Lezione 17
29 novembre 1999

Argomenti trattati
- Tipi strutturati: i `FILE`.
- L'istruzione `GOTO`.

17.1 I file

In Pascal un file è una successione di elementi dello *stesso* tipo, il cui numero non è stabilito a priori. I file disponibili in Pascal sono *file sequenziali*: non è possibile accedere direttamente ad un elemento qualsiasi, ma è necessario attraversare il file partendo dal primo elemento.

Poiché i file possono contenere una grande quantità di dati, essi vengono in genere memorizzati su memoria di massa. È possibile inoltre dichiarare file locali al programma o a una procedura, e file permenenti; questi ultimi restano memorizzati sui dispositivi di massa anche quando l'esecuzione del programma è terminata, e possono essere utilizzati successivamente dallo stesso o da altri programmi.

Un tipo *file di interi*, e una variabile `f` di tale tipo, possono essere dichiarati come:

```
TYPE
   fint = FILE OF integer;
VAR
   f: fint;
```

In questo modo, viene implicitamente introdotta una *variabile buffer*, denotata da `f^`, di tipo `integer` (cioè del tipo delle componenti del file), tramite la quale si accede di volta in volta (in lettura o scrittura) a uno degli elementi del file. Viene anche creato implicitamente un puntatore al file, che indica di volta in volta la componente accessibile. Come detto, la lettura o la scrittura avvengono sequenzialmente, a partire dal primo elemento.

Come abbiamo già visto per i file di testo, un file per essere utilizzato deve essere aperto in *lettura* o in *scrittura*. Le operazioni possibili dipendono dalla modalità di apertura utilizzata.

Lettura di un file

Le procedure e funzioni di base per la lettura di un file `f` sono:

- `reset(f)`
  Apre il file `f` in lettura, ponendo il puntatore sul primo elemento. Se `f` non è vuoto assegna a `f^` il valore del primo elemento di `f`, altrimenti pone l'indicatore di *end–of–file* di `f` a `true`.

- `get(f)`
  Sposta in avanti di una posizione il puntatore al file. Se non è stata raggiunta la fine del file, assegna al buffer `f^` il valore della componente indicata dal puntatore, altrimenti pone a `true` l'indicatore di *end–of–file*.

- `eof(f)`
  Restituisce il valore dell'indicatore di *end–of–file* del file `f`, cioè restituisce `true` nel caso sia stata raggiunta la fine del file `f`.

A titolo d'esempio, riportiamo un breve programma che calcola e visualizza la somma dei valori contenuti in un file `f` di interi presente su memoria di massa.

```
PROGRAM sommafile (output, f);

{calcola la somma dgli interi contenuti in un file}

   VAR
      f: FILE OF integer;
      somma: integer;

BEGIN
   reset(f);
   somma := 0;
   WHILE NOT eof(f) DO
      BEGIN
         somma := somma + f^;
         get(f)
      END; {while}

   writeln('La somma dei valori contenuti nel file e'' ', somma)
END.
```

La lettura di un elemento dal file può avvenire anche utilizzando la procedura standard `read`. In particolare, se `f` è dichiarato come `FILE OF tipo` e `x` è una variabile di tipo `tipo`, l'esecuzione di

```
read(f, x)
```

è equivalente all'esecuzione della sequenza di istruzioni

```
x := f^;
get(f)
```

Utilizzando `read`, il precedente programma può essere riscritto come:

```
PROGRAM sommafile (output, f);

{calcola la somma dgli interi contenuti in un file}

   VAR
      f: FILE OF integer;
      x, somma: integer;
```

```
BEGIN
   reset(f);
   somma := 0;
   WHILE NOT eof(f) DO
      BEGIN
         read(f, x);
         somma := somma + x
      END; {while}

   writeln('La somma dei valori contenuti nel file e'' ', somma)
END.
```

Scrittura di un file

Le procedure per la scrittura di un file sono

- `rewrite(f)`
  Apre il file `f` in scrittura, *cancellando* gli eventuali dati in esso contenuti. Pone il puntatore sulla prima posizione del file (dove avverrà la prima operazione di scrittura).

- `put(f)`
  Appende alla fine del file `f`, il valore contenuto nella variabile buffer `f^`. Il valore di `f^` diventa indefinito. Il puntatore al file viene fatto avanzare di una posizione.

Come esempio presentiamo un programma che legge da `input` una sequenza di interi, terminata da 0, e la scrive in un file:

```
PROGRAM scrivifile (input, f);

{legge da input una sequenza di numeri interi, che termina con 0, e la scrive in un file}

   VAR
      f: FILE OF integer;
      x: integer;


BEGIN
   {apertura del file in scrittura}
   rewrite(f);

   write('Inserire un numero intero (0 per terminare) ');
   readln(x);
   WHILE x <> 0 DO
      BEGIN
         f^ := x;
         put(f);

         write('Inserire un numero intero (0 per terminare) ');
         readln(x)
      END {while}
END.
```

Per la scrittura è possibile utilizzare anche la procedura standard `write`. In particolare, l'esecuzione dell'istruzione

```
write(f, x)
```

è equivalente all'esecuzione delle due istruzioni

```
f^ := x;
put(f)
```

File permanenti e file temporanei

Nel Pascal standard, i file esterni, cioè esistenti anche prima e/o dopo l'esecuzione del programma, vanno passati come parametri al programma, indicandoli nell'intestazione. Tutti gli altri file sono temporanei (locali al programma o al sottoprogramma in cui sono dichiarati).

