Introduzione al Pascal

Dott. Simone Zuccher

10 aprile 2007

Blaise Pascal
Clermont-Ferrand, Puy-de-Dôme, Francia, 19 giugno 1623 – Parigi, 19 agosto, 1662.
Fu matematico, fisico, filosofo e religioso.

Indice

Elenco delle figure

Elenco delle tabelle

1 Introduzione

1.1 Che cos'è il Pascal?

Il Pascal è un linguaggio di programmazione ad alto livello, come per esempio il FORTRAN (scopi scientifici), il C (scopi industriali) e il Java (applicazioni con interfacce grafiche). Questo linguaggio di programmazione è nato con lo scopo di essere utilizzato a fini didattici e non necessariamente per la realizzazione di software applicativi.

La programmazione in Pascal risulta semplice ed ordinata e "costringe" il programmatore a rispettare alcune *regole* che rendono il codice ben strutturato e di facile lettura. Infatti, tutto ciò che si utilizza deve essere dichiarato e quindi viene esclusa la possibilità di commettere errori di programmazione dovuti a digitazione errata dei nomi delle variabili (come potrebbe accadere in C e FORTRAN), perché il compilatore le rifiuta se non sono state dichiarate preventivamente.

Questa struttura ordinata e rigorosa è uno degli aspetti fondamentali che rende il Pascal un ottimo linguaggio per imparare a programmare. Anzi, proprio queste *limitazioni*, che possono risultare noiose per un programmatore esperto, sono molti utili al "programmatore in erba" in quanto forniscono una metodologia di programmazione corretta e rendono più semplice l'individuazione di errori concettuali fatti nella scrittura del codice. Un altro punto a favore del Pascal come linguaggio didattico è la semplicità con cui si scrive il codice, essendo le istruzioni del Pascal costituite da espressioni di tipo algebrico e da termini inglesi (dette parole riservate) come PROGRAM , BEGIN , END etc.

Uno svantaggio del linguaggio è certamente la scarsa flessibilità che lo rende inadatto allo sviluppo di software applicativi.

1.2 Storia del Pascal

Il Pascal è stato sviluppato all'inizio degli anni '70 da Niklaus Wirth del Politecnico di Zurigo con lo scopo di realizzare un linguaggio ad alto livello orientato all'insegnamento della programmazione strutturata.

Il linguaggio sviluppato da Wirth viene a volte definito *Pascal standard*, in realtà in seguito lo standard del linguaggio è stato definito dall'American National Standards Institute (ANSI), quindi spesso si parla di Pascal ANSI. I compilatori disponibili oggi aderiscono perfettamente allo standard ANSI, ma sempre vengono fatte delle aggiunte che rendono la programmazione migliore, anche se non sempre sono compatibili con altri compilatori.

Attualmente sono disponibili diversi compilatori, free software e non, ma sicuramente il compilatore più famoso è il Turbo Pascal creato nel 1983 dalla Borland.

L'evoluzione dell'informatica ha apportato anche al Pascal alcune modifiche: inizialmente il Pascal supportava la programmazione strutturata, oggi supporta anche la programmazione orientata agli oggetti (OOP Object Oriented Programming); la nascita di ambienti di sviluppo visuali ha dato vita al Delphi, che viene utilizzato anche nello sviluppo di software applicativi.

Oggi il Pascal viene ampiamente impiegato nel campo per cui era stato creato e cioè

la didattica, ma non viene utilizzato per la produzione di applicazioni, dove vengono preferiti linguaggi "industriali" quali il C.

1.3 Compilatori: il GNU Pascal

Sono molti i compilatori disponibili. Viene qui presentato unicamente GNU-Pascal, essendo esso disponibile gratuitamente sotto Linux.

I pregi dello GNU Pascal sono essenzialmente una perfetta aderenza allo standard A.N.S.I., una buona velocità, e tutte le altre caratteristiche che rendono unici i compilatori GNU. Per contro, la quantità di librerie che vengono fornite con questo compilatore non è molto estesa. Un'altra nota dolente è la mancanza della possibilità di scrivere codice assembly integrato all'interno del codice Pascal, cosa che altri compilatori consentono.

Naturalmente questo compilatore è fornito con licenza GPL (www.gnu.org), altrimenti non sarebbe GNU, ed inoltre esistono dei porting per altre piattaforme tra cui DOS e Win32.

Il nome del comando per i sistemi Linux è `gpc`.

Per avere un'idea sull'utilizzo del compilatore basta eseguire `man gpc` oppure, per la distribuzione Debian, `gpc --help`

```
$man gpc

GPC(1)                        GNU Tools                        GPC(1)

NAME
       gpc - GNU project Pascal Compiler (v2.0)

SYNOPSIS
       gpc [option | filename ]...

...............
...............
...............
```

```
gpc --help
Usage: gpc [options] file...
Options:
  -pass-exit-codes         Exit with highest error code from a phase
  --help                   Display this information
  --target-help            Display target specific command line options
  (Use '-v --help' to display command line options of sub-processes)
  -dumpspecs               Display all of the built in spec strings
  -dumpversion             Display the version of the compiler
  -dumpmachine             Display the compiler's target processor
  -print-search-dirs       Display the directories in the compiler's search path  -print-lib
  -print-file-name=<lib>   Display the full path to library <lib>
  -print-prog-name=<prog>  Display the full path to compiler component <prog>
  -print-multi-directory   Display the root directory for versions of libgcc
```

```
  -print-multi-lib         Display the mapping between command line options and
                           multiple library search directories
  -print-multi-os-directory Display the relative path to OS libraries
  -Wa,<options>            Pass comma-separated <options> on to the assembler
  -Wp,<options>            Pass comma-separated <options> on to the preprocessor  -Wl,<optio
  -Xassembler <arg>        Pass <arg> on to the assembler
  -Xpreprocessor <arg>     Pass <arg> on to the preprocessor
  -Xlinker <arg>           Pass <arg> on to the linker
  -save-temps              Do not delete intermediate files
  -pipe                    Use pipes rather than intermediate files
  -time                    Time the execution of each subprocess
  -specs=<file>            Override built-in specs with the contents of <file>
  -std=<standard>          Assume that the input sources are for <standard>
  -B <directory>           Add <directory> to the compiler's search paths
  -b <machine>             Run gcc for target <machine>, if installed
  -V <version>             Run gcc version number <version>, if installed
  -v                       Display the programs invoked by the compiler
  -###                     Like -v but options quoted and commands not executed
  -E                       Preprocess only; do not compile, assemble or link
  -S                       Compile only; do not assemble or link
  -c                       Compile and assemble, but do not link
  -o <file>                Place the output into <file>
  -x <language>            Specify the language of the following input files
                           Permissible languages include: Pascal c c++ none
                           'none' means revert to the default behavior of
                           guessing the language based on the file's extension

Options starting with -g, -f, -m, -O, -W, or --param are automatically
 passed on to the various sub-processes invoked by gpc.  In order to pass
 other options on to these processes the -W<letter> options must be used.

For bug reporting instructions, please see:
<URL:http://www.gnu-Pascal.de/todo.html>.
```

1.3.1 Turbo Pascal

Parlando di Pascal non si può non parlare del Turbo Pascal. Il Turbo Pascal è un compilatore prodotto dalla Borland che grazie alla sua semplicità, potenza e velocità è diventato "IL" compilatore Pascal. Le innovazioni introdotte dalla Borland oggi sono diventate degli standard, e molti compilatori si preoccupano di essere compatibili con il turbo Pascal. Purtroppo questo compilatore è disponibile solo per DOS ed è un software commerciale.

