# Tutorato: Programmazione 1 - Modulo I

Carmine Spagnuolo

October 14, 2014

Descrivere l'output del seguente programma:

**Listing 1: Welcome by ISISLab**

```
#include <stdio.h>
#define ANSI_COLOR_RED     "\x1b[31m"
#define ANSI_COLOR_GREEN   "\x1b[32m"
#define ANSI_COLOR_RESET   "\x1b[0m"
#define P(x) printf(x)
#define $ P("\a")
#define WELCOME_BY P(ANSI_COLOR_RED "WELCOME STUDENTS by" ANSI_COLOR_RESET
    )
#define ISISLab P(ANSI_COLOR_GREEN " ISISLab" ANSI_COLOR_RESET "\n")
int main()
{
                                WELCOME_BY;
  $;$;$;  $;$;$;$;$;  $;$;$;  $;$;$;$;$;  $;          $;       $;
    $;    $;             $;    $;          $;         $; $;    $;
    $;    $;             $;    $;          $;        $;   $;   $;
    $;    $;$;$;$;$;     $;    $;$;$;$;$;  $;        $;    $;  $;$;$;$;
    $;             $;    $;            $;  $;        $;$;$;$;  $;    $;
    $;             $;    $;            $;  $;        $;    $;  $;    $;
  $;$;$;  $;$;$;$;$;  $;$;$;  $;$;$;$;$;  $;$;$;$;  $;    $;  $;$;$;$;
      ISISLab;
return 0;
}
```

# 1 PROGRAMMA

1. Fondamenti di programmazione
2. Input/Output
3. Espressioni
4. Istruzioni condizionali
5. Cicli
6. Tipi di dati
7. Funzioni
8. Array
9. Durata e visibilità delle variabili

# 2 SVOLGIMENTO DEGLI ESERCIZI

Ogni esercizio è descritto da due sezioni *Descrizione* e *Output*, l'output dei programmi deve essere identico all'output presente nelle tracce (compresi caratteri di newline, spazi, tabulazione etc.).

# 3 COMPILAZIONE DEGLI ESERCIZI

I programmi possono essere compilati in ambienti Unix utilizzando il comando
`gcc nome_file.c -o nome_file_output`
In modo alternativo nel file
`esercizi-programmazione-uno-modulo-I.tar.gz`
sono disponibili tutti gli sketch dei programmi C e tutti gli eseguibili (per confrontare l'output) presenti in questo documento. Per compilare è possibile utilizzare il comando `make`:

- `make all`: compila tutti i sorgenti *.c*;
- `make nome_programma`: compila solo il programma associato al file `nome_programma.c`;
- `make clean`: rimuove tutti i file eseguibili ed oggetto.

Il progetto è dispobibile al link: `http://www.carminespagnuolo.eu/otheractivities/tutorato/programmazione-uno-modulo-I/`.

- `esercizi-programmazione-uno-modulo-I.tar.gz`, sorgenti C e Makefile.
- `esercizi-programmazione-uno-modulo-I-bin-osx.tar.gz`, eseguibili per ambiente OS X.

- `esercizi-programmazione-uno-modulo-I-bin-linux.tar.gz`, eseguibili per ambiente Linux.

Per decomprimere utilizzare il comando `tar -xvf`. Per eseguire il download del pacchetto in modo alternativo utilizzare il comando `wget`.

## 3.1 SORGENTI

Una volta decompresso il pacchetto `esercizi-programmazione-uno-modulo-I.tar.gz` sono disponibili i seguenti file C ed il makefile per la compilazione dei sorgenti:

```
esercizi-programmazione-uno-modulo-I
|------ 00_hello_world_sec_4_0.c
|------ 01_song_hello_world_sec_4_1.c
|------ 02_simple_swap_two_numbers_sec_4_2.c
|------ 03_input_simple_swap_two_numbers_sec_4_3.c
|------ 04_input_real_swap_two_integer_numbers_sec_4_4.c
|------ 04_input_real_swap_two_integer_numbers_sec_4_4_geek.c
|------ 05_ascii_value_sec_4_5.c
|------ 06_calcolatrice_sec_4_6.c
|------ 07_area_sec_4_7.c
|------ 08_espressioni_sec_4_8.c
|------ 09_simple_string_sec_4_9.c
|------ 10_somma_N_numeri_sec_4_10.c
|------ 11_intervallo_a_b_sec_4_11.c
|------ 12_potenze_del_due_sec_4_12.c
|------ 13_equazioni_di_primo_grado_sec_4_13.c
|------ 14_stampa_mese_sec_4_14.c
|------ 15_operazioni_da_valori_in_input_somma_sec_4_15.c
|------ 16_operazioni_da_valori_in_input_media_sec_4_16.c
|------ 17_operazioni_da_valori_in_input_massimo_sec_4_17.c
|------ 18_operazioni_da_valori_in_input_read_command_sec_4.18.c
|------ 19_operazioni_da_valori_in_input_switch_sec_4_19.c
|------ 20_conversione_binario_decimale_sec_4_20.c
|------ 21_conversione_binario_decimale_sec_4_21.c
|------ 22_inversione_di_cifre_sec_4_22.c
|------ 23_codifica_numero_sec_4_23.c
|------ 24_rubrica_telefonica_sec_4_24.c
|------ 25_fattoriale_iterativo_sec_4_25.c
|------ 26_fattoriale_iterativo_funzione_fatt_sec_4_26.c
|------ 27_fattoriale_ricorsivo_sec_4_27.c
|------ 28_quadrati_sec_4_28.c
|------ 29_min_max_avg_array_sec_4_29.c
|------ 30_numeri_di_fibonacci_sec_4_30.c
|------ 31_ricerca_binaria_sec_4_31.c
|------ 32_ricerca_binaria_2_sec_4_32.c
|------ 33_ricerca_binaria_ricorsiva_sec_4_33.c
|------ 34_barra_di_progresso_sec_4_34.c
|------ 35_gioco_impiccato_it_dic.txt
|------ 35_gioco_impiccato_sec_4_35.c
|------ 36_space_invaders_sec_4_36.c
|------ isislab.c
 ------makefile
```

