# Programmazione • Appello d'esame del 18 giugno 2010 • Compito B

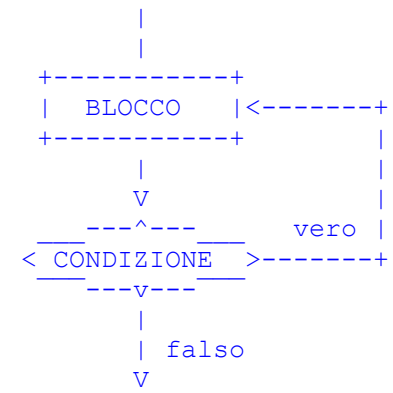
## PARTE 0 (propedeutica) ——————————————————————

### Esercizio 0.1
Descrivere il costrutto DOWHILE (sintassi e funzionamento) e darne il diagramma di flusso.

Il costrutto DOWHILE serve per eseguire una serie di istruzioni definita nel blocco almeno una volta e per ripeterla finché è verificata la condizione indicata dopo while. La sintassi è:

```
do {
  BLOCCO
} while (CONDIZIONE);
```

e il diagramma di flusso corrispondente è




### Esercizio 0.2
Implementare la funzione
`int MaxArray(int a[], int n);`
che ritorna il valore massimo di un array di lunghezza n.

```
int MaxArray (int a[], int n) {
  int max=0,i;
  for(i=0; i<n; i++)
    if(a[i] > a[max])
      max=i;
    return a[max];
}
```

### Esercizio 0.3
Dare un programma che calcoli la somma dei numeri dispari da 1 a 15.

```
main () {
  int somma=0, i;
  for(i=1; i<16; i+=2)
    somma+=i;
  printf("%d\n",somma);
}
```

## PARTE 1 (fondamentale) ——————————————————————

**Esercizio 1.1**
Scrivere l'output del seguente programma C. (Indicare con '_' underscore gli eventuali spazi e con '<n>' la riga completamente vuota)

```
int main() {

  int a=5, b=0, c=0;
  do { printf("%d ", a); a-=3; } while (a+=2);
  printf("\n");
  a=(b=1)+(c=2);
  printf("%d\n",a);

  char d;
  for (d='B'+2; d<'H'; d++)
    printf("%c ",d);
  printf("\n");

  int i;
char str[] =
{'f', 'o', 'r', 'z', 'a', '\0', 'a', 'z', 'z', 'u', 'r', 'r', 'i', '\0'};
  printf("%s\n",str);
  printf("%s\n",str+strlen(str)+1);

  str[strlen(str)]='-';
  printf("%s\n",str);

}
```

<soluzione esercizio 1.1>

**Esercizio 1.2**
Dare l'output della seguente funzione "Add5".

```
void Add5a(int x) {
  x+=5;
}

int Add5b(int x) {
  return x+5;
}

void Add5c(int *x) {
  *x += 5;
}

void Add5() {
  int x=3;
  Add5a(x);
  printf("%d ",x);
  x=Add5b(x);
  printf("%d ",x);
  Add5c(&x);
  printf("%d\n",x);
}
```

3_8_13
:

Nella soluzione il simbolo  indica la posizione finale del cursore (N.d.R.).

**Esercizio 1.3**
Descrivere l'algoritmo BUBBLESORT e le due principali ottimizzazioni che si possono fare sull'algoritmo. In tutti i casi fare esempi chiari e commentarne i passi.

L'algoritmo BUBBLESORT è un algoritmo di ordinamento che, partendo da un estremo di un array, confronta ogni coppia di valori consecutiva e li scambia qualora non fossero nell'ordine desiderato. Tale operazione ha l'effetto di spostare ad ogni ciclo il massimo (o il minimo, a seconda dell'implementazione) in un estremo. Il numero massimo di cicli da effettuare è pari al numero di elementi da ordinare. Esempio:

```
cicli     4  7  6  5  2    coppie scambiate
    1)    4  6  5  2  7    (7,6) (7,5) (7,2)
    2)    4  5  2  6  7    (6,5) (6,2)
    3)    4  2  5  6  7    (5,2)
    4)    2  4  5  6  7    (4,2)
```

Si può ottimizzare l'algoritmo in due modi;
1) poiché è garantito che alla fine di ogni ciclo l'estremo consiste nel valore desiderato, esso può essere escluso nei cicli successivi, riducendo il numero totale di operazioni eseguite;
2) si può introdurre un parametro che conti il numero di scambi effettuati in ogni ciclo e, qualora alla fine di un ciclo esso risulti pari a zero, non esegua ulteriori cicli, poiché gli elementi sono già ordinati. Per esempio,

```
1° ciclo |   2   7   3   5   9
         |         X
         |   2   3   7   5   9
         |             X
         |   2   3   5   7   9    2 scambi

2° ciclo |   2   3   5   7   9    0 scambi, gli elementi sono già ordinati
```

Poiché l'algoritmo controlla al massimo n-1 coppie per ogni ciclo ed effettua al massimo n cicli, la sua complessità nel caso medio è $O(n^2)$ ed è dunque meno efficiente di altri algoritmi di ordinamento.

**Esercizio 1.4**
Descrivere le funzionalità (cosa calcola) e discutere le assunzioni che devono essere verificate sui parametri delle seguenti funzioni. Esemplificare dettagliatamente la loro invocazione.