1.3.2 Delphi

Delphi è un ambiente di sviluppo visuale molto potente e veloce. Il linguaggio usato dal Delphi è sostanzialmente il Pascal con delle aggiunte (tipo OOP) che lo rendono più potente e flessibile.

Anche Delphi è un software commerciale prodotto dalla Borland ed è disponibile per Windows. Da un po' di tempo ne esiste una versione anche per Linux, Kylix, che non è free software ma che è comunque disponibile gratuitamente se viene utilizzato per lo sviluppo di software libero.

1.4 Perché il Pascal?

Il motivo principale che spinge all'uso del Pascal è semplicemente la voglia di imparare a programmare, la semplicità di utilizzo, la rigorosità dell'impostazione. Tutti questi fattori rendono il Pascal uno dei migliori linguaggi disponibili per imparare a *programmare come si deve*.

Un secondo motivo è la possibilità di passare a programmare in Delphi in maniera semplice e naturale, consentendo così la realizzazione di software ad un livello molto alto.

Il terzo motivo per scegliere il Pascal è per la realizzazione di applicazioni destinate ad un uso personale o che non necessitano di prestazioni al top, evitando di complicarsi la vita con linguaggi più difficili quali il C.

Solitamente il Pascal è solo il primo di una serie di linguaggi di programmazione, come è stato per l'autore. Tuttavia, il rigore del Pascal consente di non trovare grosse difficoltà quando si passa a linguaggi più complessi come il C.

2 Elementi base di un programma in Pascal

2.1 Il classico "Hello World!"

Il primo programma che ciascun programmatore scrive per vedere se ha capito i rudimenti di un linguaggio è il classico "Hello World!":

```
(1) (*Programma HelloWorld*)
(2) program HelloWorld;
(3) begin
(4)   writeln('Hello World!');
(5) end.
```

Analizziamo il programma riga per riga.

Alla riga 1 si trova un commento. Il testo scritto tra `(* *)` viene visto dal compilatore come un commento e non viene considerato ai fini della compilazione.

La riga 2 indica l'inizio del programma con la parola chiave **program** e dove "HelloWorld" rappresenta il nome del nostro programma.

Le righe 3 e 5 indicano l'inizio del blocco principale del programma, tra le parole chiave **begin end** vanno inserite le instruzioni che verranno eseguite

La riga 4 stampa a video la stringa 'Hello World', il comando `writeln('testo da stampare');` scrive a video tutto quello che si trova tra gli apici.

Dopo aver salvato le 5 righe di codice nel file `hello.pas`, si compili il programma eseguendo semplicemente

```
$ gpc -o hello hello.pas
```

Se il programma non contiene errori, nella stessa directory di `hello.pas` si trova il file eseguibile `hello`. Si noti che l'opzione `-o` seguita da `hello` ordina al compilatore di generare un file di output (da cui `-o`) di nome `hello`. Nel caso questa opzione venga omessa il nome di default per l'eseguibile è `a.out`.

2.2 Struttura di un programma in Pascal

Un programma Pascal si divide generalmente in tre parti: Intestazione, Blocco delle dichiarazioni e Blocco delle istruzioni.

L'intestazione comincia con la parola chiave **program** seguita dal nome del programma e terminata da un punto e virgola `;`:

```
program <nome_programma>;
```

Il blocco delle dichiarazioni definisce i vari dati che si usano nel programma e si può suddividere nel seguente modo:

1. Etichette (`Label`);
2. Costanti (`Const`);
3. Tipi di dato definiti dall'utente (`Type`);
4. Variabili (`Var`);
5. Procedure e Funzioni (`Procedure` e `Function`);

In seguito questi concetti saranno più chiari, per ora ci si limita a dare una visione generale del blocco dichiarativo.

Il blocco delle istruzioni contiene le istruzioni che il programma deve eseguire al fine di produrre un'azione (quale può essere il calcolo di un'operazione o la scritta di un testo a video).

Riepilogando, questa la struttura di un programma Pascal è la seguente:

1. Intestazione;
2. Dichiarazioni;
   (a) Label;
   (b) Const;
   (c) Type;
   (d) Var;
   (e) Procedure e Function;
3. Istruzioni;

Visto dal punto di vista della codice abbiamo :

```
(*Intestazione*)
program nome_programma;
(*Blocco delle dichiarazioni*)
label L1,L2.... Ln;
const ..... ;
type ..... ;
var ..... ;
procedure ..... ;
function ..... ;
(*Blocco delle istruzioni*)
begin
(*Istruzioni*)
end.
(*Fine del programma*)
```

Si noti che non è possibile mettere la parte delle istruzioni prima di quella delle dichiarazioni (per ovvi motivi), mentre è possibile invertire l'ordine di alcuni elementi nella parte dichiarativa. Tuttavia, è consiglibile non farlo. Per imparare a memoria l'ordine delle varie parti si può ricorrere alla frasetta *La Cosa Tre Volte Più Facile*, in cui ogni iniziale corrisponde ad un elemento della parte dichiarativa nell'ordine corretto.

2.3 Parole riservate

Ogni linguaggio ha una serie di *parole riservate* che servono per scopi precisi e che il programmatore non può in alcun modo ridefinire per i suoi scopi.

In tabella 1 è riportata la lista completa delle parole riservate del Pascal A.N.S.I.

AND	ARRAY	BEGIN	CASE
CONST	DIV	DO	DOWNTO
ELSE	END	FILE	FOR
FORWARD	FUNCTION	GOTO	IF
IN	LABEL	MOD	NIL
NOT	OF	OR	PACKED
PROCEDURE	PROGRAM	RECORD	REPEAT
SET	THEN	TO	TYPE
UNTIL	VAR	WHILE	WITH

Tabella 1: Tabella delle parole riservate

2.4 Identificatori

Un identificatore è il nome che si assegna ad una parte di un programma Pascal, per esempio una variabile, una costante, una funzione etc.

Un identificare deve rispettare le seguenti semplici regole:

1. Essere costituito da una sequenza qualsiasi di lettere e cifre, purché il primo carattere sia una lettera. Non c'è differenza tra maiuscole e minuscole in quanto il Pascal non è case-sensitive, cioè non distingue tra maiuscole e minuscole.

2. Non sono ammessi caratteri diversi da lettere o numeri (si noti che il simbolo *underscore* '`_`' è ammesso, mentre ovviamente il simbolo *meno* '`-`' non lo è.

3. Non è possibile usare un parola riservata come identificatore.

4. Non sono ammessi gli spazi.

Vediamo ora alcuni esempi per chiarire meglio.

`Pippo Pippo21 Pippo34 Pippo_56` sono degli identificatori validi, inoltre per quanto detto prima `PIPPO pippo Pippo` sono tutti lo stesso identificatore. `12Pippo` non è valido in quanto inizia con un numero. `P/ippo` non è valido in quanto contiene il carattere `/` che non è né lettera né numero. `Var` non è valido in quanto è una parola riservata.

