Introduzione al linguaggio PASCAL

Indice

Convenzioni

Negli esempi di codice e nelle descrizioni della sintassi forniti in questa dispensa saranno usati caratteri a spaziatura fissa e in grassetto (es. **`program`**) per indicare elementi del programma che hanno un significato predefinito; saranno usati caratteri a spaziatura fissa non in grassetto (es. `pippo`) per indicare elementi scelti dal programmatore, che vanno riportati nel programma ma potrebbero essere cambiati; infine, con caratteri in corsivo (es. *identificatore*) per indicare parti del programma che non vanno inserite alla lettera ma sostituite da opportuni elementi della sintassi del linguaggio che abbiano il significato indicato dalla parola. Ad esempio, se viene riportata un'istruzione della forma:

```
pippo := espressione ;
```

si intende che i simboli '`:=`' e '`;`' devono essere inseriti così come sono riportati; il nome '`pippo`' può essere eventualmente sostituito da un altro nome scelto dal programmatore; infine, al posto di '*espressione*' occorre inserire un'espressione valida secondo la sintassi del Pascal.
Spesso, per illustrare elementi della sintassi del linguaggio, saranno forniti per brevità dei frammenti di codice, ai quali mancano alcune parti per ottenere un programma completo e funzionante; verrà indicato esplicitamente quando l'esempio riportato costituisce un programma completo. Verranno talvolta usati dei puntini sospensivi (. . . . .) per indicare l'omissione di parti del programma il cui ruolo è facilmente ricavabile dal contesto.

Elementi di un programma

Un programma Pascal è costituito da *parole chiave*, *identificatori*, *costanti letterali*, *operatori*, *separatori*, *spazi* e *commenti*.

- Le *parole chiave* (*keywords*) sono nomi che hanno un significato predefinito nel linguaggio, e possono essere usati solo con quel significato. Tipicamente si tratta di parole della lingua inglese o di loro abbreviazioni. La seguente tabella contiene le principali parole chiave del Pascal:

and	array	begin	case
const	div	do	downto
else	end	file	for
function	goto	if	in
label	mod	nil	not
of	or	procedure	program
record	repeat	set	string
then	to	type	until
var	while	with	

- Gli *identificatori* sono nomi scelti dal programmatore per denominare parti del programma o informazioni trattate dal programma stesso. Un identificatore è costituito di lettere, cifre e trattini di sottolineatura (_), con i vincoli che il primo carattere sia una lettera e che l'identificatore non coincida con nessuna delle parole chiave. Esempi validi di identificatori sono: `pippo`, `C3PO`, `aleph0`, `i`, `temperatura_acqua`. Nel linguaggio Pascal (a differenza di altri linguaggi) non è considerata significativa la differenza tra lettere maiuscole e minuscole, per cui `C3PO` e `c3po` rappresentano il medesimo identificatore.
  Alcuni identificatori hanno un significato predefinito dal linguaggio; tuttavia è possibile ridefinirli con un significato scelto dal programmatore (a differenza di quanto avviene con le parole chiave) qualora non sia necessario usare il significato preassegnato, sebbene questa operazione non sia consigliabile in quanto riduce la comprensibilità del programma.

- Le *costanti letterali* rappresentano dei valori inseriti direttamente nel testo del programma. In particolare, i valori possono essere dei seguenti tipi:

  - Valori logici: si usano le parole **`true`** (vero) e **`false`** (falso) per indicare una costante letterale che rappresenta un valore logico.

  - Numeri interi: es. `42`, `-108`.

  - Numeri reali: la notazione per i numeri reali segue la convenzione anglosassone di usare il punto invece della virgola come separatore per i decimali: es. `3.1416`, `-0.44`. E' possibile indicare numeri molto grandi o molto piccoli attraverso la *notazione scientifica*, in cui il numero è seguito dalla lettera `E` e da un numero intero (positivo o negativo) che rappresenta l'esponente della potenza di 10 da usare come fattore di scala. Ad esempio, la costante letterale `2.97E8` rappresenta il numero 2.97 moltiplicato per 10^8, ovvero 297000000.

  - Caratteri o sequenze di caratteri (dette *stringhe* di caratteri): le costanti letterali di

questo tipo sono espresse racchiudendo i caratteri tra apici (`'`). Un utilizzo tipico di queste costanti letterali è per indicare messaggi da stampare sullo schermo. Esempi:

```
'x', 'Questa è una sequenza di caratteri.',
'Ma 9 per 9 farà 81?'
```

Nel caso in cui la sequenza debba contenere proprio il carattere di apice (`'`), occorre scrivere questo carattere due volte in modo da distinguerlo dall'apice che termina la costante letterale. Es.: `'L''apostrofo'` corrisponde alla sequenza di caratteri: `L-'-a-p-o-s-t-r-o-f-o`.

- Gli *operatori* sono simboli o parole chiave usati per rappresentare le operazioni elementari definite sui tipi di dato elementari. Più operazioni possono essere combinate per formare *espressioni* del tutto analoghe a quelle usate nell'algebra; in tal caso le operazioni vengono eseguite secondo un ordine determinato dalla *precedenza* degli operatori (es., come in algebra la moltiplicazione ha la precedenza sull'addizione), a meno che non sia indicato esplicitamente un ordine attraverso l'uso delle parentesi tonde (in Pascal si usano solo le parentesi tonde nelle espressioni, ma è possibile avere parentesi tonde all'interno di altre parentesi tonde). I principali operatori sono indicati nella seguente tabella, dove a valori più alti della precedenza corrispondono operazioni che vengono eseguite per prime:

Operatore	Precedenza	Operazione effettuata
=	1	Uguale a
<,	1	Minore di, maggiore di
<=, =	1	Minore o uguale, maggiore o uguale
<	1	Diverso da
+,-	2	Addizione, sottrazione
*	3	Moltiplicazione
/	3	Divisione con risultato reale
DIV	3	Quoziente della divisione tra interi
MOD	3	Resto della divisione tra interi
OR	2	OR dell'algebra di Boole
AND	3	AND dell'algebra di Boole
NOT	4	NOT dell'algebra di Boole

Gli operatori `+, -, *, /, DIV, MOD` sono detti *aritmetici*. Gli operatori `=, <, , <=, =, <` sono detti *relazionali*. Infine gli operatori `AND, OR e NOT` sono detti *logici*.

