Il linguaggio Pascal è nato come strumento puramente didattico, che poi si è esteso fino a raggiungere potenzialità vicine a quelle del linguaggio C.

La caratteristica più appariscente di questo linguaggio è che tutto deve essere dichiarato prima del suo utilizzo. Il vantaggio di questo tipo di approccio sta nella possibilità di escludere errori di programmazione dovuti a digitazione errata dei nomi delle variabili, perché il compilatore si rifiuta di considerarle se non sono state dichiarate preventivamente.

Dal momento che di dialetti Pascal ne esistono molti, in questo capitolo si cerca di fare riferimento allo standard ANSI, anche se potrebbe essere particolarmente riduttivo. Gli esempi che vengono proposti dovrebbero essere compatibili con i compilatori descritti nel capitolo u118, senza bisogno di configurazioni particolari.

Struttura fondamentale

Il Pascal impone una struttura nella preparazione dei sorgenti. L'esempio seguente è un programma che non fa alcunché.

```
program Nulla;

begin
end.
```

Nella prima riga dell'esempio, si può osservare la definizione del nome del programma, attraverso la direttiva '**`program`**'. Il nome, in questo caso è '**`Nulla`**', non deve corrispondere necessariamente al nome del file.

Le parole chiave ‘**begin**’ e ‘**end**’ delimitano lo spazio utilizzato per le istruzioni del programma, che in questo caso non esistono.

Il punto finale, dopo la parola chiave ‘**end**’, serve a indicare al compilatore la conclusione del programma, che può apparire solo alla fine del sorgente.

Istruzioni Pascal

Le istruzioni Pascal terminano con un punto e virgola (‘**;**’), così un’istruzione può impiegare più righe senza bisogno di utilizzare simboli di continuazione, oppure, su una riga possono apparire più istruzioni (sempre separate con il punto e virgola).

È possibile raggruppare più istruzioni attraverso i delimitatori ‘**begin**’ e ‘**end**’: il primo dei due viene seguito dalle istruzioni senza l’uso del punto e virgola, mentre il secondo termina normalmente con un punto e virgola, oppure un punto se si tratta del delimitatore che conclude il programma.

```
istruzione;
```

```
begin istruzione; istruzione; istruzione; end;
```

L’istruzione nulla può essere rappresentata da un punto e virgola isolato.

Nomi

Secondo il Pascal standard, i nomi che servono per identificare ciò che si utilizza, come variabili, procedure o funzioni, sono composti da una lettera alfabetica, seguita da una combinazione libera di altre lettere e cifre numeriche. Secondo lo standard originale non è ammissibile l’uso del trattino basso, ma la maggior parte dei compilatori ammette anche questo carattere.

La lunghezza dei nomi dovrebbe essere libera, con la limitazione che ogni compilatore è in grado di distinguere i nomi solo in base a un numero massimo di caratteri. Il valore minimo definito dallo standard è di otto caratteri.

Per quanto riguarda i nomi, il Pascal non distingue tra maiuscole e minuscole, come invece avviene nel linguaggio C.

Commenti

Il Pascal consente l’utilizzo di due tipi di delimitatore per circoscrivere i commenti: le parentesi graffe (‘**{**’ e ‘**}**’) e la coppia ‘**(***’ ‘***)**’. Generalmente non sono ammissibili i commenti annidati, cioè quelli a più livelli.

Quello che segue è l’esempio del programma che non fa alcunché, con qualche commento.

```
{
    Ecco un programma che non fa proprio nulla.
}
program Nulla;

begin
    (* è qui che ha luogo il «nulla» *)
end.
```

Esistono due tipi di delimitatori per i commenti solo perché i primi, cioè le parentesi graffe, sono difficili da ottenere nelle prime tastiere di alcuni paesi europei.

Suddivisione di un programma Pascal

Il linguaggio Pascal è un po’ rigido per ciò che riguarda la sequenza con cui possono essere descritte le varie parti che lo compongono. Si distinguono tre parti fondamentali nel file sorgente:

1. intestazione del programma -- si tratta della dichiarazione ‘**program**’ seguita dal nome;
2. dichiarazioni -- è lo spazio in cui si dichiara tutto ciò che viene usato nel programma, per esempio le variabili, le procedure e le funzioni;

3. istruzioni -- è lo spazio, delimitato dalle parole chiave ‘**begin**’ ‘**end**’, in cui si inseriscono le istruzioni del programma, ovvero è quello che in altri linguaggi di programmazione è la funzione o la procedura principale.

