

# Simulazione esame di Programmazione (mod A)

CDL in Intelligenza Artificiale e Data Analytics.

Giulio Caravagna (gcaravagna@units.it)

Simulazione Gennaio 2021

## Istruzioni

L'appello contiene **6** esercizi (A1, A2, A3, B1, B2, B3) da risolvere in 3 ore (tempo massimo).

- Scaricate il file **Appello.zip** da Moodle che contiene un template di 6 cartelle, una per esercizio.
- Le soluzioni devono essere caricate sul portale Moodle alla pagina del corso (in formato **zip**, ricompilando le cartelle che avete scaricato come template, dopo aver aggiunto le vostre soluzioni).

**Importante.** I primi 3 esercizi (A1, A2 e A3) sono considerati di *sbarramento*, le soluzioni devono essere perfette e permettono di raggiungere 18/30 (minimo per superare l'esame). I restanti 3 esercizi (B1, B2 e B3) sono opzionali, e valgono fino al raggiungimento del voto massimo di 30/30.

### Risoluzione degli esercizi di sbarramento.

- ogni esercizio va risolto partendo dall'implementazione disponibile nel template. Del template l'unica cosa che dovete modificare è il file `main.c` (come visto a lezione).
- Per esempio, se lavorate su repl.it per risolvere l'esercizio **A1**:
  - create un nuovo repl per C;
  - trascinate la cartella del template **A1** (drag and drop) nel nuovo repl - compariranno tutte le sotto-cartelle ed i files necessari;
  - risolvete l'esercizio modificando `main.c`;
  - avete dei files di input (es `input_1`, `input_2` ed `input_3`) con cui testare il vostro esercizio. I files sono disponibili all'interno della cartella `input` se li volete ispezionare. Il risultato atteso per ciascun input è memorizzato nella cartella `result`. Potete usare questi files per controllare il vostro calcolo.
  - testate la vostra implementazione usando `make test_1` (oppure `make test_2` etc.) per confrontare il risultato da voi calcolato con quello atteso.
  - quando avete finito scaricate la vostra soluzione e sostituirla al template (potete anche sostituire solamente il file `main.c` del template, perchè il resto del template non deve essere modificato).

## Esercizi di sbarramento

### Es. A1 (6 punti)

Si scriva un programma C che calcola, dato un array  $a$  in input di dimensione  $n$ , il vettore delle frequenze  $f$  tale per cui:

- $f$  ha la stessa dimensione di  $a$ ;
- $f[i] = v$  dove  $v$  è il numero di occorrenze del valore  $a[i]$  in  $a$ , solo se l'occorrenza in posizione  $i$ -esima è la prima del valore  $a[i]$  in  $a$ ; altrimenti  $f[i] = 0$ ;

Esempio di calcolo: per  $a = [1, 2, 1, 4, 1, 2]$  vale  $f = [3, 2, 0, 1, 0, 0]$

### Es. A2 (6 punti)

Si consideri un array  $a$  in input di dimensione  $n$  che memorizza una serie di valori  $a_1, a_2, \dots, a_n$  in modo sequenziale. Si consideri la seguente formula

$$s_n = \sum_{i=1}^n a_i a_{i+1}$$

Per esempio: se  $a = [1, 2, 3, 4, 5, 6]$  allora  $s_n = 70$ ;

Si scriva un programma C che calcola  $s_n$  in modo ricorsivo.

### Es. A3 (6 punti)

Dato un insieme di  $n$  valori memorizzato in un array  $a$ , si definisca una procedura C iterativa che ordina l'array in modo crescente.

Per esempio: dato  $a = [3, 1, 2, 0, 4, 1, 6]$  restituisce  $[0, 1, 1, 2, 3, 4, 6]$

## Esercizi non di sbarramento

### Es. B1 (4 punti)

Si consideri questo frammento di codice C

```
#include <stdio.h>

int f(int x, int a)
{
    int b = a;

    // P*
    if(x <= b) return(a);
    else return(b + f(x - 2, a + 1));
}

int main(void)
{
    int x = 10; // P1 - prima della chiamata di "f"
    int y = f(x, 0); // P2 - dopo la chiamata di "f"

    printf("%d", y);
}
```

Si disegni la memoria del programma al punto P1, ai punti P\* (cioè durante ogni chiamata ricorsiva di `f`), ed al punto P2 (fine della chiamata). Stabilire il valore calcolato dalla chiamata con l'ultima `printf`.

Potete risolvere questo esercizio carta e penna, e caricare uno screenshot del vostro esercizio nella cartella B1, oppure consegnare una copia cartacea della vostra soluzione.

**Nota** nel disegno della memoria si può omettere il disegno di `[f, fun]`.

### Es. B2 (3 punti)

Si consideri questo programma

```

int myFun(int arr[], int l, int r, int x)
{
    if (r >= l) {
        int mid = l + (r - l) / 2;

        if (arr[mid] == x) return(mid);
        if (arr[mid] > x) return(myFun(arr, l, mid - 1, x));
        return(myFun(arr, mid + 1, r, x));
    }

    return -1;
}

int main(void)
{
    int arr[] = { 2, 3, 4, 10, 40 };
    int n = 5;
    int x = 10;
    int result = myFun(arr, 0, n - 1, x);
}

```

Quale formula logica viene calcolata dalla funzione `myFun`? Si fornisca anche un esempio in cui `myFun` restituisce -1 dopo almeno due chiamate ricorsive.

### Es. B3 (5 punti)

Si consideri la seguente classe Python

```

class SistemaPagamento:

    def calcola_pagamento(self, impiegati):

        print('Calcolo pagamento')
        print('=====')

        for impiegato in impiegati:
            print("Payroll:", impiegato.id, ' - ', impiegato.name)
            print("Totale:" impiegato.calcola_pagamento(), "\n")

```

In questo esercizio si deve:

- Creare una classe `Impiegato` che fornisce l'interfaccia di base per gli oggetti per cui il metodo `calcola_pagamento` di `SistemaPagamento` viene definito;
- Definire un metodo per il calcolo dello stipendio che distingue tra diversi tipi di impiegato. Si considerino:
  - Gli amministrativi (`Amministrativi`), che hanno un pagamento costante settimanale;
  - Alcuni impiegati (`ImpiegatiOre`) che sono pagati ad ore, con una tariffa ed un numero di ore lavorate che sono specifici per ciascun impiegato;
  - Gli impiegati della compagnia (`ImpiegatiCommissione`) sono pagati con un salario fisso incrementato di una commissione che varia tra gli impiegati.

Stabilire la corretta gerarchia di ereditarietà per le classi sviluppate, e si crei un semplice programma che usa le suddette classi.