# 4 ESERCIZI

## 4.0 HELLO WORLD

- *Descrizione:* scrivere un programma in linguaggio C che visualizzi sullo standard output la stringa "Hello World".
- *Output:*

```
Hello World
```

## 4.1 SONG HELLO WORLD

- *Descrizione:* modificare il programma in sezione 4.0 in modo da produrre a video 3 stringhe "Hello World" intervallate da un beep di sistema.
- *Output:*

```
Hello World
Hello World
Hello World
```

- *Suggerimento:* [Alarm (Beep, Bell)] corrisponde al beep di sistema (suono) Google Key "Escape sequences in C".

## 4.2 SIMPLE SWAP TWO NUMBERS

- *Descrizione:* scrivere un programma in linguaggio C che scambia il valore di due variabili intere $x = 10$ e $y = 9$.
- *Output:*

```
x:10    y:9    <->    x:9    y:10
```

- *Suggerimento:* utilizzare una variabile di appoggio `int tmp` e il carattere '\t' per la formattazione dell'output.

## 4.3 INPUT SIMPLE SWAP TWO NUMBERS

- *Descrizione:* modificare il programma 4.2 in modo da leggere i due interi da scambiare da standard input.
- *Output:*

```
Inserire due valori interi x e y:
10
9
x:10 y:9 <->x:9 y:10
```

- *Suggerimento:* utilizzare la funzione `scanf`.

### 4.4 INPUT REAL SWAP TWO INTEGER NUMBERS

- *Descrizione:* modificare il programma in sezione 4.3 in modo da non utilizzare la variabile temporanea.
- *Output:*

```
Inserire due valori interi x e y:
10
9
x:10 y:9 <->x:9 y:10
```

- *Suggerimento:* le variabili *x* e *y* sono intere, utilizzare le operazioni aritmetiche di somma e sottrazione.
- *Geek test:* in modo alternativo provare anche ad utilizzare l'operatore di Bitwise XOR (^).

### 4.5 ASCII VALUE

- *Descrizione:* scrivere un programma in linguaggio C che chiede all'utente di inserire un carattere *c* e visualizza a video il corrispondente valore ASCII.
- *Output:*

```
Enter a character: p
ASCII value of p = 112
```

- *Suggerimento:* non utilizzare la tabella ASCII ma sfruttare gli operatori di formattazione della funzione `printf`.

### 4.6 CALCOLATRICE

- *Descrizione:* scrivere un programma in linguaggio C che calcola la somma, la differenza, il prodotto e la divisione di due numeri interi letti da standard input.
- *Output:*

```
Programma: Calcolatrice

Inserisci il primo numero: 3
Inserisci il secondo numero: 7

Numeri inseriti 3.000 e 7.000
La somma e' 10.000
```

```
La differenza e' -4.000
Il prodotto e' 21.000
La divisione e' 0.429
```

- *Suggerimento:* utilizzare variabili di tipo `float` formattando l'output in modo da visualizzare tre cifre dopo la virgola.

## 4.7 AREA

- *Descrizione:* scrivere un programma in linguaggio C che legge da standard input un numero reale *L* e calcola e visualizza a video: area del quadrato di lato *L*; area cerchio di raggio *L*; area triangolo equilatero di lato *L*. Utilizzare la direttiva al precompilatore `#define rad3_4() (sqrt(3) / 4)`.
- *Output:*

```
Inserire il valore reale L=4
Area quadrato di lato 4.0 16.000000
Area cerchio di lato 4.0 50.265484
Area triangolo equilatero di lato 4.0 6.928203
```

- *Suggerimento:* utilizzare la funzione `pow` e la constante `M_PI` presenti nella libreria `math.h`.
- *Geek test:* spiegare cosa significa `#define rad3_4() (sqrt(3) / 4)`.

## 4.8 ESPRESSIONI

- *Descrizione:* dato il seguente programma in linguaggio C descrivere l'output senza eseguire il codice.

