PASCAL: LE FUNZIONI

TRATTO DA CAMAGNI-NIKOLASSY, CORSO DI INFORMATICA, VOL. 2, HOEPLI

Informatica

Le funzioni

Le funzioni

- Oltre alle procedure in Pascal è possibile definire un secondo tipo di sottoprogramma: le funzioni.
- Hanno un comportamento simile alle funzioni matematiche.

> Ricorda che una funzione matematica è una legge che mette in corrispondenza gli elementi di due insiemi A e B in modo che a ogni elemento X di A corrisponda uno e un solo elemento di B.

- Funzioni in Pascal
  - Allo stesso modo, in Pascal, una funzione esegue delle operazioni sulle su una o più variabili del programma chiamante e restituisce a esso un valore (**e uno solo!**) come risultato.

Definizione delle funzioni

- La notazione usata per definire la funzione ha un duplice significato:
  - Il nome della funzione
  - La variabile di ritorno nella quale è contenuto il risultato dell'elaborazione (che ha lo stesso nome della funzione
- Esempio:

```
function alcubo(<parametri>):integer;
```

Definizione delle funzioni

```
function alcubo(<parametri>):integer;
```

dove:

- la parola chiave `function` individua la funzione;
- `alcubo` è il nome della funzione e della variabile che conterrà il risultato;
- `(<parametri>)` è, come per le procedure, la lista dei parametri di ingresso;
- `:integer` è il tipo della variabile `alcubo` che conterrà il risultato.

Nel corpo della funzione deve sempre essere presente un'istruzione di assegnazione che al termine dell'elaborazione scrive il risultato nella variabile che ha il nome della funzione, in modo da ritornare tale valore al programma chiamante. In questo caso sarà presente un'istruzione come la seguente:

```
alcubo:=<risultato delle elaborazioni>;
```

Esempio

- Scriviamo un programma che calcola il cubo di un numero inserito

```
0  program funzione;
1  (* primo esempio di funzione *)
2  var
3    numero,cubo:integer;
4    function alcubo(numero:integer):integer;
5    begin
6      alcubo:=numero*numero*numero;  // calcola il cubo
7    end;                             // fine funzione
8  begin                              // inizio programma principale
9    write('introduci un numero intero ');
10   readln(numero);
11   cubo:=alcubo(numero);            // chiamata funzione
12   write('il cubo del numero letto e'' ');
13   writeln(cubo);
14 end.
```

Esempio

- In particolare nel programma chiamante si ha:

```
11  cubo:=alcubo(numero);                    // chiamata funzione
```

- Cioè viene eseguita come una normale istruzione di assegnazione
- Alla variabile cubo viene assegnato il valore della variabile alcubo (che è il parametro di ritorno delle'esecuzione della funzione alcubo)

È inoltre necessario assicurarsi che:

- i parametri passati rispettino il tipo indicato nell'elenco dei parametri formali;
- il valore di ritorno venga assegnato a una variabile dello stesso tipo indicato nella dichiarazione della funzione.

Esempio: funzioni con più parametri

- Massimo di tre numeri

```
0  program funzione2;
1  (* secondo esempio di funzione *)
2  var
3    numero1,numero2,numero3,max:integer;
4    function maggiore(num1,num2,num3:integer):integer;
5    begin
6      maggiore:=num1;
7      if num2>num1
8        then
9          maggiore:=num2;
10     if num3>maggiore
11       then
12         maggiore:=num3;
13   end;                          // fine funzione
14 begin                          // inizio programma principale
15   write('introduci tre numeri interi ');
16   readln(numero1,numero2,numero3);
17   max:=maggiore(numero1,numero2,numero3);     // calcolo
18   write('il maggiore dei tre numeri e'' ');
19   writeln(max);
20 end.                           // fine programma principale
```

Funzioni con scambio dei parametri per indirizzo

- Nel caso dell'esempio precedente potevamo scambiare un quarto parametro per indirizzo:

```
function maggiore(num1,num2,num3:integer;var max:integer) boolean;
```

Scambio per indirizzo

- In questo modo si va a scrivere il risultato dell'operazione direttamente nella variabile max.