È il caso di osservare che i commenti possono essere collocati in ogni punto del file sorgente.

Output elementare

Quasi tutti gli esempi di programmazione elementare, in qualunque linguaggio di programmazione, utilizzano un’istruzione per l’output elementare.

Negli esempi che vengono mostrati inizialmente, si fa spesso uso della procedura ‘**Writeln()**’, la quale si occupa semplicemente di emettere attraverso lo standard output tutti gli argomenti forniti. L’esempio seguente serve a emettere la frase «1000 volte ciao mondo!», utilizzando due parametri: la costante numerica 1000 e la stringa « volte ciao mondo!».

```
program CiaoMondo1000;

begin
    Writeln(1000, ' volte ciao mondo!');
end.
```

Si tenga presente, in ogni caso, che ‘**Writeln**’ e ‘**writeln**’ sono la stessa cosa.

Variabili e tipi

I tipi di dati elementari del linguaggio Pascal dipendono dal compilatore utilizzato e dall’architettura dell’elaboratore sottostante. I tipi standard del Pascal ANSI sono elencati nella tabella u119.4. Il tipo ‘**char**’, non fa parte dello standard ANSI, ma è molto diffuso e così appare incluso in quella tabella.

Tabella u119.4. Elenco dei tipi di dati primitivi fondamentali in Pascal.

Tipo	Descrizione
`int`	Numeri interi positivi e negativi.
`byte`	Interi positivi di un solo byte (da 0 a 255).
`real`	Numeri a virgola mobile.
`boolean`	Valori logici booleani.
`char`	Carattere (generalmente di 8 bit).

Valori contenibili e costanti letterali

Ogni tipo di variabile può contenere un solo tipo di dati, esprimibile eventualmente attraverso una costante letterale scritta secondo una forma adatta.

I valori numerici vengono espressi da costanti letterali senza simboli di delimitazione.

- Gli interi (‘**integer**’) vanno espressi con numeri normali, senza punti di separazione di un’ipotetica parte decimale, prefissati eventualmente dal segno meno (‘**-**’) nel caso di valori negativi.
- I valori ‘**byte**’ vanno espressi come gli interi positivi, con la limitazione della dimensione massima.
- I numeri reali (‘**real**’) possono essere espressi come numeri aventi una parte decimale, segnalata dalla presenza di un punto decimale.

  Se si vuole indicare un numero reale corrispondente a un numero intero, si deve aggiungere un decimale finto, per esempio, il numero 10 si può rappresentare come ‘**10.0**’.

  Naturalmente è ammissibile anche la notazione esponenziale, come per esempio ‘**7e-2**’ che corrisponde in pratica a $7*(10^{-2})$, pari a 0,07 (scritto ‘**0.07**’).

I valori logici vengono espressi dalle costanti letterali '**TRUE**' e '**FALSE**'.

I valori carattere e stringa, vengono delimitati da coppie di apici singoli, come '**'A'**', '**'B'**', ... '**'Ciao Mondo!'**'.

Dichiarazione delle variabili

La dichiarazione delle variabili può essere fatta esclusivamente prima di un blocco '**begin**' '**end**' di un programma, di una funzione o di una procedura.

```
var nome : tipo;
```

Dalla sintassi si vede l'utilizzo della parola chiave '**var**', seguita dal nome della variabile da definire, quindi da due punti ('**:**'), infine dalla definizione del tipo di variabile.

In realtà, è possibile anche indicare un elenco di nomi, separati da virgole, quando questi devono essere tutti dello stesso tipo; inoltre, è possibile dichiarare più variabili differenti, utilizzando la parola chiave '**var**' una sola volta.

Segue la descrizione di alcuni esempi.

- ```
  var     conta   :       integer;
  ```
  Dichiara la variabile '**conta**' di tipo intero.
- ```
  var     conta,canta     :       integer;
  ```
  Dichiara le variabili '**conta**' e '**canta**' di tipo intero.
- ```
  var     conta   :       integer;
          canta   :       integer;
  ```
  Esattamente uguale all'esempio precedente.
- ```
  var
          conta   :       integer;
          lettera :       char;
  ```
  Dichiara la variabile '**conta**' di tipo intero e la variabile '**lettera**' di tipo carattere.

