^ dopo una PUT (F) puo' variare a seconda del sistema operativo e del tipo di file. Se F non ha modo di accesso DIRECT, EOF (F) deve essere TRUE prima di una PUT (F). EOF (F) e' sempre TRUE dopo una PUT (F). Se F^ e' un record con varianti, viene riportata la variante che ha dimensione maggiore.

RESET E REWRITE

Le procedure RESET e REWRITE vengono usate per riportare all'inizio la posizione corrente di un file. RESET viene usata per preparare successive operazioni di GET e READ. REWRITE viene usata per preparare successive operazioni di PUT e WRITE.

PROCEDURE RESET (VAR F);

E' una procedura primitiva intrinseca di file system. Riporta all'inizio la posizione corrente del file ed effettua una GET (F). Se il file non e' vuoto, viene letto il suo primo componente in F^ ed EOF (F) diventa FALSE. Se il file e' vuoto, il valore di F^ rimane indefinito ed EOF (F) diventa TRUE. RESET inizializza il file F prima della sua lettura. Per file DIRECT, la scrittura puo' essere fatta anche dopo una RESET.

Nel Pascal il modo in cui RESET chiude il file e lo riapre dipende dal sistema operativo. Viene emesso segnale di errore se non e' stato impostato correttamente il nome del file (come parametro del programma o mediante ASSIGN o READFN) oppure se il file non viene trovato dal sistema operativo. Se si verifica un errore durante una RESET, il file viene chiuso anche se e' stato aperto correttamente e se l'errore proviene dalla GET iniziale.

RESET (INPUT) e' effettuata automaticamente quando il programma viene inizializzato; essa e' permessa anche in modo esplicito. RESET su un file DIRECT permette sia la lettura che la scrittura, ma il file non viene creato automaticamente. La GET iniziale legge il record numero uno nei file DIRECT.

Si deve osservare che una GET (F) esplicita, effettuata immediatamente dopo una RESET (F), riporta il secondo componente del file nella variabile buffer. Comunque, una READ (F, X), dopo una RESET (F), trasferisce in X il primo componente di F poiche' READ (F, X) e' "X := F^; GET (F)".

PROCEDURE REWRITE (VAR F);

E' una procedura primitiva intrinseca di file system. Riporta all'inizio la posizione corrente del file. Il valore di F^ e' indefinito e EOF (F) diventa TRUE. Cio' e' necessario per inizializzare un file F prima della scrittura (per file DIRECT, puo' essere effettuata la lettura anche dopo una REWRITE).

Nel Pascal, il modo in cui la REWRITE chiude il file e lo riapre dipende dal sistema operativo. Se il file non esiste viene creato. Se esiste, il suo vecchio valore viene perso (a parte il caso DIRECT). Il nome del file deve essere stato impostato (come parametro di programma oppure con ASSIGN o READFN). Se si verifica un errore durante la REWRITE, il file viene chiuso. Se possibile, un file con lo stesso nome rimane inalterato quando si verifica un errore nella REWRITE; in alcuni sistemi operativi il file puo' essere pero' cancellato.

REWRITE (OUTPUT) e' fatta automaticamente quando un programma viene inizializzato; essa e' anche permessa in modo esplicito. REWRITE su un file DIRECT permette sia la lettura che la scrittura. REWRITE non fa una PUT iniziale in modo analogo alla GET di RESET.

EOF E EOLN

Le funzioni EOF e EOLN controllano rispettivamente le condizioni di fine file e fine riga. Esse ritornano un risultato booleano. In generale questi risultati indicano quando terminare la lettura di una linea o di un file.

FUNCTION EOF: BOOLEAN;
FUNCTION EOF (VAR F) : BOOLEAN;

E' una funzione primitiva intrinseca di file system. Indica se la variabile buffer F^ e' posizionata alla fine del file F per file SEQUENTIAL e TERMINAL. Quindi, se EOF (F) e' TRUE, allora il file e' appena stato scritto oppure l'ultima GET ha raggiunto la fine del file.

Per file DIRECT, se EOF (F) e' TRUE allora l'ultima operazione e' stata una WRITE (in questo caso il file puo' o non puo' essere posizionato alla fine) oppure l'ultima GET ha raggiunto la fine del file.

EOF senza parametro e' equivalente a EOF (INPUT). EOF (INPUT) non e' mai TRUE, tranne in quei sistemi operativi in cui un carattere particolare genera uno stato di fine file, oppure tranne se INPUT viene assegnato ad un altro file. La chiamata a EOF (F) permette di accedere alla variabile buffer F^.

FUNCTION EOLN : BOOLEAN;
FUNCTION EOLN (VAR F) : BOOLEAN;

E' una funzione primitiva intrinseca di file system. Indica se la posizione corrente del file e' alla fine di una riga nel file-testo F dopo una GET (F). Il file deve avere struttura ASCII.

Secondo lo standard ISO, richiamare EOLN (F) quando EOF (F) e' TRUE e' un errore. Nel Pascal questo errore, nella maggior parte dei casi, viene rilevato. Il file F deve essere di tipo TEXT.

Se EOLN (F) e' TRUE, il valore di F^ e' un carattere di spazio, ma la posizione del file e' alla fine della riga. EOLN senza parametro e' equivalente a EOLN (INPUT). La chiamata a EOLN (F) permette di accedere alla variabile buffer F^.

PAGE

La procedura PAGE aiuta a formattare i file-testo. Non e' una procedura "necessaria", come lo sono la GET e la PUT.

PROCEDURE PAGE;
PROCEDURE PAGE (VAR F);

E' una procedura primitiva intrinseca di file system. Provoca il posizionamento all'inizio di una nuova pagina quando il file-testo viene stampato. Dato che PAGE scrive nel file, devono essere soddisfatte le condizioni iniziali descritte per PUT. Il file deve avere struttura ASCII. PAGE senza parametro e' equivalente a PAGE (OUTPUT).

Se F non e' posizionato all'inizio di una linea, PAGE (F) scrive dapprima un carattere di marca di riga in F. Se F e' SEQUENTIAL o DIRECT, allora PAGE (F) scrive un carattere di avanzamento carta CHR (12). Se F e' TERMINAL l'effetto e' determinato dal sistema operativo, che in genere permette l'avanzamento carta da terminale.

VALUTAZIONE PIGRA

La valutazione pigra ("lazy") e' stata progettata per risolvere un problema ricorrente nel Pascal, cioe' per leggere da un terminale in modo naturale.

Il problema fondamentale e' che lo standard ISO definisce la procedura RESET con una GET iniziale. Anche se e' accettabile in ambiente di file sequenziali, questo tipo di pre-lettura non funziona per l'I/O interattivo.

La valutazione lazy nel Pascal permette di rinviare l'attuale input fisico (solo per file-testo) quando viene valutata una variabile buffer.

Ad esempio, se in un file normale viene eseguita RESET e poi READ, la

procedura RESET chiama la procedura GET, la quale trasferisce nella variabile buffer il primo elemento del file. Nel caso che il file sia un terminale, questo primo elemento non esiste ancora.

Occorre quindi, da terminale, impostare un carattere per poter soddisfare la GET. Solo allora puo' iniziare la fase di input.

La valutazione lazy elimina questo problema per i file-testo, assegnando alla variabile buffer un valore speciale di stato che lo indica "pieno" o "vuoto".

La condizione normale dopo una GET (F) e' di "vuoto". Lo stato indica "pieno" soltanto dopo che la variable e' stata assegnata sia in input che in output. "Pieno" implica che il contenuto della variabile corrisponde all'elemento corrente del file. "Vuoto" implica esattamente l'opposto, cioe' che il contenuto della variabile buffer non corrisponde all'elemento corrente del file e che l'input e' stato rinviato. La tabella 17-2 contiene queste regole.

Istruzione	Stato alla chiamata	Azione intrapresa	Stato all'uscita
GET (F)	Pieno	Punta al componente successivo. Diventa vuoto perche' il valore puntato non risiede nella variabile buffer	Vuoto
GET (F)	Vuoto	Trasferisce nella variabile buffer il contenuto dell'elemento corrente del file. Posiziona il file sul successivo elemento. Diventa vuoto perche' il valore puntato non risiede nella variabile buffer	Vuoto
Riferimento a F^	Pieno	Non viene intrapresa alcuna azione	Pieno
Riferimento a F^	Vuoto	Trasferisce nella variabile buffer il contenuto dell'elemento corrente del file	Vuoto

Tabella 17-2 Valutazione Lazy

Si deve osservare che RESET (F) pone prima a pieno lo stato e quindi chiama la GET per porre a vuoto lo stato senza effettuare alcun input.

Esempio di valutazione lazy con chiamata automatica a REWR1TE:

```
{1NPUT e' automaticamente un file-testo.}
{RESET (1NPUT); fatta automaticamente.}
WR1TE (OUTPUT, "Enter number: ");
READLN (1NPUT, FOO);
```

La chiamata automatica iniziale della procedura RESET richiama la procedura GET, la quale cambia lo stato della variabile da pieno a vuoto. La prima azione fisica su terminale e' l'output della WR1TE. READLN effettua una serie delle operazioni seguenti:

```
temp := 1NPUT^;
GET (1NPUT)
```

L'input fisico avviene quando ogni 1NPUT^ viene caricato e la procedura GET riporta a vuoto lo stato della variabile buffer.
READLN termina con la sequenza:

```
WHILE NOT EOLN DO GET (1NPUT);
GET (INPUT)
```

Questa operazione salta i caratteri iniziali e quello di marca di riga. La funzione EOLN richiama l'input fisico. Impostare il carattere di ritorno carrello significa porre a TRUE la condizione EOLN. Sia la procedura GET nel ciclo di WH1LE che l'ultima GET riportano lo stato a vuoto. L'ultimo input fisico e' la lettura del carattere di ritorno carrello.

I/O SIMULTANEO

Nei sistemi operativi che lo permettono, l'I/O simultaneo consente ad una GET o ad una PUT di iniziare la fase di I/O e di ritornare immediatamente al programma che ha effettuato la chiamata. Si usa soltanto per file con struttura BINARY.

Il programma puo' effettuare calcoli mentre la variabile buffer viene riempita o trasferita. La variabile buffer ha un'altro valore di stato associato, che la definisce "libera" o "occupata". Se la variabile e' occupata quando ne viene richiesto l'accesso, il programma deve attendere finche' lo stato non la indica "libera".

Ad esempio, la seguente parte di programma legge il file IN_FILE, esegue alcune operazioni con il valore corrente, e quindi lo riporta nel file OUT_FILE:

```
WHILE NOT EOF (IN_FILE) DO
   {Controlla la fine dell'input e aspetta}
   {finche' IN_FILE^ non e' libera.}

BEGIN
   READ (IN_FILE, BUFF);
   {IN_FILE^ e' libera, e viene quindi assegnata a BUFF;}
   {comincia la lettura del componente successivo.}

   OPERATE (BUFF);
   {Si effettuano operazioni durante la READ e la WRITE.}

   WRITE (OUT_FILE, BUFF);
   {Attende finche' OUT_FILE^ non e' libera,}
   {assegna quindi BUFF e comincia la scrittura.}

END
```

L'esempio precedente si avvale delle procedure READ e WRITE. Si deve osservare che le due righe seguenti sono equivalenti:

```
READ (IN_FILE, BUFF)
BUFF := IN_FILE^; GET (IN_FILE)
```

E anche le seguenti:

```
WRITE (OUT_FILE, BUFF)
OUT_FILE^ := BUFF; PUT (OUT_FILE)
```

L'I/O simultaneo viene applicato alle procedure GET e PUT, e anche a quelle READ e WRITE. Nella pratica non e' una cosa usuale, per il sistema runtime del Pascal, gestire la concorrenza. Per ulteriori dettagli sull'implementazione si puo' vedere l'Appendice A, "Caratteristiche di Implementazione", nel manuale "Linguaggio Pascal Guida Utente".

Quando si accede alla variabile buffer, sia per la valutazione lazy che per l'I/O simultaneo, il Pascal genera una chiamata di I/O a sistema. Comunque, se la variabile buffer e' passata come parametro di riferimento ad una procedura o ad una funzione, la procedura o funzione stessa puo' fare I/O sul file relativo senza eseguire queste chiamate speciali.

Passare una variabile buffer come parametro di riferimento e' un errore in Pascal, anche se viene emesso soltanto un segnale di avvertimento. GET e PUT, quando vengono usate su una variabile buffer passata come parametro di riferimento, hanno effetti indefiniti. Anche l'assegnazione dell'indirizzo di una variabile buffer ad una variabile tipo e' molto pericolosa, poiche' non fa uso dei meccanismi di valutazione lazy e di I/O simultaneo.

INPUT E OUTPUT CON FILE-TESTO

Nel Pascal Standard l'input e l'output leggibili dall'utente vengono fatti mediante file-testo. I file-testo sono file di tipo TEXT ed hanno sempre struttura ASCII. Normalmente i file-testo standard INPUT e OUTPUT vengono forniti come parametri standard nella intestazione del programma:

```
PROGRAM IN_AND_OUT (INPUT, OUTPUT);
```

Altri file-testo rappresentano generalmente alcuni dispositivi di I/O come il terminale, il lettore di scheda, una stampante oppure un file di disco del sistema operativo. Il livello Esteso permette l'uso di file addizionali non passati come parametri di programma.

Allo scopo di facilitare la gestione dei file-testo vengono fornite, oltre alle procedure GET e PUT, le quattro funzioni standard READ, READLN, WRITE e WRITELN.

- READ and READLN

  Le procedure READ e READLN leggono dati da file-testo. READ e READLN sono definite in termini della procedura piu' primitiva GET. La procedura READLN e' molto simile alla READ, a parte il fatto che legge anche il carattere di fine riga.

- WRITE e WRITELN

  Le procedure WRITE e WRITELN scrivono dati nei file-testo. WRITE e WRITELN sono definite nei termini dell'operazione piu' primitiva PUT. La procedura WRITELN scrive un carattere di marca di riga alla fine della linea. Sotto qualsiasi altro aspetto, WRITELN e' analoga alla WRITE.

Queste procedure hanno una sintassi piu' flessibile per quanto riguarda le liste di parametri permettendone, tra le altre cose, un numero variabile. Non tutti i parametri devono essere di tipo CHAR, ma possono essere anche di altri tipi, nel qual caso il trasferimento dati avviene insieme ad un'operazione implicita di conversione dati. In alcuni casi, i parametri possono includere valori addizionali di formattazione che influiscono sulla conversione dei dati.

Se la prima variabile e' una variabile file, allora e' il file ad essere letto o scritto. Negli altri casi vengono considerati automaticamente i file standard INPUT e OUTPUT rispettivamente per la lettura e scrittura.

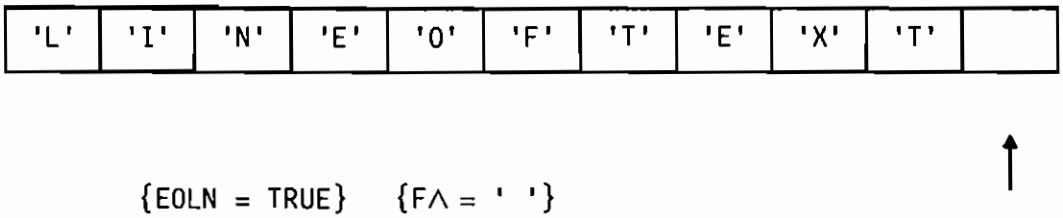
Questi due file hanno modo di accesso TERMINAL e struttura ASCII. Essi vengono predichiarati come:

```
VAR INPUT, OUTPUT:  TEXT;
```

Nel Pascal i file INPUT e OUTPUT sono trattati come tutti gli altri file-testo. Essi possono essere usati con ASSIGN, CLOSE, RESET, REWRITE e le altre procedure e funzioni. Comunque, anche se presenti come parametri di un programma, essi non vengono associati inizialmente ad alcun nome di file. Essi vengono invece assegnati al terminale utente.

RESET di INPUT e REWRITE di OUTPUT vengono fatte automaticamente, indipendentemente dalla loro presenza come parametri di programma.

I file-testo rappresentano un caso speciale tra tutti i tipi di file, nel senso che sono strutturati in righe tramite "marche di riga". Se, durante la lettura, la posizione del file viene fatta avanzare fino ad un carattere di marca di riga (il carattere seguente l'ultimo della riga), allora il valore della variabile buffer F∧ diventa un carattere spazio e la funzione standard EOLN (F) ritorna il valore TRUE.
Ad esempio:



L'avanzamento della posizione del file di un altro carattere puo' provocare una delle seguenti tre cose:

1. Se viene raggiunta la fine del file, allora EOF (F) diventa TRUE.

2. Se la linea successiva e' vuota, a F∧ viene assegnato uno spazio vuoto e EOLN (F) rimane TRUE.

3. Altrimenti, il carattere successivo viene trasferito in F∧ ed EOLN (F) diventa FALSE.

Dato che nel Pascal Standard le marche di riga non sono elementi CHAR, in teoria esse possono essere generate soltanto dalla procedura WRITELN. E' comunque possibile usare un altro carattere come marca di riga. E' quindi possibile fare la WRITE di una marca di riga, ma non la READ.

Quando un file-testo in creazione viene chiuso, all'ultima riga viene automaticamente unito un carattere di marca di riga; questo non avviene se il file e' vuoto oppure se l'ultimo carattere e' gia' una marca di riga.

Quando, in fase di lettura, viene raggiunta la fine del file, viene ritornata una marca di riga per l'ultima linea, anche se essa non era contenuta nel file.

Qualsiasi lista di dati scritta mediante WRITELN e' generalmente leggibile, con la stessa lista, da una READLN (non si deve pero' avere una LSTRING alla fine della lista).

Le domande e le risposte interattive sono molto facili nel Pascal. Per avere l'input sulla stessa linea della risposta occorre usare la WRITE per il prompt. READLN deve essere sempre usata per la risposta.

Ad esempio:

```
WRITE ('Enter command: ');
READLN (risposta);
```

Se non viene fornito alcun file, la maggior parte delle procedure di file-testo assumono il file INPUT oppure OUTPUT. Ad esempio, se 1 e' di tipo INTEGER, allora READ (1) equivale a READ (INPUT, I).

READ E READLN

PROCEDURE READ
PROCEDURE READLN

Sono procedure intrinseche del file system per I/O su file-testo. READ e READLN leggono dati da file-testo. Entrambe vengono definite nei termini della procedura piu' primitiva GET. Cioe', se P e' di tipo CHAR, allora READ (F, P) equivale a:

```
BEGIN
   P := F^
   {Assegna la variabile buffer a P.}
   GET (F)
   {Trasferisce in F^ il componente successivo di F.}
END
```

READ puo' avere piu' di un singolo parametro, come in READ (F, P1, P2, ..., Pn). Questo equivale a:

```
BEGIN
   READ (F, P1);
   READ (F, P2);
     .
     .
   READ (F, PN)
END
```

La procedura READLN e' molto simile alla READ, con la differenza che legge anche il carattere di fine riga. Al livello primitivo di GET, senza parametri, la READLN equivale a:

```
BEGIN
   WHILE NOT EOLN (F) DO GET (F);
   GET (F)
END
```

Una READLN con parametri, come in READLN (F, P1, P2,...,Pn), e' equivalente a:

```
BEGIN
   READ (F, P1, P2,...,Pn);
   READLN (F)
END
```

READLN viene spesso usata per saltare all'inizio della riga successiva. Essa puo' essere usata soltanto con file-testo (modo ASCII).

Se non viene specificato alcun nome di file, sia la READ che la READLN leggono dal file INPUT standard. Quindi non occorre specificare esplicitamente il nome INPUT. Ad esempio, le seguenti due READ hanno il medesimo effetto:

```
READ (P1, P2, P3)
{Assume INPUT in modo implicito}
READ (INPUT, P1, P2, P3)
```

A livello Standard, i parametri P1, P2, P3 devono essere di uno dei seguenti due tipi:

```
CHAR
INTEGER
REAL
```

A livello Esteso la READ accetta anche parametri dei seguenti tipi:

```
WORD
tipo enumerato
BOOLEAN
INTEGER4
tipo puntatore
STRING
LSTRING
```

Quando il compilatore legge una variabile di tipo subrange, il valore letto deve essere compreso nel campo di variabilita'. In caso contrario viene emesso un segnale di errore, indipendentemente dalla presenza o meno del relativo indicatore di controllo.

La procedura READ puo' leggere anche da un file che non sia un file-testo (cioe' BINARY). La forma READ (F, P1, P2,...,Pn) puo' essere usata su un file BINARY. Comunque questa procedura non funziona nel modo previsto dopo una SEEK su un file DIRECT.
Per file BINARY, READ (F, X), e' equivalente a:

```
BEGIN
   X := F^;
   GET (F)
END
```

FORMATI READ

Il processo di READ per tipi formattati (tutti i tipi eccetto CHAR, STRING e LSTRING) provvede prima a trasferire i dati in una LSTRING interna e quindi a decodificare la stringa per ottenerne il valore.

Vi sono tre punti importanti per quanto riguarda la READ formattata:

1. Gli spazi iniziali, le tabulazioni, i caratteri di avanzamento carta e di marca di riga non vengono considerati. Ad esempio, quando si effettua una READLN (1, J, K) ove I, J e K sono INTEGER, essi possono essere sulla stessa riga oppure su righe diverse.

2. I caratteri vengono letti se appartengono all'insieme valido per il tipo specificato. Ad esempio, "-1-2-3" viene letta come stringa di caratteri per un singolo INTEGER, ma segnala un errore quando la stringa di caratteri e' decodificata. Questo significa che gli elementi devono essere separati da spazi, tabulazioni, marche di riga oppure da caratteri non permessi nel formato.

3. I valori M e N nella READ vengono ignorati, eccetto per un valore N appartenente ad un tipo enumerato. In letture BINARY non sono accettati parametri M e N.

La maggior parte delle regole di formattazione valgono anche per la funzione DECODE.

- Tipi INTEGER e WORD

  Se P e' di tipo INTEGER, WORD, oppure un loro subrange, allora READ (F, P) implica la lettura di una sequenza di caratteri da F in modo da formare un numero in accordo alla sintassi normale del Pascal, e quindi l'assegnazione del numero a P. La notazione non decimale (16#C007, 8#74, 10#19, 2#101, #Face) viene accettata sia per INTEGER che per WORD, con una radice da 2 a 36. Se P e' di tipo INTEGER, un carattere '+' o '-' in testa viene accettato. Se P e' di tipo WORD, allora vengono accettati numeri con valore fino a MAXWORD (32768...65535).

- Tipi REAL e INTEGER4

  Se P e' di tipo REAL oppure, al livello Esteso, INTEGER4, READ (F, P) implica la lettura di una sequenza di caratteri da F in modo da formare un numero del tipo appropriato per poi assegnarlo alla variabile P. La notazione non decimale non viene accettata per numeri REAL, ma viene accettata per numeri INTEGER4. Quando viene letto un valore REAL, viene accettato un numero con separatore decimale sia in testa che in fondo: ma questa forma provoca l'emissione di un messaggio di avvertimento se usata come costante in un programma.

- Tipi Enumerati e Booleani

  A livello Esteso, se P e' un tipo enumerato o booleano, viene letto un numero come subrange di WORD e viene assegnato a P un valore tale che il numero sia l'ORD del valore di tipo enumerato. Inoltre, se P e' di tipo BOOLEAN, la lettura di una delle sequenze di caratteri 'TRUE' o 'FALSE' provoca l'assegnazione dei valori rispettivamente vero e falso a P. Il numero letto deve essere nel campo di variabilita' dei valori ORD della variabile.

  Sempre a livello Esteso, se il parametro P e' un tipo enumerato ed include la notazione :N come in READ (P::N), vengono letti caratteri dal file F in modo da formare un identificatore o un numero valido. Se i caratteri formano un numero, esso deve essere il valore ORD descritto nel precedente paragrafo; se i caratteri formano un identificatore che corrisponde ad un identificatore costante del tipo enumerato, allora il suo valore viene assegnato a P. Se la variabile e' BOOLEAN, allora la lettura delle cifre 1 o 0 provoca

l'assegnazione dei valori rispettivamente vero o falso alla variabile booleana. 'TRUE' e 'FALSE' vengono anche accettati come identificatori costanti di BOOLEAN.

Il valore attuale di N viene ignorato: l'uso della notazione N avverte il compilatore che deve memorizzare l'identificatore di tipo enumerato e renderlo disponibile alla routine READ. L'omissione della notazione N permette di risparmiare lo spazio di memoria che verrebbe usato per l'identificatore.

- Tipi di riferimento

Al livello Esteso, se P e' un tipo puntatore, un numero viene letto come WORD e quindi assegnato a P, in un modo che dipende dalla singola implementazione; in questo modo un puntatore puo' essere scritto e poi riletto ottenendo un riferimento alla stessa locazione. I tipi indirizzo dovrebbero essere letti come WORD usando la notazione .R o .S.

- Tipi stringa

Al livello Esteso, se P e' una STRING (n), allora vengono letti i successivi n caratteri in P. Marche di riga iniziali, spazi, caratteri di tabulazione o di avanzamento carta non vengono ignorati. Se viene incontrato un carattere di marca di riga prima di n caratteri, i caratteri seguenti in P vengono riempiti con spazi e la posizione del file rimane sulla marca di riga.

Se la STRING viene riempita con n caratteri prima che venga incontrato il carattere di marca di riga, la posizione nel file viene posta sul carattere seguente. In alcune implementazioni vi puo' essere un limite di 255 caratteri sulla lunghezza di una STRING letta. P puo' essere di tipo super array STRING (cioe' un parametro di riferimento oppure una variabile riferita da puntatore).

A livello Esteso, se P e' una LSTRING (n), vengono letti i successivi n caratteri in P e la lunghezza di LSTRING e' posta ad "n". Marche di riga iniziali, spazi, caratteri di tabulazione o di avanzamento carta vengono ignorati. Se viene incontrato un carattere di marca di riga prima di n caratteri letti, la lunghezza di LSTRING viene stabilita uguale al numero di caratteri letti e la posizione del file rimane sulla marca di riga.

Se LSTRING viene riempita con n caratteri prima del carattere di marca di riga, allora la posizione del file rimane sul carattere seguente. P puo' essere di tipo super array LSTRING (cioe' un parametro di riferimento oppure una variabile riferita da puntatore). READ (LSTRING) e' utile quando si leggono righe intere da un file-testo, in particolare quando si vuole conoscere la lunghezza della riga letta. Ad esempio, il modo piu' facile per copiare un file-testo e' quella di usare READLN e WRITELN con una variabile LSTRING.

Generalmente, READ e READLN non usano parametri con ampiezza di campo M: non si puo' leggere la stringa '123456' come due numeri INTEGER mediante una READ (I:3, J:3). Comunque e' possibile leggere due

elementi LSTRING (3) e quindi decodificarli per ottenere lo stesso effetto.

WRITE E WRITELN

PROCEDURE WRITE
PROCEDURE WRITELN

Sono procedure intrinseche di file system (I/O su file-testo). Scrivono dati su file-testo. WRITE e WRITELN sono definite nei termini dell'operazione piu' primitiva PUT; cioe' se P e' un'espressione di tipo CHAR ed F e' un file di tipo TEXT, allora WRITE (F, P) e' equivalente a:

```
BEGIN
   F^ := P;
   {Assegnazione di P alla variabile buffer F^}
   PUT (F)
   {Assegnazione di F^ al componente successivo del file}
END
```

WRITE puo' avere piu' di un parametro, come in WRITE (F, P1, P2,...,Pn). Cioe':

```
BEGIN
   WRITE (F, P1);
   WRITE (F, P2);
     .
     .
   WRITE (F, Pn)
END
```

La procedura WRITELN scrive un carattere di marca di riga alla fine della linea emessa. Sotto tutti gli altri punti di vista, WRITELN e' del tutto equivalente a WRITE. Quindi WRITELN (F, P1, P2,...,Pn) equivale a:

```
BEGIN
   WRITE (P1, P2,...,Pn);
   WRITELN (F)
END
```

Se WRITE o WRITELN non hanno parametri file, allora viene assunto il parametro file OUTPUT. Le seguenti coppie di istruzioni sono equivalenti:

```
WRITE (P1, P2,...,Pn)
WRITE (OUTPUT, P1, P2,...,Pn)

WRITELN (P1, P2,...,Pn)
WRITELN (OUTPUT, P1, P2,...,Pn)
```

Al livello Standard, i parametri in WRITE possono essere espressioni di uno dei seguenti tipi:

```
CHAR
INTEGER
REAL
BOOLEAN
STRING
```

Al livello Esteso, le espressioni possono essere di tipo:

```
WORD
INTEGER4
LSTRING
tipo enumerato
tipo puntatore
```

I parametri possono avere valori opzionali M e N (per ulteriori informazioni sui parametri M e N si puo' vedere "Formati WRITE", piu' oltre in questo capitolo).

Anche se la procedura WRITE puo' scrivere su file BINARY (cioe' non su file-testo), questa non e' un'operazione consigliata per file DIRECT dopo un'operazione di SEEK, dato che la READ complementare puo' non funzionare nel modo voluto.

Per i file BINARY, la WRITE (F, X) e' equivalente a:

```
BEGIN
   F := X;
   PUT (F)
END
```

E' accettata anche la forma WRITE (F, P1, P2,...,Pn). Normalmente le WRITE su file BINARY non accettano valori M e N.

FORMATI WRITE

Nei file-testo i parametri per WRITE e WRITELN possono assumere una delle seguenti forme:

```
P     P:M     P:M:N     P::N
```

I valori M e N possono essere considerati parametri valore di tipo INTEGER e vengono usati per formattare in diversi modi. II livello Esteso permette valori M ed N sia per le READ che per le WRITE e permette anche di fornire N senza M, come in:

```
P::N
```

Il loro uso non-standard e' un errore che pero' non viene rilevato al livello Standard. In alcuni viene usato solo M, N, o nessuno dei due; i valori M ed N, non usati, vengono ignorati.

L'omissione di M o N equivale all'uso di MAXINT. Ad esempio, WRITE (12 : MAXINT) usa il valore M di default (B in questo caso). Generalmente, i valori M e N non sono accettati per file BINARY. In WRITE, il valore M e'

l'ampiezza di campo che indica il numero di caratteri da scrivere. Nel Pascal-1SO M deve essere maggiore di zero e, se l'espressione richiede meno di M caratteri, deve essere riempita di caratteri di spazio a sinistra.

A livello Esteso, M puo' anche essere zero o negativo. Se e' negativo viene usato il suo valore assoluto, ma la riempitura con spazi inizia a destra invece che a sinistra. Se e' zero, non vengono scritti caratteri. Questi sono errori dello standard 1SO che pero' non vengono segnalati.