**A)**
```
int A(int a[], int n) {
  if (n==0) return 0;
  if (n==1) return a[0];
  if (n%2==1) return A(a,n/2)+a[n/2]+A(a+n/2+1,n/2);
  return A(a,n/2)+A(a+n/2,n/2);
}
```

**B)**
```
void B(float m[N][N], float a[N], float b[N]) {
  int i, j;
  for (i=0; i<N; i++) {
   b[i]=0;
   for (j=0; j<N; j++)
    b[i]+=a[j]*m[i][j];
  }
}
```

**A)** La funzione A restituisce la somma dei valori contenuti in un array di interi, di lunghezza n. Se l'array è vuoto, ritorna 0; se contiene un solo elemento, restituisce il suo valore; se contiene un numero pari di elementi

restituisce la somma delle due metà (calcolate ricorsivamente); se ne contiene un numero dispari restituisce la somma del valore centrale e delle somme dei valori precedenti e successivi (anch'essi calcolati ricorsivamente). Il parametro n deve essere pari al numero di elementi nell'array, contati a partire da 1. Un esempio di invocazione è:

```
#include <stdio.h>
main() {
  int a[10]={1,12,1,14,20,21,18,9,14,7};
  printf("La somma è %d.\n",A(a,10));
}
```

**B)** La funzione B, dati una matrice quadrata m sotto forma di array N-dimensionale e due vettori a, b con N coordinate, sostituisce a b il vettore m*a, ovvero il prodotto righe per colonne di m per a. Si assume che N sia una costante intera opportunamente definita nell'header. Un esempio di invocazione è:

```
#include <stdio.h>
#define N 3
main () {
  float mat[N][N]={0.866, 0, -0.5, 0, 1, 0, 0.5, 0, 0.866};
  float a[N]={45.4081, 11.8870, 733940.375};
  float b[N]={66.114, 97.118, 111.33};
  B(mat,a,b);
  int i;
  for(i=0; i<N; i++)
    printf("%f\n",b[i]);
}
```

**Esercizio 1.5**
Sia data la seguente definizione di struttura per il punto tridimensionale:

```
typedef struct {
  float x;
  float y;
  float z;
} Punto3D;
```

Si diano le seguenti funzioni:

```
// distanza euclidea tra due punti
float Distanza( Punto3D p1, Punto3D p2);

// punto medio di un insieme di n punti in un array
Punto3D PuntoMedio( Punto3D aPunti[], int n);
```

<soluzione esercizio 1.5>

**Esercizio 1.6**
Si descriva la famiglia di algoritmi di ricerca binaria. Quando e perché risultano più efficienti dei semplici algoritmi di ricerca lineare?

<soluzione esercizio 1.6>

**PARTE 2 (avanzata) ——————————————————————**

**Esercizio 2.1**
Si data la seguente definizione di insieme implementato da un array.

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

```
#define MAXSIZE 100

typedef struct {
  int numValori;
  int Valori[MAXSIZE];
} Insieme;
```

Si implementino funzioni:

```
Insieme* Crea();
  // crea un insieme vuoto

Insieme* Crea() {
  Insieme* p=(Insieme*) calloc(1, sizeof(Insieme));
    // calloc inizializza tutti i valori a 0
  return p;
}


void Inserisci(Insieme *s, int v);
  // inserisce il valore v nell'insieme se non è già presente!

void Inserisci(Insieme *s, int v){
  int i;
  for(i=0; i<(s->numValori); i++)
    if(s->Valori[i] == v)
      return;    // arresta la funzione se il valore è già presente
  s->Valori[i] = v;       // qui i == numValori
  s->numValori++;
}


void Svuota(Insieme *s);
  // svuota l'insieme

void Svuota(Insieme *s){
  s->numValori=0;
  /* poiché le funzioni sugli insiemi si limitato ai primi "numValori"
     elementi dell'array Valori, Valori non viene modificato */
}


void StampaValori(Insieme s);
  // stampa i valor dell'insieme a schermo

void StampaValori(Insieme s) {
  int i;
  for(i=0; i<(s.numValori); i++)
    printf("%d ",s.Valori[i]);
    /* stampa tutti i valori fino al numValori-esimo */
  printf("\n");
  return;
}


Insieme* Unione(Insieme s, Insieme q);
  // ritorna un puntatore ad un nuovo insieme unione dei due insiemi s e q
```

```
Insieme* Unione(Insieme s, Insieme q) {
  Insieme* p=Crea();
  int i,k;
  for(i=0; i<(s.numValori); i++) {
    p->Valori[i] = s.Valori[i];   // copia tutti i valori di s
    p->numValori++;
  }   // a questo punto i == numValori
  for(k=0; k<(q.numValori); k++) {
    int j,n=0;
    for(j=0; j<(s.numValori); j++)
      if(s.Valori[j] == q.Valori[k])
        n=1;      // se l'elemento q.Valori[k] è già presente in j, n=1
    if(n==0) {        // se l'elemento non è presente in j, lo aggiunge
      p->Valori[i++] = q.Valori[k];
      p->numValori++;       // qui i == numValori
    }
  }
  return p;
}


Insieme* Intersezione(Insieme s, Insieme q);
  // ritorna un puntatore ad un nuovo insieme intersezione dei due insiemi
s e q

Insieme* Intersezione(Insieme s, Insieme q) {
  Insieme *p=Crea();
  int i,j,k=0;
  for(i=0; i<(s.numValori); i++)
    for(j=0; j<(q.numValori); j++)
      if(s.Valori[i] == q.Valori[j])
        p->Valori[k++] = s.Valori[i];
  /* il doppio ciclo FOR controlla che i valori di s siano presenti
     anche in q: se sì, li aggiunge a *p */
  p->numValori = k;
    // k è stato già maggiorato di 1 dall'ultimo IF valutato positivamente
}


Insieme* Differenza(Insieme s, Insieme q);
  // ritorna un puntatore ad un nuovo insieme differenza dei due insiemi s
e q

// codice della funzione Differenza
```