IP a.a. 2023/24 - Esame del 18 Gennaio 2024

Prima di cominciare lo svolgimento leggete attentamente tutto il testo.

Questa prova è organizzata in due sezioni, in cui sono dati alcuni elementi e voi dovete progettare ex novo tutto quello che manca per arrivare a soddisfare le richieste del testo. Per ciascuna sezione, nel file zip del testo trovate una cartella contenente i file da cui dovete partire. Dovete lavorare solo sui file indicati in ciascuna parte. Modificare gli altri file è sbagliato (ovviamente a meno di errata corrige indicata dai docenti).

In questi file dovete realizzare le funzioni richieste, esattamente con la *segnatura* con cui sono indicate: nome, tipo restituito, tipo degli argomenti nell'ordine in cui sono dati. Non è consentito modificare queste informazioni. Potete invece fare quello che volete all'interno del corpo delle funzioni: in particolare, se contengono già una istruzione `return`, questa è stata inserita provvisoriamente per rendere compilabili i file ancora vuoti, e **dovrete modificarla in modo appropriato**.

Potete inoltre realizzare altre funzioni in tutti i casi in cui lo ritenete appropriato. Potete inserirvi tutti gli `#include` che vi servono oltre a quello relativo allo header con le funzioni da implementare. Attenzione però che **usare una funzione di libreria per evitare di scrivere del codice richiesto viene contato come errore** (esempio: se è richiesto di scrivere una funzione di ordinamento, usare la funzione `std::sort()` dal modulo di libreria standard `algorithm` è un errore).

Il programma principale, che esegue il test, è dato in ognuna delle sezioni. Controllate durante l'esecuzione del main, per ogni funzione, quanti sono i test che devono essere superati e controllate l'esito (se non ci sono errori deve essere `true` per tutti).

Compilare con: `g++ -std=c++11 -Wall *.cpp`

NB1: soluzioni particolarmente inefficienti potrebbero non ottenere la valutazione anche se forniscono i risultati attesi. Di contro ci riserviamo di premiare con un bonus soluzioni particolarmente ottimali.
NB2: superare positivamente tutti i test di una funzione non implica soluzione corretta e ottimale (e quindi valutazione massima).

1 Sezione 1 - Array - (max 7.5 punti)

Per questa parte lavorate nella cartella `Sezione1`. Per ogni esercizio dovete scrivere una funzione nel file specificato, fornito nel file zip, completandolo secondo le indicazioni.

Materiale dato

Nel file zip trovate

- Un file `array.h` contenente le definizioni di tipo dato e le intestazioni delle funzioni ⟵ NON MODIFICARE
- un file `mainTestArray.cpp` contenente un main da usare per fare testing ⟵ NON MODIFICARE
- un file `Es1-Funzione1.cpp` ⟵ MODIFICARE IL SUO CONTENUTO
- un file `Es1-Funzione2.cpp` ⟵ MODIFICARE IL SUO CONTENUTO
- un file `Es1-Funzione3.cpp` ⟵ MODIFICARE IL SUO CONTENUTO

1.1 Es1-Funzione1 - (2.5 punti)

```
bool arrayContainsFibonacciSeries(const int* arr, unsigned int size)
```

- INPUT:
  - `int* arr`: un array di interi,
  - `unsigned int size`: la dimensione dell'array
- OUTPUT:
  - TRUE se l'array contiene la sequenza dei primi `size` numeri della successione di Fibonacci, o se l'array è vuoto;
  - FALSE altrimenti.
- Comportamento:

  la funzione prende in input un array di interi e la sua dimensione e verifica che ogni cella dell'array contenga il valore corretto rispetto alla successione di Fibonacci definita come segue:

  $$F_n = F_{n-1} + F_{n-2}$$

  con $F_0 = 0$ e $F_1 = 1$.

  Quindi gli elementi della successione sono: 0, 1, 1, 2, 3, 5, 8, 13, e così via.

- Esempi:

  INPUT => OUTPUT

  `arr`=[] , `size`=0 => TRUE

  `arr`=[0] , `size`=1 => TRUE

  `arr`=[1] , `size`=1 => FALSE

  `arr`=[0, 1, 1, 2, 3, 5, 8] , `size`=7 => TRUE

  `arr`=[8, 1, 1, 2, 3, 5, 0] , `size`=7 => FALSE

1.2 Es1-Funzione2 - (2 punti)

```
void reverseArray(int* arr, unsigned int size)
```

- INPUT:
  - `int* arr`: un array di interi,
  - `unsigned int size`: la dimensione dell'array
- Comportamento:

  la funzione prende in input un array di interi e la sua dimensione e inverte l'ordine dei sui elementi.
- Esempi:

  `int* arr` (prima dell'esecuzione) => `int* arr` (dopo l'esecuzione)

  `arr`=[] => `arr`=[]

  `arr`=[34] => `arr`=[34]

  `arr`=[1,5,7,23] => `arr`=[23,7,5,1]

1.3 Es1-Funzione3 - (3 punti)

```
int findEquilibriumIndex(const int* arr, unsigned int size)
```

- INPUT:
  - `int* arr`: un array di interi,
  - `unsigned int size`: la dimensione dell'array
- OUTPUT: *il primo indice* di equilibrio come definito sotto oppure –1 se non esistente.
- Comportamento:

  La funzione restituisce *il primo indice* di equilibrio (se presente) o –1 se non esiste alcun indice di equilibrio.