Se la rappresentazione dell'espressione non sta in ABS (M) caratteri, vengono usate posizioni supplementari per tipi numerici, oppure il valore e' troncato verso destra per tipi stringa. Se M viene omesso oppure e' uguale a MAX1NT, viene usato un valore di default.

II valore N significa:

- il numero di cifre decimali, se P e' di tipo REAL
- la radice output, se P e' INTEGER, WORD, INTEGER4 o puntatore
- il valore numerico o l'identificatore se P e' un tipo enumerato

La maggior parte delle seguenti regole di formattazione valgono anche per la procedura ENCODE.

- Tipi INTEGER e WORD

  Se P e' di tipo INTEGER, WORD, o un loro subrange, allora la rappresentazione decimale di P viene scritta nel file. Se P e' un 1NTEGER negativo, allora viene sempre emesso un carattere '-' in testa. I valori WORD non sono mai negativi. Per valori INTEGER e WORD, il valore di M e' 8.

  Se ABS (M) e' piu' piccolo della rappresentazione del numero, allora vengono usate posizioni addizionali. N viene usato per scrivere in esadecimale, decimale, ottale, binario oppure in qualche altra base ponendo N uguale ad un numero che va da 2 a 36; questa e' un'estensione dello standard ISO Pascal. Se N non e' 10 (oppure e' omesso, oppure e' MAXINT), allora la riempitura alla sinistra avviene con zeri invece che con spazi. Omettere N, o porre N uguale a MAXINT oppure a 10, implica una radice decimale.

  I numeri decimali WORD da 32768 a 65535 vengono scritti normalmente e non nel loro equivalente negativo. Tutti i valori scritti devono essere separati da spazi o da qualche altro carattere non numerico in modo da poterli leggere come numeri separati.

- Tipi REAL e 1NTEGER4

  Se P e' di tipo REAL, viene scritta nel file una rappresentazione decimale del numero P arrotondata al numero specificato di cifre decimali. Se N viene omesso oppure stabilito uguale a MAXINT, viene scritta una rappresentazione a virgola mobile di P nel file; questa rappresentazione e' costituita da un coefficiente e da un fattore di

scala. Se viene specificato N, viene scritta nel file una rappresentazione a virgola fissa con N dopo il punto decimale. Se N e' zero, P viene scritto come intero arrotondato con il punto decimale.
Il valore di default di M per valori REAL e' 14.

Alcuni esempi di operazioni WRITE su valori REAL:

Questa Istruzione	Produce questo Output
WRITE (123.456)	' 1.2345600E+02'
WRITE (123.456:20)	' 1.2345600000000E+02'
WRITE (123.456::3)	' 123.456'
WRITE (123.456:2:3)	' 123.456'
WRITE (123.456:-20:3)	'123.456 '

Al livello Esteso, se P e' di tipo INTEGER4, allora la rappresentazione decimale di P viene scritta nel file. Il valore N viene usato per impostare la radice, come nel tipo INTEGER. Il valore di default di M e' 14.

- Tipi Enumerati e Booleani

Al livello Esteso, se P e' un tipo enumerato ed N e' omesso oppure stabilito uguale a MAXINT, allora ORD (P) viene scritto nel file come una WORD. Se viene dato il valore 1 ad N, l'identificatore costante del tipo enumerato di P viene scritto nel file, come una STRING. Si deve osservare che l'uso della notazione N provoca l'allocazione in memoria dell'identificatore costante del tipo enumerato.

Al livello Standard, se P e' di tipo BOOLEAN, allora le stringhe 'TRUE' e 'FALSE' vengono scritte nel file come STRING. Non viene mai scritto il valore ORD per tipi booleani come avviene per i tipi enumerati (anche se si puo' scrivere WRITE (ORD (P)).

- Tipi di riferimento

Al livello Esteso, se P e' un tipo puntatore, P viene scritta come WORD. Questo viene fatto nell'implementazione allo scopo di poter memorizzare un puntatore e poterlo leggere ed usare in seguito. I tipi indirizzo dovrebbero essere scritti come valori WORD con la notazione .R o .S.

- Tipi stringa

Se P e' di tipo STRING (n), il valore di P viene riportato nel file. Il valore di default di M e' la lunghezza della stringa, "n". Se ABS (M) e' minore della lunghezza della stringa, vengono scritti soltanto i primi ABS (M) caratteri. Se M e' zero, non viene scritto nulla. La parte destra della stringa viene sempre troncata, anche se M e' negativo. In alcune implementazioni puo' esservi un limite di 255 caratteri sulla lunghezza della stringa scritta.

Al livello Esteso, se P e' di tipo LSTRING (n) il valore di P viene scritto nel file. Il valore di default di M e' la lunghezza della

stringa, P.LEN. Se ABS (M) e' inferiore alla lunghezza della stringa, vengono scritti soltanto i primi ABS (M) caratteri. Se M e' zero, non viene scritto nulla. La porzione destra di LSTRING viene sempre troncata, anche se M e' negativo. Se ABS (M) e' maggiore della lunghezza corrente, allora le posizioni che rimangono vengono riempite da spazi. Si deve osservare che una stringa di spazi M puo' essere scritta come NULL:M.

I/O AL LIVELLO ESTESO

Al livello Esteso, il Pascal ha queste caratteristiche I/O addizionali:

- si puo' accedere ai tre campi FCB: F.MODE, F.TRAP, F.ERRS
- un certo numero di procedure e' predichiarato
- i file temporanei sono disponibili.

La sezione "I/O al Livello Esteso", nel Capitolo 10, contiene una descrizione dei campi FCB nel contesto dei file. Le procedure addizionali ed i file temporanei vengono descritte nel seguito.

PROCEDURE DI LIVELLO ESTESO

PROCEDURE ASSIGN (VAR F; CONSTS N : STRING);

E' una procedura di file system (I/O di livello Esteso). Assegna il nome di file di sistema operativo contenuto in STRING (o LSTRING) a un file F. Il formato del nome del file dipende dal sistema operativo. Come regola, ASSIGN non tiene conto degli spazi iniziali. ASSIGN non tiene conto neppure di tutti i nomi di file stabiliti in precedenza. Un nome di file deve essere stabilito prima della prima RESET O REWRITE su un file. ASSIGN su un file aperto (dopo RESET o REWRITE ma prima della CLOSE) produce un errore. E' ammessa l'ASSIGN sui file INPUT e OUTPUT; dato che questi file vengono aperti automaticamente, essi devono essere chiusi prima dell'ASSIGN.

PROCEDURE CLOSE (VAR F);

E' una procedura di file system (I/O di livello Esteso). Effettua la chiusura del file assicurando che l'accesso al file sia terminato correttamente. Questo e' molto importante per variabili file allocate nello stack o nello heap. Dato che questi file devono essere chiusi prima della RETURN o della DISPOSE, essi vengono chiusi automaticamente quando la RETURN o la DISPOSE liberano lo spazio nello stack o nello heap.

Anche le variabili file con l'attributo STATIC in procedure o funzioni vengono chiuse automaticamente all'uscita della procedura o della

funzione. I file allocati in modo statico a livello di programma, modulo o implementazione vengono chiusi automaticamente quando termina l'intero programma.

Se necessario, quando viene eseguita la CLOSE, i buffer di sistema operativo di un file in fase di scrittura vengono svuotati. Comunque la variabile buffer Pascal non e' PUT. Se un file-testo e' in fase di scrittura quando viene chiuso, e l'ultima riga non-vuota non termina con un carattere di marca di riga, allora ne viene aggiunta una. Se il file e' SEQUENTIAL allora viene aggiunto, se manca, un fine riga.

Si deve osservare che alcuni errori runtime possono togliere il controllo al sistema runtime del Pascal. In questi casi, i file in fase di scrittura non possono essere chiusi e le informazioni al loro interno vengono perse. Una CLOSE su un file gia' chiuso o mai aperto (nessuna RESET ne' REWRITE) e' permessa. CLOSE non viene ignorata se l'indicatore di errore e' attivo e se c'e' stato un errore precedente. CLOSE spegne l'indicatore di errore per il file e, se non si sono verificati errori, inizializza lo stato di errore.

PROCEDURE DISCARD (VAR F);

E' una procedura di file system (I/O di livello Esteso). Chiude e cancella un file aperto. DISCARD e' simile alla CLOSE, con la differenza che il file viene cancellato.

PROCEDURE READFN (VAR F: P1, P2,...,PN);

E' una procedura di file system (I/O di livello Esteso). READFN e' analoga alla READ (non READLN) con due eccezioni:

1. il parametro file F deve essere presente (viene assunto INPUT ma viene anche emessa una segnalazione).

2. se il parametro P e' di tipo FILE, viene letta da F una sequenza di caratteri in modo da formare un nome valido di file ed assegnarlo a P nello stesso modo di ASSIGN.

I parametri di altri tipi vengono letti allo stesso modo della procedura READ.

Si deve osservare che READFN e' analoga alla READ (non READLN) e non legge il carattere di marca di fine riga. Se il primo parametro in READFN e' un file di tipo qualsiasi, esso viene trattato come un file-testo dal quale vengono letti caratteri. Non viene assunto il nome INPUT per default.

READFN viene usata internamente per leggere i parametri di un programma. Essa e' utile per leggere il nome di un file ed assegnarlo a qualche altro file in una operazione.

PROCEDURE READSET (VAR F; VAR L: LSTRING, CONST S: SETOFCHAR);

E' una procedura di file system (I/O di livello Esteso). READSET legge caratteri e li trasferisce in L, finche' essi appartengono al set S e

finche' c'e' spazio in L. Se non viene fornito il parametro F viene assunto, in modo analogo a READ e WRITE, il valore INPUT. Gli spazi iniziali, i caratteri di tabulazione, di avanzamento carta e di fine riga vengono sempre ignorati.

La lettura termina al primo carattere di marca di riga, il quale non e' mai di tipo CHAR.

READSET, come ENCODE, viene usata dal sistema runtime per implementare le procedure READ formattate e per leggere nomi di file con READFN. E' utile nella lettura e nel riconoscimento caratteri, e per la gestione di input semplice.

In ambiente di memoria segmentata, i parametri L e S devono risiedere nel segmento dati di default.

PROCEDURE SEEK (VAR F; N: INTEGER4);

E' una procedure di file system (I/O di livello Esteso). A differenza dei file sequenziali normali, i file DIRECT hanno una struttura ad accesso casuale. SEEK viene usata per accedere direttamente agli elementi di tali file. Per usare un file DIRECT, il MODE deve essere impostato a DIRECT prima che il file venga aperto con RESET o REWRITE; il file, F, deve essere un file con modo di accesso DIRECT.

Se il file viene letto o scritto sequenzialmente, possono essere usate le normali procedure READ e WRITE. SEEK modifica un campo nel file F in modo che la GET o PUT successiva operi sul record numero N. Il parametro N (numero del record) puo' essere di tipo INTEGER, WORD oppure INTEGER4. Per i file-testo (struttura ASCII) i record sono righe; per altri file (struttura BINARY) i record sono i componenti del file. I record vengono numerati a partire da uno (non zero). Se F e' un file ASCII, SEEK pone lo stato di valutazione lazy a "vuoto". Se F e' un file BINARY, SEEK attende la terminazione dell'eventuale I/O e pone a "pronto" lo stato di I/O simultaneo.

SEEK viene meglio illustrata da alcuni esempi. Consideriamo ad esempio la struttura BINARY, modo di accesso DIRECT, con il seguente contenuto CHAR:

	'A'	'B'	'C'	'D'	'E'	'F'	'G'	
N =	1	2	3	4	5	6	7	8

Viene fatta una SEEK 1 implicita dopo una RESET o REWRITE. Quindi, per file DIRECT, devono essere fornite le seguenti sequenze di comandi:

```
RESET (F);
{SEEK 1 iniziale, seguita da GET;}
{F^ contiene ora 'A'.}
SEEK (F, 5);
{Posizione del file a 5; F^ contiene ancora 'A'}
C := F^
{C ora vale 'A'; C non contiene 'E'}
```

Si deve osservare che C non contiene il quinto elemento di F, come si potrebbe supporre. Per ottenere questo valore occorre eseguire la seguente sequenza di comandi:

```
RESET (F);
{SEEK 1 iniziale, seguita da GET;}
{F^ ora contiene 'A'.}
SEEK (F, 5);
{Posizione del file a 5.}
GET (F);
{F^ viene caricata con 'E'.}
C := F^
{C assume ora il valore 'E'.}
```

Come regola, si deve far sempre seguire una GET alla SEEK (F, N) per assicurarsi che la ennesima componente sia contenuta nella variabile buffer.

GET e PUT operano normalemente su file DIRECT con struttura BINARY. Comunque READ e WRITE operano solo su file ASCII (file-testo). In particolare, READ non opera su file BINARY con modo DIRECT perche' effettua una assegnazione della variabile buffer prima di richiamare la GET. D'altro lato, GET e PUT non vengono normalmente usate su file ASCII strutturati in modo DIRECT. La valutazione lazy rende la READ e la WRITE molto piu' appropriate. Si deve fare molta attenzione quando si vogliono mescolare operazioni normali sequenziali con operazioni di SEEK in modo DIRECT.

FILE TEMPORANEI

A volte un programma ha bisogno di un file per dati temporanei intermedi. In questo caso, possono essere creati file temporanei in modo indipendente dal sistema operativo. Per fare questo senza fornire il nome di un file in un formato specifico, occorre fare un ASSIGN del carattere zero come nome del file.
Ad esempio:

```
ASSIGN (F, CHR (0))
```

Il file system crea un unico nome per il file quando vede che esso e' zero.

In ambienti in cui diversi programmi condividono i file, il numero identificatore del programma in esecuzione viene a far parte del nome del file. I file temporanei vengono cancellati quando sono chiusi, esplicitamente oppure quando il file viene deallocato. RESET e REWRITE non cancellano il file.

18. UNITÀ COMPILABILI DI UN PROGRAMMA

SOMMARIO

Una unita' di compilazione e' un file sorgente che puo' essere compilato separatamente dal compilatore. In questo capitolo sono descritti i tre tipi di unita' di compilazione consentiti in Pascal: programmi, moduli e unita' di implementazione.

INDICE

INTRODUZIONE

Il compilatore Pascal puo' compilare tre tipi diversi di file sorgente: programmi, moduli ed implementazioni di unita'. Moduli e implementazioni di unita' possono venire compilati separatamente ed essere concatenati in seguito ad un programma senza dover essere ricompilati. A livello Standard si possono compilare soltanto programmi; moduli ed unita' sono permessi soltanto a livello Esteso.

Esempio di programma compilabile:

```
PROGRAM MAIN (INPUT, OUTPUT);
BEGIN
  WRITELN ('Main Program')
END. {Programma principale}
```

Esempio di modulo compilabile:

```
MODULE MOD_DEMO;
{Nessun parametro di intestazione}
  PROCEDURE MOD_PROC;
  BEGIN
    WRITELN ('Output from MOD_PROC in MOD_DEMO.')
  END;
END. {Mod_Demo}
```

Esempio di unita' compilabile:

```
INTERFACE;
  UNIT UNIT_DEMO (UNIT_PROC);
  {UNIT_PROC e' l'unico identificatore esportato}
  PROCEDURE UNIT_PROC;
END;
IMPLEMENTATION OF UNIT_DEMO;
  PROCEDURE UNIT_PROC;
  BEGIN
    WRITELN ('Output from UNIT_PROC in UNIT_DEMO.')
  END;
END. {Unit_Demo}
```

MODULE MOD_DEMO e UNIT UNIT_DEMO possono essere compilati separatamente ed inseriti di seguito nel programma principale nel modo seguente:

```
{INTERFACE e' richiesta all'inizio di ogni}
{sorgente che implementa o fa uso di un'unita'.}

INTERFACE;
  UNIT UNIT_DEMO (UNIT_PROC);
  PROCEDURE UNIT_PROC;
END;

PROGRAM MAIN (INPUT, OUTPUT);
{La clausola USES serve per collegare}
{l'implementazione ed il programma.}
USES UNIT_DEMO;

{La dichiarazione EXTERN serve per collegare}
{la procedura del modulo.}
PROCEDURE MOD_PROC; EXTERN;
BEGIN
  WRITELN ('Output from Main Program.');
  MOD_PROC;
  UNIT_PROC;
END. {Fine del programma principale.}
```

Quando viene compilato il programma MAIN, l'output e' costituito dalle seguenti parti:

1. output dal programma principale

2. output da MOD_PROC dichiarata in MOD_DEMO

3. output da UNIT_PROC dichiarata in UNIT_DEMO

Le regole che governano la costruzione e l'uso di programmi, moduli e unita' sono fornite nelle sezioni che seguono:

"PROGRAMMI"
"MODUL1"
"UNITA'"

PROGRAMMI

Ad eccezione dell'intestazione e del punto alla fine, un programma Pascal ha lo stesso formato di una dichiarazione di procedura. Le istruzioni comprese tra le parole chiave BEGIN e END vengono dette "corpo del programma".

Esempio di programma:

```
{Intestazione di programma}
PROGRAM ALPHA (INPUT, OUTPUT, A_FILE, PARAMETER);

{Sezione dichiarativa}
VAR A_FILE: TEXT; PARAMETER: STRING (10);

{Corpo del programma}
BEGIN
  REWRITE (A_FILE);
  WRITELN (A_FILE, PARAMETER);
END.
{Finisce con il punto '.'}
```

La parola "ALPHA" dopo la parola chiave riservata "PROGRAM" e' l'identificatore di programma. Esso diventa l'identificatore per una procedura PUBLIC senza parametri, a un livello superiore di tutti gli altri identificatori nel programma. Questa procedura ha pure l'identificatore PUBLIC ENTGQQ, il quale e' richiamato in fase di inizializzazione per iniziare l'esecuzione del programma.

Si puo' richiamare il corpo del programma come una procedura PUBLIC da un altro programma oppure da un modulo o da un'unita', facendo uso dell'identificatore del programma oppure di ENTGQQ come nome di procedura (questo non e' pero' consigliato). Questo significa che si puo' ridichiarare l'identificatore di un programma all'interno di un programma con le regole usuali di visibilita'. L'identificatore di programma e' allo stesso livello degli identificatori predichiarati; chiamare un programma INTEGER o READ causa quindi un errore.

I parametri del programma indicano variabili che vengono impostate fuori del programma stesso. Il programma comunica con il suo ambiente attraverso queste variabili.

A livello Standard tutte le variabili di qualsiasi tipo FILE devono essere presenti come parametri di programma, dato che non c'e' altro modo di fornire un nome di sistema operativo al file. Comunque, a livello Esteso, possono essere usate ASSIGN e READFN per assegnare nomi di file; in questo modo non e' piu' necessario passare queste variabili file come parametri.

I parametri di un programma sono completamente diversi da quelli delle procedure; essi non vengono passati come parametri alla procedura che e' il corpo del programma. Tutti i parametri di programma devono essere dichiarati nella parte dichiarativa del blocco che costituisce il programma.

Se non vi sono parametri di programma e non si fa riferimento ai file INPUT e OUTPUT, e' sufficiente usare questa forma:

```
PROGRAM <identificatore>;
```

I due file INPUT e OUTPUT ricevono un trattamento speciale quando sono passati come parametri. I loro valori non vengono impostati come tutti gli altri parametri e non devono essere dichiarati, poiche' sono gia' predichiarati. Ognuno di questi, quando e' usato sia implicitamente che esplicitamente, deve comparire come parametro di programma:

```
WRITE (OUTPUT, 'Prompt: ');   {Uso esplicito}
READLN (INPUT, P);

WRITE ('Prompt: ');           {Uso implicito}
READLN (P);
```

Il compilatore emette una segnalazione se essi vengono usati nel programma ma non vengono passati come parametri. L'unico effetto di INPUT e OUTPUT come parametri e' quello di evitare questa segnalazione.

Si possono ridefinire gli identificatori INPUT e OUTPUT. Comunque, tutte le procedure e funzioni input/output su file-testo (READ, EOLN, ecc.) usano la definizione originale. RESET (INPUT) e REWRITE (OUTPUT) vengono generate automaticamente, presenti o no come parametri di programma; esse possono anche essere generate esplicitamente.

L'inizializzazione del programma da' un valore ad ogni variabile parametro, eccetto INPUT e OUTPUT. Ogni parametro deve essere di un tipo semplice oppure STRING, LSTRING o FILE (cioe' di ogni tipo accettato dalla procedura READFN). I parametri di un programma devono essere variabili intere: non e' permessa la selezione di componenti.

Internamente, ogni parametro di programma usa il file INPUT e genera chiamate a READFN. Prima della lettura di ogni parametro viene fatta una chiamata speciale alla routine interna PPMFQQ. PPMFQQ legge i caratteri ritornati da una routine di interfaccia di sistema operativo chiamata PPMUQQ, che li legge a sua volta dalla linea di comando. PPMFQQ prende questi caratteri e li pone all'inizio del file INPUT. L'identificatore del parametro e' passato ad entrambe le routine (PPMFQQ e PPMUQQ). Alcuni sistemi operativi usano l'identificatore come prompt.

L'uso dei parametri di programma in Pascal puo' essere illustrato meglio mostrando come cambiare un programma in una procedura. Consideriamo il seguente programma:

```
PROGRAM ALPHA (INPUT, OUTPUT, P1, P2,...Pn);
<dichiarazioni>
{Includendo quelle per P1, P2,...Pn}
BEGIN
  <corpo>
END.
```

Il programma ALPHA potrebbe diventare la seguente procedura:

```
PROCEDURE ENTGQQ [PUBLIC];
<dichiarazioni>
{Includendo quelle per P1, P2,...Pn}
BEGIN
  PPMFQQ ('P1'); READFN (INPUT, P1);
  PPMFQQ ('P2'); READFN (INPUT, P2);
  .
  .
  PPMFQQ ('Pn'); READFN (INPUT, Pn);
  PPMEQQ;
  {Chiamata dopo la lettura completa dei parametri}
  <istruzioni di programma>
END;
```

L'azione della routine di interfaccia PPMFQQ dipende dal sistema operativo in questione.

Alcuni sistemi operativi possiedono elaborati meccanismi per trattare questo genere di parametri, usando menu e valori di default. Se il sistema operativo in questione rientra in questa categoria, lo stesso meccanismo si applica generalmente ai parametri di programma Pascal.

Altri sistemi operativi meno sofisticati passano al programma il rimanente della linea usata per lanciarlo; in questo caso i valori dei parametri vengono presi direttamente dalla linea stessa.

Se il sistema operativo non fornisce un meccanismo di passaggio di parametri, o se si verifica un errore durante l'uso di questo meccanismo, o ancora se non viene gestito un numero sufficiente di parametri, allora PPMFQQ gestisce da sola l'input dei parametri. Essa interfaccia in modo interattivo con l'utente, richiedendo i valori per ogni singolo parametro. Per ulteriori dettagli sul modo in cui la specifica implementazione inizializza i parametri di programma, si puo' vedere l'Appendice A, "Caratteristiche di Implementazione", nel manuale "Linguaggio Pascal Guida Utente".

MODULI

I moduli forniscono un metodo semplice per combinare diverse parti compilabili in un programma. Le unita', descritte piu' oltre nella sezione "Unita'", forniscono un metodo ancora piu' potente per ottenere lo stesso scopo.

Un modulo e' un programma senza il corpo. L'identificatore nell'intestazione del modulo ha lo stesso spazio di descrizione dell'identificatore di programma. L'intestazione puo' anche includere attributi applicabili a tutte le procedure e funzioni nel modulo. Non vi sono parametri di modulo, e neppure vi e' il corpo di un modulo. Un

modulo finisce con la parola chiave END e un punto.

Esempio di modulo:

```
MODULO BETA [PUBLIC];      {Attributi opzionali}

PROCEDURE GAMMA;
   BEGIN WRITELN ('Gamma') END;

FUNCTION DELTA: WORD;
   BEGIN DELTA := 123 END;

END.                       {Nessun corpo prima di END}
```

Dopo l'identificatore del modulo si possono fornire uno o piu' attributi (fra parentesi quadre) i quali valgono per tutte le procedure e funzioni presenti all'interno del modulo. A seconda degli attributi specificati valgono le seguenti regole:

- se non vi sono attributi viene assunto PUBLIC. Se esiste una lista di attributi (anche vuota) allora non viene assunto l'attributo PUBLIC.

- la direttiva EXTERN usata per una procedura particolare (o funzione) elimina l'attributo PUBLIC per l'intero modulo.

- EXTERN e ORIGIN non possono essere specificati come attributi di un intero modulo, anche se possono essere specificati per le singole procedure.

- se vengono usati PURE e INTERRUPT, il modulo deve contenere solo funzioni PURE e procedure INTERRUPT.

- PUBLIC e' l'attributo per default per tutte le procedure e funzioni. Tuttavia, in alcuni casi, una chiamata di procedura PUBLIC richiede maggior spesa di tempo di una chiamata locale. In altri casi l'identificatore di una procedura locale puo' essere in conflitto con un identificatore globale passato al linker. Per evitare questi problemi occorre usare PUBLIC con procedure individuali e fornire una lista di attributi vuota al modulo (cioe' MODULE BETA [];).

Anche se un modulo non ha corpo, ma solo dichiarazioni, puo' essere usato come procedura senza parametri; cioe' si puo' dichiarare l'identificatore del modulo come una procedura e richiamarlo da altri programmi, moduli o unita'. Questa procedura di modulo (a differenza di una procedura per programmi o unita') non viene mai chiamata automaticamente, poiche' il compilatore non puo' sapere se il modulo e' stato caricato e quindi non puo' sapere se generare una chiamata al modulo stesso.

In alcuni casi il compilatore genera codice di inizializzazione di modulo, che deve essere eseguito quando il modulo viene richiamato come procedura EXTERN. Se questo codice e' necessario, il compilatore emette questa segnalazione:

```
Initialize Module
```

Se compare questo messaggio occorre dichiarare il modulo come una procedura EXTERN senza parametri e chiamare una volta la procedura prima di accedere a qualche elemento del modulo (questo e' necessario se il modulo dichiara variabili FILE).

Dato un modulo M che dichiara le proprie variabili file, un programma che usa M deve essere del tipo:

```
PROGRAM P (INPUT, OUTPUT);
  .
  .
PROCEDURE M; EXTERN;
BEGIN
  M;   {La chiamata runtime inizializza le variabili file.}
  .
  .
END.
```

Se il modulo usa (USES) delle interfacce che richiedono inizializzazioni, il compilatore genera un messaggio per avvertire che il modulo deve essere dichiarato EXTERN e richiamato come descritto nel paragrafo precedente.

Se il modulo M non contiene alcuna delle proprie variabili file oppure usa unita' inizializzate, non occorre richiamare M come procedura nel corpo del programma oppure dichiararlo come una procedura EXTERN.

Alle variabili nei moduli non sono automaticamente assegnati attributi. Ad eccezione della fase di inizializzazione delle variabili file descritta sopra, le variabili nei moduli sono esattamente come le variabili nei programmi.

UNITA'

Il Pascal fornisce un modo strutturato per accedere a moduli separati. Un'unita' e' composta da due parti:

1. un'interfaccia

2. un'implementazione

L'interfaccia compare in testa all'implementazione di un'unita' ed in cima ad ogni programma, modulo, interfaccia o implementazione che fa uso di un'unita'.

Un'unita' contiene costanti, tipi, tipi super, variabili, procedure e funzioni, ognuna delle quali e' dichiarata nella sua interfaccia. Qualsiasi programma, modulo, implementazione o interfaccia puo' far uso di un'interfaccia. Un'implementazione contiene i corpi delle procedure e delle funzioni di un'unita', come pure le eventuali inizializzazioni per

l'unita'. Lo schema generale e' illustrato in figura 18-1.

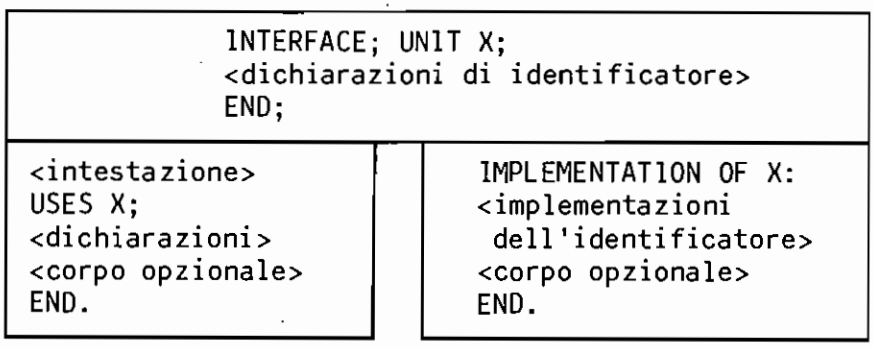


Fig. 18-1 Unita' Pascal

Quando vengono usate le unita', la loro interfaccia deve comparire prima di ogni altra cosa nel file sorgente, sia in un'implementazione che in un programma, modulo, od altra unita' che ne fa uso. Nel diagramma precedente, l'INTERFACE e' condivisa; la stessa INTERFACE esiste sia nella IMPLEMENTATION che in altri file sorgente. Analogamente, qualsiasi altro programma, modulo o unita' puo' usare l'unita' X (USE UNIT X); puo' esserci un'altra IMPLEMENTATION OF X, in linguaggio assembler, ad esempio.

Separando l'interfaccia dall'implementazione si puo' compilare un programma prima o durante la scrittura dell'implementazione. Oppure si puo' caricare un programma con una delle tante implementazioni (ad esempio una in Pascal, un'altra in assembler). Un voluminoso programma in Pascal puo' essere meglio organizzato come un programma principale ed un certo numero di unita' (alcune parti del sistema runtime del Pascal sono organizzate in modo analogo). Comunque soltanto un programma, un modulo, un'interfaccia o un'implementazione possono fare uso di un'unita', non una procedura o funzione individuali.

Un programma, modulo, implementazione o interfaccia che fanno uso di un'interfaccia devono riportarla all'inizio del file sorgente. Generalmente il file che contiene l'interfaccia e' un file separato; occorre quindi usare il metacomando $INCLUDE per richiamare questo file in fase di compilazione. Dato che esiste solo una copia originale dell'interfaccia, questo modo e' piu' facile e sicuro che inserire fisicamente l'interfaccia in tutti i posti in cui viene usata (correndo il rischio di averne diverse versioni).

In alcune applicazioni occorre avere diverse versioni della stessa interfaccia. Ad esempio, esiste una versione separata dell'interfaccia del blocco di controllo dei file Pascal per ogni sistema operativo; il file $INCLUDE viene ricopiato dalla versione di interfaccia voluta, prima

di compilare il programma che ne fa uso. Naturalmente ogni versione deve dichiarare gli identificatori comuni; puo' anche avere alcuni valori costanti usati dal metacomando $IF per le porzioni dipendenti dalla versione dell'interfaccia.