**Listing 2: 4.10**

```
#include <stdio.h>

int main()
{
  int y;
    printf("%d\n",y);

  y = (10==10);
  printf("%d\n",y);

  y = ((1 == y)+1);
  printf("%d\n",y);

  y = y << 1;
  printf("%d\n",y);
```

```
    return 0;
}
```

- *Suggerimento:* "==" corrisponde all'operatore di uguaglianza, mentre "<<" corrisponde all'operatore Bitwise di shift a sinistra (x=10=1010, x << 2, x=40=101000).

## 4.9 SIMPLE STRING

- *Descrizione:* scrivere un programma in linguaggio C che prende come argomento due stringhe e visualizza a video i primi 5 caratteri della prima stringa seguiti da i primi 3 caratteri della seconda stringa utilizzare la funzione `printf` specificandone il formato.
- *Output:*

```
./string.out Hello World
HelloWor
```

## 4.10 SOMMA N NUMERI

- *Descrizione:* scrivere un programma in linguaggio C che legge da standard input un intero e stampa a video la somma dei primi N numeri naturali, senza utilizzare istruzioni iterative.
- *Output:*

```
Inserire un intero:
10
La somma dei primi 10 numeri interi e': 55
```

- *Suggerimento:* la somma dei primi N numeri naturali è pari a $\frac{N(N+1)}{2}$.

## 4.11 INTERVALLO [A,B]

- *Descrizione:* scrivere un programma in linguaggio C che legge da standard input due interi a e b e visualizza a video il numero di interi nell'intervallo [a,b], senza utilizzare istruzioni iterative.
- *Output:*

```
Inserire l'intero a:
1
Inserire l'intero b:
5
Nell'intervallo [1, 5] ci sono 5 interi.
```

### 4.12 POTENZE DEL DUE

- *Descrizione:* scrivere un programma in linguaggio C che legge da standard input un valore intero N e visualizza a video le prime N potenze del 2.

- *Output:*

```
Inserire un intero positivo
5
2^0 = 1
2^1 = 2
2^2 = 4
2^3 = 8
2^4 = 16
2^5 = 32
```

### 4.13 EQUAZIONI DI PRIMO GRADO

- *Descrizione:* scrivere un programma in linguaggio C per risolvere equazioni nella forma $ax + b = 0$, i valori di $a$ e $b$ devono essere passati come argomenti da linea di comando. Gestire i casi di equazione determinata, indeterminata e impossibile.

- *Output:*

```
/a.out 1 1
Equazione determinata per x=-1.000000
```

- *Suggerimento:* utilizzare la funzione `atof`: `The C library function double atof(const char *str)converts the string argument str to a floating - point number (type double).`

### 4.14 STAMPA MESE

- *Descrizione:* scrivere un programma in linguaggio C che letto un intero da standard input visualizza a video il corrispondente mese dell'anno, utilizzare il costrutto condizionale di `if-else` annidato oppure il costrutto `switch/case`.

- *Output:*

```
Inserisci il numero del mese: 13
Mese Errato
```

- *Suggerimento:*

```
int a;
...
switch (a) {
case 1:
System.out.println("a==1");
break;
case 2:
.
.
default:
System.out.println("errore");
}
```

## 4.15 Operazioni da valori in input - somma

- *Descrizione:* scrivere un programma in linguaggio C che visualizza a video tre opzioni *Add*, *Sum* e *End*. L'operazione di *Add* consente di inserire un nuovo valore reale, l'opzione *Sum* visualizza a video la somma dei numeri inseriti; *End* termina l'esecuzione del programma.
- *Output:*

```
1 - Add
2 - Sum
3 - End
-------------------->1
Enter a value:
2
*****************
1 - Add
2 - Sum
3 - End
-------------------->2
Sum 2.000000
*****************
```

## 4.16 Operazioni da valori in input - media

- *Descrizione:* modificare il programma in sezione 4.15 in modo da consentire anche la stampa della media dei valori inseriti.
- *Output:*

```
1 - Add
```

```
2 - Sum
3 - Avg
4 - End
-------------------->1
Enter a value:
2
*****************
1 - Add
2 - Sum
3 - Avg
4 - End
-------------------->2
Sum 2.000000
*****************
1 - Add
2 - Sum
3 - Avg
4 - End
-------------------->3
Avg 2.000000
*****************
```

### 4.17 OPERAZIONI DA VALORI IN INPUT - MASSIMO

- *Descrizione:* modificare il programma in sezione 4.16 in modo da consentire anche la stampa del massimo dei valori inseriti. Per il calcolo del valore massimo si utilizzi l'operatore di selezione (operatore ternario):

```
x=<condizione> ? <expr1> : <expr2>

if (condizione) x= expr1;
else  x= expr2;
```

- *Output:*

```
1 - Add
2 - Sum
3 - Avg
4 - Max
5 - End
-------------------->1
Enter a value:
2
*****************
```

```
1 - Add
2 - Sum
3 - Avg
4 - Max
5 - End
-------------------->2
Sum 2.000000
*****************
1 - Add
2 - Sum
3 - Avg
4 - Max
5 - End
-------------------->3
Avg 2.000000
*****************
1 - Add
2 - Sum
3 - Avg
4 - Max
5 - End
-------------------->1
Enter a value:
10000
*****************
1 - Add
2 - Sum
3 - Avg
4 - Max
5 - End
-------------------->4
Max 10000.000000
*****************
```

### 4.18 OPERAZIONI DA VALORI IN INPUT - READ COMMAND

- *Descrizione:* modificare il programma in sezione 4.17 migliorando la lettura del codice utilizzando le funzioni.
- *Suggerimento:* suddividere il programma in funzioni ad esempio è possibile utilizzare una funzione: `void readCommand(int *);` per la stampa del menù e la lettura del comando.