2.5 Variabili e tipi

I tipi di dati elementari del linguaggio Pascal dipendono dal compilatore utilizzato e dall'architettura dell'elaboratore sottostante. I tipi standard del Pascal ANSI sono elencati nella tabella 2. Il tipo 'char', non fa parte dello standard ANSI, ma è molto diffuso e quindi viene incluso in quella tabella.

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

Tabella 2: Elenco dei tipi di dati primitivi in Pascal.

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

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

Gli interi (`integer`) vanno espressi con numeri normali, senza punti di separazione di un'ipotetica parte decimale, prefissati eventualmente dal segno meno (`-`) nel caso di valori negativi.

I valori `byte` vanno espressi come gli interi positivi, con la limitazione della dimensione massima.

I numeri reali (`real`) possono essere espressi come numeri aventi una parte decimale, segnalata dalla presenza di un punto decimale. Se si vuole indicare un numero reale corrispondente a un numero intero, si deve aggiungere un decimale finto, per esempio, il numero 10 si può rappresentare come `10.0`. Naturalmente è ammessa anche la notazione esponenziale. Quindi, per esempio, `7e-2` corrisponde a $7 \cdot 10^{-2}$, pari a 0.07.

I valori logici (`boolean`) vengono espressi dalle costanti letterali `TRUE` e `FALSE`.

I valori carattere (`char`) e stringa (`string`), vengono delimitati da coppie di apici singoli, come `'A'`, `'B'`, ... `'Hello World!'`.

2.6 Costanti e variabili

Una costante, come dice la parola, è un oggetto che rimane sempre con lo stesso valore per tutta la durata del programma.

La struttura per dichiarare una costante è la seguente:

```
const <identificatore> = <valore>;
```

Es. `const GiorniSettimana = 7;`
Es. `const Ore = 24;`

Le variabili possono essere viste come degli oggetti che possono contenere un dato, ossia un contenitore. Chiaramente il valore del dato può cambiare nel corso del programma purché il tipo rimanga sempre uguale.

Le varibili vanno dichiarate nel seguente modo:

```
var <identificatore1>,<indenticatore2> : <tipo di dato>;
```

Es. `var Nome:string;`
Es. `var NumeroIntero: integer;`

Altri esempi:
`var conta : integer;` dichiara la variabile 'conta' di tipo intero.
`var conta,canta : integer;` dichiara le variabili 'conta' e 'canta' di tipo intero.
`var conta : integer;`
`canta : integer;`
sono esattamente uguali all'esempio precedente.

Per dichiarare la variabile `conta` di tipo intero, e la variabile `lettera` di tipo carattere:

```
var
        conta   :       integer;
        lettera :       char;
```

2.6.1 Assegnazione di variabili

Per assegnare un valore ad una certa variabile si utilizza l'operatore di assegnamento ":=", la cui sintassi è la seguente:
`<nome variabile> := <espessione o valore>;`
Naturalmente l'espressione o il valore devono essere dello stesso tipo della variabile.

2.7 Dati di tipo numerico

I dati numerici si dividono in due tipi: numeri interi e numeri reali. I dati interi sono i numeri interi (con segno), a questo gruppo appartengono varibili, costanti e funzioni che producono numeri interi, il tipo intero in Pascal viene indicato con la parola chiave

Operatore	Scopo	Operandi	Risultato
`+`	Addizione	`integer`	`integer`
`-`	Sottrazione	`integer`	`integer`
`*`	Moltiplicazione	`integer`	`integer`
`/`	Divisione	`integer`	`real`
`DIV`	Divisione intera (con resto)	`integer`	`integer`
`MOD`	Resto della divisione intera	`integer`	`integer`

Tabella 3: Tabella degli operatori artimetici sul tipo **integer**.

integer. Gli operatori artimetici che possono lavorare sugli interi sono riportati in tabella 3.

Il seguente programma mostra come si dichiarano le variabili di tipo **integer** e come si usano gli operatori appena visti.

```
program ProvaInteger;

(*Fase di dichiarazione*)
var A,B,C,D: integer;
          R:Real;

begin

(* Assegnazione*)
A:=12;
B:=23;
C:=-25;

(*Operazioni Aritmetiche*)
D:=A+C; (*12-25= -13*)
D:=B-C; (*23+25= 48*)
D:=B DIV 2; (*23 div 2 = 11 con resto 1*)
D:=B MOD 2 ; (*1 e' il resto di 23:2*)
R:=B / 2 ; (*11.5*)

end.
```

I dati reali sono i numeri reali (con segno). A questo gruppo appartengono varibili, costanti e funzioni che producono numeri reali. Il tipo in Pascal viene indicato con l'identificatore **real**.

Gli operatori artimetici che possono lavorare su dati **real** sono riportati in tabella 4.

Operatore	Scopo	Operandi	Risultato
`+`	Addizione	`real`	`real`
`-`	Sottrazione	`real`	`real`
`*`	Moltiplicazione	`real`	`real`
`/`	Divisione	`real`	`real`

Tabella 4: Tabella degli operatori artimetici sul tipo **real**.

2.8 Dati di tipo char e stringhe

I dati di tipo **char**, come indica la parola, sono i caratteri cioè stringhe di un solo carattere. I caratteri ammessi sono i caratteri del codice A.S.C.I.I. standard (vedi tabella in figura 1, pagina 16, per il codice standard e 2, pagina 17, per il codice esteso). Il seguente programma mostra come si dichiarano le variabili di tipo **char** e come si usa il codice A.S.C.I.I.

```
program ProvaRealChar;
  var a,b,c : real;
      d,e: char;
      ascii:integer;
begin
  a:=23.4;
  b:=0.344E2; (*0.344 x 10^2 = 34.4*)
  c:=a+b;
  c:=a-b;
  c:=a*b;
  c:=a/b;
  d:='d';
  d:='3';
  e:='r';
  e:='A';
  e:=char (65); (*la funzione char restituisce
                  il carattere corrispondente al
                  codice ascii inserito in questo
                  caso 'A' = ascii 65 *)
  ascii:=ord('A'); (*opposta a char da 65*)
  writeln(e,' ',ascii);
end.
```

Si noti l'uso della funzione `char(integer)`che restituisce il carattere corrispondente al codice A.S.C.I.I passato come parametro. La funzione opposta a `char(integer)` è `ord(char)`, che restituisce il valore numerico corrispondente al codice ascii del carattere passato come parametro.

Le stringhe di caratteri in Pascal rappresentano un tipo di dato proprio del linguaggio che viene indicato con la parola riservata **string**. Una variabile di tipo **string** può contene al massimo una stringa di 255 caratteri. Tra i dati di tipo string intervengono

diverse funzioni e anche l'operatore "+" detto operatore di concatenazione, che non fa altro che fondere più stringhe in una sola. Il seguente programma mostra come si dichiarano le variabili di tipo **string** ed alcuni usi.

```
program ProvaStringhe;
  var S1,S2,S3:string;
begin
  S1:='Rossi';
  S2:=''; (*stringa vuota*)
  S3:='Carlo'; S2:=S3 + ' ' + S1; (* S2='Carlo Rossi'*)
  writeln(S1,' ',S2,' ',S3,' ');
end.
```

È inoltre possibile definire stringhe con ampiezza massima minore di 255 usando il seguente indentificatore: `string[dimensione_massima]`. Per esempio, `var S: string[12];` definisce una stringa con valore massimo 12 caratteri.