Operatori ed espressioni

Gli operatori sono qualcosa che esegue un qualche tipo di funzione, su uno o due operandi, restituendo un valore. Il tipo di valore restituito varia a seconda dell'operatore e degli operandi utilizzati. Per esempio, la somma di due interi genera un intero, mentre una divisione di un valore intero per un altro numero intero, genera un numero reale.

Operatori aritmetici

Gli operatori che intervengono su valori numerici sono elencati nella tabella u119.9.

Tabella u119.9. Elenco degli operatori aritmetici e di quelli di assegnamento relativi a valori numerici.

Operatore e operandi	Descrizione
op1 `+` ***op2***	Somma i due operandi.
op1 `-` ***op2***	Sottrae dal primo il secondo operando.
op1 `*` ***op2***	Moltiplica i due operandi.
op1 `/` ***op2***	Divide il primo operando per il secondo, il risultato è in virgola mobile.
op1 `div` ***op2***	Divide il primo operando per il secondo generando un risultato intero.
op1 `mod` ***op2***	Modulo: il resto della divisione tra il primo e il secondo operando.
var `:=` ***valore***	Assegna alla variabile il valore alla destra.

Una caratteristica fondamentale del Pascal è la sua attenzione nella coerenza dei tipi di dati utilizzati nelle espressioni e negli assegnamenti. Tanto per comprendere il problema con un esempio, un compilatore non dovrebbe consentire l'assegnamento di un valore in virgola mobile in una variabile intera. Naturalmente, ogni compilatore può utilizzare una politica differente, consentendo una conversione di tipo automatica in situazioni particolari.

In ogni caso, è necessario conoscere l'uso di alcune funzioni essenziali, utili per prevenire conflitti nel tipo dei dati.

```
Round(numero_reale)
```

```
Trunc(numero_reale)
```

Le due funzioni, usate in questo modo, restituiscono un valore intero a partire da un valore a virgola mobile. Nel primo caso il numero viene arrotondato, mentre nel secondo viene semplicemente troncato al valore intero.

Operatori di confronto e operatori logici

Gli operatori di confronto determinano la relazione tra due operandi. Il risultato dell'espressione composta da due operandi messi a confronto è di tipo booleano, rappresentabile in Pascal con le costanti '**TRUE**' e '**FALSE**'. Gli operatori di confronto sono elencati nella tabella u119.10.

Tabella u119.10. Elenco degli operatori di confronto. Le metavariabili indicate rappresentano gli operandi e la loro posizione.

Operatore e operandi	Descrizione
op1 `=` ***op2***	*Vero* se gli operandi si equivalgono.
op1 `!=` ***op2***	*Vero* se gli operandi sono differenti.
op1 `<` ***op2***	*Vero* se il primo operando è minore del secondo.
op1 `>` ***op2***	*Vero* se il primo operando è maggiore del secondo.
op1 `<=` ***op2***	*Vero* se il primo operando è minore o uguale al secondo.
op1 `>=` ***op2***	*Vero* se il primo operando è maggiore o uguale al secondo.

Quando si vogliono combinare assieme diverse espressioni logiche, comprendendo in queste anche delle variabili che contengono un valore booleano, si utilizzano gli operatori logici. Gli operatori logici sono elencati nella tabella u119.11.

Tabella u119.11. Elenco degli operatori logici. Le metavariabili indicate rappresentano gli operandi e la loro posizione.

Operatore e operandi	Descrizione
`not` ***op***	Inverte il risultato logico dell'operando.
op1 `and` ***op2***	*Vero* se entrambi gli operandi restituiscono il valore *Vero*.
op1 `or` ***op2***	*Vero* se uno o entrambi gli operandi restituiscono il valore *Vero*.

Nel Pascal tradizionale, le espressioni logiche vengono valutate in ogni parte, prima di definire il risultato finale di un operatore AND o di un operatore OR. Dal momento che questo metodo di risoluzione è inutilmente dispersivo, spesso i compilatori Pascal consentono di ottenere il «cortocircuito», attraverso cui si valutano solo le parti dell'espressione che sono indispensabili per arrivare al risultato finale.

Strutture di controllo del flusso

Il linguaggio Pascal gestisce un buon numero di strutture di controllo di flusso, compreso il salto *go-to* che comunque è sempre meglio non utilizzare e qui, volutamente, non viene presentato.

