Introduzione alle Funzioni

Linguaggi di Programmazione I

Mmatr. 2-3

Esercizi

- Dire cosa viene stampato:

```
char    c = 'A';
double x = 1e+33, y = 0.001;

printf("%d %d %d \n", c == 'a', c == 'b', c != 'c');   0 0 1
printf("%d \n", c == 'A' && c<= 'B' || 'C');           1
printf("%d \n", 1 != !!c == !!!c);                     1
printf("%d \n", x+y > x-y);                            0
```

Esercizi

- Scrivere un programma che prende in input 10 interi in un array e trova i primi 3 massimi spostandoli all'inizio dell'array.
- Scrivete un programma che legge un valore intero *n* e sommi gli interi da *n* a *2n* se *n* è *0* o positivo e da *2n* a *n* altrimenti. Scrivete il codice in due versioni una che utilizzi esclusivamente cicli *for* e l'altra che utilizzi esclusivamente cicli *while*.

I primi tre numeri piu` grandi...

```
#include <stdio.h>
main()
{     int i,j,max,temp, a[10];
      for(i=0; i<=9; i++)
            scanf("%d", &a[i]);
      for(i=0; i<=2; i++)
      {     max = i;
            for(j=i+1; j<= 9; j++)
                  if(a[j]>a[max])
                        max=j;
            temp= a[max];
            a[max]=a[i];
            a[i]= temp;
      }
printf("questi sono gli elementi dell'array con\ni tre
numeri piu` grandi nelle prime tre posizioni:\n");
for(i=0; i<=9; i++)
printf("%d\n", a[i]);
}
```

Esercizio 13 del cap. 4 somma da *n* a *2n* o da *2n* a *n* con "*for*"

```
#include <stdio.h>
main()
{
      int n,i, somma=0;
      scanf("%d", &n);
      if(n>=0)
            for(i=n;i<=2*n; i++)
                  somma=somma+i;
      else
            for(i=2*n;i<=n; i++)
                  somma=somma+i;
      printf("somma = %d\n", somma);
}
```

Esercizio 13 del cap. 4 somma da *n* a *2n* o da *2n* a *n* con "*while*"

```
#include <stdio.h>
main()
{
   int count,start,n, m = 0;
   scanf("%d", &n);
   (n>=0)?(count=n,start=n):(count = -n,start = 2*n);
   while(count-- >= 0)
         m = m + start++;
   printf("m = %d\n", m);
}
```

Somma

start + (start + 1) +…+ (start + n)

Introduzione alle funzioni

Utilità delle funzioni

- realizzano il metodo di programmazione *top-down*
  - –decomposizione del problema in sottoproblemi di piccola dimensione
  - – possibili riutilizzo di codice
  - – riducono la dimensione del codice
  - – facilitano la gestione di programmi estesi

Uso delle funzioni

- Una funzione viene
  - – dichiarata con il *prototipo di funzione*

    si dichiara il **nome della funzione**, il **tipo degli argomenti** ed **il tipo del valore restituito** (*prototipo*)
  - – definita

    si definisce cosa effettua la funzione, gli argomenti ed il valore restituito
  - – utilizzata (chiamata)

    si passano alla funzione gli argomenti e si utilizza il valore restituito

Il prototipo di funzione

- E' stato introdotto dall'ANSI C
- Specifica:
  - – *nome della funzione*
  - – *tipo* degli argomenti
  - – *tipo* del valore restituito
- Forma generale:

tipo nome_funzione (lista dei tipi dei parametri);

  - – Esempio: float moltiplica(float, float);

Il prototipo di funzione

- E' possibile specificare anche gli identificatori dei parametri
  - – Esempio: float moltiplica(float f1, float f2);
  - – Gli identificatori non vengono utilizzati dal compilatore
  - – Servono per la documentazione del programma
- La parola chiave *void* è usata sia per specificare che la funzione non restituisce alcun valore sia per specificare che alla funzione non vengono passati parametri
  - – Esempi: void f (int);

    char f(void);

Il prototipo di funzione

- Permette al compilatore di verificare meglio il codice
- In alcuni casi i parametri passati ad una funzione con un tipo diverso da quello previsto vengono opportunamente convertiti
  - – Esempio:

    Il prototipo double sqrt (double); ci dice che la funzione *sqrt( )* si aspetta in input un parametro di tipo *double* .