2.9 Dati di tipo boolean

Diciamo che i valori booleani sono dei valori di verità, vero e falso in italiano, **true** e **false** in inglese oppure on e off, oppure 0 e 1. Una espressione booleana assume questi valori in seguito ad una operazione logica quale può essere un confronto.

Su un dato booleano intervengono tre opeazioni fondamentali: **and**, **or** e **not**. Queste operazioni agiscono su dati booleani e restituiscono sempre un dato booleano.

Vediamo qui di seguito come funzionano queste operazioni (usiamo la denominazione inglese True e False per indicare i valori booleani)

AND	True	False
True	T	F
False	F	F

Tabella 5: Tabella dell'operatore **AND**.

OR	True	False
True	T	T
False	T	F

Tabella 6: Tabella dell'operatore **OR**.

Come si vede **and** (tabella 5) resituisce il valore **true** solo quando entrambi gli operandi sono **true**, mentre l'operazione **or** (vedi tabella 6) restituisce **false** solo quando riceve due **false** in ingresso. Il **not** (vedi tabella 7), invece, non fa altro che invertire il valore passato (negazione).

Il tutto si può anche immaginare come numeri binari `true=1` e `false=0` e le operazione **and** come moltiplicazione binaria e **or** come somma binaria.

Un'operazione che restituisce un valore booleano è il confronto. Per esempio:

NOT	
T	F
F	T

Tabella 7: Tabella dell'operatore **NOT**.

```
(12 < 10) (*Restituisce False*)
(12 > 10) (*Restituisce True*)
(12 >10) AND (3 < 1) (*Restituisce True*)
```

In Pascal i valori booleani formano un vero e propio tipo di dato contrassegnato dall'indentificatore **boolean**.

Gli operatori **and or not** intervengono su questi tipi di dati e le operazioni sono combinate per mezzo di parentesi `()`.

Gli operatori che possono lavorare su dati booleani sono riportati in tabella 8.

Significato	Operatore
Maggiore	>
Maggiore o uguale	>=
Minore	<
Minore o uguale	>=
Uguale	=
Diverso	<>

Tabella 8: Tabella degli operatori artimetici sul tipo **boolean**.

Di seguito viene riportato un esempio sull'utilizzo di dati booleani.

```
program ProvaBoolean;
  var A,B,C: Boolean;
begin
  A:=(12<15); (*vero*)
  writeln('12<15? A=',A);
  B:=(12<10); (*falso*)
  writeln('12<10? B=',B);
  writeln('A AND B: ', (A AND B));
  writeln('A OR B: ', (A OR B));
  writeln('NOT(A AND B): ', (NOT(A AND B)));
end.
```

2.10 Input ed output di base (read e write)

Il comando **read** (`read(<lista della variabili>)`) prende come parametri delle variabili, anche una solamente, legge quello che l'utente ha scritto da tastiera non appena viene premuto invio, e lo assegna alla lista delle variabili. I tipi di variabili leciti sono

integer real char string. Ogni variabile della lista è separata dell'altra mediante la virgola (“,”). Nel caso di più variabili ci sono due possibilità di realizzare l'input. Una è di scrivere tutti i valori separati da uno spazio, l'altra di digitare invio dopo ogni dato.

Il comando **write** (`write(<dati in uscita>)`) stampa a video i dati che gli vengono passati come parametro. I dati leciti sono variabili, costanti e stringhe, separati dalla virgola.

Esistono due varianti di read e write che sono rispettivament **writeln** e **readln**. Questi due comandi funzionano nello stesso modo di **write** e **read**, con l'unica differenza di portare il cursore su una nuova linea appena finito (`ln` sta appunto per “line”).

Il seguente codice dovrebbe far capire meglio l'uso di questi comandi.

```
program InputeOutput;
  var i: integer;
      r: real;
      c: char;
      s: string;
begin
  writeln('Ciao questo programma  un esempio di Input e output');
  (*Scrive a video questa stringa e va a capo*)

  writeln; (*Stampa una riga vuota*)

  write('Inserisci un numero intero :');
  readln(i);
  writeln('Hai inserito : ',i);
  writeln;

  write('Inserisci un numero reale :');
  readln(r);
  writeln('Hai inserito ',r);
  (*r viene stampato nel formato esponenziale*)
  writeln('Hai inserito : ',r:3:4);(*Dopo lo spieghiamo*)

  writeLn;

  write('Inserisci un carattere :');
  readln(c);
  writeln('Hai inserito : ',c);

  writeln;

  write('Inserisci una stringa :');
  readln(s);
  writeln('Hai inserito :',s);

  writeln;
```

```
  write('Inserimento multiplo di intero e reale :');
  readln(i,r);
  writeln('Hai inserito : ',i,' ',r);
end.
```

Per stampare i numeri reali oltre la notazione esponenziale è possibile utilizzare il comando `write(r:X:Y)`, dove `r` è il numero reale, `x` è un numero naturale che indica quanto spazio deve occupare il numero scritto (numero di caratteri) mentre `y` indica il numero di cifre dopo la virgola (in questo caso se le cifre sono di più verrà effettuato un arrotondamento).

Nel caso dei comandi **write** e **writeln** è quindi possibile formattare l'output inserendo lo spazio che un certo dato deve occupare sullo schermo. Per dire quanto spazio deve occupare un dato si usa, come visto, la variante `<variabile>:<spazio>` all'interno dei due comandi. Naturalmente se il dato necessita di più spazio la formattazione viene ignorata. Nel caso in cui il dato richiede meno spazio, questo viene preservato e il dato viene allineato a destra. Il seguente programma riassume tali concetti.

```
program Formattazione;
  var a1,a2,a3,a4,a5,a6: integer;
      r1,r2,r3,r4,r5,r6: real;
begin
  writeln('Programma di esempio per la formattazione del testo');
  writeln;
  writeln('Testo non Formattato');
  a1:=10;
  a2:=334;
  a3:=2103;
  a4:=0;
  a5:=46;
  a6:=555;
  r1:=1.54;
  r2:=22.3;
  r3:=-34.45;
  r4:=-r2;
  r5:=r1*r1;
  r6:=r5-r1;
  writeln;
  writeln('Tabella:');
  writeln;
  writeln(a1,' ',a2,' ',a3);
  writeln(a4,' ',a5,' ',a6);
  writeln;
  writeln(r1,' ',r2,' ',r3);
  writeln(r4,' ',r5,' ',r6);
  writeln;
  writeln('Testo Formattato');
```

```
  writeln;
  writeln('Tabella:');
  (*Ogni dato occupa uno spazio di 16 colonne*)
  writeln; WriteLn(a1:16,a2:16,a3:16);
  writeln(a4:16,a5:16,a6:16);
  writeln;
  WriteLn(r1:16:4,r2:16:4,r3:16:4);
  WriteLn(r4:16:4,r5:16:4,r6:16:4);
end.
```

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	□

Figura 1: Codice ASCII.