Se l'INTERFACE per UNIT X nella figura 18-1 e' contenuta nel file X.INT, allora l'unita' compilativa che fa uso dell'unita' e della sua IMPLEMENTATION deve soltanto includere ($INCLUDE) il file interfaccia all'inizio del file sorgente, come illustrato in figura 18-2.

```
{$INCLUDE:'X.INT'}

<intestazione di unita'          IMPLEMENTATION OF X;
   compilativa>
USES X;                          <implementazioni di
<dichiarazioni>                      identificatore>
<corpo opzionale>                <corpo opzionale>
END.                             END.
```

Fig. 18-2 Unita' con File X.INT

Un file sorgente Pascal di qualsiasi tipo contiene zero o piu' interfacce, separate da punto e virgola e seguite da un programma, da un modulo o da un'implementazione seguita da un carattere punto. Ognuna di queste entita' e' detta "divisione". Vedere le sezioni che seguono, "La Divisione Interfaccia" e "La Divisione Implementazione", per dettagli sulle divisioni.

Un'unita' e' costituita da un identificatore seguito da una lista di identificatori tra parentesi. Questi identificatori vengono detti costituenti dell'unita' e sono quelli forniti da un'unita' o richiesti da un programma, modulo o altra unita'. L'unita' e' preceduta dalla parola chiave UNIT per l'unita' fornita, oppure da USES per quella richiesta.

Tutti gli identificatori di unita' in un file sorgente devono essere univoci. Gli identificatori tra parentesi, comunque, possono differire nelle divisioni fornite e in quelle richieste. La corrispondenza tra gli identificatori forniti e richiesti e' posizionale nella lista (in modo simile ai parametri formali ed attuali delle procedure).

La lista degli identificatori in una clausola USES e' opzionale; quando non sono forniti, gli identificatori in UNIT vengono usati per default. Diversi identificatori in USES permettono di cambiarli in caso di conflitti tra diverse interfacce. Possono essere combinate clausole USES multiple; quindi le seguenti istruzioni sono equivalenti:

```
USES A; USES B; USES C;
USES A, B, C;
```

Si deve anche osservare che un'unita' puo' fornire codice opzionale di inizializzazione. Tale codice e' determinato dalle parole chiave BEGIN e END alla fine dell'interfaccia, e viene fornito in un corpo opzionale nell'IMPLEMENTATION.

Esempio di un'unita' che richiede codice di inizializzazione:

Il file programma, PLOTBOX:

```
{$INCLUDE:'GRAPHI'}
PROGRAM PLOTBOX (INPUT, OUTPUT);
  USES GRAPHICS (MOVE, PLOT);
  {MOVE e PLOT sono identificatori USE}
  BEGIN
    MOVE (0, 0);
    PLOT (10, 0); PLOT (10, 10);
    PLOT (0, 10); PLOT (0, 0);
  END.
```

Il file interfaccia, GRAPHI:

```
INTERFACE;
  UNIT GRAPHICS (BJUMP, WJUMP);
  {Gli identificatori esportati sono BJUMP e WJUMP.}
  {Nel programma suddetto, MOVE e PLOT sono pseudonimi}
  {di questi identificatori.}
  PROCEDURE BJUMP (X, Y: INTEGER);
  PROCEDURE WJUMP (X, Y: INTEGER);
  {Soltanto intestazioni di procedure.}
BEGIN
{BEGIN implica codice di inizializzazione.}
END;
```

Il file implementazione:

```
{$INCLUDE:'GRAPH1'}
{$INCLUDE:'BASEPL'}
{La seguente implementazione fa uso (USES) della UNIT BASEPL.}
{L'interfaccia e' quindi inclusa sopra, e l'unita' sotto.}
IMPLEMENTATION OF GRAPHICS;
{L'implementazione non e' visibile all'utente.}
  USES BASEPLOT;
    {Le procedure BJUMP e WJUMP sono implementate di seguito.}
    {Da notare che nell'intestazione sono dati}
    {solo gli identificatori.}
    {La lista dei parametri e' data nell'interfaccia.}
  PROCEDURE BJUMP;
    BEGIN DRAWLINE (BLACK, X, Y) END;
  PROCEDURE WJUMP;
    BEGIN DRAWLINE (WHITE, X, Y) END;
BEGIN
{Inizializzazione.}
  DRAWLINE (BLACK, 0, 0)
END.
```

Il file interfaccia, BASEPL:

```
INTERFACE;
  UNIT BASEPLOT (BLACK, WHITE, DRAWLINE);
    {Altri identificatori oltre agli identificatori}
    {di procedure possono essere esportati.}
    {Si deve osservare che BLACK e WHITE sono costanti esportate}
  TYPE RAINBOW = (BLACK, WHITE, RED, BLUE, GREEN);
  PROCEDURE DRAWLINE (C: RAINBOW; H, V: INTEGER);
    {Non c'e' BEGIN, quindi non c'e' un'unita' inizializzata.}
END;
```

La clausola USES puo' essere specificata soltanto dopo l'intestazione di un programma, di un modulo, di un'interfaccia o di un'implementazione. Quando il compilatore incontra una clausola USES, memorizza nella tabella dei simboli tutti gli identificatori dei costituenti (dalla clausola UNIT o dalla USES). Gli identificatori di variabili, procedure e funzioni vengono associati ai rispettivi identificatori nell'interfaccia e diventano riferimenti esterni per il linker.

Quando viene compilato il programma sopra descritto, ogni riferimento alla procedura PLOT genera un riferimento esterno a WJUMP. Comunque i riferimenti a DRAWLINE usano lo stesso identificatore per il riferimento esterno.
Le costanti ed i tipi (inclusi i tipi super array) nell'interfaccia vengono semplicemente memorizzati nella tabella dei simboli del programma (insieme ai nuovi identificatori, se ve ne sono). Quindi un tipo in un'interfaccia e' identico al corrispondente tipo nella clausola USES.
Gli identificatori di campo di record sono gli stessi nel programma, nell'interfaccia e nell'implementazione. Gli identificatori di costanti di tipo enumerato devono essere forniti esplicitamente, se necessario; essi non sono automaticamente ricavati dall'identificatore di tipo enumerato. Le etichette non possono essere fornite da un'interfaccia,

dato che l'etichetta di un GOTO deve essere nella stessa divisione del GOTO.

LA DIVISIONE INTERFACCIA

La struttura di un'interfaccia e' la seguente:

1. Una sezione di interfaccia comincia con la parola riservata INTERFACE, un numero opzionale di versione tra parentesi ed un punto e virgola.

2. Seguono la parola chiave UNIT, l'identificatore di unita', la lista tra parentesi degli identificatori (costituenti) esportati ed un altro punto e virgola.

3. Le altre unita' richieste da questa interfaccia vengono di seguito, nelle clausole USES.

4. L'ultima sezione e' la dichiarazione attuale per tutti gli identificatori forniti nella lista dell'interfaccia, che fa uso delle usuali sezioni CONST, TYPE e VAR e delle intestazioni di procedure e funzioni in ordine qualsiasi. Non sono permesse le sezioni LABEL e VALUE.

5. L'interfaccia termina con BEGIN END, se e' richiesta l'inizializzazione, o solo con END in caso contrario.

Eccetto ORIGIN, che non puo' essere usato in interfacce, puo' essere fornita la maggior parte degli attributi a variabili, procedure e funzioni. Dato che gli attributi PUBLIC o EXTERN oppure la direttiva EXTERN vengono forniti automaticamente, non bisogna specificare attributi che possono essere in conflitto (cioe' PUBLIC e EXTERN).
Di solito, gli unici identificatori da dichiarare sono i costituenti, ma vengono permessi altri identificatori. Se l'interfaccia ha bisogno di una chiamata per inizializzare l'unita', la parola chiave BEGIN genera la chiamata. L'interfaccia termina con la parola riservata END ed un punto e virgola.

Esempio di divisione interfaccia:

```
INTERFACE (3);
  UNIT KEYFILE (FINDKEY, INSKEY, DELKEY, KEYREC);
    USES KEYPRIM (KFCB, KEYREC);
    PROCEDURE FINDKEY (CONST NAME: LSTRING;
        VAR KEY: KEYREC;
        VAR REC: LSTRING);
    PROCEDURE INSKEY (CONST REC: LSTRING;
        VAR KEY: KEYREC);
    PROCEDURE DELKEY (CONST KEY: KEYREC);
    PROCEDURE NEWKEY (CONST KEY: KEYREC);
BEGIN
{Indica che l'unita' e' inizializzata.}
END;
```

In questo esempio KEYREC e' parte dell'unita' KEYPRIM, ma viene esportato come parte dell'unita' KEYFILE. KFCB fa pure parte dell'unita' KEYPRIM ma non viene esportato dall'unita' KEYFILE. Questo e' permesso ma non e' inutile dato che NEWKEY non e' nota neppure nell'implementazione dell'unita'.

La memoria disponibile in fase di compilazione limita il numero di identificatori che il compilatore puo' gestire. Questo limite puo' essere un problema se si hanno molte interfacce, specialmente nel caso in cui interfacce usano altre interfacce. Il sintomo e' il seguente messaggio:

```
Compiler Out Of Memory
```

Il messaggio viene emesso prima dell'ultima USES del programma, modulo o implementazione in compilazione. Occorre quindi ridurre il numero di identificatori nell'interfaccia contenente solo tipi (e costanti relative ai tipi) e condivisa dalle altre interfacce, ed usare soltanto questa interfaccia nelle altre.
Se viene inclusa una variabile file nell'interfaccia, l'unita' deve essere inizializzata. Il compilatore non fornisce il solito messaggio:

```
Initialize Variable
```

quando si dichiara un file in un'interfaccia. Se l'interfaccia contiene file, occorre essere sicuri di terminarle con BEGIN END per invocare l'inizializzazione.

LA DIVISIONE IMPLEMENTAZIONE

Si puo' compilare l'implementazione di un'unita' separatamente da altri programmi, moduli o unita'; occorre pero' compilarla con la sua propria interfaccia. La struttura di un'implementazione e' la seguente:

1. L'implementazione di un'interfaccia comincia con la parola riservata IMPLEMENTATION OF, seguita dall'identificatore dell'unita' e da un punto e virgola.

2. Viene poi la clausola USES, per le unita' che la richiedono per il loro proprio uso.

3. Vengono quindi le usuali clausole LABEL, CONSTANT, TYPE, VAR e VALUE con tutte le procedure e funzioni descritte come costituenti (che devono essere nel blocco piu' esterno) oppure usate internamente in qualsiasi ordine.

Le sezioni VALUE e LABEL possono comparire nell'implementazione, non nell'interfaccia.

Esempio di implementazione:

```
IMPLEMENTATION OF KEYFILE;
  USES KEYPRIM (KEYBLOCK, KEYREC);

  VAR KEYTEMP: KEYREC;

  PROCEDURE FINDKEY;
    BEGIN
      .
      {Codice per FINDKEY}
      .
    END;

  PROCEDURE INKEY;
    BEGIN
      .
      {Codice per INKEY}
      .
    END;

  PROCEDURE DELKEY;
    BEGIN
      .
      {Codice per DELKEY}
      .
    END;

BEGIN
  .
  {Qualsiasi codice di inizializzazione.}
  .
END.
```

Costanti, variabili e tipi dichiarati nell'interfaccia non vengono ridichiarati nell'implementazione. Comunque possono essere dichiarati altri tipi "privati". Procedure e funzioni dell'unita' non includono la loro lista di parametri (essa e' presente nell'interfaccia) o attributi. (Si assume l'attributo PUBLIC, a meno che la direttiva EXTERN non sia data esplicitamente).

Tutte le procedure e funzioni in INTERFACE devono essere definite in IMPLEMENTATION. Ad esse puo' comunque essere associata la direttiva EXTERN in modo che piu' IMPLEMENTATION (oppure un'IMPLEMENTATION e codice assembler) possano implementare una singola INTERFACE. Tutte le procedure e funzioni con direttiva EXTERN devono comparire per prime; il compilatore effettua un controllo ed emette una segnalazione di errore se la direttiva EXTERN manca o non e' al posto giusto.

Si puo' implementare un'unita' in linguaggio assembler; in questo caso tutte le variabili, procedure e funzioni devono generare definizioni pubbliche per il caricatore. Si possono usare anche altri linguaggi per implementare un'unita' (MS-FORTRAN o altri).

Se l'interfaccia non e' scritta in Pascal occorre fornire l'attributo di sequenza di chiamata (e naturalmente conoscere le sequenze di chiamata e la rappresentazione interna dei parametri).

Alcune unita' runtime del Pascal sono state implementate in assembler, altre in Pascal. Ogni sezione IMPLEMENTATION che non implementa tutte le procedure di interfaccia deve, all'inizio, dichiarare tali procedure e funzioni EXTERN.

Un'implementazione, come un programma, puo' avere un corpo. Il corpo viene eseguito quando il programma che usa l'unita' viene invocato; cosi' vengono effettuate tutte le inizializzazioni richieste dall'unita'. Esso include l'inizializzazione delle variabili file e la parte di inizializzazione utente. Se il file sorgente contiene diverse unita', viene richiamato il corpo di ogni inizializzazione nell'ordine in cui compare la relativa USES nel file sorgente. Comunque il codice di inizializzazione per un'unita' viene eseguito soltanto una volta, indipendentemente dal numero di clausole che fanno riferimento ad esso.

Il corpo, come in un programma, e' composto da una lista di istruzioni comprese tra BEGIN e END. In fase di inizializzazione, il numero di versione dell'interfaccia con la quale era stata compilata l'implementazione viene confrontato con il numero di versione dell'interfaccia compilata con il programma. Questi numeri devono coincidere. Questo controllo contribuisce a mantenere un programma aggiornato. Se non viene fornito alcun numero di versione, viene assunto il valore zero.

La parola chiave BEGIN prima dell'END finale indica un'unita' con inizializzazione. Se viene omessa la parola chiave BEGIN, l'implementazione non deve avere un corpo e non viene fatta alcuna inizializzazione. Un'unita' senza inizializzazione manca di:

- codice utente di inizializzazione
- garanzia di unica inizializzazione
- controllo di numero di versione

Il formato di una implementazione non inizializzata e' simile a quello di un programma:

```
IMPLEMENTATION OF <identificatore di unita'>
<dichiarazioni>
BEGIN
   <corpo>          {Codice di inizializzazione}
END.
```

Il formato di un'implementazione non inizializzata e' il seguente (simile ad un modulo):

```
IMPLEMENTATION OF <identificatore di unita'>
<dichiarazioni>
{Nessun codice di inizializzazione}
END.
```

Se l'implementazione di un'unita' non inizializzata dichiara un file o fa uso (USES) di un'interfaccia che richiede l'inizializzazione, allora il compilatore avverte di inizializzare l'implementazione. L'inizializzazione viene fatta in modo automatico se si aggiunge la parola chiave BEGIN sia all'interfaccia che all'inizializzazione. Come per i moduli, e' possibile dichiarare un'unita' non inizializzata come procedura EXTERN e quindi farla inizializzare in fase di chiamata del programma.

19. METACOMANDI

SOMMARIO

Questo capitolo descrive il metalinguaggio Pascal. Si tratta di un linguaggio di controllo per il compilatore ed e' costituito da metacomandi rivolti al compilatore per ottenere le seguenti prestazioni: gestione degli errori, controllo dell'ottimizzazione, uso del file sorgente durante la compilazione, formato del file listing.

INDICE

INTRODUZIONE

I metacomandi costituiscono il linguaggio di controllo del compilatore. Sono direttive al compilatore che permettono di controllare:

- il livello di linguaggio Pascal
- il debugging e la gestione degli errori
- il livello di ottimizzazione
- l'uso del file sorgente durante la compilazione
- il formato del file listing

Si possono specificare uno o piu' metacomandi all'inizio di un commento; i metacomandi multipli devono essere separati da spazi o da virgole. Tra i metacomandi gli spazi, le tabulazioni e la marca di riga vengono ignorati. Le seguenti definizioni sono quindi equivalenti:

```
{$PAGE:12}
{$PAGE : 12}
```

Per disabilitare i metacomandi all'interno dei commenti, si deve collocare qualunque carattere che non sia di tabulazione o di spazio davanti al primo segno di dollaro, nel modo seguente:

```
{x$PAGE:12}
```

Si possono cambiare le direttive al compilatore durante il corso di un programma. Ad esempio, la maggior parte di un programma puo' essere $LIST-, con alcune sezioni aventi $LIST+ come richiesto.
Alcuni metacomandi, come $LINESIZE, vengono normalmente applicati all'intera compilazione.

Nello scrivere programmi Pascal per altri compilatori occorre ricordare che i metacomandi sono sempre non standard e raramente trasportabili.

I metacomandi richiamano od impostano il valore di una metavariabile. Le metavariabili sono classificate come: senza tipo, intere, switch on/off e stringa.

- Le metavariabili senza tipo vengono richiamate quando usate, come in $EXTEND.
- Le metavariabli intere possono contenere un valore numerico, come in $PAGE:101.
- Gli switch on/off possono essere impostati con un valore numerico in modo che un valore maggiore di zero pone lo switch a on ed un valore minore od uguale a zero lo pone off, come in $MATHCK:1.
- Le metavariabili stringa possono contenere un valore stringa, come in $TITLE:'COM PROGRAM'.

La tabella 19-1 illustra le convenzioni notazionali usate nella descrizione dei metacomandi che seguono:

NOTAZIONE	SIGNIFICATO
	Il metacomando e' senza tipo.
+ o -	Il metacomando e' uno switch on/off. + pone il valore a 1 (on). - pone il valore a 0 (off). Il default e' indicato da + o - nell'intestazione.
:<n>	Il metacomando e' un intero
:'<testo>'	Il metacomando e' una stringa

Tabella 19-1 Notazione dei Metacommandi

Il valore stringa nel metalinguaggio puo' essere una stringa letterale o un identificatore costante di stringa. Le espressioni costanti non sono permesse per numeri e stringhe, sebbene si possa ottenere lo stesso effetto col dichiarare un identificatore costante uguale all'espressione ed usare l'identificatore nel metacomando.

Nei metacomandi, i valori booleani e le costanti ennumerate vengono trasformati nei loro valori ORD. Quindi un booleano FALSE diventa 0 e TRUE diventa 1.

Una lista alfabetica completa dei metacomandi Pascal e' fornita nell'Appendice G, "Riepilogo dei Metacomandi Pascal".

LIVELLO DI LINGUAGGIO E DI OTTIMIZZAZIONE

I metacomandi illustrati nella tabella 19-2 permettono di controllare il livello (Standard, Esteso o di Sistema) al quale il compilatore tratta il programma, ed il grado di ottimizzazione usato. Alcuni di questi metacomandi possono non essere presenti nella versione usata. Per ulteriori dettagli si puo' vedere l'Appendice A, "Caratteristiche di Implementazione", nel manuale "Linguaggio Pascal Guida Utente".

NOME	DESCRIZIONE
$EXTEND	Aggiunge le prestazioni del livello Esteso.
$INTEGER:<n>	Imposta la lunghezza del tipo INTEGER.
$REAL:<n>	Imposta la lunghezza del tipo REAL.
$ROM	Emette una segnalazione sull'inizializzazione statica.
$SIMPLE	Disabilita l'ottimizzazione globale.
$SIZE	Minimizza la dimensione del codice generato.
$SPEED	Minimizza il tempo di esecuzione del codice.
$STANDARD	Abilita il livello Standard.
$SYSTEM	Aggiunge le prestazioni del livello Esteso e Sistema.

Tabella 19-2 Livello di Linguaggio e di Ottimizzazione

Il compilatore emette una segnalazione quando incontra una prestazione il cui livello non e' stato abilitato. Il valore di default e' $EXTEND il quale permette estensioni strutturate che sono abbastanza semplici e portabili. E' anche possibile richiedere esplicitamente l'estensione $SYSTEM che e', per sua natura, a basso livello, dipendente dalla macchina e relativamente non strutturata.

$INTEGER e $REAL definiscono la lunghezza (cioe' la precisione) dei tipi standard INTEGER e REAL. $INTEGER puo' valere soltanto 2 (default), per interi su 16 bit. Si puo' comunque impostare $REAL a 4 o 8 (default), per avere il tipo REAL identico rispettivamente a REAL4 o REAL8.

L'effetto dei metacomandi $SIZE e $SPEED varia a seconda della versione dell'ottimizzatore del compilatore. Il default e' $SIZE. Se si seleziona $SIMPLE, non viene fatta alcuna ottimizzazione. $SIZE, $SPEED e $SIMPLE sono tutti mutuamente esclusivi. Quando viene specificato $ROM, il compilatore segnala che i dati statici non verrano inizializzati nelle seguenti situazioni:

1. nella sezione VALUE

2. ogni volta che l'inizializzazione statica e' dovuta a $INITCK (descritto nella sezione seguente, "Debugging e Gestione degli Errori").

DEBUGGING E GESTIONE DEGLI ERRORI

I metacomandi contenuti nella tabella 19-3 servono per il debugging e per la gestione degli errori. Essi generano anche codice di controllo runtime degli errori.

METACOMMANDO	DESCRIZIONE
$BRAVE+	Invia messaggi di errori e segnalazioni su video.
$DEBUG-	Abilita/Disabilita tutti i controlli di debug (CK nei metacomandi seguenti).
$ENTRY-	Genera chiamate di entrata/uscita da procedure per il debugger.
$ERRORS:<n>	Imposta il numero massimo di errori per pagina (default 25).
$GOTO-	Indica i GOTO come "pericolosi".
$INDEXCK+	Controlla se i valori indice dell'array, e gli indici di super array, sono compresi nel campo di variabilita'.
$INITCK-	Controlla l'uso di valori non inizializzati.
$LINE-	Genera chiamate di numero di linea per il debugger.
$MATHCK+	Controlla gli errori matematici come l'overflow e la divisione per zero.
$NILCK+	Controlla la correttezza dei valori puntatore.
$RANGECK+	Controlla la validita' dei subrange.
$RUNTIME-	Determina il contesto degli errori runtime.
$STACKCK+	Controlla l'overflow dello stack all'ingresso di procedure e funzioni.
$TAGCK-	Controlla i campi etichettati nei record con varianti.
$WARN+	Emette segnalazioni nel file listing.

Tabella 19-3 Debugging e Gestione degli Errori

Se e' presente l'indicatore di controllo quando il compilatore tratta un'istruzione, allora vengono fatti tutti i controlli relativi all'istruzione stessa. Un errore runtime effettua una chiamata al supporto runtime della routine, EMSEQQ (sinonimo di ABORT). Quando viene chiamata EMSEQQ, il compilatore passa le seguenti informazioni:

- messaggio di errore
- codice standard di errore
- valore opzionale di stato di errore, come il codice di ritorno del sistema operativo

EMSEQQ ha pure disponibili:

- il contatore di programma nel punto in cui si e' verificato l'errore
- il puntatore di stack nel punto in cui si e' verificato l'errore
- il puntatore frame nel punto in cui si e' verificato l'errore
- il numero della linea corrente (se c'e' $LINE)
- il nome della procedura o della funzione ed il nome del file sorgente ove la procedura o funzione e' stata compilata. (se c'e' $ENTRY).

Ognuno di questi metacomandi e' descritto in dettaglio nelle pagine seguenti. La maggior parte dei metacomandi in questo gruppo puo' essere fornita al compilatore sotto forma di commutatori di linee di comando. Si puo' vedere, piu' oltre, la sezione "Switch di Linea di Comando" per ulteriori dettagli.

$BRAVE+ Invia messaggi di errore e segnalazioni su terminale, (oltre a riportarle sul file listing). Se il numero di errori e segnalazioni supera la dimensione del video, occorre controllarli sul file listing.

$DEBUG- Abilita/disabilita tutti gli switch di debug (cioe' quelli che terminano con "CK"). Puo' essere utile impostare $DEBUG- all'inizio di un programma per abilitare tutti i controlli e poi abilitare selettivamente gli switch che interessano. Oppure si puo' usare questo metacomando per abilitare tutti gli switch all'inizio, e disabilitare selettivamente quelli che non interessano. Per default alcuni switch sono abilitati, altri no.

$ENTRY- Genera chiamate di ingresso e uscita da procedure. Cio' permette al debugger o al gestore di errori di riconoscere la procedura o funzione nella quale si e' verificato l'errore. Poiche' questo switch genera un notevole incremento di codice supplementare per ogni procedura o funzione, e' consigliabile usarlo solo in fase di debug. Notare che $LINE+ richiede $ENTRY+; quindi $LINE+ abilita $ENTRY e $ENTRY- disabilita $LINE.

$ERRORS:<n> Stabilisce un limite massimo al numero degli errori permessi per pagina. La compilazione termina se questo numero viene superato. Il valore di default e' 25 errori e/o segnalazioni per pagina.

$GOTO- Associa a tutti i GOTO una segnalazione che li considera 'pericolosi'. Questa segnalazione puo' essere utile in alcune circostanze:

- per incoraggiare la programmazione strutturata in un ambiente didattico
- per indicare tutte le istruzioni GOTO durante la fase di debug

$INDEXCK+ Controlla che i valori degli indici degli array, compresi i super array, siano all'interno del campo di variabilita'. Poiche' l'indicizzazione di array avviene molto spesso, il controllo viene abilitato separatamente da altri controlli di subrange.

$INITCK- Controlla la presenza di valori non inizializzati, come:

- INTEGER non inizializzati e subrange INTEGER di 2 byte con valore esadecimale 16#8000
- subrange di 1 byte di INTEGER non inizializzati con valore esadecimale 16#80
- puntatori non inizializzati con valore 1 (se e' abilitato $NILCK)
- REAL non inizializzati con valore speciale

Il metacomando $INITCK genera codice per effettuare le seguenti operazioni:

- stabilire i valori non inizializzati quando essi vengono allocati
- stabilire il valore del campo di variabilita' di INTEGER per la variabile di controllo di FOR non inizializzata quando il ciclo di FOR termina correttamente
- stabilire il valore di una funzione che ritorna uno dei valori dei tipi non inizializzati all'ingresso della funzione

$INITCK non genera mai inizializzazioni o controllo per WORD o tipi indirizzo. Le variabili allocate staticamente vengono caricate con i loro valori iniziali. $INITCK non controlla valori in un array o in un record quando l'array o il record vengono usati.

Le variabili allocate nello stack o nello heap hanno valori inizializzati mediante codice generato. $INITCK non inizializza le variabili delle classi seguenti:

- variabili menzionate nella sezione VALUE
- campi con varianti in un record
- componenti di un super array allocato con la procedura NEW

$LINE- Genera una chiamata al debugger oppure al gestore degli errori per ogni linea di codice eseguibile. Questo permette al debugger di determinare il numero della linea dove e' stato emesso l'errore. Dato che questo metacomando genera un notevole ammontare di codice supplementare per ogni riga del programma, esso dovrebbe essere specificato solamente in fasi di debug. Notare che $LINE+ richiede $ENTRY+; quindi $LINE+ abilita $ENTRY e $ENTRY- disabilita $LINE.

$MATHCK+ Controlla gli errori matematici, incluso l'overflow di INTEGER e WORD e la divisione per zero. $MATHCK non controlla un risultato INTEGER con valore -MAXINT-1 (cioe' #8000); $INITCK non si accorge se in seguito questo valore viene assegnato ed usato.

Disabilitando $MATHCK, non sempre si disabilita il controllo di overflow. Esistono comuque routine di libreria che effettuano la somma e la moltiplicazione con overflow (LADDOK, LMULOK, SADDOK, SMULOK, UADDOK e UMULOK). Per la descrizione di queste funzioni si puo' vedere "Elenco delle Funzioni e Procedure", nel Capitolo 16.

$NILCK+ Controlla l'esistenza delle seguenti condizioni:

- puntatori senza riferimento con valore NIL
- puntatori non inizializzati se e' abilitato anche $INITCK
- puntatori al di fuori del campo di variabilita'
- puntatori che indicano un blocco libero nell'area heap

$NILCK agisce tutte le volte che un puntatore viene privato del riferimento oppure passato alla procedura DISPOSE. $NILCK non effettua controlli su tipi indirizzo.

$RANGECK+ Controlla la validita' del subrange nelle seguenti condizioni:

- assegnazione ad una variabile subrange

- istruzioni CASE senza clausola OTHERWISE
- parametri attuali per le funzioni CHR, SUCC e PRED
- indici nelle procedure PACK e UNPACK
- assegnazioni set e LSTRING, e parametri di valore
- limiti superiori di super array passati alla procedura NEW

$RUNTIME- Se lo switch $RUNTIME e' abilitato durante la compilazione di una procedura o di una funzione, la "locazione di errore" e' il posto dove la procedura o funzione sono state chiamate, e non la locazione all'interno della procedura o funzione. Questa informazione e' normalmente inviata al terminale, ma e' possibile effettuare un concatenamento di una versione utente di EMSEQQ (le routine di messagi errore) allo scopo di fare qualcosa di diverso (come richiamare il debugger runtime oppure inizializzare una unita' di controllo).

$STACKCK+ Controlla l'overflow dello stack all'ingresso di una procedura o funzione, e la memorizzazione nello stack di parametri con dimensione maggiore di quattro byte. In alcune implementazioni il superamento di capacita' dello stack viene sempre controllato. In altre, non viene mai controllato in procedure con l'attributo INTERRUPT.

$TAGCK- Controlla i valori etichetta quando si accede ad un campo con varianti. Soltanto i campi etichettati con identificatori (cioe' il cui valore e' memorizzato nel record) vengono controllati.

$WARN+ Invia messaggi di avvertimento al file listing (questo e' il valore di default). Se questo switch viene disabilitato, vengono segnalati soltanto gli errori bloccanti.

CONTROLLO DEL FILE SORGENTE

Un piccolo gruppo di metacomandi fornisce il controllo sull'uso del file sorgente durante la compilazione. Questi comandi sono elencati nella tabella 19-4 e poi descritti in dettaglio.

NOME	DESCRIZIONE
$IF <costante> $THEN <testo1> $ELSE <testo2> $END	Permette una compilazione condizionale del sorgente <testo1> se la <costante> e' maggiore di zero.
$INCLUDE:'<nome-file>'	Indirizza la compilazione del file sorgente corrente verso il file chiamato.
$INCONST:<testo>	Permette l'introduzione interattiva di valori costanti durante la fase di compilazione.
$MESSAGE:'<testo>'	Permette la visualizzazione di un messaggio su video per indicare quale versione del programma e' in fase di compilazione.
$POP	Ripristina i valori memorizzati dei metacomandi
$PUSH	Memorizza i valori correnti di tutti i metacomandi.

