# Laboratorio di Calcolo per Fisici, quinta esercitazione

Canale Lp-P, Docente: Cristiano De Michele

Lo scopo della quinta esercitazione di laboratorio è di fare ancora pratica con le strutture di controllo del flusso, quali i costrutti di selezione (cioè `if ()..else`) e i cicli. Per quanto riguarda i cicli verranno utilizzati anche i costrutti `do...while ()` e `for ()`. Inoltre si farà uso dei cicli nidificati.

---

► **Prima parte:** Rivediamo anzitutto i costrutti d'iterazione `for (...)` e `do...while ()`. Il costrutto `for (...)` ha la seguente struttura:

```
for (Iini; C; Ifin)
{
  /* inserire qui le istruzioni che verranno eseguite ad ogni ciclo */
}
```

dove `Iini` è un'istruzione che verrà eseguita una sola volta prima dell'inizio del ciclo (ad es. `i=0`, con i variabile intera), `C` è la condizione per cui il ciclo termina se essa è falsa (ad es. `i < 10`) e `Ifin` è un'istruzione che viene eseguita alla fine di ogni ciclo (ad es. `i++`). La condizione `C` viene valutata prima di ogni ciclo. Un esempio quindi di ciclo `for` è il seguente:

```
for (i=0; i < 10; i++)
{
  printf("i=%d\n", i);
}
```

dove si ricorda che la variabile intera `i` va opportunamente dichiarata dopo la parentesi graffa che si trova dopo l'istruzione:

```
int main (void)
```

La struttura del costrutto `do...while ()` è invece la seguente:

```
do
  {
    /* inserire qui le istruzioni che verranno eseguite ad ogni ciclo */
  }
while (C);
```

dove la condizione `C` (ad es. `i < 10` con `i` variabile intera) viene valutata alla fine di ogni ciclo. Si noti il necessario `;` alla fine del costrutto. Un esempio quindi di costrutto `do...while ()` è il seguente:

```
i=0;
do
  {
```

```
    printf("i=%d\n", i);
    i++;
  }
while (i < 10);
```

dove si ricorda nuovamente che la variabile intera `i` andrà opportunamente dichiarata. Notare infine che nel caso del ciclo `for ()` come nel caso del costrutto `while ()` visto nella precedente esercitazione la condizione viene verificata prima di ogni ciclo, mentre nel caso del costrutto `do...while ()` la condizione viene sempre valutata alla fine di ogni ciclo.
I costrutti d'iterazione possono essere anche nidificati, ovvero si può inserire un ciclo all'interno di un ciclo come nel seguente esempio:

```
N=1;
do
  {
    printf("N=%d\n", N);
    for (i=0; i < N; i++)
      {
        printf("i=%d\n", i);
      }
    N++;
  }
while (N < 20);
```

dove per ogni valore di `N`, compreso tra 1 e 19, `i` assumerà tutti i valori compresi tra 0 e N-1. Quindi per `N=1`, `i` assumerà il solo valore 0, per `N=2` `i` assumerà i valori 0 e 1, per `N=3` `i` assumerà i valori 0, 1 e 2 e così via. In generale, si ricorda di nuovo che **tutte le variabili che si utilizzano in un codice vanno opportunamente dichiarate** dopo il `main`.
In questa prima parte dell'esercitazione dovrete modificare il programma scritto nella precedente esercitazione che contava il numero di zeri della funzione $\sin(x)/x$ nell'intervallo $[-15, 15]$ e fare in modo che calcoli tramite due cicli nidificati il numero di zeri al variare del numero di sottointervalli $N$. Una possibile implementazione dell'algoritmo per determinare il numero di zeri di una funzione, come discusso nella precedente esercitazione, è il seguente:

```
#include<stdio.h>
#include<math.h>
int main(void)
{
  int i, N=64, nzeri=0;// N qui è il numero di intervalli
  double x, s, fxL, fxR, xL, xR, dx, xmin=-15.0, xmax=15.0;
  xL=xmin;
  dx = (xmax-xmin)/N;
  i = 0;
  while (i < N)
    {
      x = xmin + dx*i;
```

```
      xR = x;
      xL = x + dx;
      if (xL==0.0)
        {
          fxL=1.0;
        }
      else
        {
          fxL=sin(xL)/xL;
        }
      if (xR==0.0)
        {
          fxR=1.0;
        }
      else
        {
          fxR=sin(xR)/xR;
        }
      s = fxL * fxR;
      if (s <= 0)
        {
          nzeri += 1.0;
        }
      i++;
    }

  printf("Numero zeri trovati=%d\n", nzeri);
}
```

Nel seguito potrete modificare questo listato o copiare nella cartella relativa alla presente esercitazione (che chiamerete `EX5`) il codice che voi avete scritto la volta scorsa. Il nome da dare al nuovo programma è `zerovar.c`. In particolare si richiede di fare quanto segue:

1. Far richiedere al programma in input il numero massimo `Nmax` di sottointervalli N.
2. Fare in modo che il programma determini il numero di zeri per un numero variabile di sottointervalli `N` compreso tra 1 e `Nmax` utilizzando due cicli nidificati.
3. Far stampare nel terminale al programma su due colonne `Nmax` e il numero corrispondenti di zeri trovati, ovvero l'output del programma dovrà essere il seguente:

```
      1      {numero di zeri per N=1}
      .
      .
      .
      Nmax   {numero di zeri per N=Nmax}

```

4. Copiare l'output del programma in un file chiamato `zerovar.dat`. Si ricorda che in tale file non deve essere inserito del testo al di fuori delle due colonne con i dati indicati al punto precedente.

