Lezione 11
8 novembre 1999

Argomenti trattati
- Struttura dei file `input` e `output`.
- Il tipo `text`.
- Le funzioni `eof` ed `eoln`.
- Esempi di programmi per il trattamento dei testi.

11.1 I file di testo

Sinora, per la comunicazione con l'ambiente esterno al programma, sono stati utilizzati i due canali standard, identificati dai nomi `input` e `output`: per leggere da `input` abbiamo fatto uso delle procedure predefinite `read` e `readln`, per scrivere su `output` ci siamo serviti invece delle procedure predefinite `write` e `writeln`.

In Pascal ogni canale di comunicazione con l'ambiente esterno al programma viene trattato come *file*. Ad esempio, la tastiera e lo schermo sono visti come file di nomi, rispettivamente, `input` e `output`, da cui si possono ricevere o su cui si possono inviare dati. Un programma può utilizzare altri dispositivi esterni (stampante, disco, ecc.). L'invio o la ricezione di dati avviene sempre trattando questi dispositivi come file. Lo studio completo dei tipi `FILE` in Pascal verrà affrontato piú avanti. Ci limitiamo per ora a considerare una particolare famiglia di file, i file di testo, cioè del tipo predefinito `text`, alla quale appartengono `input` e `output`.

Un *file di testo* non è altro che una sequenza di righe, ognuna delle quali è, a sua volta, costituita da una sequenza di caratteri che termina con un indicatore di fine linea. Il file, a sua volta, termina con un indicatore di fine file.

Per dichiarare una variabile di nome `f`, di tipo file di testo, è necessario utilizzare la dichiarazione:

```
VAR f: text;
```

Il nome del file va inoltre specificato nell'intestazione del programma, in quanto esso rappresenta un canale di comunicazione con l'esterno (quando successivamente studieremo in dettaglio i tipi `FILE`, vedremo che è anche possibile utilizzare un file "localmente" a un programma, senza in questo caso indicarne il nome nell'intestazione).

Per essere utilizzato, un file deve essere aperto in lettura o in scrittura, mediante le procedure predefinite `reset` e `rewrite`. Una volta che il file è stato aperto, un *puntatore* indica la posizione

in cui verrà effettuata la prossima operazione. I file sono *sequenziali*, pertanto il puntatore può muoversi solo in avanti. Per il file di `output` è utile immaginare il puntatore come il cursore sullo schermo che può man mano avanzare a destra di una posizione o andare all'inizio della riga successiva.

In particolare:

- `reset(f)` apre il file `f` per la lettura, ponendo il puntatore all'inizio del file;
- `rewrite(f)` apre il file `f` per la scrittura, svuotando nello stesso tempo il file dal contenuto precedente, e ponendo il puntatore all'inizio del file.

La lettura e la scrittura avvengono utilizzando le procedure standard `read`, `readln`, `write` e `writeln`. In particolare, se `c` è una variabile di tipo `char`, e il file `f` è stato aperto in lettura, sono possibili le seguenti operazioni:

- `read(f, c)`: legge dal file di nome `f` il carattere indicato dal puntatore e lo pone nella variabile di nome `c`; sposta il puntatore sul carattere successivo;
- `readln(f)`: sposta il puntatore sul file `f` all'inizio della riga successiva (saltando gli eventuali caratteri rimasti sulla riga corrente);
- `readln(f, c)`: equivale a `read(f, c)` seguita da `readln(f)`.

Se in una chiamata di `read` o `readln` non si specifica il nome del file, allora viene assunto come nome `input`. È inoltre possibile elencare piú di una variabile, come abbiamo già visto in alcuni esempi, o elencare variabili di altri tipi, come ad esempio `integer`; in questo caso verrà letta una sequenza di cifre, convertita automaticamente in numero intero, e il valore verrà posto nella variabile indicata. Tuttavia, si potrebbero avere errori in esecuzione (se ad esempio viene letta una lettera al posto di una cifra).

Nel caso in cui il file `f` sia aperto il scrittura, possiamo effettuale le operazioni:

- `write(f, c)`: scrive il carattere contenuto nella variabile di nome `c` nella posizione indicata dal puntatore sul file di nome `f`; sposta il puntatore avanti di una posizione sulla stessa riga;
- `writeln(f)`: scrive un *ritorno a capo*, cioè marca con uno o piú caratteri speciali (che rappresentano l'indicatore di fine linea) la posizione in cui si trova il puntatore, per segnalare che la riga corrente è terminata, e posiziona il puntatore all'inizio della riga successiva;
- `writeln(f, c)`: equivale a `write(f, c)` seguita da `writeln(f)`.