- I *separatori* sono simboli utilizzati per delimitare parti del programma; corrispondono ai segni di punteggiatura del linguaggio naturale. Esempi di separatori sono il punto (`.`), la virgola (`,`), i due punti (`:`) e il punto e virgola (`;`). L'uso dei separatori è rigidamente stabilito dalla sintassi del linguaggio.

- Elementi adiacenti del programma (ma non i caratteri che compongono un singolo elemento) possono essere separati da *spazi*, che vengono ignorati nell'interpretazione del programma; l'andata a capo viene considerata equivalente a uno spazio, ed è possibile inserire sequenze di spazi di lunghezza arbitraria. La separazione con spazi è necessaria quando gli elementi adiacenti sono parole chiave, identificatori o costanti letterali, mentre in altri casi può essere opzionale. Si noti che all'interno di una costante letterale di tipo carattere (o stringa) gli spazi NON vengono ignorati.

- I *commenti* sono testi descrittivi inseriti nel programma a beneficio di un eventuale lettore, per rendere più comprensibile l'algoritmo utilizzato. Essi non hanno alcun effetto sull'esecuzione. Possono essere inseriti dovunque è possibile inserire degli spazi. I commenti sono delimitati dai caratteri `{` e `}`, o in alternativa dalle sequenze di caratteri `(*` e `*)`. Esempi di commenti:

```
{ Questo è un commento }
(* Anche questo
   è un commento
*)
```

Un esempio di programma completo

Per tradizione, il primo programma di un qualsiasi linguaggio di programmazione è il programma che si limita a visualizzare sullo schermo un messaggio di saluti o qualcosa di simile. Nel linguaggio Pascal tale programma può essere formulato come segue:

```
(* Questo è il mio primo programma.
 * Lo scopo del programma è visualizzare
 * sullo schermo il messaggio 'Salve, mondo'
 *)
program Salve;
begin
  writeln('Salve, mondo');
end.
```

In questo semplice programma possiamo vedere un lungo commento iniziale, che ne descrive lo scopo; alcune parole chiave (**`program`**, **`begin`**, **`end`**), alcuni identificatori (`Salve` e `writeln`; di quest'ultimo si usa il significato predefinito); una costante letterale, che rappresenta il messaggio da visualizzare; infine vari separatori e spazi. Nei paragrafi successivi sarà spiegato più in dettaglio il significato delle varie parti di un programma e la corrispondente sintassi.

Struttura di un programma

Un programma Pascal ha la seguente struttura:

```
intestazione
sezione_dichiarativa
sezione_esecutiva
```

L'intestazione serve ad attribuire un nome al programma; il suo formato è:

```
program identificatore;
```

dove *identificatore* è un qualsiasi nome valido scelto dal programmatore.
La *sezione* o *parte dichiarativa* (che può eventualmente mancare in un programma) contiene un insieme di dichiarazioni, ovvero di indicazioni fornite all'ambiente di programmazione per specificare le caratteristiche delle informazioni trattate dal programma. La sintassi di alcune dichiarazioni sarà esaminata nei paragrafi successivi.
La *sezione* o *parte esecutiva* contiene le istruzioni che definiscono l'algoritmo che il programma deve realizzare. Il formato di questa parte è:

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

ovvero, è costituita dalla parola chiave **begin**, da una o più istruzioni, e infine dalla parola chiave **end** seguita da un punto. Le istruzioni sono separate da punti e virgola (*;*). Nei paragrafi successivi sono illustrate le principali istruzioni del linguaggio Pascal.
Il programma visto come esempio nel paragrafo precedente era costituito di un'intestazione e della sola parte esecutiva, che conteneva una singola istruzione, che come vedremo, serve a stampare qualcosa sullo schermo.

Le variabili e la loro dichiarazione

Qualsiasi programma non banale deve manipolare delle informazioni contenute nella sua memoria. Nei linguaggi ad alto livello le informazioni sono gestite attraverso le *variabili*, che rappresentano dei contenitori di informazioni. A ciascuna variabile è associato un *nome*, attraverso il quale il programma può fare riferimento all'informazione contenuta nella variabile, e un *tipo*, che determina l'insieme dei *valori* che possono essere inseriti nella variabile e delle operazioni che ha senso effettuare su tali valori. I valori delle variabili possono essere utilizzati all'interno di espressioni specificando il nome della variabile.
In Pascal le variabili usate dal programma devono essere indicate nella parte dichiarativa, specificando per ciascuna di esse il nome e il tipo. Il formato della dichiarazione delle variabili è il seguente:

```
var identificatore : tipo ;
    identificatore : tipo ;
    .....
```

dove *identificatore* indica il nome da assegnare alla variabile, e *tipo* è il nome che specifica il tipo delle informazioni. Il tipo può essere uno dei tipi predefiniti dal Pascal, riportati nella tabella sottostante, o un tipo definito dal programmatore, di cui parleremo successivamente.

