Capitolo 2

Lezione 5 marzo 2019
Esercitazione 1

2.1 Modalità di consegna e valutazione esercitazioni

Ciascuna esercitazione avrà come obiettivo l'implementazione di alcune funzioni C, una per ciascun esercizio. Saranno utilizzati strumenti automatici sia per la correzione che per l'individuazione di casi di plagio, quindi è importante che le modalità di consegna indicate vengano rispettate scrupolosamente per evitare che l'esercitazione venga considerata come non svolta.

Valutazione Ciascuna funzione consegnata sarà eseguita per diversi valori dei parametri, non noti prima della consegna. Una funzione che ritorni sempre il risultato atteso sarà valutata 1 punto mentre negli altri casi (compilazione con errori, almeno un risultato non corretto) la funzione sarà valutata 0 punti. Durante il semestre saranno assegnati 15 esercizi, suddivisi in 5 esercitazioni. Per poter consegnare il progetto finale e sostenere l'esame orale è necessario aver raggiunto almeno 9 punti (60%).

Consegna Le funzioni richieste per tutti gli esercizi della stessa esercitazione ed eventuali funzioni ausiliarie realizzate DOVRANNO essere contenute ciascuna in un file sorgente avente nome `esNL.c`, con N sostituito dal numero dell'esercitazione e L dalla lettera maiuscola (A,B,C) che identifica l'esercizio (es: es1A.c). I file sorgente consegnati (solo *.c e non *.h), contenuti in un file zip avente nome uguale al numero di matricola (es: 861234.zip), NON DEVONO contenere la funzione `main()`. Il prototipo di ciascuna funzione DEVE essere obbligatoriamente quello indicato nel testo dell'esercizio. Quando disponibili, si suggerisce di sviluppare le soluzioni partendo dai template forniti, contenenti una funzione main di esempio, una implementazione fittizia (che ritorna un risultato costante ed errato) e gli header relativi alle funzioni richieste.

2.1.1 Date di assegnazione e consegna

Esercitazione 1 5 marzo → 12 marzo

Esercitazione 2 20 marzo → 26 marzo

Esercitazione 3 4 aprile → 10 aprile

Esercitazione 4 18 aprile → 24 aprile

Esercitazione 5 2 maggio → 8 maggio

2.2 Esercitazione 1 - Esercizio A

Scrivere una funzione C che, data una soglia $0 < \varepsilon < 1$, restituisca la somma dei termini iniziali della serie armonica generalizzata ($\alpha = 2$) fino al primo termine (escluso) avente valore minore di ε. La funzione deve quindi calcolare

$$\sum_{n=1}^{k} \frac{1}{n^2} = \frac{1}{1^2} + \frac{1}{2^2} + \cdots + \frac{1}{k^2}$$

per k tale che $\frac{1}{k^2} \geq \varepsilon \wedge \frac{1}{(k+1)^2} < \varepsilon$

Ad esempio, il valore della somma sarà uguale a 1 per $\varepsilon = 0.5$, mentre sarà uguale a 1.25 per $\varepsilon = 0.2$.

Consegna Il prototipo della funzione **DEVE** essere

```
double serie_armonica(double epsilon)
```

Soluzione

```
#include "es1A.h"

double serie_armonica(double epsilon) {
  int i=1;
  double risultato = 0.0;
  double termine = 1.0;
  /* se il termine corrente è minore della soglia mi fermo */
  while ( termine >= epsilon ) {
    /* aggiorno la somma della serie aggiungendo il nuovo termine */
    risultato += termine;
    i++;
    /* calcolo il termine successivo */
    termine = 1 / ((double) i * i);
  }
  return risultato;
}
```

2.3 Esercitazione 1 - Esercizio B

Scrivere una funzione C che dato un numero reale $x \geq 1$ ed una soglia reale $\varepsilon > 0$, restituisca un'approssimazione della radice quadrata di x con un errore inferiore a ε calcolata utilizzando il metodo dicotomico. Questo metodo consiste nel dividere in due parti (per semplicità uguali) un intervallo contenente il valore esatto della radice e ripetendo più volte il procedimento sulla parte che lo contiene fino ad ottenere un intervallo di ampiezza inferiore a ε (non utilizzare la ricorsione). Il punto centrale dell'intervallo quindi è un'approssimazione della radice quadrata di x con la precisione richiesta.
Considerare che per $x \geq 1$, $a \geq 1$, $b \geq 1$ reale positivo