### 4.19 OPERAZIONI DA VALORI IN INPUT - SWITCH/CASE

- *Descrizione:* modificare il programma 4.19 migliorando la lettura del codice utilizzando il costrutto `switch/case`.

### 4.20 CONVERSIONE BINARIO DECIMALE (DIFFICILE)

- *Descrizione:* descrivere il seguente programma per eseguire la conversione da binario a decimale:

**Listing 3: 4.20**

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>

#define BASE 2

int main(int argc, char **argv)
{
  int peso=0;
  int numero=0;
  int i=0;
  for(i=strlen(argv[1])-1; i >= 0;--i)
  {
    int c=argv[1][i] - '0';
    numero+=c*pow(BASE,peso);
    peso++;
  }
  printf("Valore decimale %d\n",numero);

  exit(0);
}
```

- *Output:*

```
./20_binario_decimale 100
Valore decimale 4
```

### 4.21 CONVERSIONE BINARIO DECIMALE (SEMPLICE)

- *Descrizione:* osservando il codice del programma in sezione 4.21 scrivere un programma in linguaggio C che converte un numero binario in decimale. L'utente inserisce una cifra binaria alla volta specificando all'inizio dell'esecuzione il numero di cifre binarie. Il programma termina visualizzando a video il valore decimale.

- *Output:*

```
Numero di cifre binario: 4

Inserire le cifre binarie partendo da bit meno significativo.
Cifra binaria: 2^0->0
Cifra binaria: 2^1->1
Cifra binaria: 2^2->1
Cifra binaria: 2^3->1

Valore decimale: 14
```

## 4.22 INVERSIONE DI CIFRE

- *Descrizione:* scrivere un programma in linguaggio C per l'inversione di un numero intero positivo. Strutturare il programma in due funzioni:
  - `int leggi_intero()`, legge ripetutamente un valore da standard input finché non viene inserito un intero positivo.
  - `int inverti_cifre(int)`, inverte le cifre di un intero positivo. Es. 1234 -> 4321, 9999 -> 9999. Non utilizzare gli array.

**Listing 4: 4.22**

```
#include <stdio.h>

int leggi_intero();
int inverti_cifre(int);

int main (void){
  int inverso, intero;
  intero = leggi_intero();
  inverso = inverti_cifre(intero);
  printf(" %d <-> %d\n", intero, inverso);
  return 0;
}

int leggi_intero()
{
  //code
}
int inverti_cifre(int)
{
  //code
}
```

- *Output:*

```
Inserire un valore intero positivo:
112233
 112233 <-> 332211
```

## 4.23 CODIFICA NUMERO

- *Descrizione:* scrivere un programma in linguaggio C che chiede all'utente un numero intero a due cifre e visualizza a video la corrispondente parola Inglese.
- *Output:*

```
Enter a two-digit number: 99
You entered the number ninety-nine.
```

- *Suggerimento:* dividere il numero in input in due cifre. Utilizzare un costrutto `switch` per elaborare la prima cifra e uno per la seconda. Ricordare che le cifre da 11 a 19 devono essere trattate differentemente.

## 4.24 RUBRICA TELEFONICA

- *Descrizione:* scrivere un programma in linguaggio C che chiede all'utente un codice del tipo *11A-BN9-87U*: dove ogni parte del codice può essere o una cifra da 0 a 9 o un lettera maiuscola A,B,C,D,E,F,G,H,I,J,K,K,M,N,O,P,R,S,T,U,V,W,X,Y.
  Dato un codice il programma visualizza a video la sua codifica numerica sostituendo le lettere secondo il seguente criterio di conversione: ABC=2, DEF=3, GHI=4, JKL=5, MNO=6, PRS=7, TUV=8, WXY=9. Strutturare il programma secondo il seguente pseudocodice:
  1. Leggi un carattere $c$ finché $c$! ='\n'
  2. Stampa a video la traduzione di $c$
- *Output:*

```
Enter phone number: 11A-BN9-87U
112-269-878
```

- *Suggerimento:* tutti i caratteri differenti dalla specifica devono essere lasciati inalterati. Per la lettura di un carattere alla volta utilizzare la funzione `getchar` mentre per la visualizzazione su standard output di un carattere utilizzare la funzione `putchar`.

## 4.25 FATTORIALE ITERATIVO

- *Descrizione:* scrivere un programma in linguaggio C che calcola il fattoriale di un numero $n$ passato come argomento.
  Il fattoriale $n!$ di un intero $n$ maggiore o uguale a 2 è definito come:
  $n! = n*(n-1)*(n-2)*...*2*1$ assumendo che 1! = 1 e 0! = 1. In modo alterativo il fattoriale può essere definito come $n! = n*(n-1)!$ ovvero il fattoriale di $n$ si ottiene a partire dal fattoriale di $n-1$ moltiplicando per $n$. Utilizzare l'istruzione iterativa `for`.
- *Output:*