Le strutture di controllo permettono di sottoporre l'esecuzione di una parte di codice alla verifica di una condizione, oppure permettono di

eseguire dei cicli, sempre sotto il controllo di una condizione. La parte di codice che viene sottoposta a questo controllo, può essere un'istruzione singola, oppure un gruppo di istruzioni. Nel secondo caso, quasi sempre, è necessario delimitare questo gruppo attraverso l'uso di '**begin**' e '**end**'.

Dal momento che è comunque consentito di realizzare un gruppo di istruzioni che in realtà ne contiene una sola, probabilmente è meglio utilizzare sempre i delimitatori '**begin**' '**end**', a vantaggio dello stile e della leggibilità del codice.

Struttura condizionale: «if»

```
if condizione then istruzione
```

```
if condizione then istruzione else istruzione
```

Se la condizione si verifica, viene eseguita l'istruzione (o il gruppo di istruzioni) seguente; quindi il controllo passa alle istruzioni successive alla struttura. Se viene utilizzato '**else**', nel caso non si verifichi la condizione, viene eseguita l'istruzione che ne segue. Vengono mostrati alcuni esempi.

```
…
var     importo : integer;
…
if importo > 10000000 then Writeln( 'offerta vantaggiosa' );
```

```
…
var     importo         : integer;
        memorizza       : integer;
…
if importo > 10000000 then
begin
    memorizza := importo;
    Writeln( 'offerta vantaggiosa' );
end
else
    Writeln( 'meglio lasciar perdere' );
```

```
…
var     importo         : integer;
        memorizza       : integer;
…
if importo > 10000000 then
begin
    memorizza := importo;
    Writeln( 'offerta vantaggiosa' );
end
else if importo > 5000000 then
begin
    memorizza := importo;
    Writeln( 'offerta accettabile' );
end
else
    Writeln( 'meglio lasciar perdere' );
```

Il blocco *if-then-else* rappresenta un'unica istruzione in Pascal. In questo senso, dovrebbe apparire un punto e virgola alla fine del blocco, a terminare l'istruzione. Se si utilizzano raggruppamenti di istruzioni attraverso i delimitatori '**begin**' '**end**', le istruzioni contenute terminano con il punto e virgola, mentre il blocco, dopo la parola chiave '**end**', no, a meno che si tratti della fine dell'istruzione '**if**'.

Per osservare meglio questo particolare, si potrebbero riscrivere gli stessi esempi nel modo seguente, in cui il punto e virgola finale serve a concludere visivamente la dentellatura delle istruzioni '**if**'.

```
…
var     importo : integer;
…
if importo > 10000000 then
    Writeln( 'offerta vantaggiosa' )
;
```



```
…
var     importo         : integer;
        memorizza       : integer;
…
if importo > 10000000 then
    begin
        memorizza := importo;
        Writeln( 'offerta vantaggiosa' );
    end
else
    Writeln( 'meglio lasciar perdere' )
;
```

```
…
var     importo         : integer;
        memorizza       : integer;
…
if importo > 10000000 then
    begin
        memorizza := importo;
        Writeln( 'offerta vantaggiosa' );
    end
else
    if importo > 5000000 then
        begin
            memorizza := importo;
            Writeln( 'offerta accettabile' );
        end
    else
        Writeln( 'meglio lasciar perdere' )
;
```

Struttura di selezione: «case»

La struttura di selezione si ottiene con l'istruzione '**case**'. Si tratta di una struttura un po' troppo complessa per essere rappresentata facilmente attraverso uno schema sintattico. In generale, l'istruzione '**case**' permette di eseguire una o più istruzioni in base al risultato di un'espressione. L'esempio seguente mostra la visualizzazione del nome del mese, in base al valore di un intero.

```
…
var     mese    : integer;
…
case mese of
    1  : Writeln( 'gennaio' );
    2  : Writeln( 'febbraio' );
    3  : Writeln( 'marzo' );
    4  : Writeln( 'aprile' );
    5  : Writeln( 'maggio' );
    6  : Writeln( 'giugno' );
    7  : Writeln( 'luglio' );
    8  : Writeln( 'agosto' );
    9  : Writeln( 'settembre' );
    10 : Writeln( 'ottobre' );
    11 : Writeln( 'novembre' );
    12 : Writeln( 'dicembre' );
end;
```

È importante osservare l'uso del punto e virgola, che conclude ogni istruzione richiamata dai vari casi. La parola chiave '**end**' finale, conclude la struttura.