  L'indice di equilibrio di un array è l'indice dell'elemento tale che la somma degli elementi precedenti ad esso è uguale alla somma degli elementi successivi ad esso. La somma degli elementi è zero anche nel caso in cui non ci siano elementi precedenti o successivi.

  Ad esempio `arr`=[-7, 1, 5, 2, -4, 3, 0] ha indice di equilibrio pari a 3 (corrispondente all'elemento 2) in quanto -7+1+5 = -4+3+0.
- Esempi:

  INPUT => OUTPUT

  `arr`=[] , `size`=0 => -1

  `arr`=[5] , `size`=1 => 0

  `arr`=[2,2] , `size`=2 => –1

  `arr`=[0,0] , `size`=2 => 0

  `arr`=[1,2,1] , `size`=3 => 1

  `arr`=[1,2,3] , `size`=3 => –1

  `arr`=[0,0,0,1] , `size`=4 => 3

2 Sezione 2 - Liste - (max 8.5 punti)

Per questa parte lavorate nella cartella `Sezione2`. Per ogni esercizio dovete scrivere una funzione nel file specificato, fornito nel file zip, completandolo secondo le indicazioni.

Siano date le seguenti definizioni:

```
typedef std::string Elem;

struct Cell {
   Elem elem;
   struct Cell* next;
};

typedef Cell* List;
```

Si richiede di implementare le funzioni descritte nel seguito.

Materiale dato

Nel file zip trovate

- un file `list.h` contenente le definizioni di tipo dato e le intestazioni delle funzioni ⟵ NON MODIFICARE
- un file `mainTestList.cpp` contenente un main da usare per fare testing delle vostre implementazioni e la realizzazione (corpo) delle funzioni fornite da noi. ⟵ NON MODIFICARE
- un file `Es2-Funzione1.cpp` ⟵ MODIFICARE IL SUO CONTENUTO
- un file `Es2-Funzione2.cpp` ⟵ MODIFICARE IL SUO CONTENUTO
- un file `Es2-Funzione3.cpp` ⟵ MODIFICARE IL SUO CONTENUTO

2.1 Es2-Funzione1 - (2 punti)

```
unsigned int computeListSize(const List &l)
```

- INPUT:
  - `l`: la lista della quale calcolare la lunghezza
- OUTPUT: la lunghezza della lista espressa come numero di elementi che la compongono.
- Comportamento:

  la funzione restituisce 0 se la lista `l` è vuota oppure un valore pari numero di elementi che la compongono. La funzione non deve modificare la lista `l`.

2.2 Es2-Funzione2 - (3 punti)

```
bool insertElemInListAtIndex(List &l, Elem s, unsigned int index)
```

- INPUT:
  - `l`: la lista nella quale inserire l'elemento
  - `s`: l'elemento da inserire nella lista
  - `index` la posizione che dovrà avere l'elemento `s` dopo l'inserimento in `l`
- OUTPUT:
  - TRUE se l'inserimento va a buon fine
  - FALSE altrimenti (quando l'index non è valido). In questo caso la funzione non deve modificare la lista `l`
- Comportamento:

  la funzione inserisce l'elemento `s` nella lista `l` in posizione `index` (partendo dalla testa della lista).

- Esempi:
  Il contenuto della lista *prima* dell'inserimento è: (A)
  Inserisco l'elemento = B in pos index = 0
  Il contenuto della lista *dopo* l'inserimento è: (B,A)
  e la funzione ritorna TRUE

  Il contenuto della lista *prima* dell'inserimento è: (A)
  Inserisco l'elemento = B in pos index = 1
  Il contenuto della lista *dopo* l'inserimento è: (A,B)
  e la funzione ritorna TRUE

  Il contenuto della lista *prima* dell'inserimento è: (A)
  Inserisco l'elemento = B in pos index = 2
  Il contenuto della lista *dopo* l'inserimento è: (A)
  e la funzione ritorna FALSE (impossibile effettuare inserimento in quella posizione)

2.3 Es2-Funzione3 - (3.5 punti)

```
void deleteLastInstanceOfElemInList(List &l, Elem s)
```

- INPUT:
  - `l`: la lista da dove eliminare l'ultima istanza dell'elemento `s`
  - `s`: l'elemento da eliminare
- Comportamento:
  la funzione elimina l'ultima istanza dell'elemento `s` dalla lista `l`. Se l'elemento non è presente non fa nulla.
- Esempi:
  Il contenuto della lista è: (A,B,A,D,C,D)
  Elimino ultima istanza l'elemento = D
  Il contenuto della lista dopo l'eliminazione è: (A,B,A,D,C)

  Il contenuto della lista è: (A,B,A,D,C,D)
  Elimino ultima istanza l'elemento = A
  Il contenuto della lista dopo l'eliminazione è: (A,B,D,C,D)

  Il contenuto della lista è: (A,B,A,D,C,D)
  Elimino ultima istanza l'elemento = C
  Il contenuto della lista dopo l'eliminazione è: (A,B,A,D,D)

  Il contenuto della lista è: (A,B,A)
  Elimino ultima istanza l'elemento = Z
  Il contenuto della lista dopo l'eliminazione è: (A,B,A)