Lezione 12
9–10 novembre 1999

Argomenti trattati
- Tipi strutturati: gli array.
- Array di array.
- Array a piú dimensioni.

12.1 Gli array

Un *array* (o *vettore*) è un insieme ordinato di variabili dello *stesso tipo* ciascuna delle quali è accessibile specificando la posizione in cui si trova.

Un array viene definito dai seguenti elementi:

- un tipo scalare, che definisce l'insieme delle *posizioni* o *indici* dell'array;
- un tipo qualsiasi, il tipo delle *componenti* dell'array, cioè degli elementi che lo costituiscono.

Si supponga ad esempio di volere costruire delle tabelle per memorizzare le temperature che si sono riscontrate durante una settimana. A tale scopo si potrebbero definire i seguenti tipi:

```
TYPE
   giorni = (lun, mar, mer, gio, ven, sab, dom);
   tabtemperature = ARRAY [giorni] OF integer;
```

Il tipo strutturato `tabtemperature` è una collezione di variabili, o componenti, di tipo `integer`, ognuna delle quali è individuabile mediante un indice, di tipo `giorni`.

A questo punto è possibile dichiarare variabili con questa struttura. Se ad esempio vogliamo costruire due tabelle, una per le temperature minime, l'altra per le massime, possiamo scrivere:

```
VAR
   tmin, tmax: tabtemperature;
```

Gli array `tmin` e `tmax` sono collezioni di 7 variabili di tipo `integer`, ognuna delle quali può essere individuata scrivendo, dopo il nome dell'array, un opportuno *selettore*, che specifichi l'indice della componente, cioè della variabile desiderata. Ad esempio, la componente di `tmin` corrispondente all'indice `gio`, è `tmin[gio]`, e può essere utilizzata come una qualunque altra variabile di tipo `integer`.

Si supponga di avere dichiarato anche una variabile `giorno` di tipo `giorni` e una variabile `x` di tipo `integer`, e si consideri la seguente sequenza d'istruzioni:

```
x := 5;
giorno := mar;
tmin[sab] := 3;
tmax[dom] := x + 1;
x := 2 * tmin[sab];
tmin[giorno] := x
```

La terza istruzione assegna il valore 3 alla componente di posto `sab` dell'array `tmin`. La quarta istruzione assegna il risultato dell'espressione `x + 1` alla componente di posto `dom` dell'array `tmax`. Nella quinta istruzione, alla variabile `x` viene assegnato il doppio del valore della componente di posto `sab` di `tmin`. Nella sesta istruzione, all'interno del selettore si utilizza la variabile `giorno`; pertanto l'istruzione assegna il valore di `x` alla componente di `tmin` il cui indice è il contenuto della variabile `giorno`, che in questo caso è `mar`.

Utilizzando variabili (o piú in generale espressioni) all'interno dei selettori, è possibile far operare una stessa istruzione su componenti differenti del vettore. Ad esempio, possiamo azzerare ogni componente del vettore `tmin`, utilizzando il seguente ciclo:

```
FOR giorno := lun TO dom DO
   tmin[giorno] := 0
```

È inoltre possibile copiare completamente un array in un altro dello stesso tipo, utilizzando un unico assegnamento. Ad esempio, l'istruzione

```
tmax := tmin
```

copia il contenuto dell'intero array `tmin` nell'array `tmax`, ed è quindi equivalente al ciclo:

```
FOR giorno := lun TO dom DO
   tmax[giorno] := tmin[giorno]
```

Esempio: calcolo del numero di occorrenze di ciascuna lettera in un testo

Nella lezione precedente abbiamo costruito un programma che contava il numero di occorrenze di ogni vocale in un testo dato in input. Vogliamo ora estendere il programma in modo che sia in grado di calcolare il numero di occorrenze di ogni lettera.