Tabella 19-4 Controllo del File Sorgente

Dato che il compilatore e' sempre posizionato sul simbolo che segue quello corrente, esso gestisce i metacomandi che seguono un simbolo, prima di trattare il simbolo stesso. Questa caratteristica del compilatore e' illustrata nell'esempio:

```
CONST Q = 1;
{$IF Q $THEN}
{Q e' indefinito nei confronti della $IF.}

CONST Q = 1; DUMMY = 0;
{$IF Q $THEN}
{Ora Q e' definito.}

X := P^;
{$NILCK+}
{NILCK e' applicato a P^.}

X := P^ ;;;
{NILCK non e' applicato a P.}
{$NILCK-}
```

$IF <costante> $THEN <testo> $END

Permette una compilazione condizionale di un testo sorgente. Se il valore della costante e' maggiore di zero, viene compilato il testo sorgente che segue la $THEN, altrimenti no. E' anche possibile specificare il costrutto $IF $THEN $ELSE $END come illustrato nel seguente esempio:

```
{$IF MSDOS $THEN}
SECTOR = S12;
{$ELSE}
SECTOR = S128;
{$END}
```

Per simulare un costrutto $IFNOT si deve usare la forma seguente del metacomando:

```
$IF <costante> $ELSE <testo> $END
```

La costante puo' essere un numero letterale oppure un identificatore costante. Il testo tra $THEN, $ELSE e $END e' arbitrario; esso puo' includere commenti, linea a capo, altri metacomandi (incluse $IF nidificate), ecc. I metacomandi che appartengono al testo non compilato vengono ignorati eccetto, naturalmente, i relativi $ELSE e $END.

Esempi con l'uso di metacondizionali:

```
{$IF FPCHIP $THEN}
CODEGEN (FADDCALL, T1, LEFTP)
{$END}
{$IF COMPSYS $ELSE}
IF USERSYS THEN DOITTOIT
{$END}
```

$INCLUDE Permette al compilatore di prendere in esame testi che appartengono al file cui si fa riferimento. Quando si raggiunge la fine del file, il compilatore ritorna a compilare il testo che conteneva le $INCLUDE. La compilazione continua con la linea del testo sorgente che segue la $INCLUDE. Il metacomando $INCLUDE deve sempre essere ultimo sulla riga.

$INCONST Permette l'introduzione di valori di costanti (ad esempio quelle usate dalle $IF) durante la fase di compilazione, invece di inserirli nel file sorgente. Cio' e' utile quando vengono usati metacondizionali per compilare una versione di sorgente per un ambiente particolare, per un singolo utente oppure per una data macchina. La compilazione puo' essere interattiva oppure a lotti. Ad esempio, il metacomando $INCONST:YEAR produce il seguente prompt per la costante YEAR:

```
Inconst : YEAR =
```

Occorre soltanto dare una risposta di questo tipo:

```
Inconst : YEAR = 1983
```

La risposta deve essere di tipo WORD. L'effetto e' analogo a quello ottenuto dichiarando una costante YEAR con il valore 1983. Questo posizionamento interattivo e' equivalente alla dichiarazione:

```
CONST YEAR = 1983;
```

E' anche possibile rispondere con una stringa letterale tra apici, per creare una costante di tipo STRING(n). Ad esempio, il metacomando $INCONST:HEADER da' il segnale per un'intestazione. Ponendo una stringa tra apici si dichiara una costante stringa:

```
Inconst : HEADER = 'Processor Version 2.75'
```

$MESSAGE — Permette l'invio di messaggi su video durante la fase di compilazione. E' molto utile quando vengono usate intensivamente metacondizionali e, ad esempio, quando occorre sapere quale versione di programma si sta compilando.

Esempio di metacomando $MESSAGE:

```
{$MESSAGE: 'Message on terminal screen!'}
```

$PUSH e $POP — Permettono la creazione di un meta-ambiente, nel quale si possono memorizzare i metavalori mediante $PUSH e ripristinarli mediante $POP. $PUSH e $POP sono utili in file $INCLUDE per salvare e ripristimare metacomandi nel file sorgente principale.

CONTROLLO DEL FILE LISTING

I metacomandi elencati nella tabella 19-5 e descritti in questa sezione permettono di formattare il file listing.

METACOMANDO	DESCRIZIONE
$LINESIZE : <n>	Stabilisce la lunghezza della riga del file listing. Il valore di default e' 79 oppure 131, a seconda dell'implementazione.
$LIST+	Abilita o disabilita il listing del sorgente. Gli errori vengono sempre listati.
$OCODE+	Abilita il listing del codice oggetto deassemblato.
$PAGE+	Salta ad una nuova pagina. Il numero di linea non viene ripristinato.
$PAGE : <n>	Stabilisce il numero di pagina per la pagina successiva (non salta ad una nuova pagina).
$PAGEIF : <n>	Salta ad una nuova pagina se vengono lasciate meno di n linee sulla pagina corrente.
$PAGESIZE : <n>	Stabilisce la lunghezza del listing in linee. Il valore di default e' 55.
$SKIP : <n>	Salta n linee oppure va a fine pagina.
$SUBTITLE :'<testo>'	Stabilisce il sottotitolo della pagina.
$SYMTAB+	Riporta la tabella dei simboli nel file listing.
$TITLE : '<testo>'	Stabilisce il titolo della pagina.

Tabella 19-5 Metacomandi di Controllo del File Listing

$LINESIZE: <n> Stabilisce la massima lunghezza delle linee nel file listing. Questo valore e' normalmente 131 o 79, a seconda dell'implementazione.

$LIST+ Abilita il listing del sorgente. Tranne quando e' specificato $LIST-, i metacomandi compaiono nel file listing. Il formato del file listing e' illustrato nella sezione seguente "Formato del File Listing".

$OCODE+ Abilita il listing simbolico del codice generato. Anche se il formato varia a seconda del tipo di codice oggetto generato, esso assomiglia generalmente ad un listing

assembler, con indirizzi di codice ed operazioni mnemoniche. In molti casi gli identificatori di procedure, funzioni e variabili statiche vengono troncati nel file listing oggetto..

$PAGE+ Forza a una nuova pagina nel file listing sorgente. Il numero di pagina del file listing viene incrementato automaticamente.

$PAGE: <n> Stabilisce il numero della pagina seguente nel listing. $PAGE: <n> non forza ad una nuova pagina nel file listing.

$PAGEIF: <n> Effettua in modo condizionato una $PAGE+, se il numero corrente di linea del file sorgente piu' n e' inferiore o uguale alla dimensione della pagina corrente.

$PAGESIZE: <n> Stabilisce il numero massimo di linee in una pagina nel sorgente. Il valore di default e' di 55 linee per pagina.

$SKIP: <n> Salta n linee o va a fine pagina nel file listing sorgente.

$SUBTITLE:'<sottotitolo>'

Stabilisce il nome di un sottotitolo che compare sotto il titolo in cima ad ogni pagina del file listing sorgente.

$SYMTAB+ Quando impostato, emette informazioni sulle variabili delle varie procedure e funzioni (vedi ad esempio le linee 14 e 17 nel file listing di esempio della seguente sezione "Formato del File Listing"). Le colonne di sinistra contengono:

1. lo spiazzamento della variabile dal puntatore frame (per variabili di procedure e funzioni)

2. lo spiazzamento delle variabili nella memoria statica (per programmi e variabili STATIC)

3. la lunghezza della variabile

Il carattere '+' o '-' iniziale indica uno spiazzamento frame. Notare che questo spiazzamento e' relativo all'indirizzo piu' basso usato dalla variabile.

La prima linea del listing $SYMTAB contiene lo spiazzamento dell'indirizzo di ritorno, a partire dall'inizio del frame (zero per il programma principale), e la lunghezza del frame, dal puntatore frame alla fine comprese le variabili frontali e temporanee. Le variabili temporanee generate dal codice non vengono incluse. Per le funzioni, la seconda linea contiene lo spiazzamento, la lunghezza ed il tipo della variabile ritornata dalla funzione. Le linee rimanenti contengono

una lista delle variabili, con il loro tipo ed attributi, come illustrato nella tabella 19-6.

PAROLE CHIAVE	SIGNIFICATO
Public	Ha l'attributo PUBLIC
Extern	Ha l'attributo EXTERN
Origin	Ha l'attributo ORIGIN
Static	Ha l'attributo STATIC
Const	Ha l'attributo READONLY
Value	E' presente in una sezione VALUE
ValueP	E' un parametro valore
VarP	E' un parametro VAR o CONST
VarsP	E' un parametro VARS o CONSTS
ProcP	E' un parametro di procedura
Segmen	Usa l'indirizzamento segmentato
Regist	Parametro passato in registro

Tabella 19-6 Notazione della Tabella dei Simboli

$TITLE:'<titolo>'

Stabilisce il titolo che compare in cima ad ogni pagina del listing sorgente.

FORMATO DEL FILE LISTING

La seguente descrizione del file listing e' basata sul seguente esempio di listing:

```
Use   Title                                              PAGE   1
User  Subtitle                                           12/11/82
                                                         10:49:17
JG IC Line#  Source Line   Pascal Version 3.0  10/82
   00     1  PROGRAM foo;   {$symtab+}
   10     2  VAR i:integer; k:ARRAY [-9..0] OF integer,
             2  --------------------Warning 156,Assumed ;^
   20     3  FUNCTION bar (VAR j: integer): integer;
   20     4    VAR k: ARRAY [0..9] OF integer;
   20     5    BEGIN
+  21     6    GOTO 1;          {saltare avanti}
             6  -------^Warning 281 Label Assumed Declared
 = 21     7     i := bar (j);  {assegnazione a globale}
             8     1:             {etichetta}
 / 21     9     j := bar (i);  {globale a parametro VAR}
-  21    10     GOTO 1;        {salto indietro}
*  21    11     RETURN; GOTO 1;{altri salti}
 % 21    12     i := bar (i);  {altro riferimento globale}
   21    13     j := bar (j);  {nessun riferimento globale}
   10    14    END;
         14  ----^306 Function Assignment Not Found

Symtab   14  Offset Length Variable - BAR
             -    2     24 Return offset, Frame length
             -    2      2 (func'n return):Integer
             +    4      2 J              :Integer VarP
             -   22     20 K              :Array

   10    15  BEGIN
   11    16   i := bar (i);
   00    17  END.

Symtab   17  Offset Length Variable
                  0     24 Return offset, Frame length
                  2      2 I              :Integer
                  4     20 K              :Array

             Errors  Warns In Pass One
                  1      2
```

Ogni pagina ha un'intestazione che contiene informazioni, come titolo e sottotitolo impostati rispettivamente mediante $TITLE e $SUBTITLE. Se questi metacomandi compaiono nelle prime linee del sorgente, hanno effetto nella pagina seguente. Il numero di pagina compare nella parte destra della prima linea di intestazione. In alcune versioni compare la data e l'ora nella parte destra, rispettivamente nella seconda e terza linea. Si puo' impostare il numero di pagina mediante $PAGE: <n>, oppure iniziare una nuova pagina con $PAGE+.

La quarta linea del listing contiene le etichette delle colonne. I contenuti delle prime tre colonne sono i seguenti:

1. Colonna JG

   Questa colonna contiene i caratteri generati per informazione utente. Gli indicatori di salto (sotto J) possono contenere uno dei seguenti caratteri:

   + salto in avanti (BREAK o GOTO ad un'etichetta non ancora incontrata)

   - salto indietro (CYCLE o GOTO ad un'etichetta gia' incontrata)

   * altri salti (RETURN oppure salti misti)

   I codici per le variabili globali (non locali alla procedura o funzione) compaiono sotto la colonna G:

   = assegnazione ad una variabile non locale

   / passaggio di una variabile non locale come parametro

   % una combinazione dei due

2. Colonna IC

   La colonna IC contiene informazioni sui livelli correnti di nidificazione. La cifra sotto "I" si riferisce al livello dell'identificatore, il quale cambia con le dichiarazioni di procedure e funzioni e con le dichiarazioni di record mediante WITH. La cifra nella colonna "C" si riferisce al livello di controllo istruzioni; questo numero cambia con la coppia BEGIN e END, come pure con CASE e END, REPEAT e UNTIL. Il numero in questa colonna e' utile per trovare le parole chiave END che mancano.

   Se una linea non e' usata dal compilatore, allora queste colonne sono vuote. Si puo' quindi determinare una porzione del sorgente commentata involontariamente oppure ignorata per via della coppia $IF $END.

3. Colonna Line

   La colonna Line riporta il numero di linea. Un file $INCLUDE riporta

la sua propria sequenza di numeri di linea. Se $LINE e' abilitato, il numero di linea ed il nome del file sorgente identificano gli errori runtime.

Due tipi di messaggi del compilatore compaiono nel listing: di errore e di avvertimento. Una compilazione con errori non puo' generare codice. Una compilazione con segnali di avvertimento puo' generare codice ma il programma puo' non funzionare correttamente. I segnali di avvertimento iniziano con la parola "Warning" ed un numero (vedere ad esempio, la linea 2 nel listing sopra riportato). I segnali di errore iniziano con un numero di errore (vedere linea 14). Nell'Appendice H, "Messaggi di Errore", si ha un elenco completo di tutti i messaggi di avvertimento e di errore.

Si possono sopprimere i messaggi di avvertimento per mezzo del metacomando $WARN-; cio' pero' non e' consigliato. Il metacomando $BRAVE+ invia i messaggi di avvertimento e di errore al terminale (e al file listing). Comunque, se su un singolo schermo ve ne sono troppi, i primi vengono fatti uscire.

L'indirizzo ove e' stato riconosciuto un errore viene indicato, nel listing, con il carattere '^'. Il messaggio puo' comparire alla destra o alla sinistra del carattere '^' ed e' preceduto da una linea di caratteri '-'.

Talvolta il compilatore non segnala un errore prima del listing della linea seguente. In questo caso il numero della linea non e' in sequenza. I caratteri di tabulazione sono ammessi nel sorgente e vengono riportati inalterati nel listing. Se lo spazio di tabulazione comprende otto colonne, l'indicatore di errore (^) e' generalmente corretto. Comunque, un errore verso la fine della linea puo' essere posizionato male se la linea contiene tabulazioni.

Se il compilatore incontra un errore per il quale non trova rimedio, emette il messaggio "Compiler Cannot Continue!". Questo messaggio compare se si verifica una delle seguenti condizioni:

- La parola chiave PROGRAM (o IMPLEMENTATION, INTERFACE o MODULE) non viene trovata oppure l'identificatore di programma, di modulo o unita' mancano.

- Il compilatore incontra un "end-of-file" non previsto.

- Il compilatore trova troppi errori; il numero massimo di errori per pagina e' impostato da $ERRORS (il default e' 25).

- Il livello dell'identificatore e' troppo nidificato (per il numero massimo di nidificazione si puo' vedere l'Appendice A, "Caratteristiche di Implementazione", nel manuale "Linguaggio Pascal Guida Utente").

Quando il compilatore, per qualunque ragione, non e' in grado di continuare, scrive semplicemente il resto del programma sul file listing, con un controllo limitato sugli errori.

SWITCH DI LINEA DI COMANDO

Molti dei metacomandi di debug e gestione errori descritti precedentemente nella sezione "Posizionamento e Ottimizzazione del Livello del Linguaggio" possono essere anche forniti come switch in fase di compilazione. Gli switch possono essere inseriti su linee di comando del compilatore oppure come risposta a prompt. La tabella 19-7 riporta i metacomandi che sono disponibili anche come switch. Per ulteriori informazioni sugli switch si puo' consultare il manuale "Linguaggio Pascal Guida Utente".

SWITCH	METACOMANDO	DESCRIZIONE
/A	$INDEXCK	Controlla che gli indici degli array e dei super array siano compresi nel campo di variabilita'.
/D	$DEBUG	Abilita tutti gli switch, compresi $ENTRY e $LINE.
/E	$ENTR	Genera chiamata di ingresso e uscita da procedure per il debugger.
/L	$LINE	Genera chiamata con numero di linea per il controllo degli errori.
/I	$INITCK	Controlla l'uso dei valori non inizializzati
/M	$MATHCK	Controlla gli errori matematici, come il superamento di capacita' e la divisione per zero.
/N	$NILCK	Controlla i valori di puntatore, compreso il NIL.
/Q	$DEBUG	Disabilita tutti gli switch, compresi $ENTRY e $LINE.
/R	$RANGECK	Controlla la validita' del subrange, comprese le assegnazioni.
/S	$STACKCK	Controlla il superamento di capacita' dello stack all'entrata di procedure e funzioni.
/T	$TAGCK	Controlla i campi etichettati in record con varianti.

Tabella 19-7 Switch di Linea di Comando

Lo switch /Q, in congiunzione con gli altri, fornisce un mezzo efficace per impostare la propria compilazione nel modo voluto. Prima di tutto occorre disabilitare tutti gli altri switch e poi abilitare in modo selettivo solo quelli desiderati.

A. DIAGRAMMI SINTATTICI DEL PASCAL

DIAGRAMMI SINTATTICI

I diagrammi delle pagine seguenti illustrano gli elementi fondamentali della sintassi del linguaggio Pascal. Sono elencati secondo l'ordine con cui vengono verosimilmente usati nello scrivere un programma. La diversa forma dei contorni ha il seguente significato:

Ovali

Indicano simboli o parole riservate del Pascal, che devono essere riportati esattamente come indicato.

Rettangoli

Indicano costruzioni di livello superiore che a loro volta contengono diagrammi sintattici.

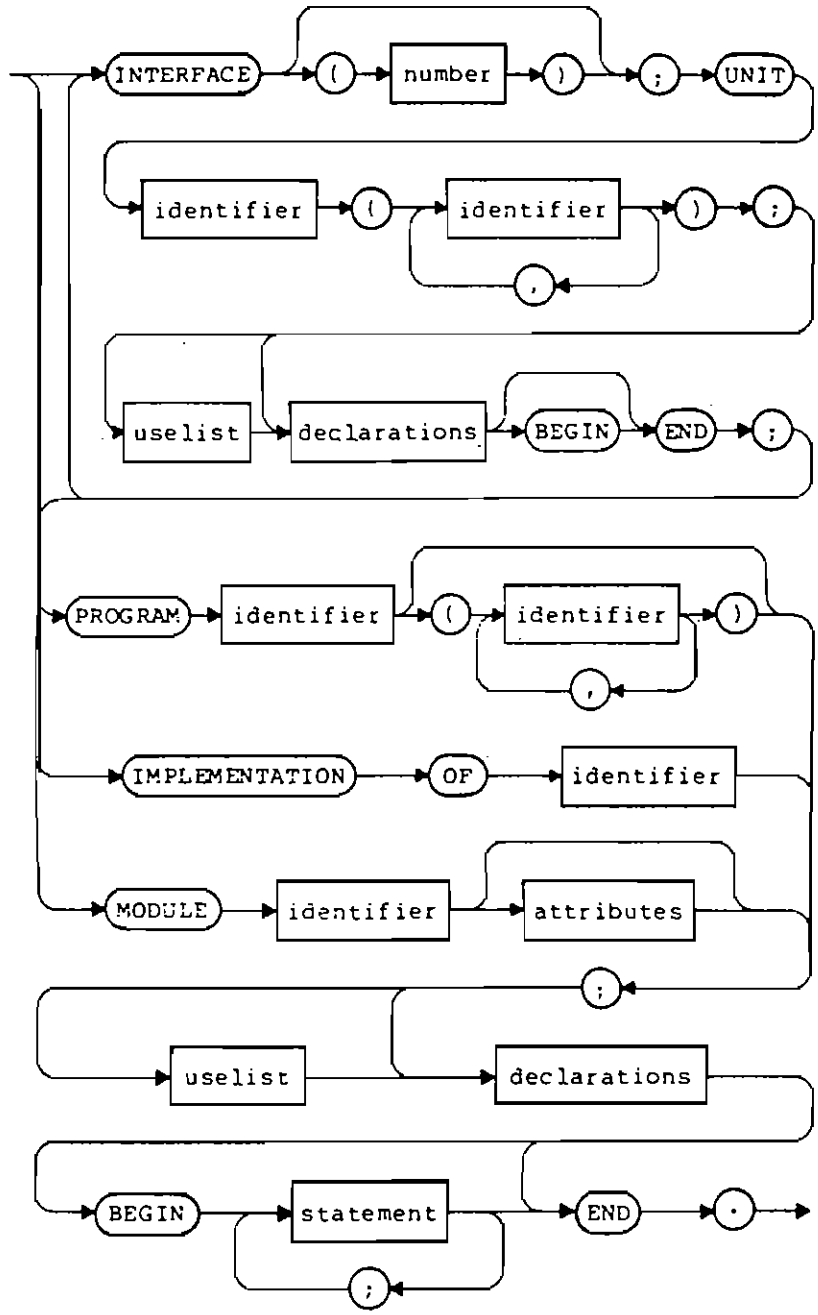
Cerchi

Indicano la punteggiatura richiesta la quale deve essere introdotta esattemente come indicato.

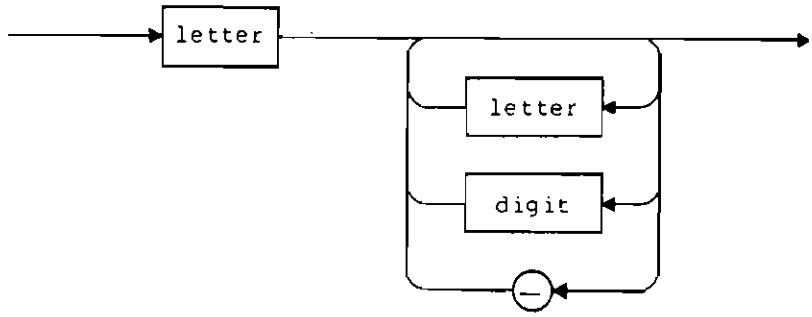
Frecce

Servono per indicare il percorso lungo il diagramma compresi eventuali loop (e cioe' ripetizioni di uno o piu' elementi di sintassi).

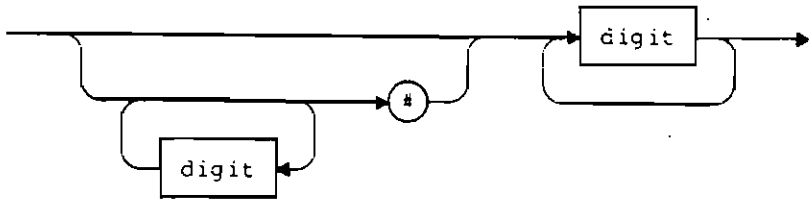
Source File


INTERFACE
(
number
)
;
UNIT
identifier
(
identifier
)
;
,
uselist
declarations
BEGIN
END
;
PROGRAM
identifier
(
identifier
)
,
IMPLEMENTATION
OF
identifier
MODULE
identifier
attributes
;
uselist
declarations
BEGIN
statement
END
.
;

