IP a.a. 2021/22 - Esame del 16 settembre 2022

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 Cognome1.cpp e Cognome2.cpp, rinominandoli rispettivamente con il vostro cognome seguito dal numero 1 o 2. 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. 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 non è permesso** (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 ogni esercizio.

1 Somma numeri

Per questa parte lavorate nella cartella `Sezione1`, nel file `<VostroCognome>1.cpp`.

Un numero positivo in base dieci è una sequenza di cifre decimali 0–9, in numero indeterminato. Essendo positivo, è garantito che non serve codificare il segno, e che almeno la cifra meno significativa è diversa da zero.

In informatica, una sequenza contenente un numero indeterminato di elementi si rappresenta normalmente come una lista linkata. Supponiamo quindi che i numeri interi siano espressi in questo modo come liste di cifre, in cui la cifra di "testa" è la MENO significativa (le unità) e le altre seguono in ordine crescente.

Dato quindi il tipo struttura `num` contenente i due campi `digit` e `next` (definita nel file header `num.h`, vedi), vogliamo scrivere le funzioni seguenti.

Funzione `int2list` (DA FARE)

```
num int2list(int inum);
```

La funzione scompone il numero intero positivo `inum` nelle sue cifre, dalla più significativa alla meno significativa, e restituisce la corrispondente lista.

Osservazione: il campo digit contiene solo valori tra 0 e 9 compresi.

Suggerimento: dividere per dieci consecutivamente, e usare anche l'operatore modulo.

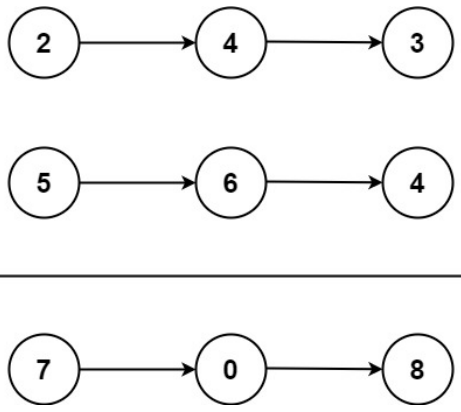
Funzione `sum` (DA FARE)

```
num int2list(num n1, num n2);
```

La funzione calcola e restituisce la somma tra `n1` e `n2` e la restituisce, sempre in forma di lista.

Osservazione: se i due addendi hanno lunghezza diversa, il più corto si può considerare completato con zeri.

Esempio:



perché 342 + 465 = 807

2 Contare i boomerang

Per questa parte lavorate nella cartella `Sezione2`, nel file `<VostroCognome>2.cpp`.

Un punto nel piano cartesiano è identificato da due coordinate x e y, quindi si rappresenta come una coppia di valori reali.

Definiamo "boomerang" una sequenza di tre punti consecutivi tale che la distanza tra il secondo e il primo sia uguale alla distanza tra il secondo e il terzo.

Ricordiamo che la distanza tra due punti (x_1, y_1) e (x_2, y_2) si calcola con il teorema di Pitagora: $d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$.

Rappresentiamo un punto con la struttura `punto` definita nello header `boomerang.h` (vedi), contenente come membri le due coordinate `x` e `y` di tipo `float`. Rappresentiamo una sequenza di N punti come un vector N-dimensionale di `struct punto`.

Funzione `contaBoomerang` (DA FARE)

```
int  contaBoomerang(std::vector);
```

Verifica tutte le sequenze di tre punti che nel vettore in ingresso sono memorizzati in posizioni consecutive; incrementa un contatore ogni volta che ne trova una che rappresenta un boomerang.

Restituisce il conteggio.

Osservazione: Lavorando in floating point, la precisione finita può dar luogo a valori che non risultano esattamente uguali anche se dovrebbero esserlo. Per confrontare l'uguaglianza usare la funzione seguente, fornita:

Funzione `equals` (fornita)

```
bool  equals(float a, float b);
```

Restituisce true se `|a-b| < eps`, dove `eps` è una tolleranza, un valore piccolo ma non nullo.