Un gruppo di casi può essere raggruppato assieme, quando si vuole che ognuno di questi esegua lo stesso insieme di istruzioni:

```
…
var     anno    : integer;
        mese    : integer;
        giorni  : integer;
…
case mese of
    1,3,5,7,8,10,12 :
        giorni := 31;
    4,6,9,11 :
        giorni := 30;
    2   :
        if ((anno mod 4 = 0) and not (anno mod 100 = 0)) or
                (iAnno mod 400 = 0) then
            giorni := 29
        else
            giorni := 28
        ;
end;
```

È anche possibile definire un caso predefinito che si verifichi quando nessuno degli altri si avvera:

```
…
var     mese    : integer;
…
case mese of
    1  : Writeln( 'gennaio' );
    2  : Writeln( 'febbraio' );
…
    11 : Writeln( 'novembre' );
    12 : Writeln( 'dicembre' );
else
    Writeln( 'mese non corretto' );
end;
```

Un intervallo di casi può essere indicato facilmente come nell'esempio seguente:

```
…
var     mese    : integer;
…
case mese of
    6..9 : Writeln( 'mesi caldi' );
    …
end;
```

Iterazione con condizione di uscita iniziale: «while»

```
while condizione do istruzione
```

La struttura '**`while`**' esegue un'istruzione finché la condizione restituisce il valore *Vero*. La condizione viene valutata prima di eseguire l'istruzione e poi ogni volta che termina un ciclo, prima dell'esecuzione del successivo.

Come sempre, al posto della singola istruzione se ne può inserire un raggruppamento, delimitato dalle parole chiave '**`begin`**' e '**`end`**'. L'esempio seguente fa apparire per 10 volte la lettera «x»:

```
program DieciX;

var contatore   : integer;

begin
    contatore := 0;
    while contatore < 10 do
    begin
        contatore := contatore + 1;
        Writeln( 'x' );
    end;
end.
```

La struttura '**`while`**' è un'istruzione singola in Pascal. Per sottolinearlo, si potrebbe cambiare la dentellatura dell'esempio appena mostrato per fare in modo che il punto e virgola finale, che chiude l'istruzione, inizi sulla stessa colonna della parola chiave '**`while`**'.

```
…
    contatore := 0;
    while contatore < 10 do
        begin
            contatore := contatore + 1;
            Writeln( 'x' );
        end
    ;
…
```

Iterazione con condizione di uscita finale: «repeat-until»

```
repeat istruzione;… until condizione;
```

La struttura '**`repeat`**' '**`until`**' permette di eseguire un gruppo di istruzioni una volta e poi di ripeterne l'esecuzione fino a quando la condizione posta alla fine continua a non verificarsi.

Ci sono quindi due diversità fondamentali, rispetto alla struttura '**`while`**': il gruppo di istruzioni viene eseguito sicuramente almeno una volta; il verificarsi della condizione implica l'interruzione del ciclo.

Per quanto riguarda la sintassi usata dal Pascal, c'è da osservare che dopo la parola chiave '**`repeat`**' possono essere collocate una serie di istruzioni, senza bisogno di un raggruppamento '**`begin`**' '**`end`**'. In questo senso, ogni istruzione termina con il suo punto e virgola.

L'esempio seguente è solo un pretesto per mostrare il funzionamento di questa struttura: visualizza 10 volte la lettera «x».


```
program DieciX;

var contatore   : integer;

begin
    contatore := 0;
    repeat
        contatore := contatore + 1;
        Writeln( 'x' );
    until contatore = 10;
end.
```

Iterazione enumerativa: «for»

```
for variabile; := inizio to fine do istruzione
```

L'istruzione '**`for`**' permette di definire un ciclo enumerativo, in cui una variabile intera viene inizializzata, quindi viene eseguita ripetitivamente l'istruzione controllata, incrementando alla fine di ogni esecuzione tale variabile e interrompendo il ciclo quando questa raggiunge il valore finale (quando la variabile ha raggiunto il valore finale, si esegue l'istruzione per l'ultima volta). L'incremento è di un'unità quando il valore finale è maggiore di quello iniziale, oppure di un'unità negativa quando il valore finale è minore di quello iniziale.

