Esame di Programmazione (mod A) - CdL AIDA
IV Appello Luglio 2021

Giulio Caravagna (gcaravagna@units.it)

1 Istruzioni

L'appello contiene **6** esercizi (`A1`, `A2`, `A3`, `B1`, `B2`, `B3`) da risolvere in 3 ore. Il template si trova su Moodle in formato ZIP, su Moodle dovete carica la vostra soluzione.

Importante. `A1`, `A2` e `A3` sono di *sbarramento* e permettono di raggiungere 18/30. `B1`, `B2` e `B3` valgono fino al raggiungimento del voto massimo di 30/30.

Risoluzione degli esercizi di sbarramento. Usando `repl.it`, risolvete l'esercizio partendo dal file `main.c` e testate il codice con i comandi `make test1`, `make test2` e `make test3`. Prima di ogni test ricordatevi di digitare `make clean`.

2 Esercizi di sbarramento (18 punti)

A1. Si scriva una funzione *iterativa* `myfunction` che prenda in input due interi positivi x ed y, e restitutisca il numero k di numeri primi compresi tra x ed y esclusi. Per esempio, se $x = 2$ ed $y = 5$ allora $k = 1$ essendo $\{3\}$ primo, mentre se $x = 2$ ed $y = 10$ allora $k = \{3\}$ essendo $\{3, 5, 7\}$ primi. Si richiede di usare una funzione ausiliaria `primo` che stabilisca *ricorsivamente* se un numero sia primo, o meno.

Suggerimento: `primo` deve essere ricorsiva, si suggerisce la definizione `int primo(int n, int i)` che restituisce `1` se `n` e' primo secondo `i`. Il calcolo prevede di valutare se i sia un divisore di n, ed eventualmente proseguire per ricorsione su `i-1`. Si ponga attenzione alla condizione di terminazione. Successivamente, si scriva `myfunction` usando `primo`, e generando i numeri nell'intervallo (x, y) con gli estremi come richiesto.

A2. Si scriva un programma C *iterativo* che calcoli, per un dato $n \geq 1$ in input, la successione di interi

$$\begin{cases} a_1 = -1 \\ a_2 = 0 \\ a_n = (2a_{n-2} - a_{n-1} + 1)n \quad \text{con } n \geq 3 \quad \text{se } a_{n-1} > a_{n-2} \\ a_n = (2n + a_{n-1} + 1)a_{n-2} \quad \text{con } n \geq 3 \quad \text{altrimenti.} \end{cases}$$

A3. Si consideri la successione

$$F_1 = 1 \qquad F_n = \frac{n+1}{F_{n-1}} \qquad n \geq 2$$

e la quantita $\boldsymbol{F} = \prod_{i=1}^{N} F_{x_i}$ calcolata a partire da un *array* $\boldsymbol{x}$ *di* N *valori non negativi* x_1, x_2, ..., x_N.

Si scriva un programma C che dato $\boldsymbol{x}$ calcoli $\boldsymbol{F}$ *iterativamente* e ogni F_i *ricorsivamente*. Per esempio, se $\boldsymbol{x} = [1, 2, 0]$ allora $\boldsymbol{F} = F_1 * F_2 * F_1$; F_1, F_2 ed F_1 sono ricorsive, il prodotto iterativo.

Suggerimento: Si consideri che F_n sono numeri con la virgola (`double`).

3 Esercizi opzionali

3.1 Es. B1 (6 punti)

In C, si vogliono definire *liste linkate* che possono memorizza un *array di interi* in ciascun loro elemento; si desidera inoltre permettere agli array di avere dimensione variabile, e.g., una lista potrebbe essere

```
[{1,2,3}] -->  [{9}] -->  [{43, 5}] // array di 3, 1 e 2 elementi
```

Si usi il sequente template per definire la `struct` necessaria ad implementare la lista.

```c
// struttura
struct elemento{

  // dato memorizzato
  ...

  // puntatore
  struct elemento * next;
};

// tipi
typedef struct elemento ElementoDiLista;
typedef ElementoDiLista * ListaDiElementi;
```

Si definiscano, secondo la `struct` sopra definita, le funzioni

```c
ListaDiElementi init(int n)
void print(ListaDiElementi lista)
```

dove *i*) `init` costruisce una lista di un singolo elemento, il quale contiene un array di $n > 0$ elementi , e *ii*) `print` stampa *ricorsivamente* la lista, mostrando ad esempio (dopo la `init`)

```c
ListaDiElementi list = init(4);

print_list(list);
// -> n = 4 | 0, 0, 0, 0,

list->next = init(12);

print_list(list);
// -> n = 4 | 0, 0, 0, 0,
// -> n = 12 | 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, %
```

Suggerimento. Se gli array hanno dimensione variabile, la gestione della memoria deve essere esplicita.

3.2 Es. B2 (3 punti)

Si consideri questo programma C

```c
int x = 6;
int * y = (int *) malloc(sizeof(int));
*y = x;

for(int i = 0; i < *y; i++)
{
  // A
  if(i %2 != 0)
  {
    int x = *y;
    i = x + 1;
    // B
  }
  else
  {
    x = x * *y;
  }
}
```

Si rappresenti la memoria del programma ai punto `A` e `B` per ciascun ciclo di esecuzione del `for`.

3.3 Es. B3 (3 punti)

Ci vengono date queste due funzioni Python

```
def f1(x)
  return x**2 + 2x

def f2(x)
  return x**2 + 2x - 1
```

Si progetti una classe `Obj` abbastanza generale da considerare tutti i polinomi di secondo grado $p(x)$, con un metodo per calcolare $p(x)$ per un x fissato, e si istanzino oggetti `o1` ed `o2` per rappresentare le computazioni fatte da `f1(x)` ed `f2(x)`.