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
128	80	Ç	160	A0	á	192	C0	└	224	E0	α
129	81	ü	161	A1	í	193	C1	┴	225	E1	ß
130	82	é	162	A2	ó	194	C2	┬	226	E2	Γ
131	83	â	163	A3	ú	195	C3	├	227	E3	π
132	84	ä	164	A4	ñ	196	C4	─	228	E4	Σ
133	85	à	165	A5	Ñ	197	C5	┼	229	E5	σ
134	86	å	166	A6	ª	198	C6	╞	230	E6	µ
135	87	ç	167	A7	º	199	C7	╟	231	E7	τ
136	88	ê	168	A8	¿	200	C8	╚	232	E8	Φ
137	89	ë	169	A9	⌐	201	C9	╔	233	E9	Θ
138	8A	è	170	AA	¬	202	CA	╩	234	EA	Ω
139	8B	ï	171	AB	½	203	CB	╦	235	EB	δ
140	8C	î	172	AC	¼	204	CC	╠	236	EC	∞
141	8D	ì	173	AD	¡	205	CD	═	237	ED	ø
142	8E	Ä	174	AE	«	206	CE	╬	238	EE	ε
143	8F	Å	175	AF	»	207	CF	╧	239	EF	∩
144	90	É	176	B0	░	208	D0	╨	240	F0	≡
145	91	æ	177	B1	▒	209	D1	╤	241	F1	±
146	92	Æ	178	B2	▓	210	D2	╥	242	F2	≥
147	93	ô	179	B3	│	211	D3	╙	243	F3	≤
148	94	ö	180	B4	┤	212	D4	╘	244	F4	⌠
149	95	ò	181	B5	╡	213	D5	╒	245	F5	⌡
150	96	û	182	B6	╢	214	D6	╓	246	F6	÷
151	97	ù	183	B7	╖	215	D7	╫	247	F7	≈
152	98	ÿ	184	B8	╕	216	D8	╪	248	F8	°
153	99	Ö	185	B9	╣	217	D9	┘	249	F9	∙
154	9A	Ü	186	BA	║	218	DA	┌	250	FA	·
155	9B	¢	187	BB	╗	219	DB	█	251	FB	√
156	9C	£	188	BC	╝	220	DC	▄	252	FC	ⁿ
157	9D	¥	189	BD	╜	221	DD	▌	253	FD	²
158	9E	₧	190	BE	╛	222	DE	▐	254	FE	■
159	9F	ƒ	191	BF	┐	223	DF	▀	255	FF	

Figura 2: Codice ASCII esteso.

3 Strutture di controllo in Pascal

3.1 If

Questa struttra di controllo del flusso di dati, **if**, è presente praticamente in tutti i linguaggi di programmazione. La sua sintassi è la seguente:

```
if (<condizione>) then <blocco1>
                  else <blocco2>
```

La condizione è un'espressione booleana: se restituisce **true** viene eseguito il primo blocco altrimenti il secondo blocco. L'**else** può anche essere omesso ed in questo caso se la condizione è vera viene eseguito il blocco altrimenti l'esecuzione prosegue dalla prima istruzione dopo il blocco. Per esempio:

```
program ProvaIF;
  var A,B:Real;
begin
  (*
    questo programma esegue la divisione
    a/b dopo avere letto i dati dall'utente,
    naturalmente dobbiamo controllare che b sia
    diverso da 0
  *)
  writeLn(' Inserisci A e B ');
  readln(A,B);

  if (B<>0) then writeln('A/B=',A/B:5:4) (*Notate che manca il ; *)
                else writeln('ERRORE : non posso dividere per 0');
end.
```

Si noti che l'ultima istruzione prima dell'**else** non va terminata con il punto e virgola (questo sempre).

3.2 Case

La struttura **case** permette di effettuare una scelta multipla in base al risultato di un'espressione oppure al contenuto di una variabile. La sintassi è la seguente:

```
case (<espressione o variabile>) of
  <listavalori1> : <Blocco1>
  <listavalori2> : <Blocco2>
  ..................
  ..................
  <listavaloriN> : <BloccoN>
else <BloccoELSE>
end;
```

La struttura funziona valutando il valore dell'espressione o della variabile e cercando il corrispondente valore tra quelli indicati nelle varie liste. Se il valore viene trovato allora viene eseguito il blocco di codice corrispondente, altrimente viene eseguito il blocco dell'**else**. Come per **if**, l'uso dell'**else** è opzionale. Se nella lista delle opzioni non viene trovato alcun valore corrispondente alla variabile del **case** allora non viene eseguito nessun blocco di codice e l'esecuzione prosegue.

```
case i of
    1,2,3 : .......
    4,5,6 : .......
    else .....
end;
```

In questo esempio `i` è una variabile di tipo **integer**. Se il suo valore è tra 1 e 3 allora viene eseguito il primo blocco, altrimenti se il suo valore è tra 4 e 6 il secondo, altrimenti viene eseguito il blocco dell'**else**.

```
case (i mod 9) of
  1,2,3,7,8: writeln('Blocco1');
  4,5,6    : writeln('Blocco2');
end;
```

In questo esempio viene valutata l'espressione `(i mod 9)` che restituisce un risultato tra 0 e 8. Se il risultato è 1,2,3,7 oppure 8 allora viene scritto a video "Blocco1", negli altri casi viene eseguito il secondo blocco e quindi viene stampato a video "Blocco2".

Un programma più complesso è il seguente.

```
program ProvaCase;
  var scelta:integer;(*variabile di valutazione*)
begin
  (*
    Questo programma mostra il funzionamento
    della struttura di controllo CASE OF
  *)
  (*
    Il programma stampa a video un menu con 3
    opzioni e poi stampa qual'e' la scelta effettuata
    dall'utente
  *)
  writeln;
  writeln(' Menu Pricipale : ');
  writeln;
  writeln(' 1 Opzione A');
  writeln(' 2 Opzione B');
  writeln(' 3 Opzione C');
  writeln;
  write(' Digita la scelta (1..3) : ');
```

```
  readln(scelta);

  case scelta of
    1 : Writeln(' Hai scelto l''opzione A !!');
    2 : Writeln(' Hai scelto l''opzione B !!');
    3 : Writeln(' Hai scelto l''opzione C !!')
  else writeln ('ERRORE : La scelta effettuata non e' corretta');
  end;
end.
```

3.3 While

Come la parola stessa induce a pensare, la struttura di controllo **while** permette di eseguire un blocco di codice finché una data condizione non cambia. Questo corrisponde a realizzare un ciclo con controllo della condizione all'inizio. La forma generale della struttura è la seguente:

```
while (<condizione>) do
  <blocco delle istruzioni>
```

dove la condizione è un'espressione di tipo booleano. Se la condizione assume il valore **true**, allora il blocco di codice (che può anche essere una singola istruzione) viene eseguito, altrimenti l'esecuzione salta alla prima istruzione dopo il blocco considerato. Pertanto, il ciclo può anche non essere mai esegiuto. Per evitare che il ciclo continui all'infinito, è necessario che all'interno del blocco ci sia un'istruzione che ad un certo punto modifica la condizione iniziale rendendola **false**.

```
(1)  i:=1;

(2)  while (i<=10) do
       begin
(3)       writeln(i);
(4)       inc(i);
       end;
```

Questa porzione di codice stampa i numeri da 1 a 10 in colonna. Da notare alla riga (4) la procedura `inc (integer)` che incrementa di 1 il valore della variabile (è equivalente a scrivere `i:=i+1;`), procedura analoga a `inc` è `dec(integer)` che decrementa di 1 il valore della variabile.