Nome del tipo	Valori ammessi

`BOOLEAN`	Valori logici (true, false)
`INTEGER`	Numeri interi in un opportuno intervallo (es.: -32768 ... 32767)
`REAL`	Numeri reali
`CHAR`	Singoli caratteri (es. `'A'`)

Esempio di dicharazioni di variabili:

```
var pippo : integer;
    area : real;
```

Questa dichiarazione definisce due variabili i cui nomi sono `pippo` e `area`, che possono contenere rispettivamente numeri interi e numeri reali. Per dichiarare più variabili dello stesso tipo è possibile usare una notazione abbreviata, in cui si specificano i nomi separati da virgole e alla fine il nome del tipo preceduto dai due punti. Esempio:

```
var i, j : integer;
    p, q : boolean;
```

Le costanti simboliche

Spesso può essere opportuno assegnare un nome a valori costanti, che non devono essere modificati dal programma, per semplificare la scrittura del programma stesso. Si pensi ad esempio all'uso di pi greco in matematica. In Pascal è possibile fare questo attraverso la dichiarazione di *costanti simboliche*, che va inserita nella parte dichiarativa del programma. Il formato è:

```
const identificatore = valore;
      identificatore = valore;
      .....
```

Esempio di dichiarazioni di costanti simboliche:

```
const pi = 3.14; (* pi greco *)
      VelLuce = 2.97E8; (*  Vel. della luce in m/s *)
```

Uno dei vantaggi dell'uso delle costanti simboliche è che si può facilmente modificare il valore usato per una costante senza dover cercare tutti i punti del programma dove viene utilizzata. Ad esempio, se si vuole usare un valore di pi greco con quattro cifre decimali, basta modificare la corrispondente definizione nella parte dichiarativa del programma.
Anche i valori delle costanti simboliche, come i valori delle variabili, possono essere utilizzati nella costruzione di espressioni.
In Pascal è predefinita la costante simbolica `maxint`, che rappresenta il numero più grande rappresentabile nel tipo **integer**.

Le istruzioni elementari

Le istruzioni di uscita

Le istruzioni di uscita, o di *output*, hanno lo scopo di presentare valori o risultati di operazioni sullo schermo del computer. Le istruzioni di uscita fondamentali sono due: **`write`** e **`writeln`**. La

sintassi è la stessa:

```
writeln(espressione, espressione, .....);
```

dove il risultato delle espressioni è visualizzato sullo schermo. La differenza tra **`write`** e **`writeln`** è che quest'ultima va a capo dopo aver scritto, e quindi successive scritte sullo schermo appariranno su una nuova linea.
Esempio:

```
writeln('9 per 9 fa ' , 9*9);
```

in questo caso vengono stampate due cose: una costante letterale che corrisponde al messaggio '9 per 9 fa ', seguita dal risultato dell'espressione 9*9, che è il numero intero 81.
Nota: i numeri reali vengono normalmente stampati usando la notazione scientifica; se si vuole che venga usata la notazione convenzionale, occorre indicare dopo l'espressione il numero di cifre decimali da usare, con il seguente formato:

```
writeln(espressione:0:num_decimali);
```

Ad esempio, per stampare 3.14 con due decimali occorre l'istruzione:

```
writeln(3.14:0:2);
```

L'istruzione di assegnazione

L'istruzione di assegnazione consente di inserire un valore in una variabile. Il formato è:

```
identificatore := espressione ;
```

dove *espressione* è una qualsiasi espressione che produca un risultato che può essere inserito nella variabile il cui nome è *identificatore* (ovvero, un valore dello stesso tipo della variabile).
Esempio:

```
area := pi*raggio*raggio;
```

calcola il risultato dell'espressione a destra del simbolo '`:=`' e lo inserisce nella variabile il cui nome è '`area`'.
Si noti che l'espressione può anche contenere un riferimento alla stessa variabile in cui si inserisce il risultato:

```
i := i + 1;
```

significa: inserisci nella variabile '`i`' il *vecchio* valore della variabile '`i`' aumentato di 1.

Le istruzioni di ingresso

Le istruzioni di ingresso, o di *input*, hanno lo scopo di leggere valori forniti dall'utente del programma. Le istruzioni di ingresso fondamentali sono due: **`read`** e **`readln`**. La sintassi è la stessa:

```
readln(variabile, variabile, .....);
```

dove i valori che l'utente scrive sulla tastiera sono inseriti nelle variabili specificate, seguendo l'ordine con cui compaiono nell'istruzione. Il valore inserito dall'utente deve essere dello stesso tipo della variabile corrispondente. La differenza tra **`read`** e **`readln`** è che quest'ultima va a capo dopo aver letto l'ultimo valore; quindi se l'utente ha scritto altre informazioni sulla stessa riga, queste verranno ignorate.
Esempio:

```
readln(raggio);
```

Un esempio di programma completo

Il seguente programma illustra tutti gli elementi del Pascal visti finora.

```
program AreaDelCerchio;
const pi = 3.1416;
var raggio: real;
    area: real;
begin
  (* Chiede all'utente di specificare il raggio *)
  write('Inserisci il raggio: ');
  (* Legge il valore del raggio *)
  readln(raggio);
  (* Esegue il calcolo dell'area *)
  area := pi * raggio * raggio;
  (* Visualizza il risultato *)
  writeln('L''area del cerchio è: ', area);
end.
```

Strutture di controllo di un programma

Le istruzioni elementari possono essere raggruppate utilizzando le *strutture di controllo* del linguaggio per esprimere algoritmi più complessi della semplice esecuzione di passi predefiniti in successione. Le strutture di controllo fondamentali sono la *sequenza*, la *selezione* e *l'iterazione* (o *ripetizione*). Di seguito verranno presentati i costrutti del linguaggio Pascal che realizzano queste strutture di controllo.