Se non viene specificato il nome del file, viene assunto come nome `output`. Inoltre anche in questo caso è possibile elencare piú di una variabile e variabili di tipi differenti (i cui valori, in fase di scrittura saranno convertiti in sequenze di caratteri). Si possono inoltre specificare ampiezze di campo, come già visto in una lezione precedente per la scrittura su `output`.

Esistono due funzioni, che restituiscono valori di tipo `boolean`, che permettono di verificare se è stata raggiunta la fine di una riga o la fine del file:

- `eof(f)`: restituisce valore `true` nel caso il puntatore sul file `f` si trovi sull'indicatore di fine file (end–of–file);
- `eoln(f)`: restituisce valore `true` nel caso il puntatore sul file `f` si trovi sull'indicatore di fine riga (end–of–line).

La chiamata delle funzioni `eof` e `eoln` senza indicazione del nome del file, si riferisce a `input`. In particolare la fine della linea di `input` corrisponde al tasto "invio". La fine del file di input corrisponde alla pressione di un tasto, o una combinazione di tasti particolare, che dipende dal sistema utilizzato. In ambiente UNIX, la fine del file di input corrisponde alla combinazione dei tasti ctrl-d premuti all'inizio di una riga; in ambiente Macintosh alla combinazione ctrl-c; in ambienti DOS/Windows al tasto `ESC` o alla combinazione ctrl-z.

Esempio 1: copia di un file

Vogliamo scrivere un programma che copi il contenuto di un file di testo `f` in un file di testo `g`.

Il programma sarà costituito da due fasi: la prima di apertura dei file, la seconda di copia.

L'apertura dei due file avviene utilizzando le procedure standard `reset` e `rewrite`, rispettivamente su `f` e su `g` (si ricordi che l'eventuale contenuto precedente di `g` verrà perso).

Una volta che i file sono stati aperti, si copieranno successivamente le righe di `f` su `g`, fino a raggiungere la fine del file. Ad alto livello si dovrà dunque eseguire il seguente ciclo:

```
WHILE f non e' finito DO
BEGIN
   copia la riga di f su cui si trova il puntatore in g
   sposta il puntatore di f sulla riga successiva
   scrivi un "a capo" su g
END
```

Le operazioni `sposta il puntatore di f sulla riga successiva` e `scrivi un "a capo" su g` vengono effettuate semplicemente con le due chiamate `readln(f)` e `writeln(g)`. Per effettuare l'operazione `copia la riga di f su cui si trova il puntatore in g` occorre ricordare che una riga è una sequenza di caratteri. Pertanto, si copiano uno alla volta i caratteri della riga nel file `g`. In altre parole, si può utilizzare il seguente ciclo:

```
WHILE la riga di f su cui si trova il puntatore non e' finita DO
BEGIN
   leggi il prossimo carattere di f
   e scrivilo su g
END
```

A tale scopo si definisce una variabile `c` di tipo `char`, per effettuare la copia del carattere. L'operazione `leggi il prossimo carattere di f` sarà semplicemente una `read(f, c)`, mentre l'operazione `scrivilo su g` sarà una `write(f, c)`.

Ecco il listato del programma completo costruito seguendo questo schema:

```
PROGRAM copiafile (f, g);

{copia il contenuto del file di testo f nel file di testo g}

   VAR
      f, g: text;
      c: char;

BEGIN
   reset(f);
   rewrite(g);
   WHILE NOT eof(f) DO
      BEGIN
         WHILE NOT eoln(f) DO
            BEGIN
               read(f, c);
               write(g, c)
            END; {while}
         readln(f);
         writeln(g)
      END {while}
END.
```

Associazione tra nome interno e nome esterno

Così come è stato scritto, il programma `copiafile` permette di copiare un file di nome `f` in un file di nome `g`. Volendo utilizzare il programma per copiare file di nomi differenti, è necessario sostituire i nomi in tutto il testo del programma. Per evitare ciò, i vari ambienti di programmazione mettono a disposizione dei metodi per associare al *nome interno* del file, cioè al nome utilizzato nel programma, come `f` e `g`, un *nome esterno*. Negli ambienti THINK Pascal e HP–Pascal, è possibile creare questa associazione, indicando, come secondo parametro di `reset` o `rewrite`, il nome esterno del file. Ad esempio, per copiare un file di nome `pippo` in un file di nome `pluto`, potremmo sostituire le due istruzioni di apertura file del programma `copiafile`, con `reset(f,'pippo')` e `rewrite(g,'pluto')`. Al posto di indicare direttamente i nomi come costanti tra apici, è possibile utilizzare variabili di tipo stringa alle quali siano stati assegnati i nomi desiderati. Nel Turbo Pascal l'associazione avviene invece chiamando, prima dell'apertura del file, una procedura di nome `assign`, che abbia come parametri rispettivamente il nome interno e il nome esterno. Nel nostro esempio si dovrebbe dunque scrivere:

```
assign(f, 'pippo');
reset(f);
assign(g, 'pluto');
rewrite(g);
```