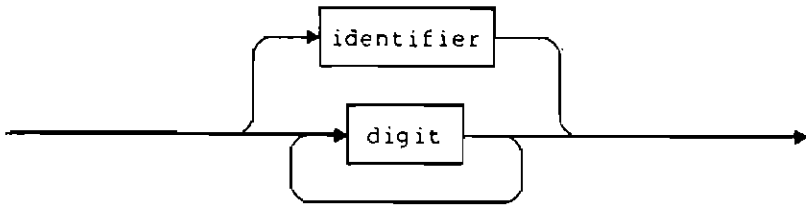
Identifier



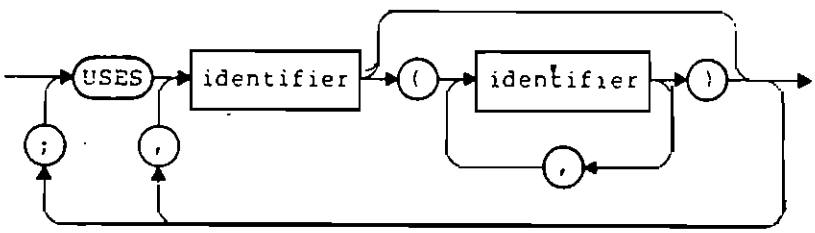
Number



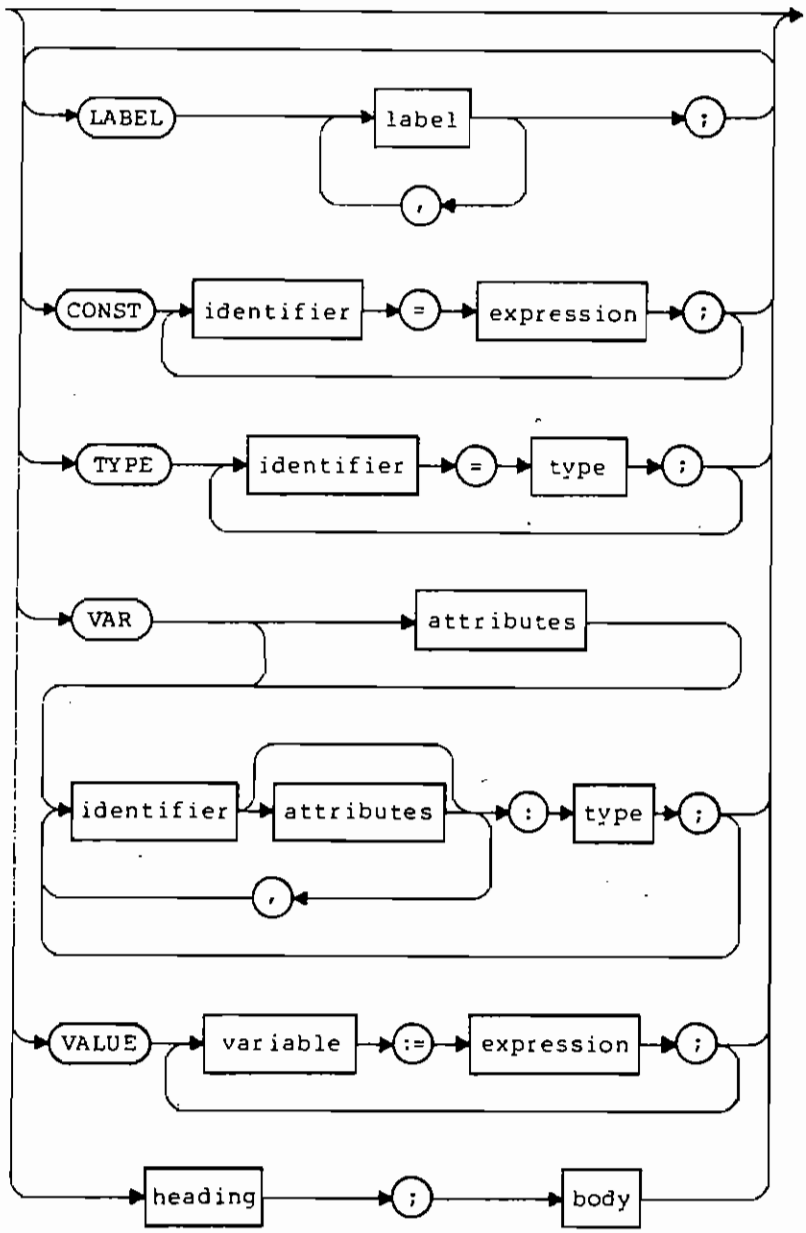
Label



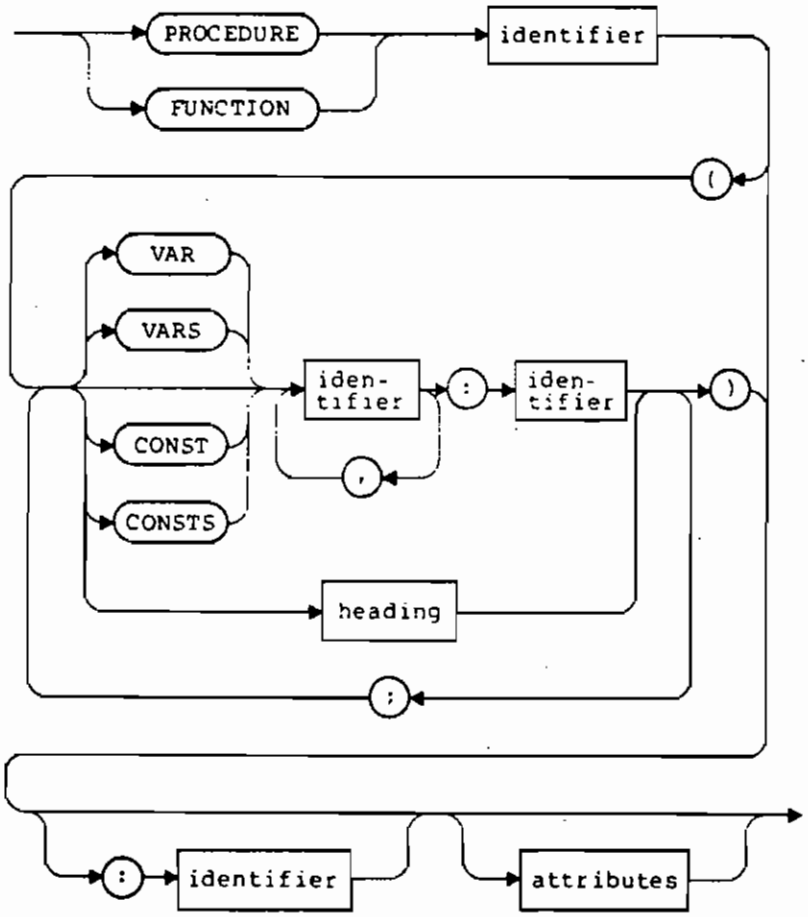
Uselist



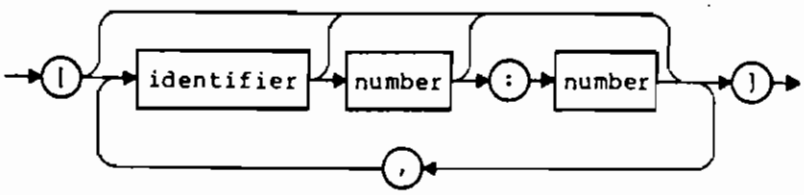
Declarations



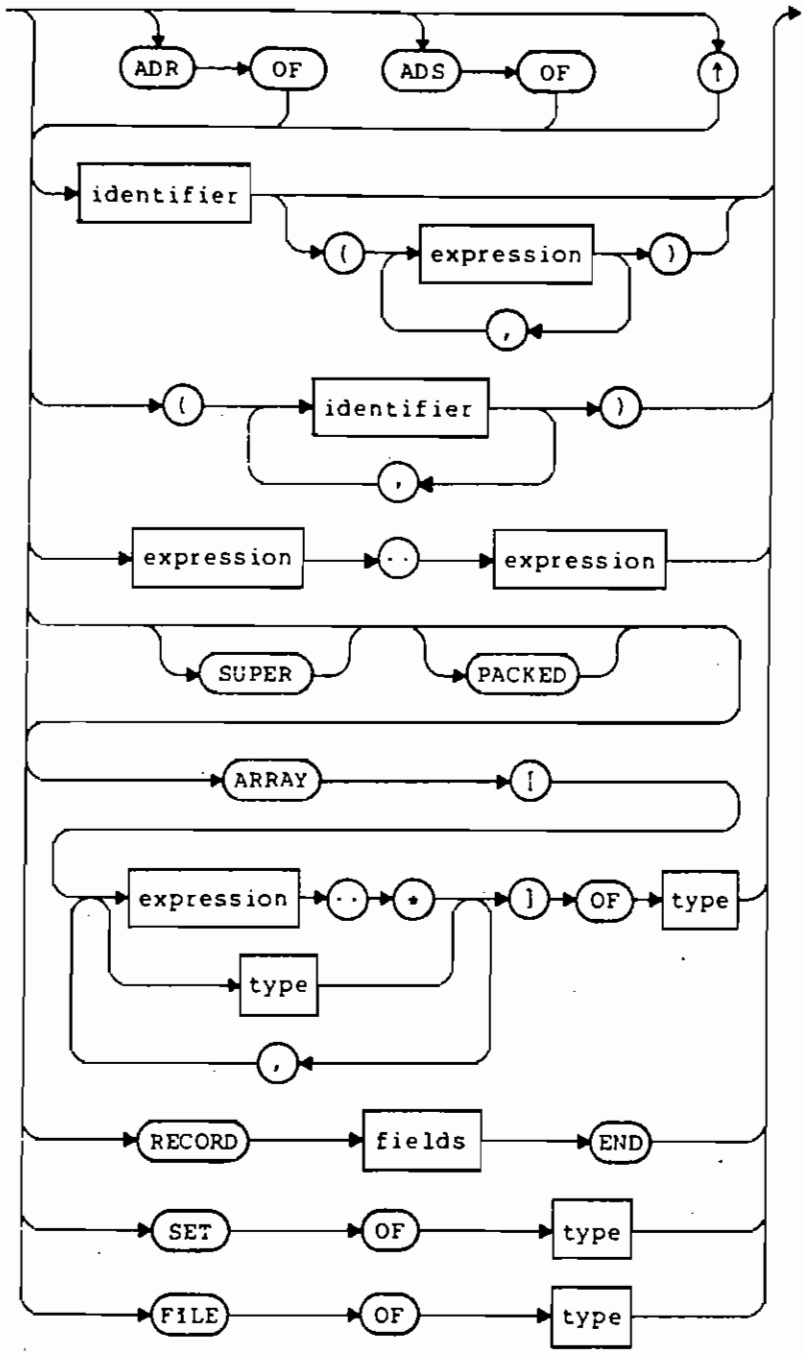
Heading



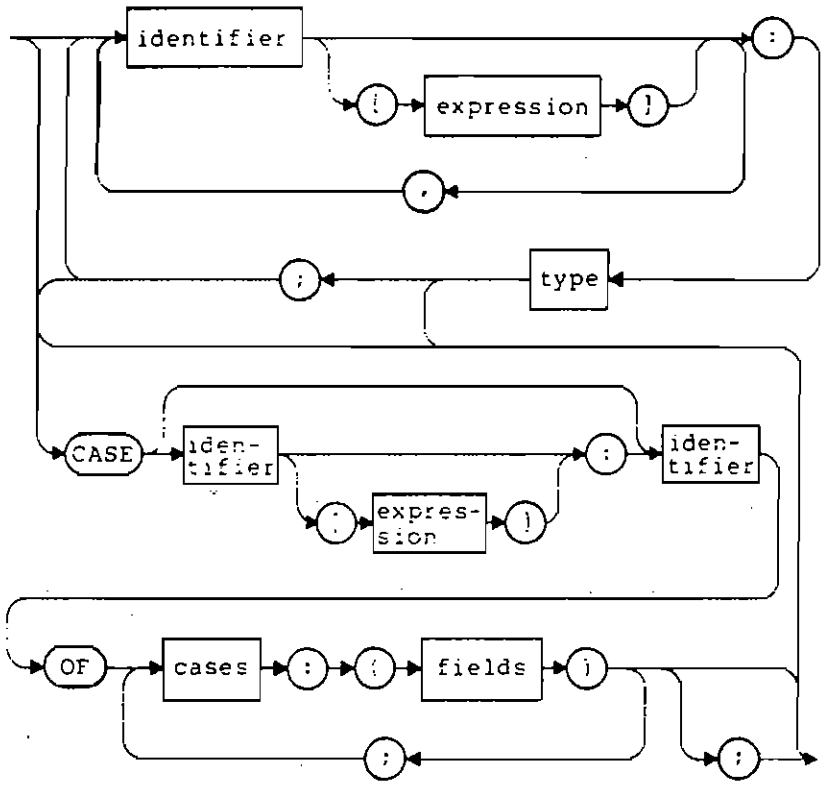
Attributes



Type


ADR
OF
ADS
OF
↑
identifier
(
expression
)
,
(
identifier
)
,
expression
..
expression
SUPER
PACKED
ARRAY
[
expression
..
*
]
OF
type
type
,
RECORD
fields
END
SET
OF
type
FILE
OF
type

Fields



Body

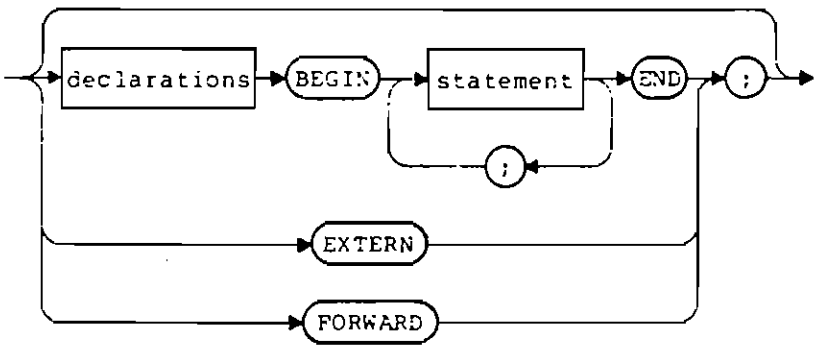
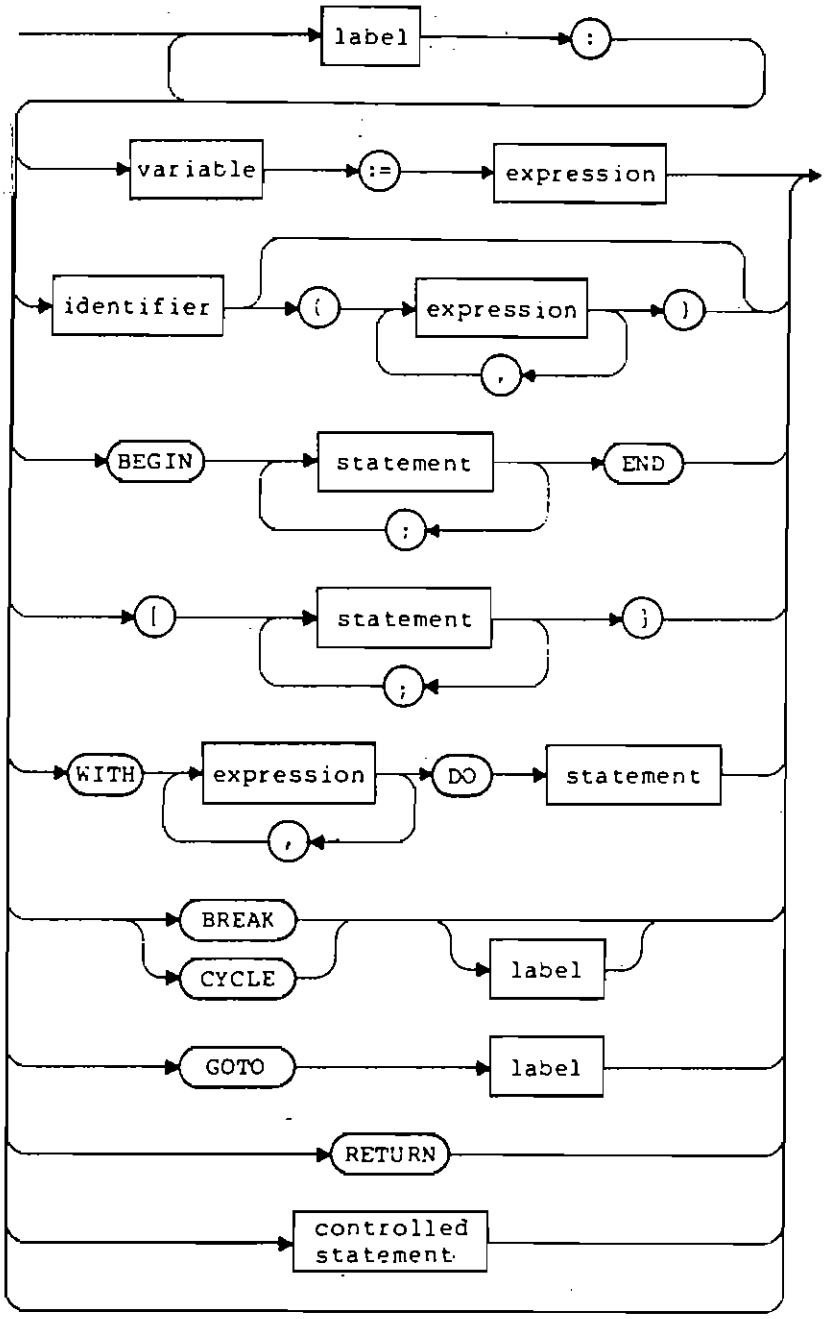
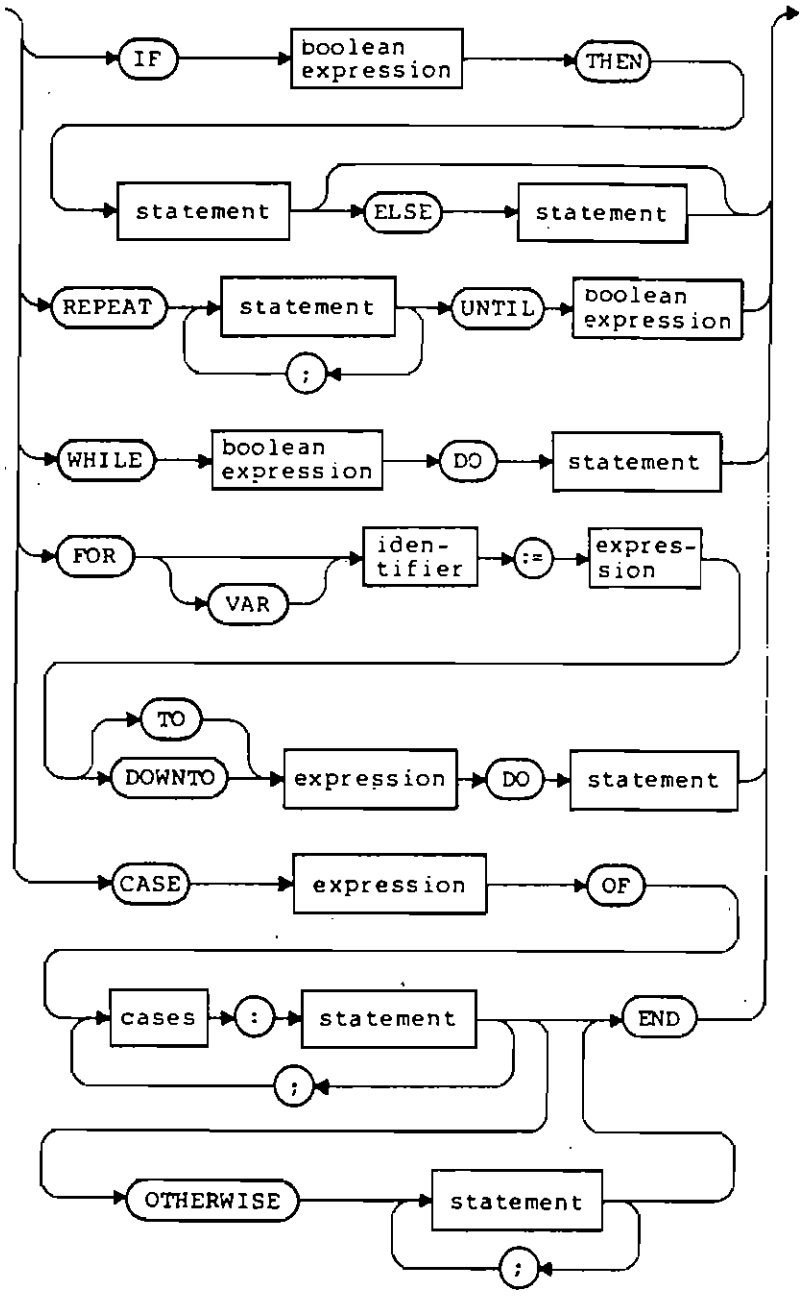


Fig. A-6

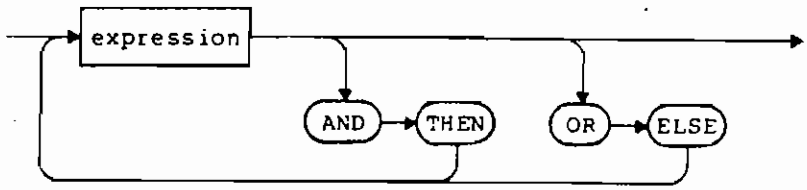
Statement


label
:
variable
:=
expression
identifier
(
expression
,
)
BEGIN
statement
;
END
[
statement
;
]
WITH
expression
,
DO
statement
BREAK
CYCLE
label
GOTO
label
RETURN
controlled
statement

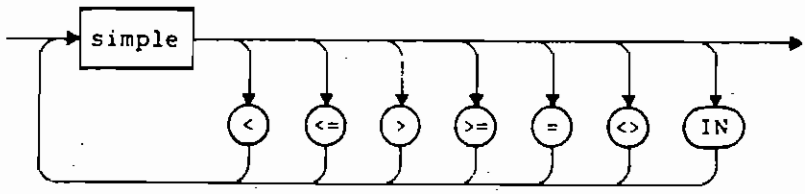
Controlled Statement



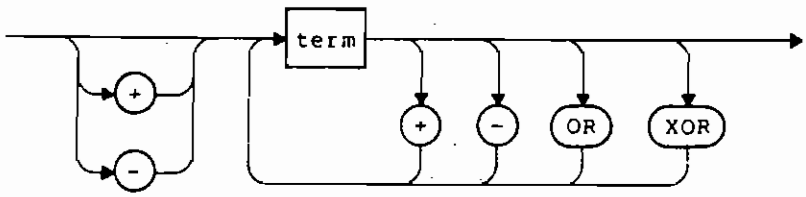
Boolean Expression



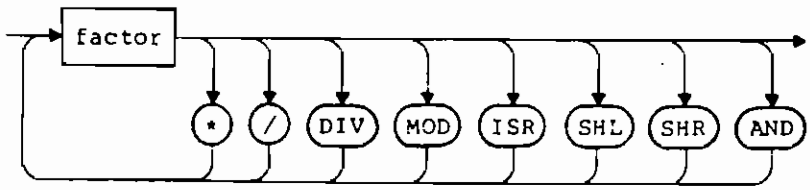
Expression



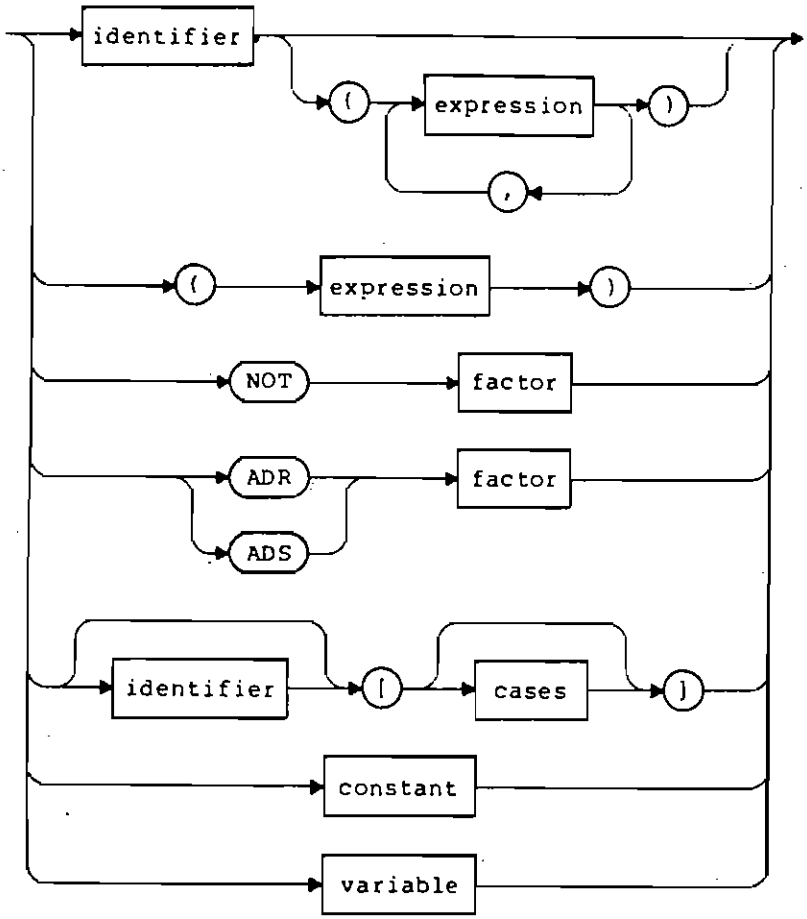
Simple



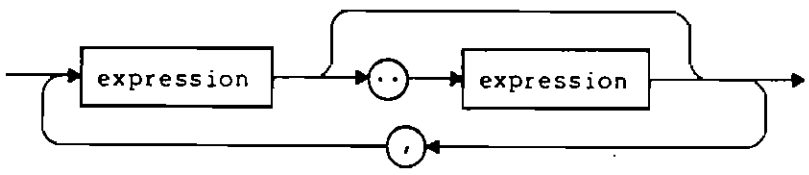
Term



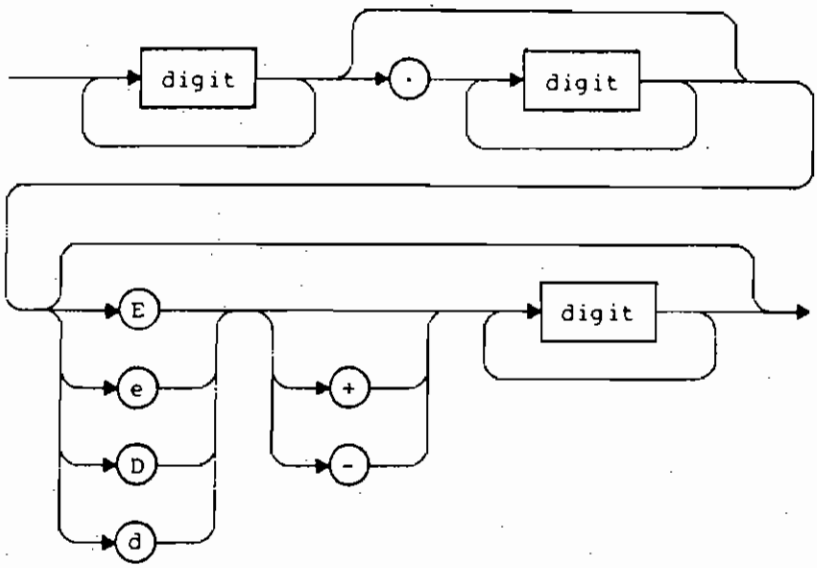
Factor



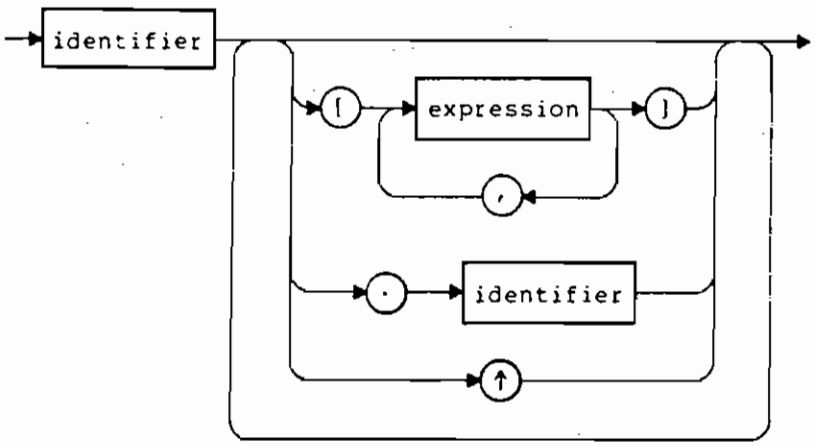
Cases



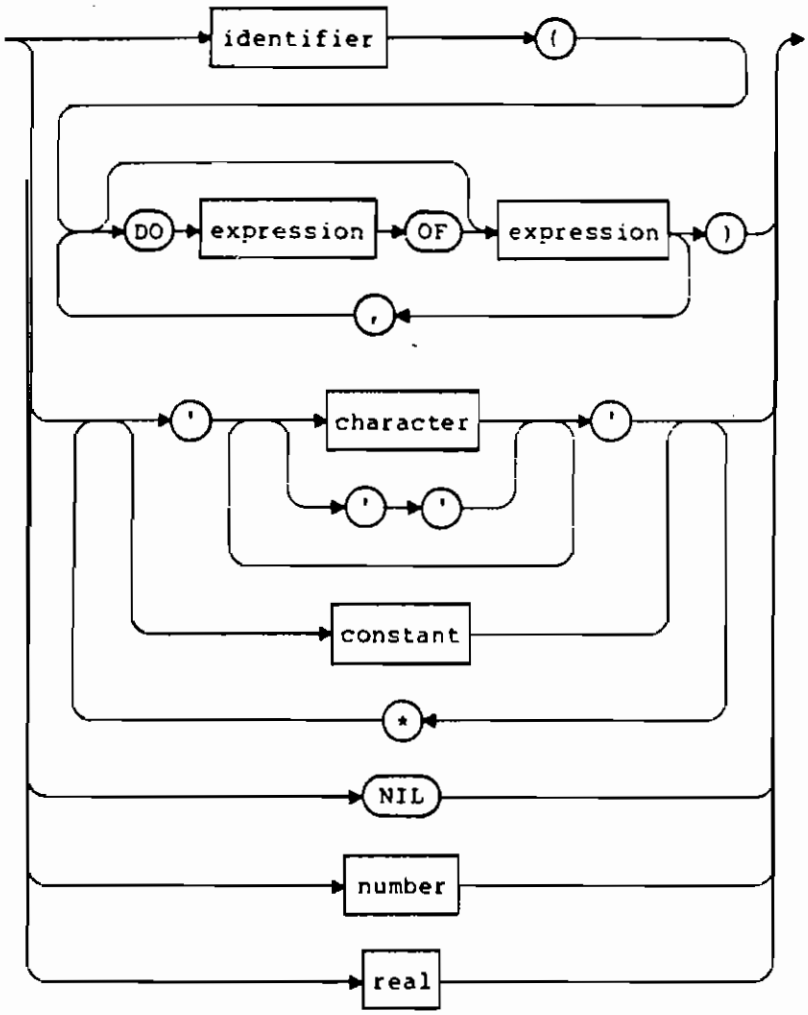
Real Number



Variable



Constant


identifier
(
DO
expression
OF
expression
)
,
'
character
'
'
'
constant
*
NIL
number
real

B. PRESTAZIONI DEL PASCAL E LO STANDARD ISO

SOMMARIO

Questa appendice descrive le differenze tra questa implementazione del Pascal e lo standard ISO, e riassume le caratteristiche del Pascal.

INDICE

IL PASCAL E LO STANDARD ISO

Lo standard ISO definisce un gran numero di condizioni di errore, ma dispone di un'implementazione particolare per la gestione degli errori documentando se l'errore non viene individuato. Qui di seguito sono descritti questi "errori non rilevati" ed altre differenze fra il Pascal e lo standard ISO. Un programma Pascal che vuole attenersi alle prescrizioni dello standard ISO deve includere entrambi i metacommandi $STANDARD e $DEBUG.

Il Pascal comprende le seguenti estensioni (di minore rilievo) rispetto agli attuali standard ISO/ANSI/IEEE:

- il punto interrogativo (?) in sostituzione della freccia in alto (^)
- il segno di sottolineatura (_) negli identificatori.

Tenendo conto del modo in cui il compilatore costruisce gli identificatori, le nuove parole riservate aggiunte ai livelli Esteso e di Sistema non possono essere usate come identificatori al livello Standard. La nuova direttiva, EXTERN, oltre a nuove funzioni predichiarate sono standard nel Pascal.

Nelle pagine seguenti sono riassunte le differenze fra il Pascal a livello Standard e gli attuali standard ISO/ANSI/IEEE:

1. Lo standard ISO richiede un separatore fra numeri e identificatori o parole chiave.

   In taluni casi, il Pascal non richiede un separatore fra numero e identificatore o parola chiave; ad esempio "100mod" e' accettato come "100 mod" senza generare errore.

2. Lo standard ISO non consente di passare un componente di una struttura PACKED come parametro di riferimento.

   Il Pascal invece consente specificatamente di passare un elemento CHAR di un PACKED ARRAY [1...n] OF CHAR come parametro di riferimento. Passare un'etichetta di campo come riferimento e' un errore non rilevato. Passare altri componenti impaccati, invece, genera normalmente un errore.

3. Lo standard ISO non include il carattere di marca di linea di file-testo nell'insieme dei valori CHAR.

   Il Pascal consente di utilizzare tutti i 256 valori a 8-bit come valori CHAR; con alcuni sistemi operativi un particolare carattere (ad esempio il ritorno carrello) funge anche da carattere di marca di linea.

4. Lo standard ISO richiede che venga assegnata una variante per tutti i possibili valori delle etichette.

   Il Pascal consente di non specificare tutti i valori di etichetta in una dichiarazione di record variante.

5. Lo standard ISO richiede che un identificatore abbia un solo significato in ciascun campo di validita'.

   In Pascal, se nello stesso campo di validita' si usa un identificatore e poi lo si ridefinisce, l'errore non viene rilevato. Nell'esempio seguente

   ```
   CONST X = Y;  VAR Y : CHAR;
   ```

   si hanno due significati per Y nello stesso campo di validita'. Generalmente il Pascal utilizza l'ultima definizione di un identificatore. Vi e' un caso ambiguo: quando si dichiara il tipo FOO in un campo di validita' e in un campo di validita' piu' interno TYPE P = ∧ FOO; FOO = TYPE. In questo caso l'intendimento del programmatore e' espresso in modo ambiguo. Il compilatore usa l'ultima definizione di FOO ed emette un segnale di avvertimento.

6. Lo standard ISO richiede che nelle procedure WRITE e WRITELN l'ampiezza del campo "M" sia un numero positivo.

   Il Pascal tratta M < 0 come se fosse M = ABS(M), ma l'espansione del campo avviene da destra anziche' da sinistra. M puo' anche essere zero quando non si vuole scrivere nulla. I parametri di file-testo READ(LN) e WRITE(LN) accettano sia il parametro M che N, ignorando quello non necessario. E' anche consentita la forma "V::N". Quando si scrive un INTEGER, il parametro N definisce la radice di output; in fase di lettura o scrittura di un tipo enumerato, il parametro N definisce il numero ordinale o ha l'opzione di identificatore costante.

7. Lo standard ISO non consente che una variabile creata con la forma lunga di NEW venga assegnata, usata in un'espressione o passata come parametro. Tuttavia questo controllo e' difficile in fase di compilazione e costoso in fase di esecuzione.

   Il Pascal consente l'assegnazione di queste variabili utilizzando la lunghezza effettiva della variabile oggetto. L'errore dello standard ISO non e' rilevato.

8. Lo standard ISO non consente di utilizzare la forma contratta di DISPOSE in una struttura allocata con la forma lunga di NEW. Lo standard ISO ammette soltanto che una variabile allocata con la forma lunga di NEW sia rilasciata con la forma lunga di DISPOSE; tutti i campi etichettati non devono essere mai modificati fra le due chiamate.

   Il Pascal consente di utilizzare le forma contratta di DISPOSE in una struttura allocata con la forma lunga di NEW e non esegue controlli su eventuali cambiamenti dei valori etichetta.

9. Lo standard ISO afferma che come conseguenza di un "cambiamento di variante" (ad esempio quando e' assegnato un nuovo valore di etichetta), tutti gli altri campi di variante risultano indefiniti.

   Pascal non definisce i campi non inizializzati all'atto

dell'assegnazione di una nuova etichetta, e cosi' non fa uso di un campo variante contenente un valore non definito.

10. Lo standard ISO non consente che una variabile con un riferimento attivo (cioè i record di un'istruzione WITH esecutiva o un effettivo parametro di riferimento) sia eliminata (se e' una variabile heap) o cambiata con una GET oppure con una PUT (se e' una variabile del buffer di un file).

    Pascal non rileva questi errori.

11. Lo standard ISO normalmente definice I MOD J come un errore se J < 0, ed il risultato di MOD e' sempre positivo anche quando I e' negativo.

    Il Pascal, per l'operatore MOD, non usa la semantica della nuova bozza dello standard. Pertanto i programmi scritti in vista della portabilita' non devono usare MOD se non quando ambedue gli operandi sono positivi.

12. Lo standard ISO non definisce gli array 'conformant'.

    Il Pascal non implementa il concetto di conformant array. I super array forniscono fondamentalmente le stesse funzionalita', ma in modo molto piu' flessibile. Il tipo super array del Pascal fornisce parametri di conformant array, come array di lunghezza dinamica allocati nello heap. I programmi scritti correttamente secondo lo standard ISO sono eseguiti correttamente, senza alcun cambiamento, sotto Pascal.

13. Lo standard ISO richiede che la variabile di controllo di un ciclo FOR sia locale nell'ambito del blocco contiguo. Ogni assegnazione a questa variabile di controllo genera un errore.

    Il Pascal consente di usare variabili non locali, ma devono essere STATIC; in tal modo la variabile di controllo di un ciclo FOR puo' essere sia locale che a livello PROGRAM. Inoltre il Pascal non considera un errore un'assegnazione alla variabile di controllo, se l'assegnazione avviene in una procedura o funzione richiamate all'interno dell'istruzione FOR.

14. Lo standard ISO richiede che l'argomento CHR sia INTEGER.

    Il Pascal accetta per CHR ogni tipo ordinale.