Il costrutto `begin ... end`

In Pascal il fatto che più istruzioni debbano essere eseguite in sequenza si esprime racchiudendo le istruzioni tra le parole chiave **`begin`** e **`end`**, separando le istruzioni con il carattere punto e virgola (`;`). E' possibile inserire in qualunque punto del programma dove sia richiesta una singola istruzione una sequenza di istruzioni racchiuse tra **`begin`** e **`end`**; in tal caso la parola **`end`** deve essere seguita dall'eventuale punto e virgola (`;`) che seguirebbe l'istruzione singola. Questo blocco di istruzioni viene anche denominato *istruzione composta*. **Nota**: l'ultima istruzione all'interno di un blocco **`begin`** ... **`end`**non necessita del punto e virgola finale.

Il costrutto `if ... then ... else`

Questo costrutto rappresenta la maniera principale per esprimere la selezione tra due istruzioni (o tra due sequenze di istruzioni, usando l'istruzione composta) sulla base di una condizione. La sintassi del costrutto è la seguente:

```
if condizione then
  istruzione1
else
  istruzione2;
```

o in alternativa:

```
if condizione then
  istruzione1;
```

Condizione rappresenta un'espressione che abbia come risultato un valore logico (e quindi usi gli operatori logici e/o relazionali). Se *condizione* ha come valore **true**, allora viene eseguita *istruzione1* e (nel caso sia presente la parte **else**) viene ignorata *istruzione2*. Se invece *condizione* ha come valore **false**, viene ignorata *istruzione1* e se è presente la parte **else** viene eseguita *istruzione2*.
Si noti che l'istruzione che precede la parola chiave **else** NON DEVE essere terminata dal punto e virgola. E' possibile specificare più di una istruzione al posto di *istruzione1*, di *istruzione2* o di entrambe utilizzando il costrutto **begin** ... **end** visto precedentemente.

Esempio:

```
if b<0 then
    writeln('Il risultato della divisione è: ', a/b)
else
    writeln('Non posso dividere per zero!');
```

Nota: nel caso in cui si usino più istruzioni **if** *annidate* una dentro l'altra senza impiegare **begin** e **end**, può sorgere qualche ambiguità nel caso in cui non tutti gli **if** abbiano una parte **else**, come illustrato dal seguente frammento di codice:

```
if a = 0 then
  if a = 0 then
    writeln('a è zero')
else
  writeln('a è negativo');
```

A quale delle due condizioni si riferisce l'**else**? Nelle intenzioni del programmatore, chiaramente alla prima condizione. Dal punto di vista del linguaggio però un **else** viene attribuito all'**if** più recente che ne sia privo; quindi, in questo caso l'**else** fa riferimento alla seconda condizione. Per evitare di incorrere in questo genere di ambiguità si raccomanda di usare sempre **begin** ... **end** qualora l'istruzione che segue il **then** o l'**else** non sia una delle istruzioni elementari: il codice dell'esempio precedente avrebbe dovuto essere formulato senza ambiguità nel modo seguente:

```
if a = 0 then
  begin
    if a = 0 then
      writeln('a è zero');
  end
else
  writeln('a è negativo');
```

Istruzioni `if` in cascata

Capita frequentemente all'interno di un programma, di dover verificare una serie di condizioni, a ciascuna delle quali è associata un'azione, fermandosi non appena viene trovata la prima condizione vera (dopo aver eseguito l'azione corrispondente). Sebbene questo caso si riconduca all'uso di istruzioni **if** annidate una dentro l'altra, risulta preferibile evitare la proliferazione eccessiva di **begin** ... **end** che pregiudicherebbe la leggibilità del programma, adottando invece il seguente formato:

```
if condizione1 then
  istruzione1
else if condizione2 then
  istruzione2
.....
else if condizioneN then
  istruzioneN
else
  istruzione_di_default;
```

dove l'istruzione dopo l'ultimo **else** viene eseguita nel caso di *default*, cioè se tutte le condizioni sono false. La parte **else** finale può essere omessa nel caso in cui non sia necessario prevedere un'istruzione di *default.*

Il costrutto `case`

Un caso particolare di selezione richiede di dover confrontare il risultato di un'espressione con più valori costanti, a ciascuno dei quali è associata un'azione. Sebbene sia possibile esprimere questo genere di selezione usando istruzioni **if** in cascata, il costrutto **case** consente un'espressione più chiara, sintetica ed efficiente. La sintassi del costrutto è la seguente:

```
case espressione of
  valore1 : istruzione1;
  valore2 : istruzione2;
  .....
  valoreN : istruzioneN;
end;
```

o in alternativa:

```
case espressione of
  valore1 : istruzione1;
  valore2 : istruzione2;
  .....
  valoreN : istruzioneN;
else
  istruzione_di_default;
end;
```

Ciascun valore può essere rappresentato da una costante letterale o da una costante simbolica. E' anche possibile specificare più valori separati da virgole, o un intervallo di valori indicando i due estremi dell'intervallo (inclusi nell'intervallo stesso) separati da due caratteri di punto (. .). Quando viene eseguito questo costrutto, viene calcolato il valore di *espressione*, che deve essere di tipo **integer** o **char** (o, più in generale, deve appartenere a un tipo *ordinale*, di cui parleremo in seguito), e viene confrontato con i valori specificati nel costrutto. Se viene trovato un valore uguale al risultato di *espressione*, viene eseguita l'istruzione corrispondente, e termina l'esecuzione del **case**. Altrimenti, se il risultato non è tra i valori specificati, viene eseguita *istruzione_di_default*, se è presente la parte **else**.