    La chiamata sqrt(6); produce il valore corretto perché il compilatore sa dal prototipo che il parametro di sqrt è di tipo *double* e quindi converte il valore 6 di tipo *int* in *double*

La definizione

• Forma generale:

tipo nome_funzione (lista dei tipi e nomi dei parametri)

{

…

}

• Esempio: float moltiplica (float f1, float f2) ← intestazione

```
{
  float prod;
  prod = f1*f2;      } ← corpo
  return (prod);
}
```

Chiamata di una funzione

• Causa l'esecuzione delle istruzioni associate al nome della funzione

• Forma generale: nome_funzione(lista degli argomenti) ;

• Esempio di chiamata ad una funzione:

printf ("valore = %d\n", val);

printf : nome della funzione

("valore = %d\n", val) : argomenti passati tra parentesi

"valore = %d\n" : primo argomento (stringa)

val : secondo argomento (un intero)

Modalità di chiamata di una funzione

- Viene valutata ogni espressione nella lista dei parametri attuali
- Se necessario il valore dell'espressione di un parametro viene convertito al tipo del parametro formale; il valore del parametro attuale viene assegnato al corrispondente parametro formale.
- Viene eseguito il corpo della funzione.
- Se viene eseguita un'istruzione di *return* il controllo ritorna all'ambiente chiamante
- Se l'istruzione *return* contiene un'espressione, il suo valore viene restituito all'ambiente chiamante dopo essere stato convertito, se necessario, al tipo di dato specificato per la funzione
- Se l'istruzione *return non* contiene alcuna espressione allora non viene restituito nessun valore all'ambiente chiamante.

Un esempio: tabella di potenze

```
#define N 7
long power (int, int);
main()
{
    int i,j;
    for (i=1; i<= N; i++)
    {
        for (j=1; j<= N; j++)
            printf ("%9ld\t", power (i , j) );
        printf ("\n");
    }
}
long power (int base, int esponente)
{
    long prod = 1;
    int i;
    for (i = 1; i<= esponente; i++)
        prod *= base;
    return (prod);
}
```

Passaggio di parametri

• *Parametri formali*

– nella definizione

• *Parametri attuali*

– nella chiamata

– corrispondono in numero e in tipo (o tipo compatibile) ai parametri formali

• La funzione viene eseguita

– con i parametri formali che prendono il valore dei parametri attuali

Chiamata per valore

• Ogni parametro viene valutato e il suo valore sostituisce "localmente" quello del corrispondente parametro formale

• Il valore di una variabile passata come parametro ad una funzione non viene modificato nell'*ambiente chiamante*

Esempio di chiamata per valore

```
int comp_sum (int);
main()
{
      int n = 5, sum;
      printf ("%d\n", n);      /*stampa 5*/
      sum =comp_sum(n);
      printf ("%d\n", n);      /*stampa 5*/
      printf ("%d\n", sum);    /*stampa 15*/
}
int comp_sum (int val)
{
      int s = 0;
      for ( ; val > 0; --val)
            s += val;
      return s;
}
```

Il valore di val viene decrementato all'interno della funzione *comp_sum*.

Ciò non comporta una modifica del valore di n nella funzione *main*

Esempio di chiamata per valore

```
int comp_sum (int);
main()
{
      int n = 5, sum;
      printf ("%d\n", n);      /*stampa 5*/
      sum =comp_sum(n);
      printf ("%d\n", n);      /*stampa 5*/
      printf ("%d\n", sum);    /*stampa 15*/
}
int comp_sum (int val)
{
      int sum = 0;
      for ( ; val > 0; --val)
            sum += val;
      return sum;
}
```

La variabile sum definita in *comp_sum* è visibile solo in *comp sum*

Funzioni di libreria matematica

• La libreria matematica è parte della libreria standard

• Esempi di funzioni matematiche disponibili nella libreria matematica:
– *double sqrt (double)* radice quadrata
– *double pow (double, double)* elevazione a potenza
– *double sin (double), double cos (double), double tan(double)* funzioni seno, coseno e tangente

• Per usare le funzioni in questa libreria occorre
– includere il file di header *math.h* con #include <math.h>
– compilare con l'opzione -lm (solo se si usano vecchie versioni di UNIX).
– Esempio: gcc -lm prova.c

Input/Output di caratteri

• *getchar*()
– macro definita in *stdio.h*
– legge un carattere da tastiera
– restituisce EOF se l'utente digita CTRL-D
• *putchar*()
– macro definita in *stdio.h*
– scrive un carattere sul video
• EOF è la costante simbolica per *End Of File*
– su molti sistemi -1 è il valore di *End Of File*
– *stdio.h* contiene la riga: #define EOF -1
– l'inclusione di *stdio.h* e l'uso di EOF rendono portabile il programma

Esempio dell'uso di getchar e putchar

```
/* Uso di getchar e putchar */
#include <stdio.h>
main()
{
        int c;
        while ( (c = getchar( ) ) != EOF)
                putchar (c);
}
```

Esempio dell'uso di getchar e putchar

```
/* Uso di getchar e putchar */
#include <stdio.h>
main()
{
    int c;
    while ( (c = getchar( ) ) != '\n')
        if (c >= 'A' && c <= 'Z')
                putchar (c + 'a' -'A');
        else
                putchar (c + 'A' -'a');
}
```

Legge fino a INVIO

• Codifiche ASCII
– A = 65... Z = 90
– a = 97 … z = 122
• Esempio di conversione:
– D (68) diventa
68 + 97 - 65 = 100 (d)
– b (98) diventa
98 + 65 - 97 = 66
(B)