SOMMARIO DELLE PRESTAZIONI DEL PASCAL

Qui di seguito sono riassunte le estensioni del Pascal rispetto allo standard ISO. A meno di indicazione contraria, sono da intendersi tutte a livello Esteso.

CARATTERISTICHE SINTATTICHE E PRAGMATICHE

1. Il metalinguaggio (livello Standard)

$BRAVE
$DEBUG
$ENTRY
$ERRORS
$EXTEND
$GOTO
$INCLUDE
$INCONST
$INDEXCK
$INTICK
$IF $THEN $ELSE $END
$INTEGER
$LINE
$LINESIZE
$LIST
$MATHCK
$MESSAGE
$NILCK
$OCODE
$OPTBUG
$PAGE
$PAGEIF
$PAGESIZE
$POP
$PUSH
$RANGECK
$REAL
$ROM
$RUNTIME
$SIMPLE
$SIZE
$SKIP
$SPEED
$STACKCK
$STANDARD
$SUBTITLE
$SYNTAB
$SYSTEM
$TAGCK
$TITLE
$WARM

2. Listing aggiuntivo (livello Standard)

   a) segnalazioni di salti, globali, livello identificatori, livello controllo, intestazione, terminatori

   b) errori testuali e messaggi di avvertimento

3. Aggiunte sintattiche

   a) ! come commento a fine linea

   b) parentesi quadre equivalenti a BEGIN/END

4. Notazione non decimale dei numeri

   a) costanti numeriche con # oppure nn# (dove nn = 2..36)

   b) DECODE/READ assume la notazione #

   c) ENCODE/WRITE con N di 2, 8, 10, 16

5. CASE con campo di variabilita' esteso

   a) per le istruzioni CASE e record con varianti

   b) OTHERWISE per tutti gli altri valori del campo di variabilita'

   c) A..B per il campo di variabilita', dei valori

TIPI E FORME DI DATI

- Tipo WORD, funzione WRD, costante MAXWORD
- Tipi REAL4 e REAL8
- Tipo INTEGER4, costante MAXINT4
- Funzioni FLOAT4, ROUND4 e TRUNC4
- Tipi indirizzo (livello Sistema)
  - . tipi ed operatori ADR e ADS
  - . parametri VARS e CONSTS
- Tipi SUPER array
  - . parametri conformant
  - . variabili di heap a lunghezza dinamica
  - . super array multidimensionali
  - . tipi super STRING e LSTRING
- Tipo LSTRING, costante NULL, campo .LEN
- Spiazzamenti espliciti di byte nei record (livello Sistema)
- Parametri di riferimento CONST e CONSTS per costanti ed espressioni
- Costanti strutturate (array, record e set)
- Funzioni estese che ritornano ogni tipo assegnabile
- Selezione variabile sui valori ritornati dalle funzioni
- Attributi

EXTERN	PORT
EXTERNAL	PUBLIC
FORTRAN	PURE
INTERRUPT	READONLY
ORIGIN	STATIC

OPERATORI E FUNZIONI INTRINSECHE

- Operatori di livello Esteso:
  - . operatori di shift: SHL SHR 1SR
  - . operatori logici a livello di bit: AND OR NOT XOR
  - . operatori di insieme: < >
- Espressioni costanti:
  - . concatenazione di stringhe costanti tramite l'operatore *
  - . espressioni numeriche, ordinali, booleane in clausole di tipo
  - Altre funzioni costanti:

CHR	UPPER
DIV	WRD
HIBYTE	*
H1WORD	+
LOBYTE	-
LOWER	<
LOWORD	< =
MOD	< >
ORD	=
RETYPE	>
SIZEOF	> =

  - Funzioni intrinseche addizionali a livello Esteso:

ABORT	LOWORD
BYLONG	RESULT
BYWORD	S1ZEOF
DECODE	UPPER
ENCODE	H1WORD
EVAL	LOBYTE
H1BYTE	LOWER

  - Funzioni intrinseche addizionali a livello di Sistema:

FILLC	MOVESL
FILLSC	MOVESR
MOVEL	RETYPE
MOVER	

  - Funzioni intrinseche che agiscono su stringhe:
    - . per STRING o LSTRING: COPYSTR POSITN SCANEQ SCANNE
    - . solo per LSTR1NG: CONCAT INSERT DELETE COPYLST
  - Funzioni di libreria dell'MS-FORTRAN REAL (livello Standard)

- Funzioni di libreria del Pascal (livello Standard):

ALLHQQ	MARKAS
BEGOQQ	MEMAVL
BEGXQQ	PLYUQQ
DATE	PTYUQQ
DISBIN	RELEAS
ENDOQQ	SADDOK
ENDXQQ	SMULOK
ENABIN	TICS
FREECT	TIME
GTYUQQ	UADDOK
LADDOK	UMULOK
LMULOK	UNLOCK
LOCKED	VECTIN

FLUSSO DI CONTROLLO E CARATTERISTICHE DI STRUTTURA

- Istruzioni di flusso di controllo: BREAK, CYCLE e RETURN
- Operatori di controllo sequenziale: AND THEN e OR ELSE in IF, WHILE, REPEAT
- Loop esteso FOR: variabile FOR VAR
- Sezione VALUE per inizializzare variabili statiche
- Sezioni in ordine misto LABEL, CONST, TYPE, VAR, VALUE
- Moduli compilabili con attributi globali
- UNIT, INTERFACE e IMPLEMENTATION:
  - numeri di versione di interfaccia, controllo della versione
  - ridenominazione opzionale dei costituenti
  - garanzia dell'unicita' dell'unita' durante l'inizializzazione
  - inizializzazione di unita' opzionale

FILE E I/O A LIVELLO ESTESO

- Dichiarazione di lunghezza di linea di file-testo, TEXT (nnn)
- READ enumerato, booleano, puntatore, STRING, LSTRING
- WRITE enumerato, puntatore, LSTRING
- Valore negativo di M per giustificare a sinistra anziche' a destra
- File temporanei

- File con modo di accesso DIRECT, procedura SEEK
- Procedure ASSIGN, CLOSE, DISCARD, READSET, READFN
- Costanti e tipi FILEMODES, modo di accesso F.MODE
- Trappole di errore, modo di accesso F.TRAP e F.ERRS
- I/O enumerato con uso di identificatore come stringa

I/O A LIVELLO DI SISTEMA

L'estensione del Pascal rispetto allo standard ISO mette a disposizione il tipo FCBFQQ, del tutto equivalente ai tipi FILE.

C. QUESTO E ALTRI PASCAL

SOMMARIO

Questa appendice descrive le differenze di implementazione tra questo Pascal e altri linguaggi Pascal. Essa descrive anche le differenze tra questo Pascal e il Pascal UCSD.

INDICE

INTRODUZIONE

Al livello Standard il Pascal e' conforme all'attuale bozza dello standard ISO. Percio', in teoria, i programmi scritti in conformita' allo standard ISO sono portabili e possono essere compilati da un qualsiasi compilatore Pascal.

In pratica, pero', la maggior parte dei programmi Pascal e' scritta con alcune prestazioni non standard. In questi casi e' necessario modificare il file sorgente per adattarlo alle convenzioni usate in Pascal.

IMPLEMENTAZIONI DEL PASCAL

Le aree in cui si manifestano le differenze tra le varie implementazioni del Pascal possono essere raggruppate nelle seguenti categorie:

- I/O interattivo

  Il Pascal implementa la valutazione "lazy" per trattare l'I/O interattivo in modo naturale. Altri Pascal possono implementare questa prestazione in modi diversi. Per esempio alcuni sistemi richiedono un READLN iniziale.

- Trattamento delle stringhe

  Il Pascal ammette il tipo super array LSTRING per trattare efficientemente le stringhe di lunghezza variabile. Per il trattamento delle stringhe lo standard ISO fornisce, invece, le procedure PACK e UNPACK; altri Pascal presentano spesso alcuni ampliamenti alle capacita' di trattamento delle stringhe prescritte dallo standard.

- Controlli del compilatore

  I controlli di compilazione implementati in forma di switch di linea di comando oppure come comandi nell'ambito di commenti del sorgente, possono variare da Pascal a Pascal. Per garantire la portabilita' occorre eliminare tutti i controlli inclusi nei commenti.

- Dimensione massima del set

  La dimensione massima del set varia da Pascal a Pascal. Alcuni Pascal limitano la dimensione del set a 16 o 64 elementi. Nel Pascal i set possono contenere fino a 256 elementi. Cio' consente di supportare il SET OF CHAR.

- Compatibilita' dei tipi

  La rigidita' delle regole sulla compatibilita' fra tipi e' variabile. In alcuni Pascal, tipi strutturalmente equivalenti con nomi

differenti sono compatibili; in altri (e nello standard ISO) non lo sono.

- Istruzioni GOTO fuori dei blocchi

  Alcuni Pascal non permettono istruzioni GOTO al di fuori dei blocchi, queste invece sono permesse nel Pascal.

- Gestione dello heap

  Anziche' usare le procedure NEW e DISPOSE per il trattamento della allocazione dinamica della memoria, alcuni Pascal usano le procedure MARK e RELEASE. Il Pascal ammette ambedue i metodi (MARKAS e RELEAS sono i due nomi usati nel Pascal per MARK e RELEASE).

- OTHERWISE nelle istruzioni CASE e nei record con varianti

  Se in un'istruzione CASE viene omessa la parola OTHERWISE, il controllo non viene passato alla successiva istruzione eseguibile, come avviene in qualche altro Pascal esteso. Inoltre alcuni altri Pascal usano le parole ELSE o OTHERS invece di OTHERWISE.

- Assegnazione di nomi ai file

  La procedura ASSIGN in PASCAL assegna ad un file un nome a livello di sistema operativo. Alcuni altri Pascal usano un secondo parametro di RESET e REWRITE per un nome di file.

- Compilazioni separate

  La maggior parte dei Pascal non accetta la direttiva EXTERN (o EXTERNAL) per procedure e funzioni. I concetti di MODULE e/o INTERFACE e IMPLEMENTATION sono supportati in molti Pascal, anche se la sintassi puo' variare. Alcuni Pascal non supportano variabili PUBLIC e EXTERN ma possono usare un approccio FORTRAN COMMON. In quest'ultimo caso, per garantire la portabilita' si devono assegnare tutte le variabili globali in una sezione VAR del Pascal, usando [PUBLIC] nel PROGRAM e [EXTERN] nel MODULE, e includere in ciascuna ($INCLUDE) la stessa dichiarazione di variabile.

- Parametri di programma

  Alcuni Pascal ignorano parametri di programma. In alcuni Pascal tutti i file devono essere parametri di programma.

- Parametri procedurali

  Parecchi Pascal non consentono di passare procedure e funzioni come parametri. Molti non permettono di passare alcuna procedura o funzione predichiarata.

IL PASCAL E IL PASCAL UCSD

Dato che il Pascal UCSD e' uno dei piu' diffusi Pascal nel campo dei microcomputer, la necessita' di conversione di UCSD in Pascal, e viceversa, si presenta con notevole frequenza. Percio' in questa sezione vengono esaminate le differenze e le somiglianze fra i due Pascal.

Il Pascal incorpora, in una forma o nell'altra, molte delle estensioni dell'UCSD. Nella tabella C-1 le estensioni UCSD sono confrontate con quelle disponibili nel Pascal.

ESTENSIONI UCSD	EQUIVALENTI DEL PASCAL
ATAN	ARCTAN
BLOCKREAD	GETUQQ
BLCOKWRITE	PUTUQQ
CLOSE	CLOSE
CLOSE (F, LOCK)	CLOSE (F)
CLOSE (F, PURGE)	DISCARD (F)
CONCAT	CONCAT
COPY	COPYLST o MOVEL
DELETE	DELETE
EXIT	RETURN o GOTO
FILLCHAR	FILLC e FILLSC
HALT	ENDXQQ
INSERT	INSERT
IORESULT, $I	campi ERRS e TRAP
LENGTH	.LEN e STR [0]
LOG	LNDRQQ
MARK	MARKAS
MEMAVAIL	MEMAVL
MOVELEFT	MOVEL e MOVESL
MOVERIGHT	MOVER e MOVESR
POS	POSITN
RELEASE	RELEAS
SCAN	SCANEQ e SCANNE
SEEK	SEEK
SIZEOF	SIZEOF
STR	ENCODE
STRING [n]	LSTRING (n)
UNIT	UNIT
File senza tipo	tipo FCBFQQ

Tabella C-1 Pascal e Pascal UCSD

Nelle note successive vengono fatti dei confronti fra punti di particolare interesse:

- Il tipo STRING [n] dell'UCSD e' logicamente simile al tipo LSTRING (n) del Pascal. Ambedue, infatti, contengono la lunghezza di una stringa a lunghezza variabile nell'elemento zero di un ARRAY OF CHAR.

- Il Pascal UCSD utilizza MARK e RELEASE per allocare le variabili puntatore nell'area heap. Gli altri Pascal di solito usano NEW e DISPOSE. Questo Pascal accetta ambedue i metodi di allocazione dinamica della memoria.

- Le unita' Pascal sono simili alle unita' UCSD, con le seguenti eccezioni: nel Pascal un INTERFACE deve apparire per primo in ogni unita' compilativa che lo utilizza; poiche' il Pascal UCSD ha un suo particolare file system, si puo' utilizzare il nome del'unita' per trovare il nome del file di interfaccia nel modo standard.

  Il Pascal richiede una lista di tutti gli identificatori trasferiti dall'unita' nella clausola UNIT, e la considera opzionale in una clausola USES. In una clausola USES, per evitare conflitti, possono essere dati identificatori diversi.

  Infine, il Pascal fornisce il codice di inizializzazione dell'unita' ed il controllo della versione di interfaccia. Nessuna di queste due prestazioni e' presente nel Pascal UCSD.

- CONCAT e' una funzione del Pascal UCSD; nel Pascal e' una procedura.

- Nel Pascal UCSD, se viene eseguita un'istruzione CASE che non seleziona nessuna istruzione, si passa all'istruzione successiva. Nel Pascal per ottenere questo effetto bisogna includere una clausole OTHERWISE vuota.

- Il Pascal UCSD permette di usare le funzioni EOF(F) e EOLN(F) anche su un file chiuso; nel Pascal invece questo genera un errore.

- Il Pascal UCSD consente il confronto fra record e array con il segno di uguale (=) e di diverso (< >). Nel Pascal occorre trascrivere (RETYPE) record e array in tipi STRING di lunghezza uguale, e quindi confrontarli come stringhe.

D. CODICI DEI CARATTERI ASCII

CODICI DEI CARATTERI ASCII

DEC	ESA	CAR	DEC	ESA	CAR
000	00H	NUL	032	20H	SPACE
001	01H	SOH	033	21H	!
002	02H	STX	034	22H	"
003	03H	ETX	035	23H	#
004	04H	EOT	036	24H	$
005	05H	ENQ	037	25H	%
006	06H	ACK	038	26H	&
007	07H	BEL	039	27H	'
008	0BH	BS	040	28H	(
009	09H	HT	041	29H	)
010	0AH	LF	042	2AH	*
011	0BH	VT	043	2BH	+
012	0CH	FF	044	2CH	,
013	0DH	CR	045	2DH	-
014	0EH	SO	046	2EH	.
015	0FH	SI	047	2FH	/
016	10H	DLE	048	30H	0
017	11H	DC1	049	31H	1
01B	12H	DC2	050	32H	2
019	13H	DC3	051	33H	3
020	14H	DC4	052	34H	4
021	15H	NAK	053	35H	5
022	16H	SYN	054	36H	6
023	17H	ETB	055	37H	7
024	18H	CAN	056	38H	8
025	19H	EM	057	39H	9
026	1AH	SUB	058	3AH	:
027	1BH	ESCAPE	059	3BH	;
028	1CH	FS	060	3CH	<
029	1DH	GS	061	3DH	=
030	1EH	RS	062	3EH	>
031	1FH	US	063	3FH	?

DEC	ESA	CAR	DEC	ESA	CAR
064	40H	@	096	60H	'
065	41H	A	097	61H	a
066	42H	B	098	62H	b
067	43H	C	099	63H	c
068	44H	D	100	64H	d
069	45H	E	101	65H	e
070	46H	F	102	66H	f
071	47H	G	103	67H	g
072	48H	H	104	68H	h
073	49H	I	105	69H	i
074	4AH	J	106	6AH	j
075	4BH	K	107	6BH	k
076	4CH	L	108	6CH	l
077	4DH	M	109	6DH	m
078	4EH	N	110	6EH	n
079	4FH	O	111	6FH	o
080	50H	P	112	70H	p
081	51H	Q	113	71H	q
082	52H	R	114	72H	r
083	53H	S	115	73H	s
084	54H	T	116	74H	t
085	55H	U	117	75H	u
086	56H	V	118	76H	v
087	57H	W	119	77H	w
088	58H	X	120	78H	x
089	59H	Y	121	79H	y
090	5AH	Z	122	7AH	z
091	5BH	[	123	7BH	{
092	5CH	\	124	7CH	|
093	5DH	]	125	7DH	}
094	5EH	^	126	7EH	~
095	5FH	_	127	7FH	DEL

DEC = decimale, ESA = esadecimale (H), CAR = carattere, LF = salto riga, FF = salto pagina, CR = ritorno a capo, DEL = cancella carattere.

E. ELENCO DELLE PAROLE RISERVATE DEL PASCAL

ELENCO DELLE PAROLE RISERVATE DEL PASCAL

Parole riservate a livello standard:

AND	NIL
ARRAY	NOT
BEGIN	OF
CASE	OR
CONST	PACKED
DIV	PROCEDURE
DO	PROGRAM
DOWNTO	RECORD
ELSE	REPEAT
END	SET
FILE	THEN
FOR	TO
FUNCTION	TYPE
GOTO	UNTIL
IF	VAR
IN	WHILE
LABEL	WITH
MOD	

Parole riservate addizionali a livello esteso:

BREAK	RETURN
CONSTS	SHL
CYCLE	SHR
IMPLEMENTATION	UNIT
INTERFACE	USES
ISR	VALUE
MODULE	VARS
OTHERWISE	XOR

Parole riservate addizionali a livello sistema:

ADR
ADS

Nomi di attributi:

EXTERN	PORT
EXTERNAL	PUBLIC
FORTRAN	PURE
INTERRUPT	READONLY
ORIGIN	STATIC

Nomi di direttive:

EXTERN
EXTERNAL
FORWARD

Logicamente le direttive sono parole riservate. Poiche' nel Pascal ISO sono consentite direttive addizionali, esse sono tutte comprese nel livello standard. EXTERN e' sia un attributo che una direttiva. EXTERNAL e' sinonimo di EXTERN in ambedue i casi. Questo consente la compatibilita' con un notevole numero di altri Pascal.

F. RIEPILOGO DELLE PROCEDURE E FUNZIONI DISPONIBILI

SOMMARIO

Questa appendice fornisce un elenco approssimativo di tutte le procedure e funzioni disponibili, insieme al nome del gruppo nel quale esse sono presentate nella sezione "Categorie delle Procedure e Funzioni Disponibili" in Capitolo 6.

INDICE

RIEPILOGO DELLE PROCEDURE E FUNZIONI DISPONIBILI

NOME	DESCRIZIONE	CATEGORIA
ABORT	Pone termine al programma	Livello Esteso
ABS	Funzione valore assoluto	Aritmetica
ACDRQQ	Funzione arco coseno REAL8	Aritmetica
ACSRQQ	Funzione arco coseno REAL4	Aritmetica
AIDRQQ	Funzione troncamento REAL8	Aritmetica
AISRQQ	Funzione troncamento REAL4	Aritmetica
ALLHQQ	Alloca un elemento nello heap	Libreria
ANDRQQ	Arrotondamento a zero REAL8	Aritmetica
ANSRQQ	Arrotondamento a zero REAL4	Aritmetica
ARCTAN	Funzione arcotangente	Aritmetica
ASDRQQ	Funzione arcoseno REAL8	Aritmetica
ASSRQQ	Funzione arcoseno REAL4	Aritmetica
ASSIGN	Assegna nome file	File system
ATDRQQ	Arcotangente (A/B) REAL8	Aritmetica
ATSRQQ	Arcotangente (A/B) REAL4	Aritmetica
A2DRQQ	Funzione arcotangente REAL8	Aritmetica
A2SRQQ	Funzione arcotangente REAL4	Aritmetica
BEGOQQ	Inizializza l'utente	Libreria
BEGXQQ	Inizializzazione generale	Libreria
BYLONG	WORD o INTEGER in INTEGER4	Livello Esteso
BYWORD	Dispone i byte in word	Livello Esteso
CHDRQQ	Coseno iperbolico REAL8	Aritmetica
CHR	Assume il carattere ASCII del valore	Conversione dati
CHSRQQ	Coseno iperbolico REAL4	Aritmetica
CLOSE	Chiude il file	File system
CNDRQQ	Funzione coseno REAL8	Aritmetica
CNSRQQ	Funzione coseno REAL4	Aritmetica
CONCAT	Concatena LSTRING	Stringa
COPYLST	Copia su LSTRING	Stringa
COPYSTR	Copia su STRING	Stringa
COS	Funzione coseno	Aritmetica
DATE	Funzione data	Libreria
DECODE	Decodifica LSTRING in variabile	Livello Esteso
DELETE	Rimuove porzioni di LSTRING	Stringa
DISBIN	Disabilita le interruzioni	Libreria
DISCARD	Chiude e cancella un file	File system
DISPOSE	Elimina un elemento nello heap	Allocazione dinamica
ENABIN	Abilita le interruzioni	Libreria
ENCODE	Codifica un espressione in LSTRING	Livello Esteso
ENDOQQ	Terminazione di utente	Libreria
ENDXQQ	Terminazione di programma	Libreria
EOF	End-Of-File booleano	File system

Tabella F-1 Procedure e Funzioni

NOME	DESCRIZIONE	CATEGORIA
EOLN	End-Of-Line booleano	File system
EVAL	Valuta le funzioni	Livello Esteso
EXDRQQ	Funzione esponenziale REAL8	Aritmetica
EXP	Funzione esponenziale	Aritmetica
EXSRQQ	Funzione esponenziale REAL4	Aritmetica
FILLC	Riempie l'area con C, relativa	Livello Sistema
FILLSC	Riempie l'area con C, segmentata	Livello Sistema
FLOAT	Converte INTEGER in REAL	Conversione dati
FLOAT4	Converte INTEGER4 in REAL	Conversione dati
FREECT	Da' il calcolo dei blocchi liberi	Libreria
GET	GET il prossimo componente del file	File system
GTYUQQ	Input terminale diretto	Libreria
HIBYTE	Ottiene un high BYTE	Livello Esteso
HIWORD	Ottiene un high WORD	Livello Esteso
INSERT	Inserisce una stringa	Stringa
LADDOK	Controllo addizione 32-bit con segno	Libreria
LDDRQQ	Funzione logaritmo in base 10 REAL8	Aritmetica
LDSRQQ	Funzione logaritmo in base 10 REAL4	Aritmetica
LMULOK	Controllo moltiplicazione 32-bit con segno	Aritmetica
LN	Funzione logaritmo naturale	Aritmetica
LNDRQQ	Logaritmo naturale REAL8	Aritmetica
LNSRQQ	Logaritmo naturale REAL4	Aritmetica
LOBYTE	Ottiene low BYTE	Livello Esteso
LOCKED	Stato di risorse "locked"	Libreria
LOWER	Ottiene il limite inferiore	Livello Esteso
LOWORD	Ottiene low WORD	Livello Esteso
MARKAS	Marca i limiti dello heap	Libreria
MDDRQQ	Funzione modulo REAL8	Aritmetica
MDSRQQ	Funzione modulo REAL4	Aritmetica
MEMAVL	Memoria disponibile	Libreria
MNDRQQ	Funzione minimo REAL8	Aritmetica
MNSRQQ	Funzione minimo REAL4	Aritmetica
MOVEL	Sposta byte a sinistra, relativo	Livello Sistema
MOVER	Sposta byte a destra, relativo	Livello Sistema
MOVESL	Sposte byte a sinistra, segmentato	Livello Sistema
MOVESR	Sposta byte a destra, segmentato	Livello Sistema
MXDRQQ	Funzione massimo REAL8	Aritmetica
MXSRQQ	Funzione massimo REAL4	Aritmetica
NEW	Alloca nuovi elementi nello heap	Allocazione dinamica
ODD	Funzione Odd booleano	Conversione dati
ORD	Ottiene valore ordinale	Conversione dati
PACK	Array CHAR Packed	Conversione dati
PAGE	Scrive una nuova pagina	File system
PIDRQQ	REAL8 elevato a potenza di INTEGER	Aritmetica

Tabella F-1. Procedure e Funzioni (continuazione)

NOME	DESCRIZIONE	CATEGORIA
PISRQQ	REAL4 elevato a potenza di INTEGER	Aritmetica
PLYUQQ	End line terminale diretto	Libreria
POSITN	Cerca la posizione di una sottostringa	Stringa
PRDRQQ	REAL8 elevato a potenza REAL8	Aritmetica
PRED	Funzione predecessore	Conversione dati
PRSRQQ	REAL4 elevato a potenza REAL4	Aritmetica
PTYUQQ	Output di terminale diretto	Libreria
PUT	Mette un valore nel file	File system
READ	Legge un file	File system
READFN	Legge un nome di un file	File system
READLN	Legge il file alla fina della riga	File system
READSET	Legge il set	File system
RELEAS	Rilascia lo spazio nello heap	Libreria
RESET	Predispone un file per la lettura	File system
RESULT	Restituisce il risultato di una funzione	Livello Esteso
RETYPE	Forza un'espressione a tipo	Livello Sistema
REWRITE	Predispone un file per la scrittura	File system
ROUND	Arrotondamento REAL	Conversione dati
ROUND4	Arrotondamento INTEGER4	Conversione dati
SADDOK	Controllo addizione 16-bit con segno	Libreria
SCANEQ	Cerca fino a trovare il carattere	Stringa
SCANNE	Cerca finchè non trova il carattere	Stringa
SEEK	Posizionamento al record del file in modo diretto	File system
SHDRQQ	Seno iperbolico REAL8	Aritmetica
SHSRQQ	Seno iperbolico REAL4	Aritmetica
SIN	Funzione seno	Aritmetica
SIZEOF	Rileva la dimensione delle struttura	Livello Esteso
SMULOK	Controllo moltiplicazione 16-bit con segno	Libreria
SNDRQQ	Funzione seno REAL8	Aritmetica
SNSRQQ	Funzione seno REAL4	Aritmetica
SQR	Funzione elevazione al quadrato	Aritmetica
SQRT	Funzione radice quadrata	Aritmetica
SRDRQQ	Radice quadrata REAL8	Aritmetica
SRSRQQ	Radice quadrata REAL4	Aritmetica
SUCC	Funzione successore	Conversione dati
THDRQQ	Tangente iperbolica REAL8	Aritmetica
THSRQQ	Tangente iperbolica REAL4	Aritmetica
TICS	Ora nelle unita' arbitrarie	Libreria
TIME	Ora nella funzione giorno	Libreria
TNDRQQ	Funzione tangente REAL8	Aritmetica

Tabella F-1 Procedure e Funzioni (cont.)

NOME	DESCRIZIONE	CATEGORIA
TNSRQQ	Funzione tangente REAL4	Aritmetica
TRUNC	Troncamento REAL	Conversione dati
TRUNC4	Troncamento INTEGER4	Conversione dati
UADDOK	Controllo addizione senza segno	Libreria
UMULOK	Controllo moltiplicazione senza segno	Libreria
UNLOCK	Sblocca una risorsa	Libreria
UNPACK	STRING non impaccata in array	Conversione dati
UPPER	Raggiungi i limite superiore	Livello Esteso
VECTIN	Definisce il vettore d'interruzioni	Libreria
WRD	Conversione al valore WORD	Conversione dati
WRITE	Scrive su file	File system
WRITELN	Scrive una linea su file	File system

Tabella F-1 Procedure e Funzioni (cont.)

G. RIEPILOGO DEI METACOMANDI PASCAL

SOMMARIO

Questa appendice fornisce una singola lista alfabetica di tutti i metacomandi descritti in "Metacomandi", Capitolo 19. I valori di default, se ve ne sono, sono indicati dopo i metacomandi.

INDICE

RIEPILOGO DEI METACOMANDI PASCAL

METACOMANDO	AZIONE
$BRAVE+	Invia messaggi al video del terminale
$DEBUG-	Disattiva tutti i controlli di errore (CK)
$ENTRY-	Genera per il debugger le chiamate di entrata e uscita dalla procedura
$ERRORS:25	Definisce il numero di errori ammessi per pagina
$EXTEND	Aggiunge prestazioni al livello Esteso
$GOTO-	I flag sui GOTO vengono considerati "pericolosi"
$IF <costante> $THEN <testo1> $ELSE <testo2> $END	Consente la compilazione condizionale del sorgente <testo1> se la <costante> è maggiore di zero
$INCLUDE: '<file>'	Commuta la compilazione sul file specificato
$INCONST: <testo>	Consente l'assegnazione interattiva di valori costanti durante la fase di compilazione
$INDEXCK+	Controlla che i valori degli indici dell'array non siano fuori del campo di variabilita'
$INITCK-	Controlla che non vengano usati valori non inizializzati
$INTEGER	Definisce la lunghezza del tipo INTEGER
$LINE-	Genera le chiamate di numerazione righe per il debugger
$LINESIZE: 79	Definisce la definizione del listing del sorgente

Tabella G-1 Metacomandi Pascal

METACOMANDO	FUNZIONE
$LIST+	Attiva o disattiva il listing del sorgente
$MATHCK+	Controlla errori di matematica
$MESSAGE:'<testo>'	Visualizza un messaggio sul terminale
$NILCK+	Controlla il valore del puntatore
$OCODE+	Attiva o disattiva il listing del codice oggetto
$PAGE+	Salta alla pagina successiva
$PAGE: <n>	Definisce il numero di pagina per quella successiva
$PAGEIF: <n>	Salta alla pagina succesiva se mancano meno di <n> righe
$PAGESIZE: 55	Definisce la lunghezza della pagina di listing del sorgente
$POP	Ripristina il valore memorizzato di tutti i metacomandi
$PUSH	Memorizza il valore attuale di tutti i metacomandi
$RANGECK+	Controlla la validità del subrange
$REAL:4	Definisce la lunghezza del tipo REAL
$ROM	Avviso della inizializzazione aritmetica
$RUNTIME-	Determina il contesto degli errori in fase di esecuzione
$SIMPLE	Disabilita le ottimizzazioni globali
$SIZE	Minimizza la grandezza del codice generato
$SKIP: <n>	Salta n righe oppure va a fine pagina
$SPEED	Minimizza il tempo di esecuzione del codice
$STACKCK+	Controlla l'overflow dello stack in fase di ingresso

Tabella G-1 Metacomandi Pascal (continuazione)

METACOMANDO	FUNZIONE
$STANDARD	Abilita solo il livello Standard
$SUBTITLE: '<subt>'	Definisce il sottotitolo delle pagine
$SYMTAB+	Invia la tabella dei simboli al listing del sorgente
$SYSTEM	Aggiunge prestazioni al livello Esteso e di Sistema
$TAGCK-	Controlla le etichette dei campi nei record con varianti
$TITLE: '<title>'	Fornisce il titolo pagina per il listing del sorgente
$WARN+	Da' messaggi di avvertimento nel listing sorgente

Tabella G-1 Metacomandi Pascal (cont.)