Nell'ambiente IRIE Pascal sono disponibili entrambe le modalità di associazione tra nome esterno e nome interno sopra descritte.

Esempio 2: calcolo del numero di occorrenze di ciascuna vocale in `input`

Vogliamo costruire un programma che legga un testo da input e stampi il numero di occorrenze di ciascuna vocale nel testo.

Il programma sarà così strutturato:

1. inizializzazione variabili;
2. lettura del testo e calcolo del numero di occorrenze;
3. stampa del risultato.

I dati principali del programma saranno cinque variabili, `na`, `ne`, `ni`, `no` e `nu`, utilizzate per contare le frequenze di ciascuna vocale.

Scriviamo questo programma utilizzando tre procedure senza parametri corrispondenti alle tre parti evidenziate sopra. Il programma avrà la seguente struttura:

```
PROGRAM FrequenzeVocali (input, output);

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

   VAR
      na, ne, ni, no, nu: integer;

...dichiarazioni delle procedure inizializza, LeggiEConta, scrivi...

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

La procedura `inizializza` non deve far altro che azzerare le variabili utilizzate come contatori. Pertanto il suo codice è:

```
PROCEDURE inizializza;
BEGIN {inizializza}
   na := 0;
   ne := 0;
   ni := 0;
   no := 0;
   nu := 0
END; {inizializza}
```

La procedura `scrivi` deve invece scrivere in `output` i valori dei contatori:

```
PROCEDURE scrivi;
BEGIN {scrivi}
   writeln('Il testo contiene ', na : 1, ' a');
   writeln('Il testo contiene ', ne : 1, ' e');
   writeln('Il testo contiene ', ni : 1, ' i');
   writeln('Il testo contiene ', no : 1, ' o');
   writeln('Il testo contiene ', nu : 1, ' u')
END; {scrivi}
```

La procedura `LeggiEConta` deve occuparsi di ricevere i caratteri da `input`, esaminarli e incrementare opportunamente i contatori. Poniamoci sempre ad alto livello, e consideriamo prima di tutto il file `input` come una sequenza di righe. Per esaminare l'intero file dovremo pertanto effettuare un ciclo tipo:

```
WHILE il file non e' finito DO
BEGIN
   esamina una riga
   spostati sulla riga successiva
END
```

Per spostarsi sulla riga successiva è sufficiente scrivere `readln`. Ricordando che una riga non è altro che una sequenza di caratteri, l'esame di ciascuna riga può essere realizzato a sua volta con un altro ciclo:

```
WHILE la riga non e' finita DO
BEGIN
   leggi il prossimo carattere
   esaminalo
END
```

Definiamo, localmente alla procedura, una variabile `c` di tipo `char`, utilizzata per memorizzare il carattere letto. Pertanto, la lettura del prossimo carattere verrà effettuata mediante `read(c)`. L'esame del carattere avverrà secondo questo schema:

```
IF il carattere e' una vocale
   THEN incrementa il contatore corrispondente
```

L'incremento del contatore corrispondente alla vocale, viene effettuato all'interno di un'istruzione `CASE`, con la quale si effettua una selezione in base alla vocale letta. La procedura risulta essere pertanto:

```
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 vocale, incrementane il contatore}
               IF (c = 'a') OR (c = 'A') OR (c = 'e') OR (c = 'E') OR
                  (c = 'i') OR (c = 'I') OR (c = 'o') OR (c = 'O') OR
                  (c = 'u') OR (c = 'U') THEN
                  CASE c OF
                     'a', 'A':
                        na := na + 1;
                     'e', 'E':
                        ne := ne + 1;
                     'i', 'I':
                        ni := ni + 1;
                     'o', 'O':
                        no := no + 1;
                     'u', 'U':
                        nu := nu + 1
                  END {case c}
            END; {while not eoln ...}

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

La lunga condizione `(c = 'a') OR (c = 'A') OR ... OR (c = 'U')` può essere sostituita dalla condizione piú compatta ed elegante `c IN ['a','A','e','E','i','I','o','O','u','U']`, che risulta vera quando il contenuto della variabile `c` di tipo `char` appartiene all'insieme di caratteri specificato a destra dell'operatore `IN`. Lo studio degli insiemi in Pascal verrà affrontato in maniera sistematica in una lezione successiva.