Il seguente programma mostra l'utilizzo del **while**.

```
program ProvaWhile;
  var num:integer;
begin
  {Questo e' un programma di prova per capire l'uso della
   struttura while.
   Questo prog. chiede all'utente di inserire dei numeri
```

```
   che vengono moltiplicati per 10 e il risultato viene
   stampato a video.
   L'inserimento dei numeri continua finche' l'utente non
   inserisce il valore 0 (zero)
  }
  write('Inserisci una serie di numeri che verranno moltiplicati * 10');
  writeln(' (inserisci 0 per fermarti)');
  writeln;
  write(' Num : ');
  readln(num);
  while (num<>0) do
  begin
    writeln(' num*10 = ',num*10);
    writeln;
    write(' Num : ');
    readln(num);
  end;
end.
```

In questo programma si mostra l'utilizzo delle graffe (`{ commento }`) per i commenti.

3.4 Repeat

La struttura di controllo **repeat** è analoga alla struttura **while** e consente di eseguire un blocco di codice finché una data condizione non cambia. La forma generale della struttura **repeat** è la seguente:

```
repeat
  <Istruzioni>
until <condizione>;
```

Le istruzioni sono contenute tra **repeat until** e quindi non necessitano degli identificatori di blocco **begin end**.

Questa struttura, a differenza del **while** ha il controllo della condizione in coda e quindi le istruzioni sono eseguite almeno una volta. Un'altra differenza sta nel fatto che il blocco delle istruzioni viene eseguito finché la condizione è **false**. Non appena la condizione assume il valore **true** il programma esce dal ciclo ed esegue la prima istruzione dopo **until**.

```
i:=1;
somma:=0;
repeat
  somma := somma + i;
  i := i + 1;
until (i>n);
```

Questo esempio mostra come calcolare la somma dei primi N numeri naturali. Si vede che il contatore i assume i valori da 1 a n, e non appena $i = n+1$ il ciclo si ferma perché

la condizione diventa vera. Alla fine, nella variabile `somma` è contenuto il valore della somma dei primi N numeri.

Il seguente programma mostra l'utilizzo del **repeat**.

```
program ProvaRepeat;
  var i,n:integer;
begin
  {Questo e' un programma di prova per mostrare l'uso
   della struttura repeat
   Questo programma stampa i numeri da 1 a N dove N
   e' inserito dall'utente
  }
  write('Inserisci N :');
  readln(n);
  i:=1;
  repeat
    writeln(i);
    inc(i);
  until (i>n);
end.
```

3.5 For

La struttura **for** permette di eseguire un ciclo per un numero di volte prestabilito. La sintassi della strutture **for** è la seguente:

```
for <variabile> := <viniziale> to <vfinale> do
<Blocco>
```

La variabile dichiarata assume tutti i valori tra `viniziale` e `vfinale`, incrementandola ogni volta di uno. Il numero di cicli eseguiti è ovviamente pari a `vfinale - viniziale + 1`.

Esiste anche una variante che permette di andare in ordine decrescente, utilizzando la parola **downto** anziché **to**:

```
for <variabile> := <vfinale> downto <viniziale> do
<blocco>
```

In questo caso `variabile` assume tutti i valori da `vfinale` a `viniziale`, diminuendola ogni volta di uno. Il numero di cicli è pari a `vfinale - viniziale + 1`.

```
for i := 1 to 10 do
    writeln(i);
```

Questo esempio stampa i numeri da 1 a 10 e, come si può notare, è molto più semplice dello stesso esempio con il **while**.

Il seguente programma mostra l'utilizzo del **for**.

```
program ProvaFor;
  var i,n,num:integer;
begin
 (*
 Queto programma e' un esempio per mostrare l'uso della struttura for.
 Questo programma chiede all'utente di inserire una lista di numeri di
 dimensione N di cui viene calcolato il quadrato
 *)
 write('Inserisci la dimensione della lista di numeri :');
 readln(n);
 for i:= 1 to n do
 begin
   write('Inserisci il ',i,' numero : ');
   readln(num);
   writeln(num,'^2=',num*num);
 end;
end.
```

4 Procedure e funzioni

4.1 Sintassi generale

Il Pascal, come quasi tutti i linguaggi, permette la programmazione per moduli. Ogni modulo ha un compito preciso, è indipendente dagli altri, e si comporta come se fosse un programma all'interno del programma principale (da qui il termine *sottoprogramma* o *subroutine*).

Il Pascal ha due metodi per realizzare sottoprogrammi: procedure e funzioni. Le procedure (**procedure**) sono utilizzate quando il sottoprogramma deve eseguire delle azioni senza necessariamente restituire dei valori. Le funzioni (**function**), invece, restituiscono un dato dopo una serie di elaborazioni. Quindi, mentre per le procedure non c'è bisogno di definire per esse un tipo di dato, una funzione è di un certo tipo. La sintassi di funzioni e procedure è la seguente:

```
procedure <nome procedura> (<lista parametri>);
  <Corpo della procedura>

function <nome funzione> (<lista parametri>): <tipo restituito>;
  <corpo della funzione>
```

4.2 Corpo della procedura e della funzione, esempi

Siccome sottoprogramma è un programma vero e proprio, la struttura di una funzione o di una procedura è uguale a quella di un generico programma (vedi 2.2). Ricapitolando si ha:

1. Intestazione;

2. Dichiarazioni;
    (a) Label;
    (b) Const;
    (c) Type;
    (d) Var;
    (e) Procedure e Function;
3. Istruzioni;

È possibile definire all'interno di una subroutine delle altre subroutine, permettendo così un'ulteriore modularizzazione del programma.

La seguente procedura chiede all'utente di inserire due numeri di cui calcola la somma e stampa a video il risultato.

```
procedure Somma ;
  var A,B: integer;
begin
 write('Inserisci A e B : ');
 readln(A,B);
 writeln(A,'+',B,'=',A+B);
end;
```

Per quanto riguarda le funzioni, il valore in uscita si assegna come il solito utilizzando la sintassi per l'assegnazione (`:=`), ovvero `<nome funzione> := <valore uscita>;`. Per esempio, la seguente funzione SQR calcola il quadrato del parametro A e ne restituisce il valore:

```
function SQR (A:integer):integer;
  begin
    SQR := A*A;
  end;
```

4.3 Lista dei parametri in entrata ed in uscita

Ad ogni sottoprogramma sono passati dei parametri che forniscono una sorta di interfaccia tra programma e modulo. La sintassi di dichiarazione dei parametri è la seguente:

```
(var1,var2, ,varn:tipo1;varj, , ,varm:tipo2;...);
```

I parametri del sottoprogramma possono essere usati come qualunque altra variabile all'interno della subroutine.

Supponiamo di volere un modulo che ricevuti in ingresso due numeri interi A e B ne calcoli il prodotto e lo metta in una variabile C. Questo può essere fatto con una **procedure** o con una **function**, come mostrano i seguenti esempi.