L'esempio già visto, in cui veniva visualizzata per 10 volte una «x», potrebbe tradursi nel modo seguente, attraverso l'uso di un ciclo '**`for`**'.

```
program DieciX;

var contatore   : integer;

begin
    for contatore := 1 to 10 do
        Writeln( 'x' )
    ;
end.
```

Come sempre, al posto di controllare una singola istruzione, se ne può gestire un gruppo, attraverso l'uso dei delimitatori '**`begin`**' e '**`end`**'. L'esempio già visto, potrebbe eventualmente tradursi nel modo seguente:

```
…
    for contatore := 1 to 10 do
        begin
            Writeln( 'x' );
        end
    ;
…
```

Procedure e funzioni

Il linguaggio Pascal distingue due tipi di subroutine: procedure e funzioni. In pratica, le procedure sono funzioni che non restituiscono alcun valore.

La dichiarazione e descrizione delle procedure e delle funzioni deve essere fatta all'interno della parte iniziale del programma, dedicata alle dichiarazioni. Procedure e funzioni possono chiamarsi a vicenda e, in ogni caso, perché la chiamata possa essere valida, occorre che la procedura o la funzione sia stata dichiarata precedentemente.

Ci sono situazioni in cui non è possibile descrivere una funzione o una procedura prima di quella chiamante. In tali casi, è possibile dichiarare una funzione senza descriverla immediatamente.

Struttura

Per il linguaggio Pascal, le procedure e le funzioni sono dei sottoprogrammi veri e propri, tanto che anche in questo caso si distinguono tre parti: intestazione, dichiarazioni e istruzioni. In particolare, l'intestazione può includere anche la dichiarazione, a meno che questa non sia separata per renderla visibile ad altre procedure e funzioni precedenti.

```
procedure nome[(parametro_formale[…])];
```

```
function nome[(parametro_formale[…])] : tipo;
```

La sintassi che appare sopra rappresenta la dichiarazione di una procedura e di una funzione. Come si può osservare, a parte la parola chiave iniziale, la funzione ha alla fine l'indicazione del tipo di dati che restituisce.

Se la procedura o la funzione non richiede l'indicazione di parametri, allora non è necessario specificare alcun ***parametro formale***, quindi non sono necessarie nemmeno le parentesi tonde.

Dopo la dichiarazione della funzione o della procedura, vanno indicate le dichiarazioni, per esempio le variabili utilizzate, nello stesso modo già visto per il programma.

Infine vanno poste le istruzioni, all'interno di un raggruppamento '**begin**' '**end**'. A differenza del raggruppamento analogo che riguarda il blocco principale del programma, la parola chiave '**end**' è conclusa con un punto e virgola invece che con il punto.

La funzione restituisce un valore, attraverso l'assegnamento a una variabile ipotetica che ha lo stesso nome della funzione.

Segue la descrizione di alcuni esempi.

- ```
  procedure CiaoCiao;
  begin
      Writeln('Ciao a tutti');
      Writeln('ciao ciao ciao');
  end;
  ```
  Si tratta di una procedura elementare che non utilizza alcun parametro e si limita a emettere un messaggio di saluto.
- ```
  function CiaoCiao : boolean;
  begin
      Writeln('Ciao a tutti');
      Writeln('ciao ciao ciao');
      CiaoCiao := TRUE;
  end;
  ```
  Si tratta di una funzione elementare che non utilizza alcun parametro e si limita a emettere un messaggio di saluto, restituendo sempre il valore booleano *Vero*.

Campo di azione

Sia le variabili che le procedure e le funzioni, hanno un campo di azione. Le variabili dichiarate nella parte introduttiva di un programma, prima della dichiarazione di procedure e funzioni, sono accessibili al corpo del programma e a tutte le procedure e funzioni. Le variabili dichiarate nella parte introduttiva di una procedura o di una funzione, hanno effetto locale, non essendo visibili all'esterno; se queste hanno nomi già utilizzati per le variabili globali, di fatto ne impediscono l'accesso.

Le procedure e le funzioni, in qualità di sottoprogrammi, possono contenere anche la dichiarazione di sottoprocedure e sottofunzioni. In tal caso, tali subroutine sono accessibili solo dal codice contenuto nella procedura o funzione in cui sono dichiarate. Nello stesso modo, le variabili locali delle procedure o delle funzioni sono accessibili anche alle rispettive sottoprocedure e sottofunzioni.

Forward

Si è accennato al fatto che, perché una chiamata possa essere valida, occorre che la procedura o la funzione in questione sia stata dichiarata prima, cioè in una posizione precedente all'interno del sorgente.

In presenza di chiamate ricorsive tra più procedure o funzioni, diviene impossibile che ogni chiamata si riferisca sempre a qualcosa di definito e descritto in precedenza.