Ecco il listato completo del programma:

```
PROGRAM FrequenzeVocali (input, output);

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

   VAR
      na, ne, ni, no, nu: integer;

   PROCEDURE inizializza;
   BEGIN {inizializza}
      na := 0;
```

```
   ne := 0;
   ni := 0;
   no := 0;
   nu := 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 vocale, incrementane il contatore}
               IF c IN ['a','A','e','E','i','I','o','O','u','U'] THEN
                  CASE c OF
                     'a', 'A':
                        na := na + 1;
                     'e', 'E':
                        ne := ne + 1;
                     'i', 'I':
                        ni := ni + 1;
                     'o', 'O':
                        no := no + 1;
                     'u', 'U':
                        nu := nu + 1
                  END {case c}
            END; {while not eoln ...}

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


PROCEDURE scrivi;
BEGIN {scrivi}
   writeln('Il testo contiene ', na : 1, ' a');
   writeln('Il testo contiene ', ne : 1, ' e');
   writeln('Il testo contiene ', ni : 1, ' i');
   writeln('Il testo contiene ', no : 1, ' o');
   writeln('Il testo contiene ', nu : 1, ' u')
END; {scrivi}
```

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

Esempio 3: conteggio del numero di parole in `input`

Vogliamo scrivere un programma che legga un testo da `input` e scriva in `output` il numero di parole incontrate. Per semplicità considereremo come parola una sequenza di lettere, delimitata da qualsiasi altro carattere. Ad esempio la sequenza `aa54bc` corrisponderà a due parole.

Anche in questo caso il programma è costituito da una fase di inizializzazione, da una fase di lettura e conteggio e da una fase di scrittura del risultato. Il dato fondamentale è una variabile, `nparole`, di tipo `integer`, utilizzata per contare il numero di parole.

Schematizziamo brevemente la fase di lettura e conteggio. Come nel caso precedente, essa sarà costituita da un ciclo principale in cui si esaminano le righe:

```
WHILE il file non e' finito DO
BEGIN
   esamina una riga
   spostati sulla riga successiva
END
```

L'esame di ciascuna riga verrà a sua volta realizzato mediante un altro ciclo:

```
WHILE la riga non e' finita DO
BEGIN
   leggi il prossimo carattere
   esaminalo
END
```

Utilizziamo, come sempre, una variabile `c` di tipo `char`, per memorizzare il carattere letto. L'esame di un singolo carattere è essenzialmente un test del tipo:

```
IF e' finita una parola
   THEN incrementa il contatore delle parole
```

Osserviamo che la fine di una parola viene individuata quando, dopo avere letto una sequenza di lettere, si trova un carattere che non è una lettera, oppure quando dopo una sequenza di lettere si trova la fine della riga. Introduciamo dunque una variabile di nome `InParola` di tipo `boolean`, che vale `true` quando ci si trova all'interno di una parola. La fine di una parola viene dunque trovata quando il valore di `InParola` è `true` e il carattere che viene letto non è una lettera, oppure quando il valore di `InParola` è `true` e viene raggiunta la fine di una riga.

Possiamo dunque riscrivere l'esame di una carattere come:

```
{se il carattere e' una lettera allora ti trovi in una parola}
IF (c >= 'a') AND (c <= 'z') OR (c >= 'A') AND (c <= 'Z') THEN
   InParola := true
{se il carattere non e' una lettera, e se prima eri in una parola}
{allora non sei piu' in una parola; incrementa il contatore}
ELSE IF InParola THEN
   BEGIN
      InParola := false;
      nparole := nparole + 1
   END
```

Inoltre, alla fine di una riga, occorrerà anche controllare se è finita una parola con il test:

```
IF InParola THEN
   nparole := nparole + 1
```

La variabile `InParola` deve essere inizializzata a `false` all'inizio di ogni riga.