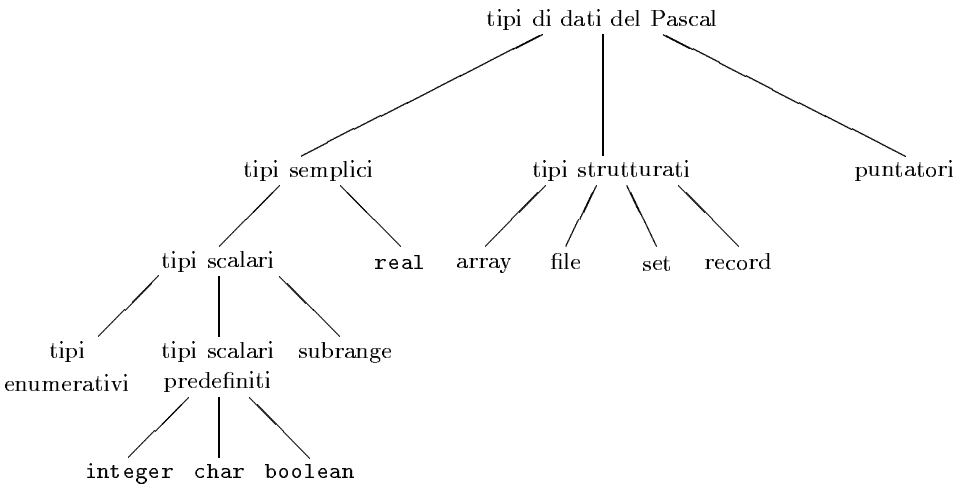
Per copiare un file `f` in un file `g` dello stesso tipo *non* è possibile utilizzare l'assegnamento `g := f`, ma è necessario copiare un elemento per volta. Inoltre i file *non* possono essere passati come parametri per valore ai sottoprogrammi.

L'associazione tra il nome esterno del file e il nome interno al programma dipende dall'ambiente utilizzato. Si veda a questo proposito quanto presentato nel caso del tipo `text`.

Spesso le componenti dei file sono `RECORD` costituiti da *piú campi*. Pertanto, la variabile buffer `f^` di un tale file sarà un record, di cui sarà possibile selezionare un campo specificando, come sempre, il relativo nome. Inoltre con le istruzioni `read` e `write` è possibile leggere e scrivere sul file un intero record.

17.2 Organizzazione dei tipi del Pascal

La seguente figura riepiloga l'organizzazione dei tipi del linguaggio Pascal:



17.3 L'istruzione `GOTO`

L'istruzione `GOTO` indica che l'elaborazione prosegue in un altro punto del programma. Dopo la parola chiave `GOTO` deve essere indicata una *etichetta* o *label*, costituita da un numero intero. L'etichetta deve essere dichiarata nella sezione `LABEL` del programma principale o di un sottoprogramma (le dichiarazioni di `LABEL` precedono tutte le altre dichiarazioni). Inoltre, il programma deve contenere un'istruzione preceduta dall'etichetta utilizzata.

Consideriamo ad esempio il seguente programma, che stampa i numeri da 0 a 10:

```
PROGRAM p (output);

   LABEL
      99, 555;

   CONST
      max = 10;

   VAR
      i: integer;

BEGIN
   writeln('Inizio programma');
   i := 0;
555:
   IF i > max THEN
      GOTO 99;
   writeln(i);
   i := i + 1;
   GOTO 555;
99:
   writeln('Fine programma')
END.
```

Quando si incontra l'istruzione `GOTO 555`, l'esecuzione prosegue dall'istruzione preceduta dall'etichetta `555`, cioè dall'istruzione `IF`. Analogamente, l'istruzione `GOTO 99` provoca un salto all'istruzione `writeln` finale.

Non presentiamo altri dettagli relativi all'istruzione `GOTO` del Pascal. È stato dimostrato che ogni programma contenente istruzioni `GOTO` può essere riscritto utilizzando esclusivamente le tre strutture di controllo fondamentali (sequenza, selezione e iterazione). Poiché il Pascal è un linguaggio strutturato, l'istruzione `GOTO` risulta inutile: per ogni programma Pascal contenente l'istruzione `GOTO` esiste un programma Pascal equivalente che non la contiene. Inoltre l'uso dell'istruzione `GOTO` rende poco comprensibile la struttura dei programmi. Per questo motivo è bene evitare l'utilizzo di questa istruzione.

Esercizi

1. Scrivere un programma che copi il contenuto di un file di interi `f` presente su memoria di massa in un file `g`.

2. Scrivere un programma che copi il contenuto di un file di interi `f` presente su memoria di massa *alla fine* di un file di interi `g` presente anch'esso su memoria di massa (poichè l'apertura di `g` in scrittura con `rewrite` cancella tutto il contenuto di `g`, è necessario copiare preliminarmente il contenuto di `g` in un file temporaneo `h`).