Esempio:

```
case votoEsame of
  1 .. 17:  writeln('L''esame non è stato superato!');
  18 .. 23: writeln('L''esame è stato superato con un voto basso');
  24 .. 26: writeln('L''esame è stato superato con un voto medio');
  27 .. 30: begin
              writeln('Complimenti!');
              writeln('L''esame è stato superato con un voto alto!');
            end;
else
  writeln('Errore! Il voto non è compreso tra 1 e 30!!!');
end;
```

Il costrutto `while`

Il costrutto **while** consente di realizzare la ripetizione ciclica (o *iterazione*) di una o più istruzioni in base al verificarsi di una condizione; per questo motivo questo costrutto viene anche detto *ciclo***while**. La sintassi dell'istruzione **while** è la seguente:

```
while condizione do
  istruzione;
```

dove *condizione* è un'espressione che ha un risultato di tipo logico, e *istruzione* è un'istruzione singola o un'istruzione composta racchiusa tra **begin** e **end**, che viene detta *corpo* del ciclo. L'esecuzione di un istruzione **while** comporta i passi illustrati di seguito:

1. Viene valutata la *condizione*. Se il risultato è **false**, l'esecuzione del **while** termina ignorando *istruzione*.
2. Altrimenti, se la condizione è **true**, viene eseguita *istruzione* e si ritorna al passo 1.

In questo modo, *istruzione* viene ripetuta fintantoché *condizione* rimane vera. Ovviamente, *istruzione* deve modificare in qualche modo le variabili da cui dipende il valore di *condizione*, affinché la ripetizione possa avere termine.

Esempio:

```
(* Questo programma completo stampa i numeri da 1 a 10. *)
program StampaNumeri;
var i:integer;
begin
  i := 1;
  while i <= 10 do
    begin
      writeln(i);
      i := i+1;
    end;
end.
```

Nota: un errore molto frequente consiste nell'inserire un punto e virgola dopo la parola chiave **do**. In questo caso, il punto e virgola viene interpretato come *istruzione vuota* (ovvero, istruzione che non fa assolutamente nulla), e viene associato come *istruzione* al **while** invece dell'istruzione da eseguire ad ogni iterazione. La conseguenza di questo errore è che il **while** esegue un ciclo infinito (perché l'istruzione vuota non modifica le variabili da cui dipende la condizione) in cui

non esegue alcuna istruzione.

Il costrutto `repeat ... until`

In un ciclo **while** la condizione viene controllata sempre *prima* di eseguire l'istruzione da ripetere. Questo significa che se la condizione è falsa in partenza, l'instruzione non viene eseguita nemmeno una volta. In alcune situazioni, la verifica della condizione richiede alcune informazioni che sono calcolate dalle stesse istruzioni che fanno parte del corpo del ciclo. Per gestire tali situazioni il Pascal prevede un altro costrutto per i cicli, il costrutto **repeat** ... **until**, la cui sintassi è:

```
repeat
  istruzione;
  istruzione;
  .....
until condizione;
```

L'esecuzione del ciclo **repeat** ... **until** è costituita dai seguenti passi:

1. Vengono eseguite in sequenza le istruzioni comprese tra **repeat** e **until**.
2. Viene valutata *condizione*; se il risultato è **true**, termina l'esecuzione del ciclo. Altrimenti ricomincia dal passo 1.

Si notino le seguenti differenze rispetto al ciclo **while**:

- La condizione viene valutata *dopo* l'esecuzione del corpo del ciclo
- Il corpo del ciclo viene eseguito sempre almeno una volta
- Il corpo del ciclo viene ripetuto se la condizione è **false**, mentre nel **while** è ripetuto se la condizione è **true**.

Esempio:

```
(* Questo frammento legge una variabile di tipo
   carattere, ripetendo la lettura finché l'utente
   non inserisce il carattere 's' o il carattere 'n'.
*)
repeat
  write('Confermi? (s/n): ');
  readln(c); (* c è di tipo CHAR *)
until (c='s') or (c='n');
```

Il costrutto `for`

In molti algoritmi è necessario ripetere una o più istruzioni per ciascun valore di un intervallo di numeri interi. Sebbene sia possibile realizzare questa operazione utilizzando i costrutti iterativi presentati precedentemente, come si è visto nell'esempio riportato nel paragrafo dedicato al costrutto **while**, nel Pascal (e in molti altri linguaggi di programmazione) esiste un costrutto dedicato specificamente a questo tipo di ripetizione. La sintassi di questo costrutto è:

```
for variabile := valore_minimo to valore_massimo do
  istruzione;
```

dove *variabile* è una variabile dichiarata di tipo **integer**, e *valore_minimo* e *valore_massimo* sono gli estremi dell'intervallo (inclusi nell'intervallo stesso) per il quale si vuole ripetere *istruzione*. Tali valori possono essere specificati attraverso costanti o anche attraverso espressioni. L'esecuzione dell'istruzione **for** comporta i seguenti passi:

1. In *variabile* viene inserito *valore_minimo*
2. Se *variabile* è maggiore di *valore_massimo* l'esecuzione del **for** termina
3. Viene eseguita *istruzione*
4. Il valore di *variabile* viene aumentato di 1 e si ricomincia dal passo 2.