Utilizzando di nuovo gli insiemi, la condizione `(c >= 'a') AND (c <= 'z') OR (c >= 'A') AND (c <= 'Z')` può essere riscritta come:

```
c IN ['a'..'z','A'..'Z']
```

Ecco il listato completo del programma:

```
PROGRAM ContaParole (input, output);

{conta il numero di parole nel file di input}

   VAR
      nparole: integer;
      c: char;
      InParola: boolean;


BEGIN {ContaParole}
   nparole := 0;

   {esamina il testo}
   WHILE NOT eof DO
      {esamina una riga}
      BEGIN
         InParola := false;
         WHILE NOT eoln DO
            {esamina un carattere}
            BEGIN
               read(c);

               {se il carattere e' una lettera allora ti trovi in una parola}
               IF c IN ['a'..'z','A'..'Z'] THEN
                  InParola := true
               {se il carattere non e' una lettera, e se prima eri in una parola}
               {allora non sei piu' in una parola; incrementa il contatore}
               ELSE IF InParola THEN
                  BEGIN
                     InParola := false;
                     nparole := nparole + 1
                  END

            END; {while not eoln ...}

         {se prima di finire la riga eri in una parola, incrementa il contatore}
         IF InParola THEN
            nparole := nparole + 1;
         {passa alla riga successiva}
         readln
```

```
    END; {while not eof ...}
  writeln('Sono state lette ', nparole : 1, ' parole')
END. {ContaParole}
```

Per esercizio si può studiare come migliorare il programma, considerando ad esempio come parola una sequenza di lettere delimitata da spazi, da caratteri di punteggiatura o da parentesi.

Esercizi

1. Scrivere un programma che legga un file di testo e lo visualizzi sullo schermo, numerandone ciascuna riga. Ad esempio, dato il file:

   ```
   Nel mezzo del cammin di nostra vita
   mi ritrovai per una selva oscura
   che' la diritta via era smarrita.
   Ahi quanto a dir qual era e' cosa dura
   esta selva selvaggia e aspra e forte
   che nel pensier rinova la paura!
   Tant'e' amara che poco e' piu' morte;
   ma per trattar del ben ch'i' vi trovai,
   diro' de l'altre cose ch'i' v'ho scorte.
   Io non so ben ridir com'i' v'intrai,
   tant'era pien di sonno a quel punto
   che la verace via abbandonai.
   ```

   si dovrà produrre in output:

   ```
    1  Nel mezzo del cammin di nostra vita
    2  mi ritrovai per una selva oscura
    3  che' la diritta via era smarrita.
    4  Ahi quanto a dir qual era e' cosa dura
    5  esta selva selvaggia e aspra e forte
    6  che nel pensier rinova la paura!
    7  Tant'e' amara che poco e' piu' morte;
    8  ma per trattar del ben ch'i' vi trovai,
    9  diro' de l'altre cose ch'i' v'ho scorte.
   10  Io non so ben ridir com'i' v'intrai,
   11  tant'era pien di sonno a quel punto
   12  che la verace via abbandonai.
   ```

2. Scrivere un programma che legga un file di testo e lo visualizzi sullo schermo, convertendo tutte le lettere minuscole in maiuscole: Ad esempio, l'output prodotto sul file dell'esempio precedente, dovrà essere:

   ```
   NEL MEZZO DEL CAMMIN DI NOSTRA VITA
   MI RITROVAI PER UNA SELVA OSCURA
   CHE' LA DIRITTA VIA ERA SMARRITA.
   AHI QUANTO A DIR QUAL ERA E' COSA DURA
   ESTA SELVA SELVAGGIA E ASPRA E FORTE
   CHE NEL PENSIER RINOVA LA PAURA!
   TANT'E' AMARA CHE POCO E' PIU' MORTE;
   ```

```
MA PER TRATTAR DEL BEN CH'I' VI TROVAI,
DIRO' DE L'ALTRE COSE CH'I' V'HO SCORTE.
IO NON SO BEN RIDIR COM'I' V'INTRAI,
TANT'ERA PIEN DI SONNO A QUEL PUNTO
CHE LA VERACE VIA ABBANDONAI.
```

Nota: se la variabile `c` di tipo `char` contiene una lettera minuscola, la lettera maiuscola equivalente può essere ottenuta come risultato dell'espressione `chr(ord(c)-ord('a')+ord('A'))`.

3. Modificare il programma dell'esercizio precedente, in modo che scriva il proprio output su un file di testo.

4. Il comando `wc` di UNIX conta il numero di righe, di parole e di caratteri presenti in un file di testo. Si costruisca un programma che ne simuli il comportamento.

5. Modificate il programma che conta il numero di occorrenze di ciascuna vocale in un testo, in modo che conti il numero di occorrenze di ogni lettera.