Un approccio immediato consiste nell'estendere l'insieme di variabili, introducendo, accanto ai contatori `na, ne, ni, no, nu` per le vocali, un contatore per ogni consonante. È necessario poi modificare le procedure di inizializzazione, di scrittura, e i controlli all'interno della procedura di conteggio, inserendo istruzioni corrispondenti alle nuove variabili introdotte. All'interno della procedura di conteggio avremo come costrutto `CASE` un lungo elenco di istruzioni di incremento, che differiscono solo nella scelta del contatore da incrementare, che dipende dalla lettera considerata:

```
IF c IN ['a'..'z','A'..'Z'] THEN
   CASE c OF
      'a', 'A':
         na := na + 1;
      'b', 'B':
         nb := nb + 1;
      'c', 'C':
         nc := nc + 1;
      ...
   END {case c}
```

Osserviamo che le variabili `na, nb, nc, ...` sono tutte dello stesso tipo (`integer`) e hanno una stessa funzionalità (quella di contatore), riferita ad oggetti (lettere) differenti. Possiamo quindi pensare di organizzarle in una struttura a tabella in cui ogni posizione si riferisce a una lettera dell'alfabeto, e contiene il numero di occorrenze di tale lettera nel testo. In altre parole, possiamo far uso di un array. Le componenti dell'array sono i contatori, e quindi sono di tipo `integer`. Ogni contatore è associato a una lettera, pertanto gli indici possibili saranno le lettere dell'alfabeto. Definiamo dunque i tipi:

```
TYPE
   lettere = 'a'..'z';
   tablettere = ARRAY[lettere] OF integer;
```

e dichiariamo come variabile del programma principale:

```
VAR
   nlettere: tablettere;
```

Le procedure `inizializza` e `scrivi` possono essere riscritte utilizzando dei semplici cicli, con variabile di controllo di tipo `lettere`. Ad esempio, la procedura `inizializza` è costituita da un ciclo che azzera le componenti dell'array:

```
PROCEDURE inizializza;
   VAR
      lettera: lettere;

BEGIN {inizializza}
   FOR lettera := 'a' TO 'z' DO
      nlettere[lettera] := 0
END; {inizializza}
```

Per quanto riguarda la parte di incremento dei contatori, occorre verificare se il carattere letto (contenuto nella variabile `c`) sia una lettera, e in tal caso incrementarne il contatore. Ciò può essere fatto utilizzando semplicemente la variabile `c` per selezionare la componente dell'array che deve essere incrementata. In altre parole, è sufficiente scrivere

```
nlettere[c] := nlettere[c] + 1
```

Poiché il testo in ingresso potrebbe contenere sia lettere maiuscole che minuscole, che non vanno distinte nel conteggio, dopo avere letto il carattere, si opera una conversione delle maiuscole in minuscole, come segue:

```
IF c IN ['A'..'Z'] THEN
   c := chr(ord(c) - ord('A') + ord('a'))
```

Pertanto, la parte di codice relativa al controllo del carattere letto e all'incremento del contatori può essere riscritta come

```
{se il carattere e' una lettera maiuscola, allora trasformala in minuscola}
IF c IN ['A'..'Z'] THEN
   c := chr(ord(c) - ord('A') + ord('a'));

{se il carattere e' una lettera, allora incrementane il contatore}
IF c IN ['a'..'z'] THEN
   nlettere[c] := nlettere[c] + 1
```

Ecco il listato completo del programma:

```
PROGRAM FrequenzeLettere (input, output);

{conta la frequenza di ciascuna lettera nel file di input}

   TYPE
      lettere = 'a'..'z';
      tablettere = ARRAY[lettere] OF integer;

   VAR
      nlettere: tablettere;

   PROCEDURE inizializza;
      VAR
         lettera: lettere;

   BEGIN {inizializza}
      FOR lettera := 'a' TO 'z' DO
         nlettere[lettera] := 0
   END; {inizializza}


   PROCEDURE LeggiEConta;
      VAR
         c: char;

   BEGIN {LeggiEConta}
      WHILE NOT eof DO
         {esamina una riga}
         BEGIN
            WHILE NOT eoln DO
                   {esamina un carattere}
               BEGIN
                  read(c);

                  {se il carattere e' una lettera maiuscola, allora trasformala in minuscola}
                  IF c IN ['A'..'Z'] THEN
                     c := chr(ord(c) - ord('A') + ord('a'));

                  {se il carattere e' una lettera, allora incrementane il contatore}
                  IF c IN ['a'..'z'] THEN
                     nlettere[c] := nlettere[c] + 1

               END; {while not eoln ...}

            {passa alla riga successiva}
            readln
         END {while not eof ...}
   END; {LeggiEConta}


   PROCEDURE scrivi;
      VAR
```

```
        l: lettere;

   BEGIN {scrivi}
      FOR l := 'a' TO 'z' DO
         writeln('Il testo contiene ', nlettere[l] : 1, ' occorrenze della lettera ', l)
   END; {scrivi}


BEGIN {FrequenzeLettere}
   inizializza;
   LeggiEConta;
   scrivi
END. {FrequenzeLettere}
```

12.2 Array di array

Abbiamo visto che mentre il tipo dell'indice di un array deve essere un tipo scalare, il tipo delle componenti può essere un tipo qualsiasi. Dunque il tipo delle componenti può essere a sua volta un array (purché definito prima).