Procedure:

```
procedure prod(A,B:integer;VAR C:integer);
  begin
    C:=A*B;
  end;
```

Function:

```
function prod(A,B:integer):integer;
  begin
    Prod := A*B;
  begin;
```

Si sarà notato che nella procedura `prod` il parametro C è stato dichiarato come `var` al contrario degli altri. Questo perché in Pascal esistono *parametri in ingresso* e *parametri in uscita*. La differenza è la seguente: quando ad un sottoprogramma si passa un parametro *in ingresso*, per esempio la variabile A, il sottoprogramma crea una variabile *locale* con lo stesso nome e sulla quale lavora durante l'esecuzione del sottoprogramma stesso. Quindi, ogni modifica effettuata sul parametro non si ripercuote al di fuori del sottoprogramma. Le modifiche sui parametri *in uscita*, invece, sono effettive perché non viene passata una copia del dato ma il dato vero e propio. Per dichiarare un parametro *in uscita*, che quindi deve avere le stesse caratteristiche delle variabili, si usa la parola riservata **var**.

```
procedure Prod(A,B: integer; VAR C : integer);
begin
(0)    C:=A*B;
(1)    A:=12;
(2)    B:=32;
(3)    Writeln(A,' ',B);
end;
```

Si noti che, essendo i parametri A e B in ingresso, le modifiche alle righe `(1)` e `(2)` non hanno effetto, ossia se `A=10` e `B=2` prima della procedura, allora `C=20` e `A` e `B` rimagono con valore 10 e 2 dopo la chimata della procedura. Al contrario, all'interno della **procedure** `Prod` i valori di `A` e `B` cambiano per effetto di `(1)` e `(2)`, come si può notare dalla stampa effettuata dall'istruzione `(3)`. Se C non fosse variabile (**var**), la modifica della riga `(0)` non avrebbe alcun effetto, e quindi per ogni `A` e `B`, `C` sarebbe sempre uguale a 0 (che è il valore di default di ogni variabile non inizializzata).

4.4 Invocazione di procedure e funzioni

Per richiamre una **procedure** basta eseguire

```
<nome procedura> (<parametri attuali>);
```

Si noti che i parametri attuali sono quelli forniti al momento dell'invocazione e ovviamente possono chiamarsi in modo diverso da quelli formali, utilizzati nella dichiarazione della **procedure**. L'unico vincolo è che i parametri attuali sia dello stesso tipo dei parametri formali. Tra i parametri attuali possono esseri passati anche le costanti (poiché non

variano durante l'esecuzione del sottoprogramma). Se `prod` è la **procedure** introdotta nella sezione precedente, tutte le seguenti chiamate sono valide

```
...
...
var A,B,C,D,E,F: integer;
...
A:=10;
B:=12;
prod(A,B,C);
prod(10,12,C);
D:=6;
E:=7;
prod(D,E,F);
prod(12,A,B);
```

La prima calcola in C il prodotto delle variabili A e B, la seconda calcola il prodotto di 10 e 12 in C la terza mette in F il prodotto di D per E,mentre la quarta mette in B il prodotto A per 12. Ovviamente, i parametri attuali vengono trasformati in quelli formali, ossia nella seconda chiamata A=10 e B=12, mentre nella terza A=D e B=E e C=F.

L'invocazione di una **function** va invece fatta con un'assegnazione ad una variabile dello stesso tipo:

```
<varialbile>:=<nome Funz> (<parametri attuali>);
```

Naturalmente, a `<variabile>` viene assegnato il valore restituito dalla funzione. Esempi:

```
c:= Prod(12,23);
a:=12;
c:= Prod(12,a);
b:=24;
c:= Prod(a,b);
```

5 Gestione di files

Il **file** è uno strumento che permette di memorizzare informazioni in modo permanente, su dispositivi di memoria di massa, che non vengono persi alla fine dell'esecuzione del programma. La sintassi per la dichiarazione di un tipo file è la seguente:

```
type <NomeTipoFile> = file of <tipodatidelfile>;
var F : <NomeTipoFile>;
```

Esistono due tipi di file: sequenziali e ad accesso diretto. Mentre nei file sequenziali l'acceso ad un singolo elemento, supponiamo nella posizione i avviene solo dopo avere letto gli altri $i-1$ elementi precedenti, nei file ad accesso diretto è possibile accedere direttamente all'elemento desiderato.

Le operazioni che si possono eseguire sui file sono:

- Creazione: essa stabilisce una corrispondenza tra il nome reale del file (nome esterno ossia quello sul disco) e il nome interno al programma (nome interno).
- Apertura: in lettura o in scrittura del file.
- Lettura o scrittura sul file.
- Chiusura.

5.1 Creazione del file: la procedura assign

Prima di elaborare un file si deve procedere alla sua creazione, che equivale a creare una corrispondenza tra il nome reale del file e il nome interno dello stesso. Per eseguire questa operazione si usa la procedura `assign`, la cui sintassi generale è la seguente

```
procedure assign(var <NomeInterno>:<file>;<NomeEsterno>:string);
```

Questa procedura è la prima ad essere chiamata quando si intende utilizzare i file. Chiariamo con alcuni esempi.

```
type FileInteri = file of integer;
var F : FileInteri;
...
assign (F,'file/interi.dat');
```

In questo caso il nome interno del file `interi.dat` contenuto nella sottodirectory `file` è `F`. Il nome esterno può indicare il percorso (path) relativo alla posizione del programma, come nell'esempio, oppure il path reale contenete l'intero percorso del file (del tipo `/home/username/file/prova.dat`).

5.2 Apertura del file: reset e rewrite

Dopo la creazione, per effettuare una qualsiasi operazione di lettura o scrittura bisogna aprire il file. Per aprire il file in *scrittura* si usa la procedura **rewrite**, invece per aprirlo in *lettura* si usa la procedura **reset**.

Quando si accedere ad un singolo elemento del file è come se ci fosse un puntatore in grado di muoversi tra i vari elementi e leggere o scrivere su di essi. Per comodità chiamiamo *testina* questo puntatore e aggiungiamo che per i file sequenziali questa testina non è libera di muoversi avanti e indietro ma per posizionarsi sull'elemento i deve prima leggere gli $i-1$ elementi precedenti. Si noti che la numerazione degli elementi del file parte da 0 e non da 1, quindi un file formato da N elementi va da 0 a $N-1$.

Quando viene aperto un file in scrittura viene istantaneamente creato un file vuoto e la testina posizionata in scrittura all'inizio del file. Quindi, l'apertura del file in scrittura provoca l'immediata perdita dei dai in esso contenuti.

Per scrivere un dato sul file si usa la procedura **write** che scrive il dato e porta avanti la testina. La sintassi generale di **rewrite** e **write** è la seguente

```
procedure rewrite(var <NomeInterno>:<file>)
procedure write(var <nomeinterno>:<file>;<Dato>:<Tipodati>)
```

Molto importante dopo la fase di scrittura è la chiusura del file. Quest'ultima consiste nell'aggiunta del marcatore EOF (End of File), ovvero un carattere che indica alla testina che il file è terminato e quindi non è più possibile leggere altri elementi. La chiusura avviene tramite la procedura **close**.

```
procedure close(<NomeInterno>:<file>)
```

Per sapere invece se ci si trova alla fine del file si utilizza la funzione `eof(<NomeInterno>)`. Essa restituisce il valore booleano **true** se la fine del file è stata raggiunto, **false** altrimenti.