H. MESSAGGI DI ERRORE

SOMMARIO

Questa appendice elenca tutti i numeri ed i messaggi di errore che è probabile incontrare usando il compilatore Pascal ed il sistema runtime.

INDICE

INTRODUZIONE

Le condizioni di errore possono essere riunite nelle seguenti categorie:

- avvertimenti in fase di compilazione
- errori rilevati in fase di compilazione
- errori interni del compilatore
- errori (sia in fase di compilazione che di esecuzione) definiti dallo standard ISO ma non rilevati dal Pascal
- errori di runtime del file system
- errori di runtime del file system rilevati soltanto se il relativo switch e' attivo.
- errori di runtime del file system rilevati comunque.

Gli errori del linker sono specifici del linker per il sistema operativo utilizzato e sono percio' riportati nel manuale "Linguaggio Pascal Guida Utente".

Le condizioni di errore possono essere:

- errore non rilevato
- errore rilevato dal compilatore
- errore rilevato dal sistema runtime.

Un errore e' "rilevato" se il compilatore o il sistema runtime individuano l'errore e danno un messaggio. Un avvertimento ("warning") e' un errore individuato dal compilatore ma stabilito in modo che il sorgente compilato possa girare correttamente. Gli errori di sostituzione (ad esempio usare due punti (:) invece di un segno di uguale (=)) ed alcuni altri errori di sintassi (ad esempio l'uso di punto e virgola (;) prima di un ELSE) sono errori comuni che generano solo un messaggio di avvertimento e sono fissati dal compilatore. Occorre comunque rivedere il sorgente e correggere gli errori, altrimenti vengono emessi gli stessi messaggi ogni volta che si compila.

Gli errori delle fase di compilazione comprendono tutte le condizioni descritte in questo manuale come "invalido", "illegale", "non permesso", e cosi' via. Lo standard ISO definisce un certo numero di errori che sono definiti come "errori non rilevati" nel Pascal ma che possono essere individuati in altre implementazioni.

ERRORI FRONT END DEL COMPILATORE

Gli errori front end ed i messaggi di avvertimento sono costituiti da un numero e da un messaggio. La maggior parte dei messaggi riportano una fila di barrette ed una freccia che punta alla locazione dell'errore; tre di questi messaggi (128, 129 e 130) appaiono solo dopo il corpo della routine in cui si sono verificati. La parola "WARNING" identifica gli avvertimenti come tali; tutti gli altri messaggi riportano errori di programma.

Il front end riesce a rimediare alla maggior parte degli errori; cioe' corregge la condizione e prosegue con la compilazione. Vi sono comunque alcuni errori di front end dai quali il compilatore non puo' uscire. In questi casi compare il messaggio:

Compiler Cannot Continue !

Il compilatore allora puo' fare ben poco, oltre che listare il resto del programma. Questi errori si manifestano nelle seguenti circostanze:

- si hanno piu' errori di quelli definiti dal metacomando $ERRORS

- si incontra un end-of-file inatteso

- i domini degli identificatori sono nidificati troppo profondamente

- il compilatore non riesce a trovare le parole chiave PROGRAM, MODULE o IMPLEMENTATION

- il compilatore non riesce a trovare gli identificatori PROGRAM, MODULE o IMPLEMENTATION

- si genera un errore di file system. Il messaggio riporta il nome del file e una delle seguenti frasi:

  HARD DATA (errore di controllo somma)
  DISK FULL (il disco e' pieno)
  FILE ACCESS (il file non e' stato trovato)
  FILE SYSTEM (altro errore o errore interno)

Il front end puo' anche individuare uno di questi due errori di runtime del compilatore:

- Errore: Compiler Out of Memory

  Questo di solito avviene quando sono stati dichiarati troppi identificatori.

- Errore: Compiler Internal Error

  Indipendentemente dal programma sorgente compilato, questo errore non deve mai verificarsi.

Se prima di un messaggio appare la parola "WARNING", il file di codice intermedio generato dal front end e' corretto. La condizione che ha

prodotto l'avvertimento non e' grave, ma e' considerata poco sicura. I messaggi che indicano errori veri bloccano ogni scrittura sui file intermedi, che vengono eliminati quando il front end e' terminato.

Il messaggio di errore "Compiler" significa che qualche controllo di coerenza interno ha dato risultati sfavorevoli. Indipendentemente dal sorgente in compilazione, questo messaggio non dovrebbe mai apparire.

L'elenco seguente di errori front end del compilatore comprende numero e messaggio di errore, con un breve chiarimento sulla condizione che genera il messaggio.

Codice Messaggio

101 Invalid Line Number

Ci sono troppe linee nel file sorgente (il limite massimo e' 32767).

102 Line Too Long Truncated

Nella riga ci sono troppi caratteri (il limite attuale e' di 142 caratteri).

103 Identifier Too Long Truncated

La lunghezza di un identificatore supera quella massima consentita dal sistema operativo e percio' viene troncata. Per sapere quale e' la lunghezza massima consentita si puo' vedere l'Appendice A, "Caratteristiche di Implementazione", del manuale "Linguaggio Pascal Guida Utente".

104 Number Too Long Truncated

Una costante numerica e' troppo lunga e percio' viene troncata. Le costanti numeriche hanno la stessa lunghezza massima degli identificatori.

105 End Of String Not Found

Le riga e' terminata prima di incontrare il segno di virgolette (") di chiusura.

106 Assumed String

Il compilatore ha incontrato il segno di virgolette (") o di apici rovesciati (`) ed ha assunto che tali caratteri racchiudano una stringa. Conviene usare solo gli apici singoli (').

107 Unexpected End Of File

Durante lo scanning il compilatore ha trovato un end-of-file inaspettato in un metacomando, in un numero o in un'altra locazione non valida.

108 Meta Command Expected Command Ignored

Il compilatore ha trovato il carattere dollaro ($) all'inizio di un comando, ma non l'identificatore di metacomando.

109 Unknown Meta Command Ignored

Il compilatore ha trovato un identificatore di metacomando che non riconosce o che non e' valido in questa versione del Pascal.

110 Constant Identifier Unknown Or Invalid Assumed Zero

L'identificatore di costante che segue un metacomando e' sconosciuto (come in $DEBUG: A) oppure non e' una costante del tipo giusto. Il compilatore ha sostituito con zero il valore sconosciuto o non corretto.

112 Invalid Numeric Constant Assumed Zero

La costante che segue un metacomando e' numerica (ed esempio $DEBUG: 123456) ma ha formato erroneo o e' fuori campo. Il compilatore ha sostituito con zero il valore non corretto.

113 Invalid Meta Value Assumed Zero

Il valore che segue un metacomando non e' ne' una costante ne' un identificatore. Il compilatore ha sostituito con zero il valore non corretto.

114 Invalid Meta Command

Il compilatore si aspettava uno dei seguenti segni dopo un metacomando: +, -, o : ma non lo ha trovato. Il metacomando viene ignorato dal compilatore.

115 Wrong Type Value For Meta Command Skipped

Il valore seguente il metacomando era un intero, e invece doveva essere una stringa (o viceversa). Il metacomando e' stato ignorato dal compilatore.

116 Meta Value Out Of Range Skipped

Il valore intero dato per il metacomando $LINESIZE era minore di 16 o maggiore di 160. Oppure "n" non e' ne' 4 ne' 8 per $REAL:n e non e' 2 per $INTEGER. In ognuno di questi casi il compilatore ignora il metacomando.

117 File Identifier Too Long Skipped

Il valore della stringa dato per il nome del file in un metacomando $INCLUDE era troppo lungo. Il metacomando e' stato ignorato. Il massimo ammesso e' 96 caratteri.

118 Too Many File Levels

Nel metacomando $INCLUDE vi sono troppi livelli di file nidificati. Il metacomando $INCLUDE e' ignorato.

119 Invalid Initialize Meta

Un metacomando $POP non ha il corrispondente metacomando $PUSH.

120 CONST Identifier Expected

Il compilatore non ha trovato alcun identificatore dopo il metacomando $INCONST. Il metacomando $INCONST e' ignorato.

121 Invalid INPUT Number Assumed Zero

L'input dell'utente invocato con $INCONST era invalido e si assume che sia uguale a zero.

122 Invalid Meta Command Skipped

Il compilatore ha trovato un metacomando $IF ma non quello seguente $THEN o $ELSE. Il comando $IF viene ignorato.

123 Unexpected Meta Command Skipped

Il compilatore ha trovato un metacomando $THEN non correlato ad alcun metacomando $IF. Il metacomando $THEN e' ignorato.

124 Unexpected Meta Command

Il compilatore ha trovato un metacomando non compreso fra delimitatori di commento, ma lo elabora ugualmente.

126 Invalid Real Constant

Il compilatore ha trovato una costante di tipo REAL con un punto decimale in testa o in coda. Il valore della costante e' comunque accettato.

127 Invalid Character Skipped

Il compilatore ha trovato nel file sorgente un carattere che non e' accettabile nel testo di programma.

128 Forward Proc Missing: <procedure>

Il compilatore ha trovato una procedura o una funzione dichiarata FORWARD ma non puo' trovare la procedura o la funzione stessa. Questo messaggio appare nella zona della tabella dei simboli del file di listing.

129 Label Not Encountered: <label>

Il compilatore non trova alcun uso possibile di un'etichetta

dichiarata nella sezione etichetta. Questo messaggio appare nella zona della tabella dei simboli del file listing.

130 Program Parameter Bad: <parameter>

Il compilatore ha incontrato questo parametro di programma che non e' mai stato dichiarato oppure e' di tipo inaccettabile. Questo messaggio appare nella zona dei simboli del file listing.

133 Type Size Overflow

Il tipo di dati dichiarato presuppone una struttura piu' grande del massimo consentito di 65534 byte.

134 Constant Memory Overflow

L'allocazione di memoria costante ha superato il massimo di 65534 byte.

135 Static memory Overflow

L'allocazione di memoria statica ha superato il massimo consentito di 65534 byte.

136 Stack Memory Overflow

L'allocazione di memoria per lo stack ha superato il massimo consentito di 65534 byte.

137 Integer Constant Overflow

Il valore di un tipo INTEGER, di un'espressione costante con segno, e' fuori campo di variabilita'.

138 Word Constant Overflow

Il valore di un tipo WORD o di altre espressioni costanti senza segno e' fuori campo di variabilita'.

139 Value Not In Range For Record

In una costante strutturata, nella forma lunga di una procedure NEW, DISPOSE o SIZEOF o in altre applicazioni, il valore dell'etichetta di record non e' nel campo della variante.

140 Too Many Compiler Labels

Il compiler ha bisogno di etichette interne ed il programma e' troppo lungo. Bisogna spezzare il programma in parti piu' piccole.

141 Compiler

142 Too Many Identifier Levels

Il livello di dominio dell'identificatore supera il 15. Questo e' un errore pericoloso.

143 Compiler

144 Compiler

Questo errore puo' verificarsi se il formato del file PASKEY non e' corretto.

145 Identifier Already Declared

Il compilatore ha trovato un identificatore dichiarato piu' di una volta nello stesso livello di dominio.

146 Unexpected End Of File

Durante l'analisi il compilatore ha trovato un fine-file fuori posto: in un'istruzione, una dichiarazione ecc.

147 : Assumed =

Il compilatore ha trovato un segno di due punti (:) che avrebbe dovuto essere un segno di uguale (=), e prosegue come se avesse incontrato il segno corretto.

148 = Assumed :

Il compilatore ha trovato un segno di uguale dove si aspettava invece i due punti, e prosegue come se vi fosse il segno corretto.

149 := Assumed =

Il compilatore ha trovato un due punti seguito da un segno di uguale dove si aspettava di trovare solo un segno di uguale, e procede come se vi fosse il simbolo corretto.

150 = Assumed :=

Il compilatore ha trovato un segno di uguale dove si aspettava due punti seguiti da uguale; procede come se avesse trovato il segno corretto.

151 [Assumed (

Il compilatore ha trovato una parentesi quadra di apertura dove si aspettava una tonda, e prosegue come se fosse presente il segno corretto.

152 (Assumed [

Il compilatore ha trovato una parentesi tonda di apertura dove si aspettava una quadra, e prosegue come se avesse trovato il segno corretto.

153) Assumed]

Il compilatore ha trovato una parentesi tonda di chiusura dove si aspettava di trovarne una quadra, e prosegue come se avesse trovato il segno corretto.

154] Assumed)

Il compilatore ha trovato una parentesi quadra di chiusura dove si aspettava una tonda, e prosegue come se fosse presente il segno corretto.

155 ; Assumed ,

Il compilatore ha trovato un punto e virgola dove si attendeva una virgola, e prosegue come se avesse trovato il segno corretto.

156 , Assumed ;

ll compilatore ha trovato una virgola dove si attendeva un punto e virgola e prosegue come se ci fosse il segno corretto.

162 Insert Symbol

Il compilatore non ha trovato il simbolo che si aspettava, ma prosegue come se ci fosse. Questo messaggio non deve mai apparire in quanto segnala un errore, sia pur non grave, del compilatore.

163 Insert ,

Il compilatore non ha trovato una virgola dove si aspettava, ma procede come se ci fosse.

164 Insert ;

il compilatore non ha trovato un punto e virgola dove si aspettava, ma procede come se ci fosse.

165 Insert =

Il compilatore non ha trovato un segno di uguale dove si aspettava, ma procede come se ci fosse.

166 Insert :=

Il compilatore non ha trovato i due punti seguiti de uguale dove si aspettava di trovarli, ma prosegue come se ci fossero.

167 Insert OF

Il compilatore non ha trovato un OF dove si aspettava, ma procede come se ci fosse.

168 Insert]

Il compilatore non ha trovato una parentesi quadra di chiusura, ma prosegue come se ci fosse.

169 Insert)

Il compilatore non ha trovato una parentesi tonda di chiusura dove si aspettava, ma prosegue come se ci fosse.

170 Insert [

Il compilatore non ha trovato una parentesi quadra di apertura dove si aspettava, ma prosegue come se ci fosse.

171 Insert (

Il compilatore non ha trovato una parentesi tonda di apertura dove si aspettava, ma prosegue come se ci fosse.

172 Insert DO

Il compilatore non ha trovato un DO dove si aspettava. Prosegue come se ci fosse.

173 Insert :

Il compilatore non ha trovato il segno di due punti dove si aspettava. Prosegue come se ci fosse.

174 Insert .

Il compilatore non ha trovato un punto (.) dove si aspettava di trovarne; prosegue come se ci fosse.

175 Insert ..

Il compilatore non ha trovato un doppio punto (..) dove se si aspettava di trovarlo; prosegue come se ci fosse.

176 Insert END

Il compilatore non ha trovato un END dove si aspettava, ma prosegue come se ci fosse.

177 Insert TO

Il compilatore non ha trovato un TO dove si aspettava, ma prosegue come se ci fosse.

178 Insert THEN

Il compilatore non ha trovato un THEN dove si aspettava, ma prosegue come se ci fosse.

179 Insert *

Il compilatore non ha trovato un asterisco dove si aspettava, ma prosegue come se ci fosse.

185 Invalid Symbol Begin Skip

Il compilatore ha trovato il simbolo che si aspettava, ma solo dopo aver incontrato alcuni simboli non validi. I simboli non validi, compresi fra il punto in cui appare il messaggio 185 e quello in cui appare il messaggio 186, vengono saltati.

186 End Skip

Il compilatore ha trovato il simbolo che si aspettava, ma solo dopo aver incontrato alcuni simboli non validi. I simboli non validi, compresi fra il punto in cui appare il messaggio 185 e quello in cui appare il messaggio 186, vengono saltati.

187 End Skip

Questo messaggio indica sempre la fine della porzione di testo sorgente che e' stata saltata con qualsiasi messaggio escluso il 185.

188 Section Or Expression Too Long

Il compilatore ha raggiunto il suo limite. Bisogna provare a riorganizzare il programma o spezzare l'espressione in piu' parti con assegnazioni a valori intermedi.

189 Invalid Set Operator Or Function

Il file sorgente contiene un uso non corretto di un operatore o una funzione set non corretta (per esempio l'operatore MOD o la funzione ODD con dei set).

190 Invalid REAL Operator Or Function

Il file sorgente contiene un uso non corretto di un operatore o di una funzione di REAL (per esempio l'operatore MOD o la funzione ODD con dei real).

191 Invalid Value Type For Operator Or Function

Per esempio l'operatore MOD o la funzione ODD con un tipo enumerato.

195 Compiler

196 Zero Size Value

Il file sorgente contiene il record vuoto "RECORD END" come se avesse una dimensione.

197 Compiler

198 Constant Expression Value Out Of Range

Il valore di un'espressione costante in un indice di array, in un'assegnazione subrange o in altri subrange, e' fuori del campo di variabilita'.

199 Integer Type Not Compatible With Word Type

In un'espressione si e' tentato di mescolare valori INTEGER con valori WORD. Questo errore comune indica una confusione fra valori aritmetici con segno e senza segno. Occorre modificare il valore positivo con segno in uno senza segno utilizzando WRD (), oppure cambiare quello senza segno (<MAXINT) in uno con segno con ORD ().

201 Types Not Assignment Compatible

Si e' tentato di usare dei tipi incompatibili in una istruzione di assegnazione o in un parametro di valore. Per le norme di compatibilita' fra tipi si puo' vedere "Compatibilita di Tipo", nel Capitolo 7.

202 Types Not Compatible In Expression

Si e' tentato di usare, in un'espressione, tipi non compatibili. Per le norme di compatibilita' si puo' vedere "Compatibilita' di Tipo", nel Capitolo 7.

203 Not Array Begin Skip

Una variabile seguita da una parentesi quadra o tonda di apertura non e' un array. Il compilatore salta la parte di codice compresa fra questo punto e quello in cui appare il messaggio 187.

204 Invalid Ordinal Expression Assumed Integer Zero

L'espressione contiene un tipo che e' sbagliato o non ordinale. Il compilatore assume che il valore dell'espressione sia zero.

205 Invalid Use Of PACKED Components

Un componente di una struttura PACKED non ha indirizzo (non puo' essere in un byte di confine) e non puo' essere passato per riferimento.

206 Not Record Field Ignored

Una variabile seguita da un punto non e' un record, un indirizzo o un file; e' stata percio' ignorata dal compilatore.

207 Invalid Field

Un nome valido di campo non segue una variabile record e un punto; il compilatore lo ha ignorato.

208 File Dereference Considered Harmful

Quando il compilatore calcola l'indirizzo di una variabile buffer di un file, non puo' compiere le operazioni che di solito fa sulle variabili buffer (ad esempio valutazione lazy per i file testo, o concorrenza per i file binari). Poiche' la variabile buffer a questo indirizzo puo' non essere valida, questo situazione viene considerata pericolosa.

209 Cannot Dereference Value

La variabile seguita da una freccia non e' un puntatore, un indirizzo o un file. Percio' il compilatore non puo' togliere riferimento al valore indicato.

210 Invalid Segment Address

Una variabile risiede in un indirizzo segmentato, ma e' necessario un valore di default dell'indirizzo del segmento. Puo' essere necessario fare una copia locale della variabile.

211 Ordinal Expression Invalid Or Not Constant

Il compilatore ha trovato un'espressione invalida o non costante la' dove si aspettava un'espressione costante ordinale.

214 Out Of Range For Set 255 Assumed

Il compilatore ha trovato un elemento di una costante set il cui valore ordinale e' superiore a 255, ed assume percio' il valore 255.

215 Type Too Long Or Contains File Begin Skip

Il compilatore ha trovato una costante strutturata che supera i 255 byte oppure e' (o contiene) un tipo FILE o LSTRING.

216 Extra Array Components Ignored

Il compilatore ha trovato una costante array che ha troppi componenti in relazione al tipo di array. I componenti in eccesso sono ignorati.

217 Extra Record Components Ignored

Il compilatore ha trovato una costante record che contiene troppi componenti in relazione al tipo di record. I componenti in eccesso sono ignorati.

218 Constant Value Expected Assumed Zero

Il compilatore ha trovato un valore non costante in una costante

strutturate ed ha assunto il valore uguale zero.

220 Compiler

221 Components Expected For Type

Il compilatore ha trovato troppo pochi componenti per il tipo di una costante strutturata.

222 Overflow 255 Components In String Constant

Il compilatore ha trovato una costante stringa che supera i 255 byte.

223 Use NULL

Occorre usare la costante predichiarata NULL anziche' due virgolette ("").

224 Cannot Assign With Supertype Lstring

Un super array LSTRING non puo' essere la fonte o la destinazione di un'assegnazione.

225 String Expression Not Constant

La concatenazione fra stringhe mediante asterisco si applica solo a costanti.

226 String Expected Character 255 Assumed

Il compilatore ha trovato una costante stringa senza caratteri, forse derivante dall'uso di NULL, e percio' ha assunto il valore CHR (255).

227 Invalid Address Of Function

Un'assegnazione o un altro indirizzo di riferimento al valore della funzione non e' all'interno del dominio della funzione. Oppure RESULT e' usato al di fuori del dominio della funzione.

228 Cannot Assign To Variable

L'assegnazione alle variabili di controllo READONLY, CONST, o FOR non e' consentita.

230 Unknown Identifier Assumed Integer Begin Skip

Il compilatore ha trovato un identificatore sconosciuto per il quale vuole un indirizzo, e percio' e' saltato ad una virgola, punto e virgola o parentesi destra.

231 VAR Parameter Or WITH Record Assumed Integer Begin Skip

Il compilatore ha trovato un simbolo non valido dove gli serve un

indirizzo e percio' e' saltato ad una virgola, punto e virgola o parentesi chiusa.

232 Cannot Assign To Type

La destinazione di un'assegnazione e' un file: non puo' essere assegnata per qualche altra ragione.

233 Invalid Procedure Or Function Parameter Begin Skip

Il compilatore ha trovato un uso non corretto di una funzione o procedura intrinseca. L'errore puo' essere uno dei seguenti:

- il primo parametro di NEW o DISPOSE non e' una variabile puntatore
- il valore delle etichette di record delle procedure NEW, DISPOSE, SIZEOF non e' stato trovato
- il super array nelle procedure NEW, DISPOSE, SIZEOF ha troppi limiti
- il super array in una procedure NEW, DISPOSE o SIZEOF ha troppo pochi limiti
- al super array per una procedura NEW o SIZEOF non e' stato dato alcun limite
- si e' tentato di usare una funzione WRD o ORD su di un valore non di tipo ordinale
- si e' tentato di usare le funzioni LOWER o UPPER in un valore o tipo non valido
- PACK o UNPACK in un super array o file, oppure un array che e' (o non e') impaccato come ci si aspettava
- il primo parametro di un RETYPE non e' un identificatore di tipo
- il parametro di una funzione RESULT non e' un identificatore di funzione
- si e' tentato di usare una funzione o procedura intrinseca non disponibile in questa versione di Pascal
- ORD o WRD di un valore INTEGER4 e' fuori campo di variabilita'
- il parametro dato per HIWORD o LOWORD non e' ne' un ordinale ne' INTEGER4.

234 Type Invalid Assumed Integer

Il parametro dato per READ, WRITE, ENCODE o DECODE non e' del

tipo INTEGER, WORD, INTEGER4, REAL, BOOLEAN, enumerato, o puntatore; oppure il parametro dato per READ o WRITE non e' di uno di questi tipi o del tipo FILE. Il compilatore ha assunto che fosse del tipo INTEGER. Questo errore si puo' anche verificare se un parametro di programma non ha un tipo leggibile, nel qual caso si verifica un errore alla parola chiave BEGIN del programma principale.

235 Assumed File INPUT

Poiche' il primo parametro di un READFN non e' un file, e' assunto INPUT.

236 Invalid Segment For File

Il parametro di file deve risiedere sempre nel segmento di default.

237 Assumed INPUT

INPUT non era stato dato come parametro di programma ed e' stato assunto.

238 Assumed OUTPUT

OUTPUT non era stato dato come parametro di programma ed e' stato assunto.

239 Not Lstring Or Invalid Segment

La destinazione di un READSET, ENCODE, o DECODE deve essere una LSTRING nel segmento di default. Una (o ambedue) di queste condizioni e' mancante.

242 File Parameter Expected Begin Skip

La procedura READSET si aspettava un parametro di file testo ma non lo ha trovato. Il compilatore ha ignorato la procedura e riparte dal punto in cui appare il messaggio 187.

243 Character Set Expected

La procedura READSET si aspettava un parametro SET OF CHAR ma non lo ha trovato.

244 Unexpected Parameter Begin Skip

Il compilatore ha trovato piu' di un parametro dato per EOF, EOLN, o PAGE; ha ignorato quelli in eccesso.

245 Not Text File

Si e' tentato di usare EOLN, PAGE, READLN, o WRITELN in qualche file che non e' un file testo.

248 Size Not Identical

La funzione RETYPE non puo' operare come richiesto finche' i parametri passati non sono della stessa lunghezza.

249 Procedural Type Parameter List Not Compatible

Le liste di parametri per i parametri procedurali sia formali che effettivi non sono compatibili. Cioe' il numero di parametri, il tipo della funzione risultato, il tipo di qualche parametro o gli attributi sono differenti.

250 Cannot Use Procedure With Attribute

Si e' tentato di chiamare una procedura con l'attributo INTERRUPT direttamente o indirettamente. INTERRUPT non lo consente.

251 Unexpected Parameter Begin Skip

Il compilatore ha trovato una parentesi tonda aperta che indica una funzione o una procedura, ma non ha trovato dei parametri e percio' e' saltato al punto in cui appare il messaggio 187.

252 Cannot Use Procedure Or Function As Parameter

Si e' tentato di passare questa procedura o funzione intrinseca come parametro; la cosa non e' permessa.

253 Parameter Not Procedure Or Function Begin Skip

Il compilatore si aspettava un parametro procedurale ma non lo ha trovato. Salta al punto in cui appare il messaggio 187.

254 Supertype Array Parameter Not Compatible

Il parametro attuale passato non e' dello stesso tipo o non e' derivato dallo stesso tipo super del parametro formale.

255 Compiler

256 VAR Or CONST Parameter Type Not Identical

I tipi dei parametri di riferimento attuali e formali non sono, come dovrebbero essere, identici.

257 Parameter List Size Wrong Begin Skip

Il compilatore ha trovato troppi o troppo pochi parametri nelle liste. Se ne ha trovati troppi ha saltato quelli in eccesso.

258 Invalid Procedural Parameter To EXTERN

Una procedura o una funzione (che non e' ne' PUBLIC ne' EXTERN) e' stata passata come parametro ad una procedura o funzione dichiarata EXTERN. Il compilatore invoca la procedura o funzione

attuale con delle chiamate intrasegmento e percio' non puo' passarle ad un segmento di codice esterno.

259 Invalid Set Constant For Type

Il set non e' costante, i tipi base non sono identici o la costante e' troppo lunga.

260 Unknown Identifier In Expression Assumed Zero

L'identificatore di un'espressione non e' definito o e' mal scritto.

261 Identifier Wrong In Expression Assumed Zero

L'identificatore di un'espressione non e' corretto (ad esempio un identificatore di tipo file) percio' e' stato assunto uguale a zero.

262 Assumed Parameter Index Or Field Begin Skip

Dopo l'errore 260 o 261 tutti i caratteri compresi fra parentesi quadre o tonde, oppure i trattini seguiti da un identificatore, vengono saltati.

265 Invalid Numeric Constant Assumed Zero

Qualche costante letterale assunta come INTEGER o INTEGER4 contiene un errore di decodifica. Il numero e' troppo grande, ha un carattere non valido ecc. La costante non corretta e' stata assunta uguale o zero.

267 Invalid Real Numeric Constant

In qualche costante letterale assunta tipo REAL c'e' un errore di decodifica. Il numero e' troppo grande, un carattere non e' corretto ecc.

268 Cannot Begin Expression Skipped

E'stato eliminato un simbolo che non puo' dare inizio ad una espressione.

269 Cannot Begin Expression Assumed Zero

E'stato sostitiuto con zero un simbolo che non puo' dare inizio ad una espressione.

270 Constant Overflow

Il divisore in una funzione DIV o MOD e' la costante zero (INTEGER o WORD). La cosa non e' permessa.

272 Word Constant Overflow

Una costante WORD meno una constante WORD ha dato come risultato un valore negativo.

275 Invalid Range

Il limite inferiore di un subrange e' piu' grande del limite superiore (es 2..1).

276 CASE Constant Expected

Il compilatore si aspettava un valore della costante per l'istruzione CASE o per una variante record, ma non lo ha trovato.

277 Value Already In Use

In un'istruzione CASE o in un record variante il valore e' gia' stato assegnato (come ad esempio in CASE 1..3 : XXX ; 2 : YYY ; END).

279 Label Expected

Il compilatore si aspettava un'etichetta ma non l'ha trovata.

280 Invalid Integer Label

Un'etichetta usa notazioni non decimali (ad esempio 8#77) che non sono ammesse.

281 Label Assumed Declared

Il compilatore ha trovato un'etichetta che non appare nella sezione LABEL.

283 Expression Not Boolean Type

L'espressione che segue un IF, un WHILE o un UNTIL deve essere BOOLEAN.

284 Skip To End Of Statement

Il compilatore ha trovato, e ha saltato, una clausola ELSE o UNTIL inaspettata.

285 Compiler

286 ; Ignored

Il compilatore ha trovato, ed ha ignorato, un punto e virgola prima di un ELSE (il punto e virgola non e' necessario in questo caso).

288 : Skipped

Il compilatore ha trovato ed ignorato un segno di due punti (:)

dopo un'istruzione OTHERWISE (i due punti non sono necessari in questo caso).

289 Variable Expected For FOR Statement Begin Skip

Il compilatore si aspettava un identificatore di variabile dopo uno statement FOR ma, non trovandolo, e' saltato al punto in cui appare il messaggio 187.