Funzioni predefinite

- Sono funzioni che sono state scritte da altri utenti e che fanno parte di librerie specifiche
- Esempio: sqr(), sqrt(), esp(), ecc…
- Per utilizzarle bisogna inserire nel codice il riferimento alla libreria di appartenenza
- È necessario inserire nella parte dichiarativa il seguente codice:
  - **uses** <*nome_libreria*>;

Funzioni predefinite

Funzione	Argomento	Risultato
`sqr(x)`	intero o reale	Calcola il quadrato del numero *x*.
`sqrt(x)`	intero o reale	Calcola la radice quadrata del numero *x*.
`exp(x)`	intero	Calcola e^x.
`int(x)`	reale	Calcola la parte intera di *x* restituendo un reale.
`trunc(x)`	reale	Calcola la parte intera di *x* restituendo un intero.
`round(x)`	reale	Arrotondamento al valore intero più vicino a *x*.
`randomize`	nessun argomento	Genera un nuovo seme per i numeri casuali.
`random`	nessun argomento	Genera un numero casuale tra 0 e 1.
`random(x)`	intero	Genera un numero casuale tra 0 e *x*.
`abs(x)`	intero o reale	Calcola il valore assoluto di un numero *x*.
`sin(x)`	reale	Seno del numero *x*.
`cos(x)`	reale	Coseno del numero *x*.
`ln(x)`	reale	Logaritmo naturale del numero *x*.

Unit	Applicazione
`system`	Sovrintende alla fase di esecuzione e viene chiamata automaticamente da tutti i programmi.
`graph`	Insieme di funzioni e procedure per disegnare.
`dos`	Supporta le funzioni del sistema operativo DOS.
`crt`	Contiene le procedure per la gestione del video.
`printer`	Predispone l'utilizzo della stampante.

Esempio: radice quarta di un numero

```
0  program funzione3;
1  var
2    numero,radice4:real;
3    function sqrt4(numero:real):real;// inizio funzione
4    begin
5      if numero>0
6        then
7          sqrt4:=sqrt(sqrt(numero))   // radice della radice
8        else
9          sqrt4:=0;                   // errore
10   end;                               // fine funzione
11 begin                               // inizio programma principale
12   write('introduci un numero positivo ');
13   readln(numero);
14   radice4:=sqrt4(numero);           // calcolo radice quarta
15   write('la radice quarta è' ');
16   writeln(radice4:8:2);             // output formattato
17 end.                                // fine programma principale
```

Esempio: radice quarta di un numero

L'istruzione:

```
7   sqrt4:=sqr(sqr(numero))  // radice della radice
```

è un esempio di passaggio diretto del risultato di una funzione come parametro di ingresso a una seconda funzione, cioè una funzione di una funzione.

Esercizi 1-2, pag. 32

1 Che cosa fa la seguente funzione che utilizza le funzioni `pred(x)` e `succ(x)` che rispettivamente ritornano il numero precedente e successivo di x?

```
function pippo(a,b:integer):integer;
begin
  while a>0 do
   begin
      b:=succ(b);
      a:=pred(a);
   end;
  pippo:=b;
end;
```

2 Come per l'esercizio precedente, che risultato dà questa funzione?

```
function etabeta(a,b:integer):integer;
begin
  while a>0 do
    begin
      a:=pred(a);
    b:=succ(b);
    end;
  etabeta:=a*b;
end;
```

Esercizi 3-4, pag. 32

3 Che cosa fa la seguente funzione?

```
function pluto (e,b:integer):integer;
var p:integer;
begin
  p:=1;
  if e<>0
    then
      repeat
        p:=p*b;
        e:=e-1;
      until e=0;
  pluto:=p;
end;
```

4 Che cosa fa questo segmento di codice che utilizza la funzione `fatt()` che calcola il fattoriale di un numero?

```
readln(n);
k:=0;
while k<n do
begin
 a:=n-k;
 w:=fatt(n);
 y:=fatt(a);
 z:=fatt(k);
 bin:=w div(y*z);
 write(bin,' ');
 k:=k+1
end;
```

Esercizi 5, pag. 32

5 Qual è il contenuto della variabile z al termine dell'esecuzione della procedura h?

```
procedure h;
var x,y,z:integer;
 function f(var
x:integer;y:integer):integer;
 var z:integer;
 begin
   x:=y*2;
   z:=2x+y;
   f:=x+y+z
 end;
begin
  x:=1;y:=2;z:=3;
  z:=z+f(y,x)
end;
```

Problemi 1-6, pag. 33

1 Scrivi una funzione che ricevendo un numero in ingresso ritorna `TRUE` se tale numero è primo.

2 Scrivi una funzione che ricevendo in ingresso due date ritorna il numero di giorni trascorsi tra di esse.

3 Scrivi una funzione che ricevendo in ingresso due ore espresse in hh:mm:ss ritorna il numero di secondi trascorsi tra di esse.