Si considerino ad esempio le seguenti definizioni:

```
TYPE
   mesi = (gen, feb, mar, apr, mag, giu, lug, ago, sep, ott, nov, dic);
   tabmesi = ARRAY [1..31] OF integer;
   anno = ARRAY [mesi] OF tabmesi;
```

La prima definizione introduce il tipo enumerativo `mesi`, con le dodici costanti elencate. La seconda definizione introduce il tipo `tabmesi`, un array di 31 variabili `integer`, individuabili utilizzando i valori del subrange `1..31` del tipo `integer`. Infine, la terza definizione introduce il tipo `anno`, un array di 12 componenti, individuabili mediante valori di tipo `mesi`; ogni componente di questo array è di tipo `tabmesi`, cioè a sua volta un array.

Supponiamo di dichiarare le seguenti variabili:

```
VAR
   x, y: anno;
   z: tabmesi;
   m: mesi;
   g: 1..31;
```

Possiamo selezionare le singole componenti dell'array `x` di tipo `anno`, specificando nel selettore un valore di tipo `mesi`. Ad esempio `x[gen]` è la componente di posto `gen` dell'array `x`. Tale componente è di tipo `tabmesi`. Pertanto possiamo selezionare al suo interno la componente di posto `3`, che sarà di tipo `integer`, aggiungendo l'opportuno selettore, cioè scrivendo `x[gen][3]`. Riepilogando:

- `x` è di tipo `anno`, cioè `ARRAY [mesi] OF tabmesi`;
- `x[m]` è la componente di posto `m` di `x`; dunque il tipo di `x[m]` è `tabmesi`, cioè `ARRAY [1..31] OF integer`;
- `x[m][g]` è la componente di posto `g` di `x[m]`; dunque il tipo di `x[m][g]` è `integer`;

Consideriamo alcuni esempi di assegnamenti:

- `x[gen][3] := 10` è un'istruzione di assegnamento di un valore di tipo `integer` che assegna il valore `10` al terzo giorno del mese `gen` dell'anno `x`;

- `y[mar] := x[apr]` è un'istruzione di assegnamento di un valore (strutturato) di tipo `tabmesi` che copia l'intero contenuto dell'array `x[apr]` nell'array `y[mar]`, cioè i dati relativi al mese `apr` dell'anno `x` nel mese `mar` dell'anno `y`;
- `x[dic] := z` è ancora un'istruzione di assegnamento di un valore (strutturato) di tipo `tabmesi` che copia l'intero contenuto dell'array `z` nell'array `x[apr]`;
- `y := x` è un'istruzione di assegnamento di un valore (strutturato) di tipo `anno` che copia l'intero contenuto dell'arrary `x` nell'array `y`.

12.3 Array a piú dimensioni

Negli esempi presentati sinora abbiamo utilizzato esclusivamente array monodimensionali, in cui cioè ogni elemento è individuato da un unico indice. È possibile definire anche array a piú dimensioni, per rappresentare tabelle o matrici, specificando un tipo scalare per ogni dimensione dell'array. Ad esempio, le matrici di interi, formate da 100 righe e 100 colonne, possono essere definite mediante il tipo:

```
TYPE
   matrice = ARRAY [1..100,1..100] OF integer;
```

La selezione di un elemento all'interno di una variabile `x` di tipo `matrice` avverrà specificando una coppia di indici, scrivendo ad esempio `x[5,8]`. Si osservi che il tipo `matrice` è differente dal tipo `matrice2` seguente:

```
TYPE
   vet = ARRAY [1..100] OF integer;
   matrice2 = ARRAY [1..100] OF vet;
```

Sia le variabili di tipo `matrice` che quelle di tipo `matrice2` permettono di memorizzare 10000 valori di tipo `integer`. Tuttavia, le variabili di tipo `matrice` sono array con componenti di tipo `integer` selezionabili con una coppia di valori, entrambi nel subrange `1..100`, mentre le variabili di tipo `matrice2` sono array con componenti di tipo `vet`, ciascuna delle quali selezionabile con un indice nel subrange `1..100`; all'interno di ciascuna componente è possibile poi selezionare, mediante un altro indice, un elemento di tipo `integer`. Pertano, se `x` è di tipo `matrice` e `y` di tipo `matrice2`, un valore `integer` memorizzato in `x` è individuabile mediante un selettore a due indici, come `x[i,j]`, mentre un valore `integer` memorizzato all'interno di una componente di `y` è individuabile mediante l'uso di un doppio selettore, come `y[i][j]`.