```
Il fattoriale del numero 4 e' pari a 24
```

- *Suggerimento:* per leggere il valore di *n* passato come argomento utilizzare la funzione `atoi` disponibile nella libreria `stdlib.h`: `int n=atoi(argv[1]);`.

## 4.26 Fattoriale Iterativo - Funzione fatt

- *Descrizione:* modificare il programma in sezione 4.25 inserendo il calcolo del fattoriale in una funzione `int fatt(int)`.

## 4.27 Fattoriale ricorsivo

- *Descrizione:* modificare il programma in sezione 4.26 modificando la funzione `fatt` per calcolare il fattoriale in modo ricorsivo.

## 4.28 Quadrati

- *Descrizione:* il seguente programma in linguaggio C chiede all'utente un numero *n* e visualizza a video tutti i quadrati compresi tra 1 e *n*; Ad esempio se $n = 100$ il programma visualizza a video 4, 16, 36, 64, 100. Modificare il programma in modo da sostituire il costrutto iterativo `for` con il costrutto `while`.

**Listing 5: 4.28**

```c
#include <stdio.h>

int main(void)
{
  int i, n;

  printf("Enter limit on maximum square: ");
  scanf("%d", &n);

  for (i = 2; i * i <= n; i += 2)
    printf("%d\n", i * i);

  return 0;
}
```

- *Output:*

```
Enter limit on maximum square: 100
4
16
36
64
100
```

```
With while:
4
16
36
64
100
```

## 4.29 MIN MAX AVG ARRAY

- *Descrizione:* completare il seguente codice in linguaggio C inserendo il codice delle funzioni:
  1. `int maxValue(int*,int)`, calcola il massimo di un array di dimensione N;
  2. `int minValue(int*,int)`, calcola il minimo di un array di dimensione N;
  3. `int avgValue(int*,int)`, calcola la media dei valori di un array di dimensione N.

  Testare le funzioni compilando in modalità DEBUG (de-commentare la linea #define DEBUG).

**Listing 6: 4.22**

```
#include<stdio.h>
#include <stdlib.h>
#include<limits.h>

#define N 1000
//Attiva la compilazione in debug, rimuovere il commento per la fase
    di debug
//#define DEBUG

int maxValue(int *,int);
int minValue(int *,int);
float avgValue(int *,int);

int main()
{
  int *array;
  int i=0;
  int n;
#ifdef DEBUG
  n=3;
  array=(int*)malloc(sizeof(int)*n);
  array[0]=0;
  array[1]=1;
  array[2]=2;
#else
  n=N;
  array=(int*)malloc(sizeof(int)*N);
  for(;i<N;i++)
    array[i]=rand()%N;
```

```
#endif
  printf ("MAX = %d\n",maxValue(array,n));
  printf ("MIN = %d\n",minValue(array,n));
  printf ("AVG = %f\n",avgValue(array,n));
return 0;
}
```

- *Output:*

```
MAX = 2
MIN = 0
AVG = 1.000000
```

## 4.30 NUMERI DI FIBONACCI

- *Descrizione:* scrivere un programma in linguaggio C che visualizza a video l' n-esimo numero di Fibonacci.
  L'n-esimo numero di Fibonacci è definito attraverso la seguente definizione ricorsiva:
  1. $F(n) = 0$, se $n = 0$;
  2. $F(n) = 1$, se $n = 1$;
  3. $F(n) = \mathrm{F(n\text{-}1)} + \mathrm{F\ (n\text{-}2)}$, se $n \geq 2$;

  Gestire il caso in cui l'utente inserisce un numero negativo.

- *Output:*

```
Inserire  un intero >= 0: 10
F(10) = 55.
```

- *Geek test:* utilizzare variabili di tipo long.

## 4.31 RICERCA BINARIA

- *Descrizione:* il seguente programma in linguaggio C dato un array di interi prima li ordina utilizzando l'algoritmo *Quick Sort* e successivamente cerca un elemento nell'array utilizzando l'algoritmo di *Ricerca Binaria*. Spiegare il funzionamento della funzione `binarysearch`.

**Listing 7: 4.22**

```
#include<stdio.h>
#include <stdlib.h>


#define N 1000
//Attiva la compilazione in debug, rimuovere il commento per la fase
    di debug
```