5. Utilizzando python graficare il numero di zeri che si ottengono al variare del numero di sottointervalli `N`: qual è il valore minimo di $N$ per cui vengono trovati tutti gli zeri?

6. Per $N = 3, 4, 5, 6$ Quanti zeri trovate? Perché? *Scrivete le risposte a queste ultime due domande su un file chiamato `risposte.txt` nella cartella `EX5`.*

► **Seconda parte:** In C anche l'input come l'output è di norma bufferizzato, questo vuol dire che se una `scanf` richiede l'immissione di un intero e viene immesso invece un numero con la virgola, dei caratteri possano rimanere nel buffer anche dopo che l'istruzione `scanf` ha terminato la sua esecuzione. Per evitare comportamenti inattesi con successive richieste di dati input, è bene svuotare tale buffer di input e a tale scopo si può usare la seguente istruzione C:

```
while (getchar()!='\n');
```

che va inserita dopo l'istruzione `scanf` con cui si sarà richiesto d'inserire una qualche variabile. Inoltre l'istruzione restituisce un valore intero che sarà 1 se il numero inserito è stato correttamente convertito e assegnato alla variabile indicata come argomento e preceduta dal simbolo `&`, oppure 0 se la conversione e l'assegnazione non sono potute avvenire. Ad esempio, se si usa la seguente istruzione:

```
a=1;
printf("Immetti un intero:\n");
res = scanf("%d", &a);
printf("res=%d a=%d\n", res, a);
```

dove `a` e `res` sono due variabili intere che vanno opportunamente dichiarate, immettendo alla richiesta del calcolatore i caratteri `a11`, la `printf` produrrà il seguente output:

```
res=0 a=1
```

ovvero non verrà assegnato alcun valore alla variabile `a` poiché la sequenza di caratteri immessa non è un intero. Se invece si immettessero i caratteri `11` otterremmo come output:

```
res=1 a=11
```

Il valore della variabile `res` può quindi essere usato per determinare se i caratteri immessi rappresentano il tipo di dato richiesto o meno.
Torniamo ora alla discussione di questa seconda parte dell'esercitazione. Dato un sottointervallo $[x_L, x_R]$ degli N in cui si è suddiviso l'intervallo $[-15, 15]$, se esso è sufficientemente piccolo da garantire la presenza di un solo zero al suo interno, come miglior stima $x_0$ ed errore $\epsilon$ dello zero possiamo prendere $x_0 = (x_L + x_R)/2$ e $\epsilon = (x_R - x_L)/2$. Modificate il precedente programma in modo che:

1. Il programma richieda l'immissione in input dell'intero `N`, ovvero del numero di sottointervalli in cui dividere l'intervallo $[-15, 15]$.

2. Il buffer di input venga svuotato a seguito dell'immissione di `N`.

3. Il programma richieda nuovamente l'immissione di `N` se quest'ultimo non sia un intero ben formato (utilizzando il valore restituito dall'istruzione `scanf` come discusso sopra) oppure se esso sia minore di 1. *Suggerimento:* per implementare tale controllo dovrete utilizzare un costrutto d'iterazione.

4. Utilizzando $N = 20$ il programma stampi nel terminale su tre colonne il numero dello zero (intero $\geq 1$), la migliore stima di tale zero e l'incertezza di tale stima. L'output del programma dovrà quindi essere del tipo:

```
1  x1  eps1
.
.
.
n  xn  epsn
```

   dove `xn` è l'n-esimo zero e `epsn` la corrispondente incertezza.

Dopo aver fatto girare il programma dovrete:

1. Copiare l'output del programma in un file chiamato `zeroerr.dat`. Si ricorda che in tale file non deve essere inserito del testo al di fuori delle due colonne con i dati indicati, come indicato sopra.

2. Fare un grafico con python del file `zeroerr.dat` includendo anche le barre d'errore per ogni zero. Per fare questo potete sostituire nello script creato nella prima parte le due righe in cui sono presenti istruzione `np.loadtxt` e `plt.plot` con

```
x, y, dy = np.loadtxt('zeroerr.dat', usecols=(0,1,2), unpack=True)
plt.errorbar(x,y, yerr=dy, fmt='.b', capsize=4, ecolor='r', label='zeroerr.dat')
```

   Notare che `ecolor` è il colore delle barre d'errore (in questo caso il colore è `r` cioè rosso), inoltre il valore assegnato a `fmt` è lo stesso che usavate nel comando `plt.plot` per specificare il tipo di linea o i simboli da usare per i punti e il colore da usare. In questo caso il `.` indica che utilizzeremo dei puntini per rappresentare i punti del grafico il cui colore sarà `b` ovvero blu. Il parametro `capsize` invece indica la larghezza della barra orizzontale che compare nelle barre d'errore. Altre informazioni relative all'uso del comando python `plt.errorbar` le potete trovare sul sito `https://matplotlib.org/`. Infine ricordatevi di rinominare opportunamente le label degli assi cartesiani e inserite una legenda con il comando:

```
plt.legend()
```