Per risolvere il problema, si può dichiarare una procedura o una funzione prima della sua descrizione effettiva, attraverso l'uso della parola chiave '**forward**', come nell'esempio seguente:

```
…
procedure MiaProcedura(…);
forward;
…
…
procedure MiaProcedura;
begin
    …
end;
…
```


La dichiarazione della procedura o della funzione deve contenere la dichiarazione di tutti i parametri formali, mentre la descrizione è assente.

Parametri formali e chiamata per valore o per riferimento

La descrizione dei parametri formali, all'interno della dichiarazione di una procedura o di una funzione, richiede la definizione del nome delle variabili e del tipo relativo. Il campo di azione di queste variabili è locale.

```
…
procedure MiaProcedura( primo,secondo : integer;
                        terzo         : char);
begin
    …
end;
…
```

L'esempio mostra la dichiarazione di una procedura che utilizza tre parametri formali, denominati casualmente proprio: '**primo**', '**secondo**' e '**terzo**'. I primi due sono di tipo '**integer**', mentre l'ultimo è di tipo '**char**'.

Come si può osservare, la dichiarazione dei parametri formali è molto simile alla dichiarazione delle variabili, con la differenza che ciò avviene all'interno di parentesi tonde, oltre al fatto che (per il momento) manca la parola chiave '**var**'.

Una procedura o una funzione in cui i parametri formali siano stati dichiarati in questo modo, riceve una copia dei dati nel momento della chiamata, senza poter riflettere all'indietro le modifiche che a questi dovesse applicare. Si ha in pratica una chiamata per valore.

È possibile dichiarare una procedura o una funzione in cui la chiamata sia per riferimento, in modo da riflettere all'indietro le modifiche, utilizzando la parola chiave '**var**'.

```
…
procedure MiaProcedura( primo       : integer;
                        var secondo : integer;
                        terzo       : char);
begin
    …
end;
…
```

L'esempio mostra una variante in cui si dichiara che il secondo parametro formale, '**secondo**', riflette all'indietro le modifiche che dovessero essergli apportate all'interno della procedura.

Chiamata e parametri attuali

La chiamata di una procedura o di una funzione, avviene semplicemente nominandola e facendola seguire dall'indicazione dei ***parametri attuali***, cioè dei valori che si vuole siano passati per l'elaborazione.

La differenza fondamentale tra procedure e funzioni sta nel fatto che le chiamate alle prime vengono utilizzate come istruzioni pure e semplici, mentre le seconde, vanno inserite all'interno di espressioni.

Merita un minimo di attenzione anche il tipo di chiamata: per valore o per riferimento. Nel primo caso, non si pongono problemi di alcun tipo, dal momento che la funzione o la procedura chiamata non può alterarli; se invece si tratta di una chiamata per riferimento, occorre fare attenzione che il parametro attuale, usato nella chiamata, non sia una costante, perché questo genererebbe un errore irreversibile.

```
…
var     MioNumero : integer;
…
procedure MiaProcedura( primo       : integer;
                        var secondo : integer;
                        terzo       : char);
begin
    …
    secondo := 777;
    …
end;
…
{ inizio del programma }
begin
    MiaProcedura( 123, MioNumero, 'C' );
    Writeln( MioNumero );
end.
```

L'esempio mostra una chiamata a una procedura in cui uno dei parametri deve essere chiamato per riferimento. In tal caso, il parametro attuale corrispondente, utilizzato nella chiamata, è necessariamente una variabile.

I/O elementare

Per le operazioni di I/O elementare, cioè per l'utilizzo di standard output e standard error, si hanno a disposizione due coppie di procedure: '**`Write()`**' e '**`Writeln()`**'; '**`Read()`**' e '**`Readln()`**'. La prima coppia per emettere qualcosa attraverso lo standard output, la seconda per leggere qualcosa dallo standard input.

Anche se non è ancora stato affrontato l'argomento stringhe, è opportuno anticipare che per inserire un apice singolo all'interno di una costante stringa, basta indicarne due consecutivi. Per esempio, la stringa seguente,

```
'questa è la ''vera'' verità'
```

Si traduce in:

```
questa è la 'vera' verità
```

Procedure «Write()» e «Writeln()»

```
Write(elemento_da_visualizzare [ : dimensione [ : decimali ] ] [ , … ] )
```

```
Writeln(elemento_da_visualizzare [ : dimensione [ : decimali ] ] [ , … ] )
```

Le procedure '**`Write()`**' e '**`Writeln()`**' permettono di emettere attraverso lo standard output il contenuto di tutti i parametri che gli vengono forniti. A seconda dei tipi di dati utilizzati, vengono effettuate tutte le conversioni necessarie a ottenere un risultato stringa.