```
//#define DEBUG

int binarysearch(int *, int, int);
void quicksort(int *, int, int);
int partition( int *, int, int);

int main()
{
  int *array;
  int i=0;
  int n,x;
#ifdef DEBUG
  n=3;
  array=(int*)malloc(sizeof(int)*n);
  array[0]=0;
  array[1]=1;
  array[2]=2;
  x=2;
#else
  n=N;
  array=(int*)malloc(sizeof(int)*N);
  for(i=1;i<N;i++)
  {
    array[i]=(rand()%N);
  }
  x=rand()%N;
#endif
  quicksort(array,0,n);

  printf("Verifica ordinamento:");
  for(i=0;i<n-1;i++)
    {
      if(array[i] > array[i+1])
        {
          printf(" errore ordinamento\n");
          return -1;
        }
    }
  printf("array ordinato\n");

  int index= binarysearch(array,n,x);

  if( array[index] != x)
    printf("Elemento %d non trovato !\n",x);
  else
    printf("Elemento %d trovato in posizione %d!\n",x,index);

return 0;
}

int binarysearch(int A[], int n, int x) {
    int a = 0, b = n;

  while (a <= b) {
```

```
        int m = (a + b)/2;

  if (A[m] == x) return m;
         else
     if (A[m] < x)
      a = m + 1;
            else
       b = m - 1;
    }
    return -1;
}

void quicksort(int a[], int l, int r)
{
   int j;

   if( l < r )
   {
       j = partition( a, l, r);
       quicksort( a, l, j-1);
       quicksort( a, j+1, r);
   }
}

int partition( int a[], int l, int r)
{
   int pivot, i, j, t;
   pivot = a[l];
   i = l; j = r+1;

   while( 1)
   {
    do ++i; while( a[i] <= pivot && i <= r );
    do --j; while( a[j] > pivot );
    if( i >= j ) break;
    t = a[i]; a[i] = a[j]; a[j] = t;
   }
   t = a[l]; a[l] = a[j]; a[j] = t;
   return j;
}
```

---

- *Output:*

```
Verifica ordinamento: array ordinato
Elemento 230 trovato in posizione 218!
```

- *Geek test:* analizzare anche il codice della funzione `quicksort` corrispondente al noto algoritmo di ordinamento *Quick Sort*.

## 4.32 Ricerca Binaria - 2 - Difficile/Geek

- *Descrizione:* modificare il codice del programma in sezione 4.31 in modo da sostituire il costrutto `while` nella funzione `binarysearch` con il costrutto `for`.
- *Suggerimento:* utilizza l'operatore ternario nella clausola di aggiornamento del ciclo for.

## 4.33 Ricerca Binaria - 3

- *Descrizione:* modificare il codice del programma in sezione 4.31 in modo da rendere la funzione `binarysearch` ricorsiva.
- *Suggerimento:* per essere ricorsiva il prototipo della funzione `binarysearch` dovrebbe essere nella forma `int binarysearch(int*,int,int,int)`.

## 4.34 Barra di progresso

- *Descrizione:* scrivere un programma in linguaggio C che dato uno array bi-dimensionale A[N][N], rappresentante N Processi ognuno composto da N ($N \geq 10$) task, stampa a video una barra di progresso per ogni processo in sequenza. La matrice $A$ è definita da elementi $A[i, j] = s$ dove $s$ è la quantità di tempo del sotto-task $j$ del processo $i$. E' possibile simulare l'esecuzione utilizzando la funzione `sleep(secondi)`. L'output del programma deve essere una matrice nella forma descritta nella sezione *Output*.

**Listing 8: 4.34**

```
#include <stdlib.h>
#include <unistd.h>
#include <math.h>

#define N 10

#define DEBUG
#define MAX_TASK_TIME 2

void printProgressBar(int**,int);

int main()
{
  int **A;

  int i, j,interval;

  int cifren=log10(N)+1;

  A=(int**)malloc(sizeof(int*)*N);
  for(i=0;i<N;i++)
  {
    A[i]=(int*)malloc(sizeof(int)*N);
    for(j=0;j<N;j++)
```

```
    {
#ifdef DEBUG
      A[i][j]=0;
#else
      A[i][j]=rand()%MAX_TASK_TIME;
#endif
    }
  }
  printf("ID");
  for(i=0;i<cifren-2;i++)printf(" ");
  printf("|0%%");

  interval=(N-6);
  for(i=0;i<interval;i++)
    printf("-");
  printf("100%%|\n");