291 FOR Variable Not Ordinal Or Static Or Declared In Procedure

Il compilatore ha trovato in uno statement FOR una variabile di controllo non corretta. In particolare, la variabile di controllo e' (ma non dovrebbe esserlo) una delle seguenti:

- tipo REAL, INTEGER4 o un altro tipo non ordinale
- il componente di un tipo array, record o file
- il tipo referente a puntatore o a tipo indirizzo
- nello stack o nello heap, a meno che non sia specificamente indicato
- dichiarata non localmente, a meno che non lo sia nella memoria statica
- un parametro di riferimento (VAR o VARS)
- una variabile con un attributo ORIGIN segmentato.

292 Skip To :=

Il compilatore si aspettava un'assegnazione in un'istruzione FOR; non avendolo trovato e' saltato al successivo :=.

293 GOTO Invalid

Il GOTO o l'etichetta contengono un'istruzione GOTO non valida.

294 GOTO Considered Harmful

Come prescritto, se il metacomando $GOTO e' attivo il compilatore ha trovato un'istruzione GOTO.

296 Label Not Loop Label

L'etichetta dopo un'istruzione BREAK o CYCLE non e' un'etichetta loop (cioe' non indirizza un'istruzione FOR, WHILE o REPEAT).

297 Not In Loop

Il compilatore ha trovato un'istruzione BREAK o CYCLE al di fuori degli statement FOR, WHILE, o REPEAT.

298 Record Expected Begin Skip

Il compilatore si aspettava una variabile record nell'istruzione WITH e, non trovandolo, e' saltato al punto in cui appare il messaggio 187.

300 Label Already In Use Previous Use Ignored

Il compilatore ha trovato un'etichetta che appariva gia' in un'altra istruzione, percio' ha ignorato il precedente impiego dell'etichetta.

301 Invalid Use Of Procedure Or Function

Il compilatore ha trovato un parametro di procedura usato come funzione, o un parametro di funzione usato come procedura.

303 Unknown Identifier Skip Statement

Il compilatore ha trovato un identificatore sconosciuto (o forse mal scritto) all'inizio di un'istruzione, ed ha ignorato l'intera istruzione.

304 Invalid Identifier Skip Statement

Il compilatore ha trovato all'inizio di un'istruzione un identificatore non valido; ha percio' ignorato l'intera istruzione.

305 Statement Not Expected

Il compilatore ha trovato un MODULE oppure un IMPLEMENTATION non inizializzato con un corpo delimitato fra le parole riservate BEGIN e END.

306 Function Assignment Not Found

Il compilatore si aspettava un'assegnazione del valore della funzione in qualche punto del corpo della funzione stessa, ma non e' riuscito e trovarla.

307 Unexpected END Skipped

Il compilatore ha trovato un END senza il relativo BEGIN, CASE o RECORD, e quindi lo ignora.

308 Compiler

309 Attribute Invalid

Il compilatore ha trovato un attributo valido solo per procedure e funzioni passate ad una variabile, un attributo valido solo per una variabile passata ad un procedura o funzione oppure una combinazione non valida di attributi (es. PUBLIC e EXTERN).

310 Attribute Expected

Il compilatore si aspettava un attributo valido dopo la parentesi quadra destra, ma non lo ha trovato.

311 Skip To Identifier

Il compilatore ha saltato un simbolo non valido (cioe' inaspettato) per raggiungere l'identificatore successivo.

312 Identifier Expected

Il compilatore ha trovato, dove si aspettava una lista di identificatori, qualcosa che non e' un identificatore.

314 Ientifier Expected Skip To ;

Il compilatore si aspettava la dichiarazione di un nuovo identificatore; non l'ha trovata ed e' saltato al successivo punto e virgola.

315 Type Unknown Or Invalid Assumed Integer Begin Skip

Il tipo di ritorno per un parametro o per una funzione non e' corretto; e cioe' non e' un identificatore o non e' dichiarato; oppure il valore parametro o il valore della funzione e' un file o un super array. Il compilatore ha assunto che il tipo sia INTEGER ed e' saltato al punto in cui appare il messaggio 187.

316 Identifier Expected

Il compilatore si aspetta, nella lista parametri, un identicatore dopo la parola PROCEDURE o FUNCTION; ma non riesce a trovarlo.

318 Compiler

319 Compiler

320 Previous Forward Skip Parameter List

Il compilatore ha trovato una definizione di una procedura o funzione FORWARD (o INTERFACE) che ripete, senza che occorra, la lista parametri o il tipo di ritorno funzione.

321 Not EXTERN

Il compilatore ha trovato una procedura o una funzione con l'attributo ORIGIN, ma privo dell'attributo EXTERN.

322 Invalid Attribute With Function Or Parameter

Il compilatore ha trovato un uso scorretto della procedura INTERRUPT: ha parametri o e' una funzione.

323 Invalid Attribute With Procedure Or Function

Il compilatore ha trovato una procedura o una funzione nidificata che ha degli attributi oppure e' dichiarata EXTERN. Nessuna di queste due condizioni e' consentita.

324 Compiler

325 Already Forward

Si e' tenato di usare FORWARD due volte per la stessa procedura o funzione.

326 Identifier Expected For Procedure Or Function

Il compilatore si aspetta un identificatore dopo le parole chiave PROCEDURE o FUNCTION, ma non riesce a trovarlo.

327 Invalid Symbol Skipped

Il compilatore ha trovato e ignorato una direttiva FORWARD o EXTERN in un'interfaccia.

328 EXTERN Invalid With Attribute

Il compilatore ha trovato una procedura EXTERN dichiarata anche PUBLIC. Cio' non e' consentito.

329 Ordinal Type Identifier Expected Integer Assumed Begin Skip

Il compilatore si aspetta un identificatore di tipo ordinale per un tipo etichetta di record, ma non riesce a trovarlo. Ha percio' saltato quello dato nel file sorgente e ha assunto il tipo INTEGER.

330 Contains File Cannot Initialize

E'stato usato un file in un record con varianti. Questo e' consentito, ma e' considerato poco sicuro e percio' non e' inizializzato automaticamente con la solita chiamata NEWFQQ.

331 Type Identifier Expected Assumed Character

Il compilatore si aspetta un identificatore di tipo ordinale. Non riuscendo a trovarlo, assume che quello che ha trovato sia di tipo CHAR.

333 Not Supertype Assumed String

Il compilatore ha trovato qualcosa che assomiglia all'indicatore di un super array. Tuttavia il tipo di identificatore non e' adatto al tipo di super array; cosi' il compilatore assume che sia l'identificatore di un super array STRING.

334 Type Expected Integer Assumed

Il compilatore si aspetta una clausola o un identificatore di tipo e, non trovandolo, assume che il tipo atteso sia INTEGER.

335 Out Of Range 255 For Lstring

Il compilatore ha trovato un indicatore LSTRING il cui limite superiore e' piu' grande di 255.

336 Cannot Use Supertype Use Designator

Un tipo super array puo' essere usato solo come parametro di riferimento o puntatore di riferimento. Alle altre variabili non puo' essere dato un tipo super array. Si usa un indicatore di super array.

337 Supertype Designatore Not Found

Il compilatore si aspetta un indicatore di super array che dia il valore del limite superiore del super array, ma non lo trova.

338 Contains File Cannot Initialize

Il compilatore ha trovato un super array di un tipo file. Anche se consentito, cio' e' considerato poco sicuro e percio' l'inizializzazione non avviene automaticamente con la normale chiamata NEWFQQ.

339 Supertype Not Array Skip To ; Assumed Integer

In una clausola di tipo il compilatore si aspetta la parole chiave ARRAY dopo SUPER. Non avendola trovata, assume che il tipo sia INTEGER e salta al punto e virgola successivo.

340 Invalid Set Range Integer Zero To 255 Assumed

Il compilatore ha trovato un campo di variabilita' non valido per il tipo base di un insieme, ed assume che esso debba essere di tipo INTEGER con variabilita' fra 0 e 225.

341 File Contains File

Il compilatore ha trovato,ma non lo accetta, un tipo file che contiene un tipo file sia indirettamente che direttamente.

342 PACKED Identifier Invalid Ignored

Il compilatore si aspetta una delle parole ARRAY, RECORD SET oppure FILE dopo la parola riservata PACKED. Non e' consentito nessun tipo di identificatore dopo PACKED.

343 Unexpected PACKED

Il compilatore ha trovato la parola chiave PACKED usata in uno

dei tipi non strutturati.

345 Skip To ;

Il compilatore si aspetta un punto e virgola alle fine di una dichiarazione (che non si trova a fine riga). Esso assume che il punto e virgola successivo sia la fine della dichiarazione.

346 Insert ;

Il compilatore si aspetta un punto e virgola alla fine della dichiarazione (che coincide con la fine della riga): non trovandolo, inserisce un punto e virgola dove si aspettava che ci fosse.

347 Cannot Use Value Section With ROM Memory

Se e' attivato il metacomando $ROM, non e' possibile avere anche una sezione VALUE.

348 UNIT Procedure Or Function Invalid EXTERN

Una dichiarazione EXTERN indispensabile avviene piu' tardi, quando dovrebbe essere in un IMPLEMENTATION (ogni procedura o funzione di interfaccia non implementata deve essere dichiarata EXTERN fin dall'inizio).

350 Not Array Begin Skip

In una sezione VALUE la variabile seguita da parentesi quadra aperta non e' un array.

351 Not Record Begin Skip

La variabile seguita da un punto nella sezione VALUE non e' un tipo record.

352 Invalid Field

Nelle sezione VALUE l'identificatore assunto come campo non e' nel record.

353 Constant Value Expected

In una sezione VALUE una variabile e' stata inizializzata non come costante ma come qualche altra cosa.

354 Not Assignment Operator Skip To ;

In una sezione VALUE manca l'operatore di assegnazione.

355 Cannot Initialize Identifier Skip To ;

In una sezione VALUE un simbolo non e' una variabile dichiarata a questo livello nella memoria fissa (STATIC). Oppure e' un

attributo ORIGIN o EXTERN illegale.

356 Cannot Use Value Section

Una sezione VALUE e' stata erroneamente inserita nella INTERFACE anziche' nell'IMPLEMENTATION.

357 Unknown Forward Pointer Type Assumed Integer

L'identificatore per il referente a un tipo di riferimento dichiarato precedentemente in questa sezione TYPE (o VAR) non e' mai stato dichiarato da solo.

358 Pointer Type Assumed Forward

La sezione TYPE include un puntatore o dei tipi indirizzo per i quali il tipo referente e' gia' stato dichiarato in un dominio che lo ingloba. Poiche' l'identificatore per il tipo referente e' stato dichiarato anche piu' tardi nella stessa sezione TYPE, il compilatore usa la seconda definizione. Nell'esempio seguente e' usato il tipo forward REAL.

```
PROGRAM OUTSIDE;
TYPE A = WORD;
PROCEDURE B;
TYPE C = ^A;
A = REAL;
```

359 Cannot Use Label Section

Il compilatore ha trovato una sezione LABEL erroneamente inclusa in un INTERFACE anziche' in un IMPLEMENTATION.

360 Forward Pointer To Supertype

Il referente di un tipo di riferimento dichiarato in questa sezione TYPE e' un tipo super array. La dichiarazione del tipo super array non avviene se non dopo il riferimento.

361 Constant Expression Expected Zero Assumed

Un'espressione in una sezione CONST non e' una costante.

362 Attribute Invalid

Una sezione VAR mescola erroneamente attributi PUBLIC o ORIGIN con EXTERN. Oppure ORIGIN appare fra le parentesi quadre dell'attributo dopo la parola chiave VAR.

364 Contains File Initialize Module

Il compilatore ha trovato in un modulo una variabile file non inizializzata. Per inizializzare i file bisogna chiamare il modulo come una procedura senza parametri.

365 Origin Variable Contains File Cannot Initialize

Il compilatore ha trovato una variabile file non inizializzata in un modulo. Poiche' le variabili ORIGIN non sono mai inizializzate, l'utente deve inizializzare lui stesso questo file.

366 UNIT Identifier Expected Skip To ;

ll compilatore si aspetta un identificatore dopo la parola chiave USES, ma non lo trova.

367 Initialize Module To Initialize Unit

Per inizializzarlo, il modulo deve essere chiamato come una procedura (una clausola USES lancia una chiamata di inizializzazione).

368 Identifier List Too Long Extra Assumed Integer

In una clausola USES con una lista di identificatori, il compilatore ha trovato nella lista piu' identificatori di quelli costituenti l'interfaccia. Quelli eccedenti sono assunti come identificatori di tipo identici a INTEGER.

369 End Of UNIT Identifier List Ignored

In una clausola USES con lista di identificatori il compilatore ha trovato nella lista meno identificatori di quelli costituenti l'interfaccia. I rimanenti costituenti dell'interfaccia non sono forniti come parte della clausola USES.

371 UNIT Identifier Expected

Dopo la frase "INTERFACCIA; UNIT" manca un identificatore.

372 Compiler

Il compilatore si aspetta, ma non trova, la parola chiave UNIT in una INTERFACE

373 Identifier In UNIT List Not Declared

Uno degli identificatori della lista dell'UNIT di interfaccia non e' dichiarata nel corpo dell'interfaccia.

374 Program Identifier Expected

Dopo la parola chiave PROGRAM o MODULE manca un identificatore. Questo e' un errore pericoloso.

375 UNIT Identifier Expected

Dopo la frase "IMPLEMENTATION OF" manca l'identificatore di unita'. Questo e' un errore pericoloso.

376 Program Not Found

Il compilatore si aspetta una delle parole riservate PROGRAM MODULE o IMPLEMENTAION. Questo e' un errore pericoloso. (Questo errore si puo' verificare se il file sorgente non e' un'unita' compilativa Pascal).

377 File End Expected Skip To End

Il compilatore ha trovato un testo sorgente addizionale dopo quella che sembra essere la fine e percio' ha ignorato tutto quello che si trova oltre al punto creduto finale.

378 Program Not Found

Il compilatore si aspetta, ma non riesce a trovarlo, il corpo principale di un'unita' compilativa oppure l'END finale.

ERRORI BACK END DEL COMPILATORE

La fonte principale di errori di back end sono errori utente derivanti sia dall'ottimazzatore che dal generatore di codice. Infatti vi sono pochissimi errori di questo tipo. Sono tutti correlati con limitazioni che non possono essere scoperte dal front end.

Gli errori di back end provocano un arresto immediato, mentre sullo schermo appare un codice di errore e un approssimativo numero di riga del listing.

Gli errori di back end sono elencati qui di seguito:

Codice Messaggio

1 Attempt To Divide By Zero

Per esempio A DIV 0

2 Overflow During Integer Constant Folding

Per esempio MAXINT + A + MAXINT

3 Expressions Too Complex/Too Many Internal Labels

Provare a spezzare in piu' parti l'espressione con assegnazioni di valori intermedi

4 Too Many Procedure And/Or Functions

Solo Pcode. Provare a suddividere l'unita' compilativa in moduli o unita'

5 Range Error

Il numero e' troppo grande per adattarsi alla destinazione

ERRORI INTERNI DEL COMPILATORE

Tutti gli errori etichettati "Compiler" nella sezione "Errori Front End del Compilatore" sono errori interni che non dovrebbero verificarsi mai.

Anche il back end del compilatore esegue un gran numero di controlli di congruenza, questi controlli dovrebbero sempre avere esito positivo e non dare mai un errore interno. Qualora si verificassero degli errori, i messaggi di errore interno o di back end avrebbero lo stesso formato

*** Internal Error NNN

NNN e' il numero dell'errore interno che puo' variare da 1 a 999. C'e' ben poco che l'utente possa fare quando si verifica un errore interno, se non riferirlo e cercare di modificare il programma nei dintorni della riga dove l'errore si e' manifestato.

ERRORI RUNTIME DI FILE SYSTEM

I codici di errore di file system variano da 1000 e 1099. I codici di errore vanno nel campo ERRC del file control block. I codici di errore da 1100 a 1199 identificano errori del file system del Pascal.

Gli errori di file system hanno tutti lo stesso formato:

<error type> error in file <filename>

seguito dal codice errore e, in alcune versioni, da uno stato dell'errore che e' una parola di ritorno di errore del sistema operativo. Il campo <error type> e' basato sul campo ERRS del file control block nel modo seguente:

Codice Messaggio

1 Hard Data

Errori difficili di dati (parita', CRC, controllo, etc.)

2 Device Name

Nome di formato o numero di unita'/device/volume non valido

3 Operation

Operazione non valida: GET se EOF; RESET di una stampante, etc.

4 File System

Errore interno del file system, ERRS > 15 etc

5 Device Offline

Unita'/device/volume non piu' disponibili

6 Lost File

Lo stesso file non e' piu' disponibile

7 File Name

Sintassi non valida, nome troppo lungo, oppure non esistono nomi temporanei etc.

8 Device Full

Disco pieno, directory piena, tutti i canali gia' allocati

9 Unknown Device

Unita'/device/volume non trovati

10 File Not Found

Non e' stato trovato neanche il file

11 Protected File

Filename duplicato; file protetto da scrittura

12 File In Use

File gia' utilizzato, blocco di concorrenza, file gia' aperto

13 File Not Open

File chiuso, I/O per chiudere FCB

14 Data Format

Errore di formato dei dati, errore di decodifica, errore di campo di variazione

15 Line Too Long

Overflow di buffer, riga troppo lunga.

ERRORI RUNTIME NEL SISTEMA OPERATIVO (1000 - 1099)

Codice	Messaggio
1000	Write error when writing end of file
1001	Unknown device name
	Solo su CP/M-80 e CP/M-86. Si verifica quando non e' mai stato assegnato in un RESET o REWRITE nessun filename
1002	Filename extension with more than 3 characters
1003	Error during creation of new file (disk or directory full)
1004	Error during open of existing file (file not found)
1005	Filename with more than 8 or zero characters
1006	Device cannot do input or output (solo per CP/M-80 e CP/M-86)
1007	Total filename lenght over 21 characters
1008	Write errors when advancing to next record
1009	File too big (over 65535 logical sectors)
1010	Write errors when seeking to direct record
1011	Attempt to open a random file to a non-disk device
1012	Forward space or back space on a non-disk device
	E' un errore solo FORTRAN
1013	Disk or directory full error during forward space or back space.
	Errore solo del FORTRAN

ERRORI DI FILE SYSTEM NEL PASCAL (1100 - 1199)

Codice	Messaggio
1100	ASSIGN or READFN of filename to open file.
	Questo errore e' rilevato solo per file testo.
1101	Reference to buffer variable of closed textfile.

1102 textfile READ or WRITE call to closed file.

1103 READ when EOF is true (SEQUENTIAL mode).

1104 READ to REWRITE file, or WRITE to RESET file (SEQUENTIAL mode).

1105 EOF call to closed file.

1106 GET call to closed file.

1107 GET call when EOF is true (SEQUENTIAL mode).

1108 GET call to REWRITE file (SEQUUENTIAL mode).

1109 PUT call to closed file.

1110 PUT call to RESET file (SEQUUENTIAL mode).

1111 Line too long in DIRECT textfile.

1112 Decode error in textfile READ BOOLEAN.

1113 Value out of range in textfile READ CHAR.

1114 Decode error in textfile READ INTEGER.

1115 Decode error in textfile READ SINT (integer subrange).

1116 Decode error in textfile READ REAL.

1117 LSTRING target not big enough in READSET.

1118 Decode error in textfile READ WORD.

1119 Decode error in textfile READ BYTE (word subrange).

1120 SEEK call to closed file.

1121 SEEK call to file not in DIRECT mode.

1122 Encode error (field width > 255) in textfile WRITE BOOLEAN.

1123 Encode error (field width > 255) in textfile WRITE INTEGER.

1124 Encode error (field width > 255) in textfile WRITE REAL.

1125 Encode error (field width > 255) in textfile WRITE WORD.

1126 Decode error (field width > 255) in textfile READ INTEGER4.

1127 Encode error (field width > 255) in textfile WRITE INTEGER4.

ALTRI ERRORI RUNTIME

I codici di errore non correlati al file system variano da 2000 a 2999. In taluni casi i metacomandi pilotano l'esecuzione dei controlli da parte del compilatore. Negli altri casi e' il compilatore che controlla sempre. La lista qui di seguito indica quale metacomando (se esiste) pilota il controllo degli errori.

ERRORI DI MEMORIA (2000 - 2049)

Poiche' lo stack e lo heap crescendo avanzano uno verso l'altro, tutti gli errori di memoria sono fra loro correlati e interdipendenti; per esempio un overflow di stack puo' generare un errore "Heap Is Invalid" se $STACKCK e' disattivato e se lo stack va in overflow.

Codice Messaggio

2000 Stack Overflow

Lo stack (frame) va fuori memoria (out of memory) mentre sta chiamando una procedura o una funzione. Questa condizione e' controllata se il metacomando $STACKCK e' attivo, e puo' essere controllato anche in altri casi.

2001 No Room In Heap

L'heap e' andato fuori memoria per l'aggiunta di una nuova variabile durante la procedura NEW (GETHQQ). Questo errore e' sempre rilevato.

2002 Heap Is Invalid

Durante una procedura NEW (GETHQQ) l'algoritmo di allocazione scopre che la struttura di heap e' sbagliata. Questo errore e' rilevato sempre.

2003 Heap Allocator Interrupted

Una procedura di interruzione ha interrotto una NEW (GETHQQ) e ha fatto lei stessa una call NEW. L'allocatore di Heap modifica lo Heap, che cosi' si trova nella sezione critica. Questo errore non e' rilevato in tutte le versioni.

2004 Allocation Internal Error

Si e' manifestato un inatteso ritorno con errore mentre GETHQQ stava richiedendo, da sistema operativo, dello spazio aggiuntivo su heap.

2031 NIL Pointer Reference

DISPOSE or $NILCK+ hanno trovato un puntatore con un valore NIL (ad esempio, 0).

2032 Uninitialized Pointer

DISPOSE o $NILCK+ hanno trovato un puntatore non inizializzato (valore 1). Questo avviene solo se il metacomando $1N1TCK e' attivo.

2033 Invalid Pointer Range

DISPOSE o $NILCK hanno trovato un puntatore che non punta lo heap o e' comunque non valido. (Potrebbe aver puntato un blocco rilasciato, rimosso dallo heap e ridato indietro al sistema.)

2034 Pointer To Disposed Var

DISPOSE o $N1LCK+ puntano ad un blocco di Heap che e' gia' stato rilasciato. Non e' valido chiamare due volte DISPOSE per la stessa variabile.

2035 Long DISPOSE Sizes Unequal

In una forma lunga di DISPOSE la lunghezza effettiva della variabile non e' uguale alla lunghezza basata sui valori di etichetta passati.

ERRORI ARITMETICI ORDINALI (2050 - 2099)

Codice Messaggio

2050 No CASE Value Matches Selector

In uno statement CASE senza clausole OTHERWISE, nessuna delle istruzioni di salto ha un valore della costante CASE uguale al valore della espressione del selettore. Questo errore e' controllato solo se il metacomando SRANGECK e' attivo.

2051 Unsigned Divide By Zero

Un valore WORD e' stato diviso per zero. Questo errore e' controllato solo se il metacomando $MATCHCK e' attivo.

2052 Signed Divide By Zero

Un valore INTEGER e' stato diviso per zero. Questo errore e' controllato solo se e' attivo il metacomando $MATHCK.

2053 Unsigned Math Overflow

Un risultato di WORD e' fuori dai valori estremi (zero e MAXWORD) ammessi. Questo errore e' controllato solo se il metacomando

$MATHCK e' attivo.

2054 Signed Math Overflow

Un risultato INTEGER e' fuori dai limiti consentiti (tra -MAXINT e +MAXINT). Questo errore e' controllato solo se il metacomando $MATHCK e' attivo.

2055 Unsigned Value Out Of Range

Il valore sorgente per una assegnazione o per un parametro e' fuori del campo di variabilita' per il valore destinazione. La destinazione puo' essere un subrange di WORD (incluso BYTE), oppure CHAR, oppure un tipo enumerato. Questo errore puo' verificarsi solo nelle funzioni SUCC e PRED e quando e' assegnata la lunghezza della stringa. Tutte queste condizioni sono controllate se e' attivo il metacomando $RANGECK.

L'errore si verifica anche quando l'indice di un array e' fuori dai limiti e l'array ha un tipo indice senza segno. Questa condizione e' controllata quando e' attivo il metacomando $INDEXCK.

2056 Signed Value Out Of Range

Questo errore e' simile al messaggio 2055, ma si riferisce al tipo INTEGER ed ai suoi subrange.

2057 Uninitialized 16 Bit Integer Used

E' stata usata, senza prima esser stata inizializzata, una variabile INTEGER o una variabile subrange INTEGER a 16 Bit, oppure dette variabili hanno assunto valore di -32768. Questa condizione e' controllata solo se il metacomando $INITCK e' attivo.

2058 Uninitialized 8 Bit INTEGER Used

E' stata usata una variabile SINT, oppure una variabile subrange INTEGER a 8 bit, senza essere stata prima assegnata; oppure tali variabili hanno il valore -128 che non e' valido. Questa condizione e' controllata solo se il metacomando $INITCK e' attivo.

2084 Integer Zero To Negative Power

Si e' tentato di elevare zero ad una potenza negativo (errore solo del MS-FORTRAN).

ERRORI ARITMETICI DI TIPO REAL (2100 - 2149)

Codice Messaggio

2100 REAL Divide by Zero

Un valore reale e' diviso per zero. Questo errore e' rilevato sempre.

2101 REAL Math Overflow

Un valore reale e' troppo grande per la rappresentazione. Questo errore viene rilevato sempre.

2102 SIN or COS Argument Range

Il parametro per la funzione SIN o COS e' troppo grande per portare ad un risultato significativo. Questo errore e' rilevato solo nei sistemi 8080.

2103 EXP Argument Range

Il parametro per una funzione EXP e' troppo grande e percio' porta ad un risultato che non si adatta alla rappresentazione. Questo errore e' rilevato solo nei sistemi 8080.

2104 SQRT of Negative Argument

Il parametro per una funzione di radice quadrata e' inferiore a zero. Questo errore e' sempre rilevato.

2105 LN of Non-Positive Argument

Il parametro di una funzione di logaritmo naturale e' minore o uguale a zero. Questo errore e' sempre rilevato.

2106 TRUNC/ROUND Argument Range

Il parametro REAL di una funzione TRUNC, TRUNC4, ROUND, ROUND4 e' al di fuori del campo di variabilita' di INTEGER. Questo errore e' sempre rilevato.

2131 Tangent Argument Too Small

Il parametro di una funzione TANRQQ e' tanto piccolo che il risultato non e' significativo. Questo errore e' sempre rilevato.

2132 Arcsin or Arccos of REAL > 1.0

Il parametro per una funzione ASNRQQ o ACSRQQ e' maggiore di uno. Questo errore e' sempre rilevato.

2133 Negative Real To Real Power

Il primo argomento di una funzione PRDRQQ o PRSRQQ e' minore di

zero. Questo errore e' sempre rilevato.

2134 Real Zero To Negative Power

Si e' tentato di elevare lo zero ad una potenza negativa. Questo errore e' sempre rilevato.

2135 REAL Math Underflow

La significativita' di una espressione REAL e' stata ridotta a zero.

2136 REAL Indefinite (Uninitialized Or Previous Error)

E' stato incontrato il valore REAL detto "infinito". Questo puo' verificarsi se il metacomando $1N1TCK e' attivo e viene usato un valore REAL non inizializzato, oppure se un precedente errore, come parte di un errore mascherato, pone la variabile nello stato non inizializzato.

2137 Missing Arithmetic Processor

Il programma e' stato concatenato dall'utente con la libreria runtime disposta per l'utilizzo del coprocessore numerico 8087, ma nel sistema non e' presente il coprocessore. Bisogna concatenare di nuovo il programma con la libreria runtime che emula l'aritmetica in virgola mobile (floating point).

2138 REAL IEEE Denormal Detected

E' stato generato un numero reale tanto piccolo che, a causa della perdita di significativita', non e' piu' valido.

2139 REAL Precision Loss

Un'operazione aritmetica sul coprocessore numerico ha generato la perdita di precisione numerica nel risultato di un'operazione.

2140 REAL Arithmetic Processor Instruction Illegal Or Not Emulated

E' stato fatto un tentativo di eseguire un'istruzione illegale sul comprocessore numerico, oppure l'emulatore in virgola mobile non puo' emulare un'istruzione legale del coprocessore.

ERRORI DI TIPO STRUTTURATO (2150 - 2199)

Codice Messaggio

2150 Stringa Too Long In COPYSTR

La stringa di origine per una funzione intrinseca COPYSTR e' troppo grande per la stringa di destinazione. Questo errore e' sempre rilevato.

2151 Lstring Too Long In Intrinsic Procedure

La destinazione LSTRING e' troppo piccola in una procedura INSERT, DELETE, CONCAT o COPYLIST. Questo errore e' sempre rilevato.

2180 Set Element Greater Than 255

Il valore in un insieme strutturato eccede il massimo di 255. Questo errore e' sempre rilevato.

2181 Set Element Out Of Range

Il valore di un'assegnazione di insieme o il valore del parametro insieme e' troppo grande per il set di destinazione. Questo errore e' rilevata solo se e' attivo il metacomando $RANGECK.

ERRORI INTEGER4 (2200 - 2249)

Codice Messaggio

2200 Long Integer Divide by Zero

Un valore INTEGER4 e' diviso per zero. Questo errore e' sempre rilevato.

2201 Long Integer Math Overflow

Un valore INTEGER4 e' troppo grande per la rappresentazione. Questo errore e' sempre rilevato.

2234 Long Integer Zero To Negative Power

Si e' tentato di elevare lo zero ad una potenza negativa. (Errore rilevato solo dall'MS-FORTRAN).

ALTRI ERRORI (2400 - 2999)

Codice Messaggio

2400 Illegal Pcode

Questo e' un errore interno che puo' manifestarsi solo nei sistemi p-code.

2450 Unit Version Number Mismatch

Durante l'inizializzazione dell'unita' si e' scoperto che l'utente (uno di quelli con clausola USES) e l'implementazione dell'interfaccia sono stati compilati con versioni di interfaccia diverse. Questo errore e' sempre rilevato.

AVVISO

La Ing. C. Olivetti & C. S.p.A. si riserva il diritto di apportare modifiche al prodotto descritto in questo manuale in qualsiasi momento e senza preavviso.

Questo materiale è stato preparato da Olivetti esclusivamente per l'uso da parte dei propri clienti.

Olivetti garantisce che il presente materiale costituisce, alla data di edizione, la più aggiornata documentazione da essa elaborata relativa al prodotto cui si riferisce.

E' inteso che l'uso di detto materiale avviene da parte dell'utente sotto la propria responsabilità.

Nessuna ulteriore garanzia viene pertanto prestata da Olivetti (in particolare per eventuali imperfezioni, incompletezze e/o difficoltà operative), restando espressamente esclusa ogni sua responsabilità per danni diretti o indiretti comunque derivanti dall'uso di tale documentazione.