Per aprire il file in *lettura* si usa la procedura **reset**, che posiziona la testina sul primo elemento del file. La lettura avviene con la procedura **read**, che legge il dato e porta la testina avanti.

```
procedure reset(var <NomeInterno>:<file>)

procedure read(var <NomeInterno>:<file>;var <Dato>:<Tipodati>)
```

Nell'esempio qui sopra **read** legge dal file `file` e mette il dato letto nella variabile `dato`.

```
type FileInteri = FILE OF integer;
var F : FileInteri;
...
assign(F,'prova.dat');
rewrite(F);
for i := 1 to 10 do write(F,i);
close(F);
```

Questo esempio mostra come scrivere i numeri da 1 a 10 sul file `prova.dat`. Per leggere i dati contenuti in esso nel file:

```
reset(F);
while (not eof(F)) do
  begin
    read(f,i);
    writeln(i);
  end;
```

Nell'esempio precedente la riga contenente **while** esegue un ciclo fino alla fine del file, quando la funzione eof restituisce **true** e quindi `not eof(f)` è **false** e il ciclo termina.

5.3 File sequenziali ad accesso diretto

Per evitare confusione vanno chiariti due aspetti dei file: l'organizzazione e il metodo di accesso. L'organizzazione indica come il file è disposto fisicamente sul disco, ovvero in modo sequenziale o random. Nell'organizzazione sequenziale tutti gli elementi sono dipsosti in modo sequenziale, uno dietro l'altro, nell'organizzazione random invece sono disposti in modo non sequenziale. Si noti che tutti i file gestisti dal Pascal sono organizzati in maniera sequenziale.

Il metodo di accesso, invece, è il modo con cui si accede ai file che può essere sequenziale o diretto. Nel primo caso per accedere all'elemento i bisogna leggere tutti gli $i-1$ elementi precedenti, invece con l'accesso diretto è possibile leggere direttamente l'elemento desiderato. Il Pascal A.N.S.I. prevede solo l'uso di file sequenziali ad accesso sequenziale, quelli visti nella sezione precedente, ma quasi tutte le implementazioni prevedono anche l'uso di file sequenziali ad accesso diretto.

I file sequenziali ad accesso diretto si comportano esattamente come quelli visti in precedenza, quindi tutte le fasi sono uguali e gestite dalle stesse procedure e funzioni. L'unica differenza sta in nella procedura **seek** che è in grado di muovere la testina sulla posizione desiderata. La sua sintassi è la seguente.

```
procedure Seek(var <NomeInterno>:<file>;<posizione>:integer)
```

Si ricordi che le posizioni vanno da 0 a $N-1$, dove N è la dimensione del file. Altre due funzioni utili per la gestione dei file in Pascal sono

```
function FileSize(var <NomeInterno> : <file>):integer

function Pos(var <NomeInterno> : <file>):integer
```

filesize restituisce la dimensione del file, invece **pos** restituisce la posizione corrente della testina.

Il seguente programma analizza in modo completo la gestione dei file.

```
program Prova_file;
  var F:file of integer;
      num:integer;
begin
  (*
  Questo programma e' un esempio che mostra l'uso dei file in Pascal
  Il programma legge una serie di numeri inseriti dall'utente e li memorizza su
  un file.
  In seguito il file viene letto e i dati stampati a video
  *)

  writeln('Inserire una serire di numeri ');
  writeln ('Per fermarsi inserite 0 ');

  (*Creazione ed apertura in scrittura del file*)
  assign(F,'dati.dat');
  rewrite(F);

  (*Inserimento dei dati*)
  writeln;
  write('Inserisci numero : ');
  readln(num);
  while (num<>0) do
```

```
    begin
      write(F,num);
      write('Inserisci numero : ');
      readln(num);
    end;

  (*Chiusura del File*)
  Close(F);

  (*Apertura in lettura e lettura dei dati*)
  reset(F);
  writeln;
  writeln('Hai inserito ',FileSize(F),' numeri');
  writeln;
  writeln('Lettura dei dati memorizzati :');

  while (not eof(F)) do
    begin
      read(F,num);
      Writeln(num);
    end;

  Close(F);
end.
```

5.4 File di testo

Il Pascal permette anche di usare i file di testo, che formano un tipo indicato con il nome **text**. I file di tipo text si comportano come dei file di caratteri quindi scrivere `var F:Text;` è equivalente a scrivere `var F:file of Char;` Siccome i file di testo si comportano come tutti gli altri file, valgono tutte le cose dette in precedenza su di essi. I file di testo hanno in più due funzioni che ne permettono l'elaborazione:

```
procedure readln (var <NomeInterno>:<file>)

procedure writeln(var <NomeInterno>:<file>)
```

La procedura **writeln** inserisce nel file il carattere di `EOLN` (End Of LiNe) che indica la fine di una riga. Conseguentemente, la procedura **readln** salta il carattere di `EOLN` e porta la testina all'inizio della nuova riga. Per sapere se la testina si trova o no su un carattere di `EOLN` esiste la funzione **eoln** che restituisce **true** se la testina è su un carattere di EOLN, **false** nel caso contrario. La sua sintassi è

```
function Eoln(var <nomeinterno>:Text):boolean

(1) VAR F : Text;
    ................
```

```
(2) Assign(F,'prova.txt');
    ................
(3) Rewrite(F);
(4) Write(F,'d');
(5) Writeln(F);
(6) Close(F);
```

Questo esempio mostra come creare (2) e aprire in scrittura (3) un file di testo, come scrivere un carattere (4) e come mettere un marcatore di `EOLN` con la procedura **writeln** (5). Con l'istruzione (6) il file viene chiuso e marcato con un `EOF`.

```
(1) VAR F : Text;
        c : char;
    ................
(2) Assign(F,'prova.txt');
    ................
(3) Reset(F);
(4) Read(F,c);
(5) Close(F);
```

In questo esempio viene letto un carattere da un file di testo (4).

Il seguente esempio completo mostra, tramite l'utilizzo di due cicli **while**, come effettuare la lettura e la scrittura su un file di testo.

```
program Prova_Text;
  var T:TEXT;
      stringa: string;
      c: char;

begin
  (*
  Questo e' un programma di esempio per mostrare l'uso dei file di testo.
  Il programma legge del testo e lo memorizza in un file che in seguito viene
  visualizzato
  *)

  (*Creazione e apertura in scrittura*)
  assign(T,'testo.txt');
  rewrite(T);

  (*Fase di scrittura testo*)
  writeln('Inserire un testo (digitare <*> per terminare)');
  Writeln;
  stringa:='';

  while  (stringa<>'*') do
    begin
```

```
      readln(stringa);
      if (stringa<>'*') then Writeln(T,stringa);
    end;
    close(T);

    (*Apertura in lettura*)
    reset(T);

    (*Lettura del testo*)
    Writeln;
    Writeln('Il testo inserito  il seguente :');
    Writeln;

    while (not eof(T)) do
      begin
        if (eoln(T)) then
          begin
            readln(T);
            writeln;
          end
        else
          begin
            read(T,c);
            Write(c);
          end;
      end;

      Close(T);
end.
```