Si può facilmente verificare che il numero di volte che l'istruzione viene ripetuta è 0 se *valore_minimo valore_massimo*, altrimenti è dato dalla formula:

```
valore_massimo - valore_minimo + 1
```

E' possibile anche usare il costrutto **for** facendo percorrere in ordine decrescente i valori compresi nell'intervallo, usando la sintassi:

```
for variabile := valore_massimo downto valore_minimo do
  istruzione;
```

Esempio:

```
(* Stampa i numeri da 1 a 5 in ordine crescente *)
for i := 1 to 5 do
  writeln(i);

(* Stampa i numeri da 10 a 6 in ordine decrescente *)
for i := 10 downto 6 do
  writeln(i);
```

Dichiarazioni di tipo

In aggiunta ai tipi predefiniti dal linguaggio, il programmatore può definire nuovi tipi attraverso opportune dichiarazioni di tipo da inserire nella parte dichiarativa del programma. La sintassi di una dichiarazione di tipo è la seguente:

```
type identificatore = costruttore_di_tipo ;
     identificatore = costruttore_di_tipo ;
     .....
```

dove *identificatore* rappresenta il nome che viene assegnato al tipo, e *costruttore_di_tipo* rappresenta un costrutto del linguaggio che specifica come il nuovo tipo è definito in termini dei tipi già esistenti. Nel seguito verranno illustrati i principali costruttori di tipo del Pascal.
I tipi predefiniti dal linguaggio sono tipi *atomici*, ovvero rappresentano informazioni non decomponibili in informazioni di tipo più semplice (almeno dal punto di vista del linguaggio Pascal). I tipi definiti dal programmatore possono essere atomici ma anche *strutturati*, cioè composti da più informazioni di tipo più semplice. Nel Pascal, i tipi atomici definiti dall'utente sono i *tipi enumerativi* e i *tipi subrange*. I principali tipi strutturati sono gli *array* e i *record*.
Nota: è possibile utilizzare un costruttore di tipo direttamente all'interno di una dichiarazione di variabile, al posto del nome del tipo, per definire variabili di un nuovo tipo senza bisogno di attribuire a questo tipo un nome.

Tipi enumerativi

In un tipo enumerativo esiste un numero finito di valori possibili, ciascuno dei quali è indicato

attraverso un identificatore. La sintassi per la dichiarazione di un tipo enumerativo è:

```
type identificatore = (identificatore1, ....., identificatoreN);
```

ovvero, il costruttore di tipo è formato da una lista di identificatori separati da virgole tra parentesi tonde.

Esempio:

```
type Mese = (Gennaio, Febbraio, Marzo, Aprile,
             Maggio, Giugno, Luglio, Agosto,
             Settembre, Ottobre, Novembre, Dicembre);
     GiornoSettimana = (lun, mar, mer, gio, ven, sab, dom);
```

Una volta definito un tipo enumerativo, è possibile dichiarare variabili usando questo tipo. Le operazioni definite su queste variabili includono l'assegnazione e il confronto di uguaglianza (operatori **=** e **<**). Inoltre, l'ordine con cui sono elencati i vari identificatori definisce una *relazione d'ordine* nel tipo, in base alla quale possono essere usati gli operatori di disuguaglianza (<,<=, , =). Sono inoltre predefinite le operazioni **pred** e **succ** che calcolano rispettivamente il predecessore e il successore di un elemento nella sequenza specificata nella definizione, e l'operazione **ord** che calcola la posizione del valore nella stessa sequenza (il valore di **ord(**x**)** è 0 se x è il primo elemento della lista, 1 se è il secondo e così via).
Gli identificatori dei valori del tipo possono essere usati come costanti letterali all'interno del programma.
Nota: non sono definite per i tipi enumerativi le operazioni di ingresso/uscita.

Esempio:

```
(* Questo frammento, che usa la dichiarazione del
   tipo GiornoSettimana vista prima, stampa
   per ciascun giorno tra lunedì e venerdì
   la posizione del giorno nella sequenza dei
   giorni della settimana
*)
var giorno: GiornoSettimana;
begin
  giorno := lun;
  while giorno <= ven do
    begin
      writeln( ord(giorno) ); (* Stampa la posizione del giorno *)
      giorno := succ(giorno); (* Passa al giorno successivo *)
  end;
end.
```

Tipi ordinali

Sono detti *tipi ordinali* i tipi per i quali è possibile definire una relazione "elemento successivo". I tipi enumerativi appena visti sono tipi ordinali, così come lo sono i tipi predefiniti **integer**(in questo caso l'elemento successivo di x è x+1), **boolean** (**true** è il successore di **false**) e **char** (i caratteri sono organizzati in una sequenza predefinita, che rispecchia l'ordine alfabetico, per cui il successore di 'A' è 'B' ecc.).
Su tutti i tipi ordinali sono predefinite le funzioni **ord**, **pred** e **succ** presentate per i tipi enumerativi.

Tipi *subrange*

I tipi *subrange*, o intervallo, rappresentano intervalli di valori all'interno di un tipo ordinale. La

sintassi della dichiarazione di questi tipi è:

```
type identificatore = minimo .. massimo ;
```

dove *minimo* e *massimo* sono i valori estremi dell'intervallo considerato (inclusi nell'intervallo).

Esempio:

```
type NumeroNaturale = 0 .. maxint;
     Lettera = 'A' .. 'Z';
     GiornoMese = 1 .. 31;
```

Sui tipi subrange sono definite le stesse operazioni disponibili sui tipi ordinali da cui è ricavato l'intervallo (quindi su un subrange di interi sono disponibili tutte le operazioni aritmetiche, etc.).