Si ricorda che è possibile utilizzare anche piú di due indici e che gli indici possono essere di tipi scalari differenti tra loro.

Utilizzando array bidimensionali, anziché array di array, possiamo riscrivere la precedente definizione del tipo `anno`, senza introdurre il tipo `tabmesi` come:

```
TYPE
   mesi = (gen, feb, mar, apr, mag, giu, lug, ago, sep, ott, nov, dic);
   anno = ARRAY [mesi,1..31] OF integer;
```

Consideriamo inoltre le seguenti dichiarazioni di variabili

```
VAR
   x, y: anno;
   m: mesi;
   g: 1..31;
```

Rivediamo, con queste nuove dichiarazioni, alcuni degli esempi di assegnamento presentati in precedenza:

- per assegnare il valore `10` in corrispondenza del terzo giorno del mese `gen` dell'anno `x`, scriveremo `x[gen,3] := 10`;
- per copiare il valore dell'intero array `x` nell'array `y` è sufficiente scrivere `y := x`;
- per copiare i dati relativi al mese `apr` dell'anno `x` nel mese `mar` dell'anno `y` è necessario utilizzare il ciclo

```
FOR g := 1 TO 31 DO
   y[mar,g] := x[apr,g]
```

12.4 Alcune considerazioni finali sugli array

È importante ricordare che in Pascal la dimensione degli array è *fissata* nel programma. Questo permette al compilatore di conoscere la quantità esatta di memoria da riservare durante l'esecuzione.

Molti linguaggi di programmazione permettono di utilizzare come indici di array solo valori interi. Il Pascal permette invece di utilizzare per gli indici tipi scalari qualsiasi. È bene sfruttare questa opportunità che in molte situazioni si rivela utile a semplificare la stesura dei programmi e a renderli piú leggibili (e quindi piú facilmente modificabili). A titolo d'esempio, si pensi a quante operazioni di cambio di rappresentazione (da `char` a `integer` e viceversa) sarebbero state necessarie se si fosse usato un subrange del tipo `integer`, come tipo degli indici nel programma per il calcolo della frequenza delle lettere in un testo.

Si ricordi che affinché i programmi siano scritti bene, non è sufficiente che "girino". Un programma scritto bene dovrà prima di tutto risultare facilmente comprensibile sia all'autore, che agli altri. Questa caratteristica è di fondamentale importanza per ottenere software che sia facilmente riutilizzabile ed espandibile e quindi per ridurre, complessivamente, i costi di produzione dei programmi.

Per rendere i programmi piú facilmente modificabili è indispensabile l'utilizzo di costanti. Si supponga ad esempio di utilizzare in un programma un tipo array, con indice subrange di `integer`, come:

```
TYPE
   tab = ARRAY [1..100] OF ...;
VAR
   t: tab;
```

Sicuramente nel programma si troveranno delle istruzioni in cui compaiono le limitazioni che sono state date sugli indici degli array; molto spesso, ad esempio, si avranno dei cicli `FOR`:

```
FOR i := 1 TO 100 DO
   operazioni che coinvolgono t[i]
```

Volendo cambiare la lunghezza dell'array, occorrerà sostituire ovunque nel programma il numero 100 con la nuova lunghezza. È dunque opportuno fissare la lunghezza in una costante, definendo:

```
CONST
   lung = 100;
TYPE
   tab = ARRAY [1..lung] OF ...;
VAR
   t: tab;
```

In questo caso il ciclo verrà scritto come:

```
FOR i := 1 TO lung DO
   operazioni che coinvolgono t[i]
```

Per modificare la lunghezza dell'array (e i relativi cicli), sarà sufficiente cambiare la costante all'inizio del programma. In alcune situazioni potrebbe risultare opportuno definire anche l'indice inferiore dell'array mediante una costante.

Infine, quando non si sappia a priori quante componenti dell'array verranno effettivamente utilizzate, si può sovradimensionare l'array, ed utilizzare all'interno del programma una variabile che indichi il valore dell'indice massimo effettivamente utilizzato.