4 Scrivi un programma che permetta di calcolare l'area di un cerchio, di un quadrato e di un triangolo equilatero richiamando tre funzioni. L'utente inserisce un numero che viene utilizzato rispettivamente come raggio e lato sia del triangolo sia del quadrato. Infine confronta i tre valori ottenuti indicando la figura con area maggiore.

5 Scrivi un programma che realizza un convertitore di base: leggendo un numero in formato decimale il programma deve offrire all'utente la possibilità di convertire una sequenza di numeri terminante con 0 (e inserita dall'utente stesso) in:

- binario
- ottale
- esadecimale

mediante una scelta operativa eseguita tramite un menu.

6 Dati due rettangoli R_1 (a_1, b_1, c_1, d_1) e R_2 (a_2, b_2, c_2, d_2) scrivi le seguenti funzioni:

- `funzione1`: calcola la lunghezza della base e dell'altezza del più piccolo rettangolo contenente entrambi i rettangoli dati;
- `funzione2`: restituisce 1 se R_1 è contenuto in R_2, altrimenti 0 (un rettangolo R_1 è contenuto nel rettangolo R_2 se ogni punto di R_1 appartiene a R_2);
- `funzione3`: riconosce se due rettangoli si intersecano o meno.

Nota: a_1, b_1, c_1, d_1 e analogamente a_2, b_2, c_2, d_2 sono i punti dei vertici, quindi coppie di coordinate cartesiane del tipo $a_1(x_1, y_1)$ ecc.

Problemi 7-10, pag. 33

7 Scrivi una funzione che ricevendo due numeri in ingresso calcola il minimo comune multiplo utilizzando una procedura che calcola la scomposizione in fattori.

8 Scrivi un programma che disegna sullo schermo il triangolo di Tartaglia dopo aver definito una funzione che calcola i coefficienti binomiali di una data potenza.

9 Scrivi un programma che riceva le temperature massime e minime di ogni giorno del mese di gennaio. Le temperature vengono inserite in due array che vengono poi visualizzati su richiesta dell'utente. Sempre su richiesta dell'utente, viene calcolata la media tra le temperature massime e il giorno in cui si è avuta la differenza minore tra temperatura massima e minima.

10 Scrivi un programma che implementi il gioco nel quale l'utente deve indovinare un numero segreto con una quantità massima di tentativi. La funzione `generaNumero()` permette di generare casualmente un numero di quattro cifre, mentre la funzione `indovinaNumero()`:

- consente di inserire un numero;
- confronta il numero da indovinare con quello inserito e visualizza il messaggio "troppo grande" o "troppo piccolo";
- se il numero inserito è corretto, ritorna la funzione `vero`, altrimenti ritorna `falso`. Il programma principale controlla se i tentativi ammessi sono esauriti, se il numero segreto è stato individuato o meno, e ripropone una nuova partita.

Le funzioni ricorsive

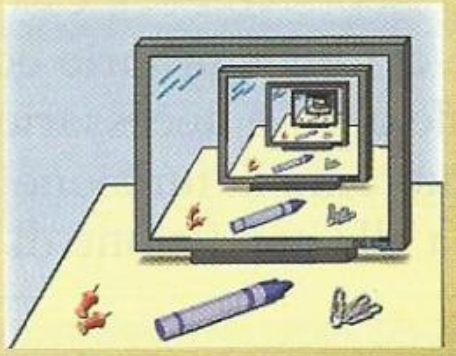
La ricorsione

- Una **funzione matematica** è definita ricorsivamente quando nella sua definizione compare un riferimento a essa
  - Esempio: il **Fattoriale** di un numero, **n! = n x (n-1)!**

    4! = 4x3! = 4x3x2! = 4x3x2x1! = 4x3x2x1x0! =

    = 4x3x2x1x1= 24

Un esempio pratico di ricorsione è quello per cui, guardando un programma televisivo, assistiamo a una scena nella quale c'è un televisore che trasmette lo stesso programma che stiamo vedendo e il televisore viene ripreso dalla telecamera: nel televisore si vede un televisore che ha all'interno un televisore… e così via.



