IP a.a. 2023/24 - Esame del 19 Febbraio 2024

Prima di cominciare lo svolgimento leggete attentamente tutto il testo.

Questa prova è organizzata in tre 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.

Vi diamo un file zip che contiene per ogni esercizio, un file per completare la funzione da scrivere e un programma principale per lo svolgimento di test. Ad esempio, per l'esercizio 1, saranno presenti un file es1.cpp e un file es1-test.cpp. Per compilare dovrete eseguire `g++ -std=c++11 -Wall es1.cpp es1-test.cpp -o es1-test`. E per eseguire il test, `./es1-test`. Dovete lavorare solo sui file indicati in ciascuno esercizio. 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).

Per ciascuno esercizio, vi diamo uno programma principale, che esegue i test. Controllate durante l'esecuzione del programma, quanti sono i test che devono essere superati e controllate l'esito (se non ci sono errori deve essere `true` per tutti).

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 - Funzione semplice - (max 2.5 punti)

1.1 Esercizio 1 (2.5 punti)

Per questo esercizio, lavorate sul file `es1.cpp`.

Materiale dato

Nel file zip trovate

- Un file `funz.h` contenente le definizioni di tipo dato e le intestazioni delle funzioni ⟵ NON MODIFICARE
- un file `es1-test.cpp` contenente un main da usare per fare testing ⟵ NON MODIFICARE
- un file `es1.cpp` ⟵ MODIFICARE IL SUO CONTENUTO

```
unsigned int numIterSequenzaSiracusa(unsigned int vinit)
```

- INPUT:
  - `unsigned int vinit`: il valore iniziale per la sequenza di siracusa
- OUTPUT: Il numero di iterazioni prima che la sequenza di Siracusa raggiunga un valore inferiore o uguale a 1 partendo dal valore iniziale `vinit`.
- Comportamento:

  la funzione prende in input un intero non negativo `vinit` e calcola i valori sucessivi della sequenza di Siracusa definita come segue:

$$S_n = \begin{cases} S_{n-1}/2 & \text{se } S_{n-1} \text{ è pari} \\ 3 * S_{n-1} + 1 & \text{altrimenti} \end{cases}$$

  con $S_0 =$`vinit`. La funzione ritorna il primo valore di n tale che $S_n \leq 1$.

  Ad esempio:

  - Se S_0 vale 4, i valori successivi della sequanza di Siracusa prima di raggiungere un valore inferiore o uguale a 1 sono: 4, 2, 1.

  - Se S_0 vale 5, i valori successivi della sequanza di Siracusa prima di raggiungere un valore inferiore o uguale a 1 sono: 5,16,8,4,2,1.

- Esempi:

  INPUT => OUTPUT

  `vinit=0` => `0`

  `vinit=1` => `0`

  `vinit=2` => `1`

  `vinit=4` => `2`

  `vinit=5` => `5`

2 Sezione 2 - Array - (max 5 punti)

2.1 Esercizio 2 (2 punti)

Per questo esercizio, lavorate sul file `es2.cpp`.

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 `es2-test.cpp` contenente un main da usare per fare testing ⟵ NON MODIFICARE
- un file `es2.cpp` ⟵ MODIFICARE IL SUO CONTENUTO

```
bool ascendingSequence(const int* arr,unsigned int dim)
```

- INPUT:
  - `int* arr`: un array di interi,
  - `unsigned int dim`: la dimensione dell'array `arr`
- INPUT:
  - `true` se i valori del array sono ordinati in modo crescente
  - `false` altrimenti
- Comportamento:

  la funzione verifica se l'array `arr` è in ordine crescente senza modificarlo.
- Esempi:

  INPUT => OUTPUT

  `arr=[]` => `true`

  `arr=[3]` => `true`

  `arr=[3,2]` => `false`

  `arr=[2,2,3]` => `true`

  `arr=[-1,2,5,100]` => `true`

  `arr=[2,5,3,100]` => `false`

  `arr=[2,2,3,3,4]` => `true`

2.2 Esercizio 3 (3 punti)

Per questo esercizio, lavorate sul file `es3.cpp`. È dato la seguente definizione.

```
struct dyn_array {
   unsigned int* A;
   unsigned int D;
};
```

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 `es3-test.cpp` contenente un main da usare per fare testing ⟵ NON MODIFICARE
- un file `es3.cpp` ⟵ MODIFICARE IL SUO CONTENUTO

```
dyn_array indexOfInArray(int v,const int* arr,unsigned int dim)
```