  printProgressBar(A,N);
  return 0;
}

void printProgressBar(int **A,int n)
{

}
```

- *Output:*

```
Progress
ID|0%----100%|
1 |##########|
2 |##########|
3 |##########|
4 |##########|
5 |##########|
6 |##########|
7 |##########|
8 |##########|
9 |#######
```

- *Geek test:* analizzare il codice del listato 1 e provare a stampare le righe della matrice alternando il colore rosso e verde. Analogamente provare ad alternare il colore dei simboli di #.

## 4.35 GIOCO DELL'IMPICCATO

- *Descrizione:* scrivere un programma in linguaggio C che consente all'utente di giocare al "Gioco dell'impiccato". L'obiettivo del gioco è indovinare una certa parola inserendo

il numero minore di caratteri. Il programma visualizza a video la parola nascosta nella forma "parola->_____" e chiede all'utente di inserire un carattere: solo se il carattere inserito fa parte dei caratteri che formano la parola il programma visualizza a video in chiaro il carattere "carattere= a, parola->_a___a". Il programma termina quando l'utente ha indovinato tutti i caratteri che formano la parola.
Il codice nel listato 9 implementa tutte le funzionalità addizionali del gioco, come ad esempio la lettura di una parola da un dizionario di parole (disponibile nei sorgenti). A partire dal codice disponibile implementare il ciclo principale del gioco.

**Listing 9: 4.35**

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX_WORD_LENGHT 300
#define LINE_NUM_WORDS 60454
#define FILE_NAME "35_gioco_impiccato_it_dic.txt"
#define DEBUG

void getWord(char *,int);

int main()
{
  int i;
  char word[MAX_WORD_LENGHT];
  char game_segreto[MAX_WORD_LENGHT],c,n;
  int num_wor;
  printf("Inserisci il numero della parola [1-%d]:",LINE_NUM_WORDS);
  scanf("%d",&num_wor);
  scanf("%c",&n);
  getWord(word,num_wor);
#ifdef DEBUG
  printf("La parola segreta e':%s\n",word);
#endif
  sprintf(game_segreto,"%s",word);
  printf("Player 2. Devi indovinare la parola segreta!\n");

  //code here


  printf("Indovinato la parola e':%s\n",word);
  return 0;
}

void getWord(char *word,int num)
{
  FILE *fd;
  char buf[MAX_WORD_LENGHT];
  char *res;
  int i;
  fd=fopen(FILE_NAME, "r");
  if( fd==NULL ) {
    perror("Errore in apertura del file");
```

```
      exit(1);
    }
    int selected_index=1;
    while(1) {
      res=fgets(buf, MAX_WORD_LENGHT, fd);
      if(selected_index==num)
      {
          for(i=0;i<strlen(res);i++)
            word[i]=res[i];
          word[i]='\0';
        break;
      }
      if( res==NULL )
        break;
      selected_index++;
    }
    fclose(fd);
  }
```

---

- *Output:*

```
Inserisci il numero della parola [1-60454]:35764
La parola segreta e':programmazione

Player 2. Devi indovinare la parola segreta!
La parola e':_______________
Inserisci un carattere:n
n
La parola e':_____________n_
Inserisci un carattere:pppp
p
La parola e':p____________n_
Inserisci un carattere:p
La parola e':p____________n_
Inserisci un carattere:rrrr


La parola e':p____________n_
Inserisci un carattere:r
La parola e':pr__r_______n_
Inserisci un carattere:r
La parola e':pr__r_______n_
Inserisci un carattere:ooo
o
La parola e':pro_r______on_
Inserisci un carattere:o
La parola e':pro_r______on_
```

```
Inserisci un carattere:ggg
g
La parola e':progr______on_
Inserisci un carattere:g
La parola e':progr______on_
Inserisci un carattere:r
r
La parola e':progr______on_
Inserisci un carattere:a
a
La parola e':progra__a__on_
Inserisci un carattere:m
m
La parola e':programma__on_
Inserisci un carattere:m
m
La parola e':programma__on_
Inserisci un carattere:z
z
La parola e':programmaz_on_
Inserisci un carattere:i
i
La parola e':programmazion_
Inserisci un carattere:eeee
e

Indovinato la parola e':programmazione
```

- *Suggerimento:* per leggere un carattere alla volta all'interno di un ciclo utilizzare due `scanf` consecutive:

  ```
  \scanf("%c",&c); scanf("%c",&n);}.
  ```

  In tal modo è possibile leggere correttamente i caratteri senza errori dovuti al carattere $'\backslash n'$. Provare ad utilizzare solo una `scanf`.

### 4.36 SPACE INVADERS

- *Descrizione:* scrivere un programma in linguaggio C che consente all'utente di giocare a una versione molto molto molto semplificata del gioco "Space Invaders". Il programma consente all'utente di spostare la navicella utilizzando i caratteri "d=destra" e "a=sinistra"; dopo ogni spostamento viene invocata una funzione `check_collision` che verifica la collisione di un raggio laser (virtuale) con un nemico. Il programma utilizza una matrice di caratteri `game_matrix` di dimensione $N \times N$. I nemici sono

identificati dal carattere "*" e sono presenti solo nelle prime N/3 righe della matrice. Nella riga N colonna "pos" della matrice viene posizionata l'astronave identificata dal carattere "∧". Completare il codice nel listato 10 implementando le funzioni:

- `void check_collision(char** game_matrix,int n,int pos);`: verifica la collisione del raggio laser lanciato dall'astronave verso un nemico. Sostituisce nella matrice `game_matrix` il carattere "*" del primo nemico a partire dal basso con il carattere "X".
- `int status_game( game_matrix,int n,);`: verifica la terminazione del gioco. Il gioco termina quando tutti i nemici sono stati eliminati.
- `int status_game( game_matrix,int n,);`: elimina dal gioco tutti i nemici distrutti. Sostituisce nella matrice `game_matrix` il carattere "X" con il carattere " ".

