IP a.a. 2023/24 - Esame del 13 Giugno 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 forniamo 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 specifici per quella funzione. 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 implementare 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 `SI` 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 punti)

1.1 Esercizio 1 (2 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 returnLucasNumberInPos(unsigned int pos)
```

- INPUT:
  - `unsigned int pos`: posizione del pos-esimo numero di Lucas
- OUTPUT: Il valore del pos-esimo numero di Lucas.
- Comportamento:

  la funzione prende in input un intero non negativo `pos` e calcola i valore del pos-esimo numero di Lucas:

  la successione di Lucas, è una successione di numeri interi positivi in cui ciascun numero è la somma dei due precedenti e i primi due termini della successione sono, per definizione, $L_0 = 2$, $L_1 = 1$. Questa successione ha quindi una definizione ricorsiva secondo la regola:

  $L_0 = 2$

  $L_1 = 1$

  $L_n = L_{n-1} + L_{n-2}$

  I primi valori di questa successione sono: $[2, 1, 3, 4, 7, 11, 18, 29, 47, 76, 123, 199, 322, 521, 843, 1364, 2207]$

- Esempi:

  INPUT => OUTPUT

  `pos=0` => `2`

  `pos=1` => `1`

  `pos=2` => `3`

  `pos=4` => `7`

  `pos=15` => `1364`

2 Sezione 2 - Array - (max 7 punti)

2.1 Esercizio 2 (3.5 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 ascendingSequenceEvenOdd(const int* arr, unsigned int dim)
```

- INPUT:
  - `int* arr`: un array di interi,
  - `unsigned int dim`: la dimensione dell'array `arr`
- OUTPUT:
  - `true` se entrambe le seguenti condizioni sono vere
    - i valori dell'array di tipo PARI sono ordinati (tra di loro) in modo crescente
    - i valori del l'array di tipo DISPARI sono ordinati (tra di loro) in modo crescente
  - `false` altrimenti
- Comportamento:

  la funzione verifica se tutti i valori pari (even in inglese) nell'array `arr` sono in ordine crescente (tra di loro) e tutti i valori dispari (odd in inglese) nell'array `arr` sono in ordine crescente (tra di loro). Quindi la funzione può ritornare `true` anche quando complessivamente l'array non è ordinato (vedi ultimo esempio con il -1 in fondo, dopo il 2): quello che conta è l'ordinamento tra i valori pari e, separatamente, i valori dispari. La funzione non modifica l'array stesso. Il valore 0 e' considerato come pari. Una sequenza di valori vuota è sempre considerata come ordinata.

- Esempi:

  INPUT => OUTPUT

  `arr=[]` => `true`

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

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

  `arr=[4,2]` => `false`

  `arr= [1,0]` => `true`

  `arr= [0,1]` => `true`

  `arr=[3,1]` => `false`

  `arr=[1,3]` => `true`

  `arr=[1,1,1,1]` => `true`

  `arr=[-3,-2,0,2,-1]` => `true`

2.2 Esercizio 3 (3.5 punti)

Per questo esercizio, lavorate sul file `es3.cpp`. È data 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 indexOfEvenInArray(const int* arr,unsigned int dim)
```

- INPUT:
  - `int* arr`: un array di interi,
  - `unsigned int dim`: la dimensione dell'array `arr`
- OUTPUT: una struct `ret` di tipo `dyn_array` dove nel campo `ret.A` abbiamo un array di dimensione `ret.D` che contiene gli indici di `arr` dove si possono trovare valori di tipo PARI (Even in inglese).
- Comportamento:

  la funzione cerca gli indici dell'array `arr` dove si trovano valori pari e li salva in una struct di tipo `dyn_array`. La funzione non modifica l'array `arr`.
- Esempi:

  INPUT (`int* arr`) => OUTPUT (`dyn_array ret`)

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

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

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

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

3 Sezione 3 - Liste - (max 7 punti)

Per ogni esercizio dovete scrivere una funzione nel file specificato, fornito nel file zip, completandolo secondo le indicazioni.
Siano date le seguenti definizioni:

```
// tipo contenuto della Cella
typedef std::string Elem;

// tipo Cell della lista
struct Cell {
   Elem elem;
   struct Cell* next;
};

// tipo lista
typedef Cell * List;
```

Si richiede di implementare le funzioni descritte nel seguito.

3.1 Esercizio 4 - (1.5 punti)

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

```
void tailInsert(List &list, Elem newElem)
```

- INPUT:
  - `list`: la lista a cui aggiungere il nuovo elemento,
  - `newElem`: l'elemento da inserire
- OUTPUT: nessuno
- Comportamento:
  la funzione inserisce l'elemento `newElem` in coda alla lista `list`. `list` puo' contenere gia' valori oppure essere vuota (cioe' uguale a `NULL`): gestire entrambi i casi; `newElem` invece si suppone sempre valorizzato correttamente.
- Esempi:
  fornendo come parametri `newElem` = Rosso e `list` = (Giallo,Verde) dopo l'esecuzione della funzione si avra' `list` = (Giallo,Verde,Rosso)

3.2 Esercizio 5 - (2 punti)

```
unsigned int elementIstancesCount(List list, Elem toFind)
```

- INPUT:
  - `list`: la lista da visitare
  - `toFind`: l'elemento di cui contare le istanze nella lista
- OUTPUT: numero di istanze dell'elemento `daTrovare` in `list`
- Comportamento:
  la funzione conta il numero di istanze di un elemento in una lista.
  NB: non usare i `Vector` per l'implementazione
- Esempi:
  - fornendo in input `list` = `NULL` e l'elemento da ricercare Giallo la funzione ritorna 0
  - fornendo in input `list` = (Rosso,Giallo) e l'elemento da ricercare Rosso la funzione ritorna 1
  - fornendo in input `list` = (Giallo,Giallo,Rosso,Rosso) e l'elemento da ricercare Giallo la funzione ritorna 2

3.3 Esercizio 6 - (3.5 punti)

```
std::string lessFrequentFind(List list)
```

- INPUT:
  - `list`: la lista dove trovare l'elemento meno frequente
- OUTPUT: l'elemento meno frequente
- Comportamento:
  la funzione ritorna l'elemento nella lista con meno istanze. Se la lista è vuota ritorna `""`.
  Nel caso ci siano 2 o piu' elementi "meno frequenti", si ritorna il primo incontrato nella lista.
  NB: non usare i `Vector` per l'implementazione

- Esempi:
  - fornendo in input `list` = `NULL` ritorna `""`
  - fornendo in input `list` = (Giallo,Giallo,Rosso) ritorna "Rosso"
  - fornendo in input `list` = (Giallo,Giallo,Rosso,Rosso,Verde,Verde) ritorna "Giallo"