Tipi *array*

Un *array* (in inglese: schiera) rappresenta una collezione di informazioni del medesimo tipo, ciascuna associata a un *indice* che appartiene a un intervallo di numeri interi (in Pascal è possibile comunque usare come indici anche gli altri tipi ordinali). La dichiarazione di un tipo *array* usa la seguente sintassi:

```
type identificatore = array [ indice_minimo .. indice_massimo ] of tipo_base
;
```

dove *indice_minimo* e *indice_massimo* sono gli estremi (inclusi) dell'intervallo di possibili indici, e *tipo_base* è il tipo degli elementi di cui è costituito l'array. Il *tipo_base* può essere sia uno dei tipi predefiniti del Pascal che un tipo definito dall'utente (sia atomico che strutturato). Il numero di elementi di un array è calcolabile come: `indice_massimo - indice_minimo +1`.
Su una variabile di tipo array è possibile accedere ai singoli elementi usando la notazione:

```
 variabile [ espressione ]
```

dove il risultato di *espressione* viene usato come indice per selezionare l'elemento corrispondente dell'array. Un elemento di un array può essere usato come una variabile appartenente al *tipo_base* dell'array.

Esempio:

```
(* Il seguente programma completo usa un array di interi
   per memorizzare un elenco di numeri, di cui viene poi
   calcolato il minimo
*)
program MinimoConArray;
const MaxIndice = 100;
type  ElencoInteri = array [ 1 .. MaxIndice ] of integer;
var   elenco : ElencoInteri;
      n, i, min: integer;
begin
  (* Legge l'elenco da tastiera *)
  write('Quanti numeri vuoi inserire? ');
  readln(n);
  for i := 1 to n do
    begin
      write('Inserisci il numero di posizione ', i, ': ');
      readln( elenco[i] );
    end;

  (* Ora calcola il minimo *)
  min := elenco[1];
  for i := 2 to n do
```

```
    if elenco[i] < min then
      min := elenco[i];

  (* Stampa il minimo *)
  writeln('Il minimo dell''elenco è: ', min);
end.
```

Array multidimensionali

Un array può avere come tipo base un qualsiasi altro tipo, compresi altri tipi array definiti dall'utente. Array i cui elementi sono array possono essere usati ad esempio per rappresentare matrici, tabelle multidimensionali e in generale altre informazioni in cui i singoli elementi dell'informazione sono individuati attraverso una coppia (o in generale una tupla) di indici. Per questo genere di informazioni il Pascal consente una notazione più sintetica per la dichiarazione e l'uso, attraverso il concetto di *array multidimensionale*. Un array multidimensionale a *N* dimensioni è una collezione di elementi dello stesso tipo, ciascuno dei quali è individuato specificando *N* indici che appartengono a *N* intervalli di numeri interi. La definizione di un array multidimensionale usa la seguente sintassi:

```
type identificatore = array [indice1_minimo .. indice1_massimo,
                             .....
                             indiceN_minimo .. indiceN_massimo]
                      of tipo_base;
```

Gli elementi di un array multidimensionale possono essere usati specificando tra parentesi quadre gli indici separati da virgole:

```
variabile [indice1, ....., indiceN]
```

Esempio:

```
(* Il seguente frammento dichiara e usa una matrice 3x4 *)
type Matrice = array[1..3, 1..4] of real;
var  M : Matrice;
begin
  M[1, 1] := 1.0;
  .....
```

Il tipo *stringa di caratteri*

Per gestire sequenze di caratteri si usa il tipo *string* (una sequenza di caratteri viene detta *stringa* di caratteri), simile al tipo array. La dichiarazione di un tipo stringa usa la seguente sintassi:

```
type identificatore = string [ lunghezza_massima ];
```

dove *lunghezza_massima* rappresenta il massimo numero di caratteri della sequenza. La specifica della lunghezza massima può essere omessa insieme alle relative parentesi quadre; in questo caso si assume come lunghezza massima 255, che è il più alto valore specificabile. Con variabili di tipo stringa sono definite le seguenti operazioni:

- assegnazione, attraverso il valore di un'altra variabile o di una costante letterale di tipo stringa
- accesso a un singolo carattere, con una notazione simile a quella usata dagli array: il primo carattere ha indice 1, il secondo ha indice 2 e così via; quindi se x è una variabile

di tipo stringa, x**[**1**]** rappresenta il primo carattere della stringa contenuta nella variabile

- calcolo della lunghezza *effettiva* della stringa (in contrapposizione alla lunghezza massima) attraverso la funzione predefinita **length**
- concatenazione di stringhe, attraverso l'operatore +
- confronto tra stringhe; nel caso di operatori di disuguaglianza (es. <) il confronto viene effettuato attraverso il concetto di ordinamento *lessicografico*

Esempio:

```
(* Il seguente programma completo conta il numero di vocali
   in una frase. Si noti che le lettere maiuscole sono
   considerate diverse dalle lettere minuscole in una stringa,
   per cui il programma deve specificare entrambi i casi.
   Inoltre non sono gestite correttamente le vocali
   accentate.
*)
program ContaVocali;
type Frase = string[150];
var  f : Frase;
     i : integer ;
     v : integer ;
begin
  write('Inserisci la frase: ');
  readln(f);
  v := 0;
  for i:=1 to length( f ) do
    case f[i] of
      'a', 'e', 'i', 'o', 'u',
      'A', 'E', 'I', 'O', 'U':  v:=v+1;
    end;
  writeln('Il numero di vocali è: ', v);
end.
```

Tipi *record*

Un *record* rappresenta un aggregato di informazioni (dette *campi* del record), eventualmente di tipo diverso, individuate attraverso un identificatore. La sintassi della dichiarazione di un tipo record è:

```
type identificatore = record
                        identificatore1 : tipo1 ;
                        .....
                        identificatoreN : tipoN ;
                      end;
```

dove *identificatore1 ..... identificatoreN* sono i nomi assegnati ai campi del record, e *tipo1 ..... tipoN* sono i tipi corrispondenti, che possono essere sia tipi predefiniti che tipi definiti dall'utente in una precedente dichiarazione.
In una variabile di tipo record si può accedere ai singoli campi usando la notazione:

```
variabile . identificatore_campo
```