Esercizi

1. Modificare il programma che stampa il numero di occorrenze di ciascuna lettera in un testo in modo che:
   - non stampi nessun messaggio per le lettere che appaiono zero volte nel testo;
   - nei messaggi relativi alle lettere che appaiono nel testo una sola volta, stampi la parola *occorrenza* al singolare anziché al plurale;
   - dopo avere stampato il numero di occorrenze di ciascuna lettera, indichi la vocale e la consonante piú frequenti, il numero totale di vocali e il numero totale di consonanti presenti nel testo.
2. Per ognuno dei seguenti frammenti di codice individuare delle dichiarazioni di variabile ed eventualmente di tipo in modo che le istruzioni che vi appaiono risultino corrette. Se ciò non è possibile spiegare il motivo.
   - ```
     FOR i := 'a' TO 'z' DO
         v[i]:= ord(i);
     ```
   - ```
     FOR i := 'a' TO 'z' DO
         v[i]:= chr(i);
     ```
   - ```
     FOR i := 'a' TO 'z' DO
         v[i]:= chr('i');
     ```
   - ```
     a := c > d;
     b := c MOD d;
     x[a] := chr(c-d);
     ```
   - ```
     a := c > d;
     b := c/d;
     x[a] := chr(c-d);
     ```
   - ```
     a := c > d;
     c := c/d;
     x[a] := chr(c-d);
     ```
   - ```
     z := 'a';
     x := y[z];
     writeln(x[1]);
     ```
   - ```
     q[q[i]] := i;
     ```
   - ```
     v[i]:= ord(true <> (a > v[i]));
     ```
   - ```
     v[a > b]:= a-b;
     v[b < a]:= b-a;
     ```
   - ```
     v[v[v[v[2]]]]:= a;
     ```
     ```

- `v[v[v[v[2]]]]:= 'a';`

3. Costruire un programma che legga un file di testo e scriva in output una tabella che indichi il numero di occorrenze di ciascuna coppia di lettere (si può indicare sulle righe la prima lettera e sulle colonne la seconda).

4. Per ognuno dei seguenti frammenti di codice individuare delle dichiarazioni di variabile ed eventualmente di tipo in modo che le istruzioni che vi appaiono risultino corrette. Se ciò non è possibile spiegare il motivo.

   - ```
     FOR i := 'a' TO 'z' DO
         v[i,ord(i)]:= chr(ord(i));
     ```
   - ```
     FOR i := 'a' TO 'z' DO
         v[i][ord(i)]:= chr(ord(i));
     ```
   - ```
     v[i]:= x;
     x[j]:= 'a';
     k:= i+j;
     v[k][k]:= x[k];
     ```

5. Per ognuno dei seguenti frammenti di codice individuare delle dichiarazioni di variabile ed eventualmente di tipo in modo che le istruzioni che vi appaiono risultino corrette dal punto di vista della compatibilità dei tipi. Se ciò non fosse possibile spiegare il motivo.

   - `i:= (x[y > z] + x[i]) > z`
   - `j:= (k + h) DIV a[chr(k)]`
   - `l:= (s + ord(t)) / a[l]`
   - ```
     q[i]:= s;
     s[ord(i)]:= i;
     r:= q;
     ```
   - ```
     p:= 'a';
     a:= ord(p) = ord('p');
     v[p]:= x;
     x[a]:= 1;
     ```
   - ```
     y := succ(100);
     x := chr(y);
     z[x,y] := x;
     ```
   - ```
     x := y;
     x[y[j]] := y[x[i]] + 1;
     ```

6. Dopo avere scritto le dichiarazioni opportune, riscrivere ognuno dei seguenti frammenti di programma, sostituendo i cicli `FOR` prima con cicli `WHILE` e poi con cicli `REPEAT`. Indicare inoltre quante volte verrà eseguita l'istruzione `writeln` interna ai cicli.

   - ```
     FOR k:= 'z' DOWNTO 'a' DO
         FOR j:= 'a' TO k DO
            writeln(x[k][j]:2);
     ```
   - ```
     FOR w:= 1 TO 100 DO
         FOR y:= false TO true DO
            writeln(s[w,y]*2);
     ```

     ```
     FOR w:= inizio TO fine DO
         FOR p:= q TO m DO
            writeln(s[p,w]/5);
     ```
     ```