Le funzioni ricorsive

FUNZIONE RICORSIVA

In informatica una funzione si dice ricorsiva quando al suo interno è presente una chiamata a se stessa.

- Secondo tale definizione però sembra che l'elaborazione non termina mai
- In realtà lo schema di una funzione ricorsiva è costituito da due elementi distinti:
  - La condizione di terminazione
  - Il passo ricorsivo

Struttura di una funzione ricorsiva

- Una funzione ricorsiva presenta la seguente struttura

```
if (<condizione di terminazione>)
  then
    istruzione finale                    // fine ricorsione
  else
    passo di avvicinamento               // passo ricorsivo
    chiamata ricorsiva a se stessa
```

Esempio: fattoriale di un numero

- Il fattoriale di un numero è: n! = n x (n-1)!
- Cioè il fattoriale è definito mediante il fattoriale stesso

```
0  function fatt(n:integer):integer;
1  var
2    m:integer;
3  begin
4    if n=0                    // test che verifica la condizione di uscita
5      then
6        fatt:=1;              // fine ricorsione
7      else
8        begin
9          m:=(n-1);           // passo di avvicinamento
10         fatt:=n*fatt(m);    // chiamata ricorsiva
11       end;
12 end;
```

Osservazione: l'istruzione 10 può essere sostituita con `fatt:=n*fatt(n-1);`

Trace table

- Supponiamo di chiamare la funzione fatt dal main program

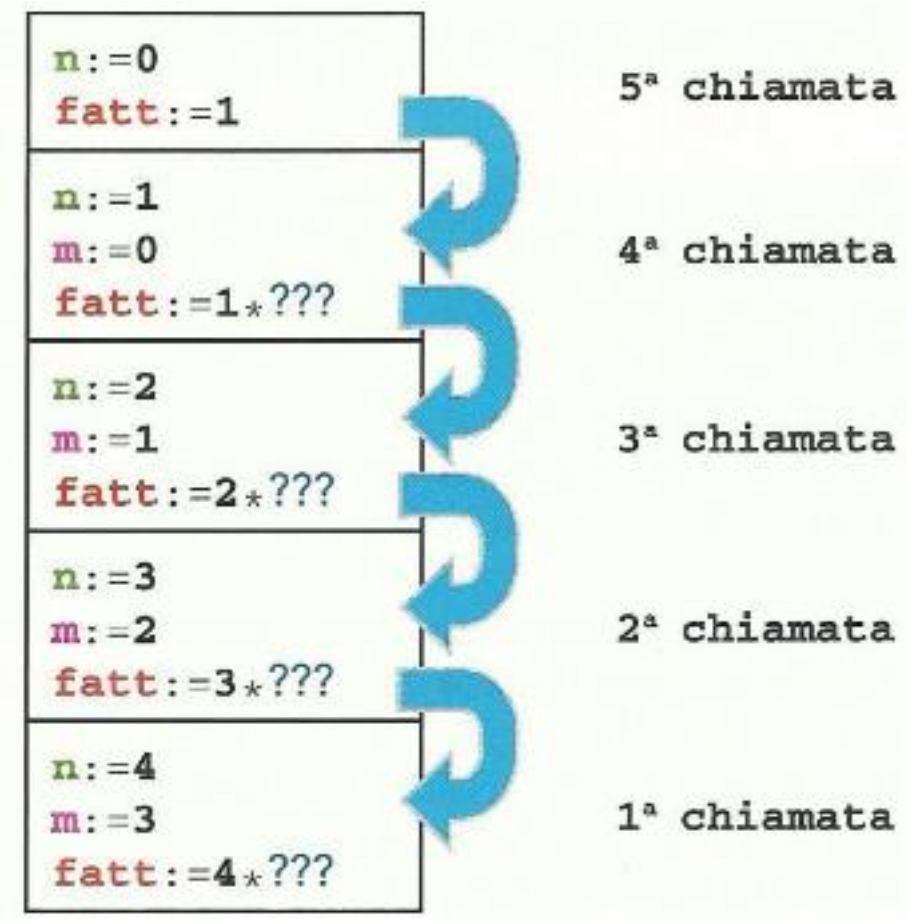
```
writeln('il fattoriale di 4 è', fatt(4) )
```