**Listing 10: 4.36**

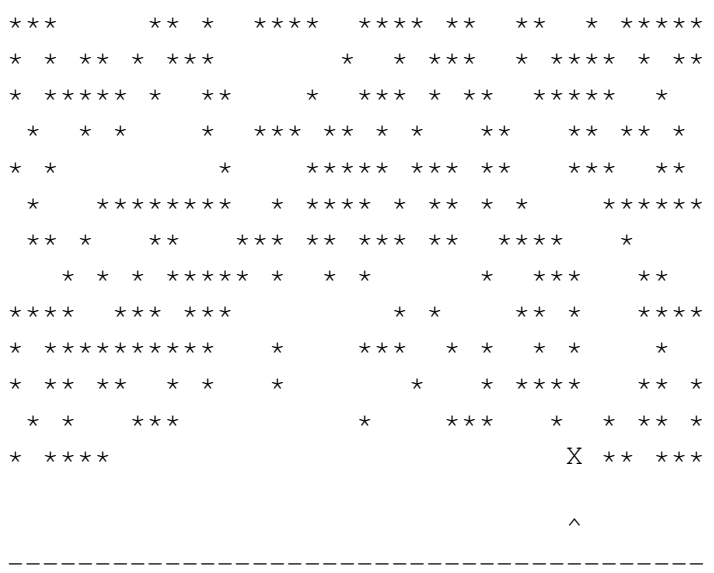
```
#define ANSI_COLOR_RED     "\x1b[31m"
#define ANSI_COLOR_GREEN   "\x1b[32m"
#define ANSI_COLOR_YELLOW  "\x1b[33m"
#define ANSI_COLOR_RESET   "\x1b[0m"
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <termios.h>
//#define DEBUG
#define N 40
void init_matrix(char**,int);
void check_collision(char**,int,int);
int status_game(char**,int);
void clear_collision(char**,int,int);
void drawMatrix(char **,int,int);

char getch();
int main()
{
  int i,j;
  char c,n;
  char **game_matrix;
  int pos=N/2;
  game_matrix=(char**)malloc(sizeof(char*)*N);
  for(i=0;i<N;i++)
    game_matrix[i]=(char*)malloc(sizeof(char)*N);

  init_matrix(game_matrix,N);
  system("clear");
  while(1)
  {
    if(status_game(game_matrix,N))
    {
      system("clear");
```

```
      printf("******HAI VINTO!******\n\n\n**FINE DEL TUTORATO!**\n\n\
          n\n");
      exit(0);
    }
    drawMatrix(game_matrix,N,pos);
    for(i=0;i<N;i++)printf("-");printf("\n");
    do
    {
      c=getch();
      n=getch();
      fflush(stdin);

    }while(c!='a' && c!='d');

    game_matrix[N-1][pos]=' ';

    clear_collision(game_matrix,N,pos);
    if(c=='a') pos--;
     else pos++;

    if(pos>N-1) pos--;
      else if(pos<0) pos++;

    check_collision(game_matrix,N,pos);
    system("clear");
  }
  exit(0);
}
int status_game(char**game_matrix,int n )
{
  //code here
}
void check_collision(char**game_matrix,int n ,int pos)
{
  //code here
}
void clear_collision(char**game_matrix,int n ,int pos)
{

  //code here
}
void drawMatrix(char **game_matrix,int n,int pos)
{
  int i,j;
    for(i=0;i<n;i++)
    {
      if(i<n/3 || i>n-3)
      {
        for(j=0;j<n;j++)
        {
          if(i==n-1 && j==pos)  game_matrix[i][j]='^';

          if(game_matrix[i][j]=='*')
```

```
              printf(ANSI_COLOR_GREEN "%c" ANSI_COLOR_RESET,
                  game_matrix[i][j]);
            else
              if(game_matrix[i][j]=='X')
                printf(ANSI_COLOR_RED"%c"ANSI_COLOR_RESET,game_matrix
                    [i][j]);
              else
                if(game_matrix[i][j]=='^')
                  printf(ANSI_COLOR_YELLOW "%c"ANSI_COLOR_RESET,
                      game_matrix[i][j]);
                else
                  printf("%c",game_matrix[i][j]);
        }
        printf("\n");
      }
    }
}
void init_matrix(char **game_matrix,int n)
{

  int i,j;
  for(i=0;i<n;i++)
  {
    for(j=0;j<n;j++)
    {
      if(i<n/3)
      {
        if((rand()%2)==1)
          game_matrix[i][j]='*';
        else
          game_matrix[i][j]=' ';
      }else
          {
            game_matrix[i][j]=' ';
          }
    }
  }
}
char getch() {
        char buf = 0;
        struct termios old = {0};
        if (tcgetattr(0, &old) < 0)
                perror("tcsetattr()");
        old.c_lflag &= ~ICANON;
        old.c_lflag &= ~ECHO;
        old.c_cc[VMIN] = 1;
        old.c_cc[VTIME] = 0;
        if (tcsetattr(0, TCSANOW, &old) < 0)
                perror("tcsetattr ICANON");
        if (read(0, &buf, 1) < 0)
                perror ("read()");
        old.c_lflag |= ICANON;
        old.c_lflag |= ECHO;
        if (tcsetattr(0, TCSADRAIN, &old) < 0)
```

```
                perror ("tcsetattr ~ICANON");
        return (buf);
}
```

- *Output:*



- *Suggerimento:* prima di iniziare ad implementare le funzioni provare ad eseguire il programma (eseguibile) `36_space_invaders` disponibile nel pacchetto.
- *Geek test:* provare ad utilizzare una matrice di dimensione $\frac{N}{3} \times N$ anziché una di dimensione $N \times N$. Inoltre provare ad impostar il colore dei caratteri: "*" a verde; "X" rosso e "∧" giallo.