- $1 \leq \sqrt{x} \leq x$
- $a \leq \sqrt{x} \leq b \iff a^2 \leq x \leq b^2$

Ad esempio, per `x=10.0, eps=0.1`, verrebbero esaminati i seguenti intervalli

```
a=1.000    (a+b)/2=5.500    b=10.000
a=1.000    (a+b)/2=3.250    b= 5.500
a=1.000    (a+b)/2=2.125    b= 3.250
a=2.125    (a+b)/2=2.687    b= 3.250
a=2.687    (a+b)/2=2.968    b= 3.250
a=2.968    (a+b)/2=3.109    b= 3.250
a=3.109    (a+b)/2=3.179    b= 3.250
a=3.109    (a+b)/2=3.144    b= 3.179
```

Al termine dell'ultima iterazione abbiamo individuato gli estremi di un intervallo contenente la radice quadrata ed avente ampiezza 0.07, inferiore alla precisione richiesta, quindi la funzione ritornerà il valore (a+b)/2=3.144

Consegna Il prototipo della funzione **DEVE** essere

```
double radice_quadrata(double x, double epsilon)
```

Soluzione

```
#include <math.h>

double radice_quadrata(double x, double epsilon) {
  double a = 1,
         b = x,
         c;
  /* ripeti se l'ampiezza dell'intervallo e' maggiore della percisione richiesta */
  while ( b - a > epsilon ) {
    c = (a+b)/2;
    if (c * c == x) {
       a = b = c;  /* c e' il valore esatto della radice quadrata di x */
    } else if ( c * c > x) {
      b = c; /* la radice quadrata di x e' maggiore di c, cerca in [c,b] */
    } else {
      a = c;  /* la radice quadrata di x e' minore di c, cerca in [a,c] */
    }
  }
  /* ritorna il punto centrale dell'intervallo */
  return (a+b)/2;
}
```

2.4 Esercitazione 1 - Esercizio C

La congettura di Goldbach afferma che ogni numero intero pari maggiore di 2 può essere scritto come somma di due numeri primi (non necessariamente distinti).

Scrivere una funzione C che dato un numero n ritorni:

- 0 se n non è un intero pari maggiore di 2
- il più piccolo numero primo p_1 tale per cui $p_2 = n - p_1$ sia un numero primo

  il più piccolo dei numeri primi che appartenga ad i quali la congettura di Goldbach sia soddisfatta
  Ad esempio:
  $10 = 3 + 7 = 5 + 5 \rightarrow$ return 3
  $12 = 5 + 7 \rightarrow$ return 5
  $14 = 3 + 11 = 7 + 7 \rightarrow$ return 3
  $24 = 17 + 7 = 19 + 5 = 13 + 11 \rightarrow$ return 5

Consegna Il prototipo della funzione **DEVE** essere

```
int goldbach(int n)
```

Soluzione

```
#include "es1C.h"

int is_prime(int n) {
  int i;
  /* (i numeri negativi), 0, 1 ed i numeri pari maggiori di 2 non sono numeri primi */
  if ((n <= 1) || (n % 2 == 0 && n > 2))
    return 0;
  /* verifica se il n e' divisibile per i numeri dispari maggiori o uguali a 3 */
  /* almeno uno dei fattori primi di n deve essere minore o uguale alla sua radice quadrata */
  /* una volta raggiunto il valore della radice quadrata ci possiamo fermare */
  for(i = 3; i <= sqrt(n); i+= 2) {
    if (n % i == 0)
```

```
13        return 0;
14    }
15    return 1;
16  }
17
18  int goldbach(int n) {
19    int pr;
20    /* verifica se il valore del parametro e' valido */
21    if ((n <= 2) || (n % 2 == 1))
22      return 0;
23    /* per ogni numero pr tra 2 ed n verifica se pr e (n-pr) sono due numeri primi */
24    for(pr = 2;pr < n;++pr)
25      if (is_prime(pr) && is_prime(n - pr))
26        /* pr < (n - pr), altrimenti sarebbe (n-pr) stato individuato in una iterazione precedente
      */
27        return pr;
28    /* nel caso la congettura sia falsa... ritorna 0 */
29    return 0;
30  }
```