Esempio:

```
type Persona = record
                 nome : string[30];
```

```
                cognome : string[30];
                eta : integer;
              end;
var p: Persona;
begin
  p.nome := 'Mario';
  p.cognome := 'Rossi';
  p.eta := 42;
  ......
```

Definizione di sottoprogrammi

Un *sottoprogramma* rappresenta una parte del programma che realizza una specifica operazione. Una volta che l'operazione è definita mediante un sottoprogramma, essa può essere utilizzata come se fosse una delle operazioni elementari predefinite dal linguaggio. I sottoprogrammi si dividono in due categorie:

- sottoprogrammi di tipo *funzione*, che producono un risultato che può essere utilizzato all'interno di un'espressione del linguaggio. Ad esempio, i sottoprogrammi che calcolano il valore di funzioni matematiche saranno di questo tipo
- sottoprogrammi di tipo *procedura*, che non calcolano un risultato utilizzabile direttamente in un'espressione. Ad esempio, un sottoprogramma il cui scopo è stampare qualcosa sullo schermo sarà di tipo procedura

Un sottoprogramma può avere dei *parametri formali* che rappresentano informazioni che devono essere comunicate ad esso al momento in cui viene richiesto lo svolgimento della sua operazione; i parametri formali sono dei nomi usati all'interno della definizione del sottoprogramma al posto dei valori veri e propri che dovranno essere comunicati al momento dell'esecuzione; nelle istruzioni che richiamano il sottoprogramma saranno poi di volta in volta specificati i *parametri attuali*, ovvero i valori veri e propri su cui il sottoprogramma dovrà lavorare.

Dichiarazione di un sottoprogramma

La dichiarazione di un sottoprogramma va inserita nella parte dichiarativa del programma. La sintassi per dichiarare un sottoprogramma è, per il tipo *funzione* è:

```
function nome_funzione ( identificatore1:tipo1; .....;
identificatoreN:tipoN ): tipo_risultato;
parte_dichiarativa
parte_esecutiva
```

mentre per un sottoprogramma di tipo procedura la sintassi è:

```
procedure nome_procedura ( identificatore1:tipo1; .....;
identificatoreN:tipoN );
parte_dichiarativa
parte_esecutiva
```

dove *nome_funzione* e *nome_procedura* sono gli identificatori assegnati ai sottoprogrammi; *identificatore1 ..... identificatoreN* sono i nomi dei parametri formali e *tipo1 ..... tipoN* sono i tipi corrispondenti; *tipo_risultato* è il tipo del valore calcolato dalla funzione. La *parte_dichiarativa* (opzionale) contiene le stesse dichiarazioni che possono comparire nella parte dichiarativa del

programma, ma in questo caso le dichiarazioni hanno effetto solo sul singolo sottoprogramma. La *parte_esecutiva* contiene le istruzioni che realizzano l'operazione assegnata al sottoprogramma; è costituita da una sequenza di istruzioni racchiuse tra **begin** e **end**, ma a differenza del programma principale, l'**end** finale è seguito da un punto e virgola. Nella parte esecutiva di una funzione è possibile specificare qual'è il valore finale prodotto dalla funzione con un'istruzione di assegnazione in cui al posto della variabile si usa il nome della funzione.

Esempio:

```
(* Questa funzione calcola il quadrato
   di un numero
*)
function Quadrato(x:real):real;
begin
  Quadrato := x*x;
end;

(* Questa procedura stampa a video una
   tabellina
*)
procedure StampaTabellina(n:integer);
var i:integer;
begin
  for i:=1 to 10 do
    writeln(n, ' per ', i, ' = ', n*i);
end;
```

Invocazione di un sottoprogramma

Un sottoprogramma viene richiamato specificando il nome e, tra parentesi tonde, i parametri attuali separati da virgole. La corrispondenza tra i parametri attuali e i parametri formali è realizzata in base alla posizione: il primo parametro attuale specificato nella lista è assegnato al primo parametro formale e così via. Per i sottoprogrammi di tipo funzione l'invocazione deve essere inserita all'interno di un'espressione (ad esempio in un'assegnazione), mentre per le procedure deve essere usata come una istruzione a sé, eventualmente seguita dal punto e virgola. I parametri attuali possono essere specificati attraverso espressioni comunque complesse.

Esempio:

```
(* In questo frammento c'e' la parte esecutiva di un
   programma che contiene le dichiarazioni di
   sottoprogramma viste nell'esempio precedente
*)
begin
  (* Calcola e stampa il quadrato di 9 *)
  writeln('Il quadrato di 9 è: ', Quadrato(9));
  (* Stampa la tabellina del 7 *)
  StampaTabellina(7);
end.
```

Dichiarazioni globali e dichiarazioni locali

Le dichiarazioni di tipi, costanti simboliche e variabili presenti nella parte dichiarativa del programma principale sono *globali*, ovvero gli oggetti dichiarati sono visibili e utilizzabili in tutto il programma, compresi i suoi sottoprogrammi. Invece le dichiarazioni presenti nella parte dichiarativa di un sottoprogramma sono *locali* al sottoprogramma stesso, ovvero hanno effetto solo sulla sua parte esecutiva. Nel caso in cui un identificatore sia dichiarato sia nella parte dichiarativa globale che in quella locale a un sottoprogramma, il sottoprogramma vedrà solo la dichiarazione locale, che in questo caso nasconde la dichiarazione globale.