- La trace table è la seguente

	Istruzioni		chiamata ricorsiva	n	m	fatt
	chiamata della funzione dal principale		1	4		fatt(4)
4	test di uscita = 0	FALSO				
9	passo di avvicinamento				3	
10	chiamata ricorsiva		2	3		4*fatt(3)
4	test di uscita = 0	FALSO				
9	passo di avvicinamento				2	
10	chiamata ricorsiva		3	2		3*fatt(2)
4	test di uscita = 0	FALSO				
9	passo di avvicinamento				1	
10	chiamata ricorsiva		4	1		2*fatt(1)
4	test di uscita = 0	FALSO				
9	passo di avvicinamento				0	
10	chiamata ricorsiva		5	0		1*fatt(0)
4	test di uscita = 0	VERO				
6	< uscita dalla funzione >					1

Esecuzione della funzione ricorsiva

- Dopo cinque chiamate ricorsive si ha:



Esecuzione della funzione ricorsiva

- Alla quinta iterazione si giunge alla condizione di terminazione
- Ogni funzione restituisce il valore calcolato alla funzione chiamante procedendo a ritroso fino ad arrivare al main program e dunque termina l'esecuzione della funzione ricorsiva

La generazione del risultato avviene quindi solo alla fine dell'eliminazione ("spilaggio") dei record di attivazione dalla pila, in quanto il risultato utile inizia a essere generato solamente quando viene eseguito il ramo non ricorsivo. Le chiamate ricorsive decompongono via via il problema, ma non calcolano alcunché: si inizia a sintetizzare il risultato solo dopo che si sono aperte tutte le chiamate, quindi "a ritroso", mentre le chiamate si chiudono con l'istruzione di ritorno.

Altra soluzione

- Il fattoriale poteva essere calcolato anche implementando un codice che più si avvicina alla formulazione matematica

```
function fatt(n:integer):integer;
if (n=0)
  then fatt:=1                    // fine ricorsione
  else fatt:=(n*fatt(n-1));  // avvicinamento e chiamata ricorsiva
```

$$\begin{cases} n! \leftarrow 1 & \text{per } n = 0 \\ n! \leftarrow n * (n-1)! & \text{per } n > 0 \end{cases}$$

Esercizi 1-4, pag. 40

Qual è il risultato dei seguenti codici con la seguente chiamata `ali(5)`?

1
```
function ali(num:integer):integer;
begin
 write(num);
 if(num<1)
  then
   ali:=num
  else
   ali:=ali(num-1);
end;
```

2
```
function ali(num:integer):integer;
begin
 write(num);
 if(num<2)
  then
   ali:=num
  else
   ali:=ali(num-2)*2;
end;
```

3
```
function ali(num:integer):integer;
begin
 write(num);
 if(num<2)
  then
   ali:=num
  else
   ali:=ali(num-1)+ali(num-2);
end;
```

4
```
function ali(num:integer):integer;
begin
 if(num<2)
  then
   write(num)
  else
   ali:=ali(num-1)+ali(num-2);
end;
```

Problemi 1-7, pag. 40

1. Leggi un numero *n* e calcola la somma dei primi *n* interi utilizzando una funzione ricorsiva.
2. Leggi un numero *n* e inverti le cifre utilizzando una funzione ricorsiva.
3. Leggi un numero *n* e calcola la potenza *n*-esima utilizzando una funzione ricorsiva.
4. Esegui il calcolo del coefficiente binomiale e rappresentalo per $n = 10$ in forma di piramide come riportato nel testo cinese del 1303 di Chu Shih-Chien, ma ricordato come Triangolo di Tartaglia o di Pascal.
5. Leggi un numero *n* e visualizza la sequenza di Fibonacci utilizzando una funzione ricorsiva.
6. Leggi una parola lunga al massimo sei caratteri diversi tra loro e visualizzane tutti gli anagrammi. Effettua quindi tutte le permutazioni semplici calcolabili mediante la formula $Pn = n!$
7. Leggi due numeri *n* e *M* ed esegui la moltiplicazione tra due numeri interi utilizzando solo somma, sottrazione, moltiplicazione per 2 e divisione per 2.