- INPUT:
  - `int v`: un valore da cercare,
  - `int* arr`: un array di interi,
  - `unsigned int dim`: la dimensione dell'array
- OUTPUT: una struct `ret` di tipo `dyn_array` dove nell campo `ret.A` abbiamo un array di dimensione `ret.D` che contiene gli indici di `arr` dove si possa trovara il valore `v`.
- Comportamento:

  la funzione cerca gli indici del array `arr` dove si può trovare il valore `v` e li mette in una struct di tipo `dyn_array`. La funzione non modifica l'array `arr`.
- Esempi:

  `int v` e `int* arr`=> OUTPUT `dyn_array ret`

  `v`=3 e `arr`=`[]` => `ret.A`=`[]` e `ret.D`=0

  `v`=2 e `arr`=`[4,5,1]` => `ret.A`=`[]` e `ret.D`=0

  `v`=2 e `arr`=`[2,6,1]` => `ret.A`=`[0]` e `ret.D`=1

  `v`=2 e `arr`=`[2,6,2,1,2,2]` => `ret.A`=`[0,2,4,5]` e `ret.D`=4

3 Sezione 3 - Liste circolari - (max 8.5 punti)

Per questa sezione, siano date le seguenti definizioni per costruire liste di interi:

```
typedef int Elem;

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

typedef Cell* List;
```

Diciamo che una lista è **circolare** se l'ultimo elemento della lista ha nel suo campo `next` l'indirizzo del primo elemento della lista. Una lista circolare vuota ha come valore `nullptr`.

3.1 Esercizio 4 (2 punti)

Per questo esercizio, lavorate sul file `es4.cpp`.

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 `es4-test.cpp` contenente un main da usare per fare testing ⟵ NON MODIFICARE
- un file `es4.cpp` ⟵ MODIFICARE IL SUO CONTENUTO

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

- INPUT:
  - `const List l`: una lista **circolare**
- OUTPUT:
  - il numero di elementi che compongono la lista
- Comportamento:
  la funzione restituisce 0 se la lista `l` è vuota oppure un valore pari al numero di elementi che la compongono. La funzione non deve modificare la lista `l`. Se un intero appare due volte nella lista, allora è contatto come due elementi.

3.2 Esercizio 5 (3 punti)

Per questo esercizio, lavorate sul file `es5.cpp`.

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 `es5-test.cpp` contenente un main da usare per fare testing ⟵ NON MODIFICARE
- un file `es5.cpp` ⟵ MODIFICARE IL SUO CONTENUTO

```
bool allDiffInCircList(const List l)
```

- INPUT:
  - `l`: una lista **circolare**
- OUTPUT:
  - `true` se tutti valori della lista sono diversi.
  - `false` altrimenti
- Comportamento:
  la funzione verifica se tutti gli interi nella lista circolare sono diversi.
- Esempi:
  Se la lista è vuota, la funzione ritorna `true`.
  Se il contenuto della lista è: (1), la funzione ritorna `true`.
  Se il contenuto della lista è: (1,2,1,3), la funzione ritorna `false`.
  Se il contenuto della lista è: (1,2,1,3,2), la funzione ritorna `false`.

3.3 Esercizio 6 (3.5 punti)

Per questo esercizio, lavorate sul file `es6.cpp`.

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 `es6-test.cpp` contenente un main da usare per fare testing ⟵ NON MODIFICARE
- un file `es6.cpp` ⟵ MODIFICARE IL SUO CONTENUTO

```
void deleteAllCircList(List &l,int v)
```

- INPUT:
  - `l`: una lista **circolare**

  - `v`: il valore da eliminare

- Comportamento:

  la funzione elimina tutte le istanze dell'elemento `v` dalla lista circolare `l`. Se l'elemento non è presente non fa nulla. La lista deve rimanere circolare.

- Esempi:

  Il contenuto della lista è: (1,2,1,4,3,4)

  Elimino l'elemento = 4

  Il contenuto della lista dopo l'eliminazione è: (1,2,1,3)

  Il contenuto della lista è: (1,2,1,4,3,4)

  Elimino l'elemento = 1

  Il contenuto della lista dopo l'eliminazione è: (2,4,3,4)

  Il contenuto della lista è: (1,1,1)

  Elimino l'elemento = 3

  Il contenuto della lista dopo l'eliminazione è: (1,1,1)

  Il contenuto della lista è: (1,1,1)

  Elimino l'elemento = 1

  Il contenuto della lista dopo l'eliminazione è: ()