Se un parametro attuale, fornito nella chiamata, viene indicato seguito da due punti ('`:`') e quindi da un numero, si stabilisce lo spazio (espresso in colonne) che questo deve utilizzare nell'output. Se si specifica tale dimensione, l'informazione viene rappresentata allineandola a destra. Questa possibilità di definire la dimensione viene utilizzata prevalentemente per i dati numerici e in questo senso sta la logica dell'allineamento a destra.

Se si vuole rappresentare un valore numerico con decimali, è abbastanza importante fissare la dimensione della visualizzazione, aggiungendo anche l'indicazione delle colonne da riservare alla parte decimale. Diversamente, la rappresentazione risulterebbe in notazione esponenziale.

L'unica differenza tra le due procedure, sta nel fatto che '**`Writeln()`**' aggiunge automaticamente, alla fine della stringa visualizzata, il codice di interruzione di riga, in modo da riportare il cursore all'inizio della riga successiva.

Segue la descrizione di alcuni esempi.

- ```
  var totale : integer;
  …
  totale := 1950000;
  …
  Write('Totale:', totale:11);
  ```
  Emette la stringa seguente, senza portare a capo il cursore alla fine:

  ```
  Totale:    1950000
  ```

- ```
  var totale : real;
  …
  real := 1234.5678;
  …
  Writeln('Totale:', totale:11:5);
  ```
  Emette la stringa seguente, portando a capo il cursore alla fine.

  ```
  Totale: 1234.56780
  ```


Procedure «Read()» e «Readln()»

```
Read( variabile [ , … ] )
```

```
Readln( variabile [ , … ] )
```

Le procedure '**`Read()`**' e '**`Readln()`**' permettono di leggere dallo standard input dei valori per le variabili che vengono indicate come parametri della chiamata. I dati inseriti, vengono distinti in base all'inserimento di spaziature, così come avviene di solito con gli argomenti di un comando del sistema operativo.

È importante che i dati inseriti siano compatibili con il tipo delle variabili utilizzate, altrimenti si rischia di ottenere un errore irreversibile durante il funzionamento del programma.

La differenza tra le due procedure sta nel fatto che '**`Readln()`**' dovrebbe restituire l'eco del codice di interruzione di riga, quando si preme [*Invio*] per concludere l'inserimento dei dati, mentre '**`Read()`**' no. In pratica, può darsi che il compilatore non riesca a distinguere tra le due procedure, comportandosi sempre nello stesso modo.

Segue la descrizione di alcuni esempi.

- ```
  var totale : integer;
  …
  Write('Inserisci il totale: ');
  Read(totale);
  …
  ```
  Emette l'invito a inserire un valore e quindi lo attende dallo standard input.

- ```
  var capitale : integer;
  var tasso    : real;
  …
  Write('Inserisci di seguito il capitale e il tasso: ');
  Read(capitale,tasso);
  …
  ```
  Emette l'invito a inserire due valori consecutivi: un intero e un valore decimale.

Struttura del sorgente: le dichiarazioni

È già stato accennato alla struttura di un sorgente Pascal: del programma, delle procedure e delle funzioni. Si tratta di tre parti fondamentali:

1. intestazione del programma, dichiarazione della procedura o della funzione;
2. dichiarazioni;
3. istruzioni.

Il punto più delicato è la definizione della parte delle dichiarazioni, dato che nel Pascal originale esiste un ordine preciso nel tipo di istruzioni che possono esservi inserite. Si tratta di dichiarazioni:

1. '**`label`**'
2. '**`const`**'
3. '**`type`**'
4. '**`var`**'
5. '**`procedure`**'
6. '**`function`**'

La maggior parte di queste dichiarazioni non è ancora stata descritta. In particolare, '**`label`**', dal momento che serve a realizzare dei salti incondizionati senza ritorno (*go-to*), non viene descritta in questi capitoli sul Pascal.

«

- Gordon Dodrill, *Pascal Language Tutorial*
  http://www.geocities.com/hotdogcom/ptutor/paslist.html
  http://packetstormsecurity.nl/programming-tutorials/Pascal/pascal-tutorial/paslist.htm