Esame 20240223

Esercizio 3

(1) **Esercizio 3 v1**

ESSAY | marked out of 10 | penalty 0 | File picker

Data una struttura dati `Stack` che rappresenta uno stack di interi (si veda il file `esercizio3.cpp`), e una serie di operazioni su questa struttura (e.g., `initStack`, `isEmpty`, `push`, `pop`, `top`, `printStack`, `deleteStack`), si vuole implementare una nuova funzione `stackOperator` che prende come argomento un puntatore a `Stack` `s` e ritorna un nuovo puntatore a `Stack` `r`. Lo stack `r` è ottenuto a partire dallo stack `s` aggiungendo dopo ogni elemento la somma degli elementi che lo precedono in `s`. Ad esempio se lo stack `s` passato in input è `<1, 2, 4, 5>` (il valore `1` è al top dello stack), allora lo stack restituito in output sarà: `<1, 1, 2, 3, 4, 7, 5, 12>`.

La funzione `stackOperator` deve **lasciare inalterato il contenuto** di `s` alla fine dell'esecuzione (*ma lo può modificare se ritenuto necessario per la realizzazione*).

La funzione `stackOperator` **non deve creare strutture intermedie** (e.g., array, stack, liste, ...) **dove memorizzare il contenuto dello stack** `s`. Valutare con attenzione le scelte implementative relative alle modalità di passaggio dello stack `s` alla funzione `stackOperator`.

Il file `esercizio3.cpp` contiene l'implementazione della struttura `Stack`, di alcuni metodi di utilità, e un `main` con alcuni esempi e alcune invocazioni della funzione `stackOperator`. Di seguito è riportato l'output di esecuzione.

```
marco > a.out
Original before: 1 2 4 5
Result StackOperator: 1 1 2 3 4 7 5 12
Original after: 1 2 4 5
Original before: 35 50 48 86 6 98 26 7 57 27
Result StackOperator: 35 35 50 85 48 133 86 219 6 225 98 323 26 349 7 356 57 413 27 440
Original after: 35 50 48 86 6 98 26 7 57 27
```

Note:

- Scaricare il file `esercizio3.cpp`, modificarlo per inserire la dichiarazione e la definizione della funzione `stackOperator`, e **caricare il file sorgente risultato delle vostre modifiche a soluzione di questo esercizio** nello spazio apposito.
- All'interno di questo programma **non è ammesso** l'utilizzo di variabili globali o di tipo `static` e di funzioni di libreria al di fuori di quelle definite in `iostream`, `cstdlib`.
- Si ricorda che, gli esempi di esecuzione sono puramente indicativi, e la soluzione proposta NON deve funzionare solo per l'input fornito, ma deve essere robusta a variazioni compatibili con la specifica riportata in questo testo.
- Si ricorda di inserire solo nuovo codice e di **NON MODIFICARE** il resto del programma (pena annullamento dell'esercizio).

esercizio3.cpp

Information for graders:

(2) **Esercizio 3 v2**

ESSAY | marked out of 10 | penalty 0 | File picker

Data una struttura dati `Stack` che rappresenta uno stack di interi (si veda il file `esercizio3.cpp`), e una serie di operazioni su questa struttura (e.g., `initStack`, `isEmpty`, `push`, `pop`, `top`, `printStack`, `deleteStack`), si vuole implementare una nuova funzione `stackOperator` che prende come argomento un puntatore a `Stack s` e ritorna un nuovo puntatore a `Stack r`. Lo stack `r` è ottenuto a partire dallo stack `s` aggiungendo dopo ogni elemento la differenza degli elementi che lo precedono in `s`. Ad esempio se lo stack `s` passato in input è `<1, 2, 4, 5>` (il valore `1` è al top dello stack), allora lo stack restituito in output sarà: `<1, -1, 2, -3, 4, -7, 5, -12>`.

La funzione `stackOperator` deve **lasciare inalterato il contenuto** di `s` alla fine dell'esecuzione (*ma lo può modificare se ritenuto necessario per la realizzazione*).

La funzione `stackOperator` **non deve creare strutture intermedie** (e.g., array, stack, liste, ...) **dove memorizzare il contenuto dello stack** `s`. Valutare con attenzione le scelte implementative relative alle modalità di passaggio dello stack `s` alla funzione `stackOperator`.

Il file `esercizio3.cpp` contiene l'implementazione della struttura `Stack`, di alcuni metodi di utilità, e un `main` con alcuni esempi e alcune invocazioni della funzione `stackOperator`. Di seguito è riportato l'output di esecuzione.

```
marco > ./a.out
Original before: 1 2 4 5
Result StackOperator: 1 -1 2 -3 4 -7 5 -12
Original after: 1 2 4 5
Original before: 35 50 48 86 6 98 26 7 57 27
Result StackOperator: 35 -35 50 -85 48 -133 86 -219 6 -225 98 -323 26 -349 7 -356 57 -413 27 -440
Original after: 35 50 48 86 6 98 26 7 57 27
```

Note:

- Scaricare il file `esercizio3.cpp`, modificarlo per inserire la dichiarazione e la definizione della funzione `stackOperator`, e **caricare il file sorgente risultato delle vostre modifiche a soluzione di questo esercizio** nello spazio apposito.
- All'interno di questo programma **non è ammesso** l'utilizzo di variabili globali o di tipo `static` e di funzioni di libreria al di fuori di quelle definite in `iostream`, `cstdlib`.
- Si ricorda che, gli esempi di esecuzione sono puramente indicativi, e la soluzione proposta NON deve funzionare solo per l'input fornito, ma deve essere robusta a variazioni compatibili con la specifica riportata in questo testo.
- Si ricorda di inserire solo nuovo codice e di **NON MODIFICARE** il resto del programma (pena annullamento dell'esercizio).

esercizio3.cpp

Information for graders:

(3) **Esercizio 3 v3**

ESSAY | marked out of 10 | penalty 0 | File picker

Data una struttura dati `Stack` che rappresenta uno stack di interi (si veda il file `esercizio3.cpp`), e una serie di operazioni su questa struttura (e.g., `initStack`, `isEmpty`, `push`, `pop`, `top`, `printStack`, `deleteStack`), si vuole implementare una nuova funzione `stackOperator` che prende come argomento un puntatore a `Stack s` e ritorna un nuovo puntatore a `Stack r`. Lo stack `r` è ottenuto a partire dallo stack `s` aggiungendo prima di ogni elemento la somma degli elementi che lo precedono in `s`. Ad esempio se lo stack `s` passato in input è `<1, 2, 4, 5>` (il valore `1` è al top dello stack), allora lo stack restituito in output sarà: `<12, 5, 7, 4, 3, 2, 1, 1>` (*si noti che l'ordine è invertito rispetto a* `s`).

La funzione `stackOperator` deve **lasciare inalterato il contenuto** di `s` alla fine dell'esecuzione (*ma lo può modificare se ritenuto necessario per la realizzazione*).

La funzione `stackOperator` **non deve creare strutture intermedie** (e.g., array, stack, liste, ...) **dove memorizzare il contenuto dello stack** `s`. Valutare con attenzione le scelte implementative relative alle modalità di passaggio dello stack `s` alla funzione `stackOperator`.

Il file `esercizio3.cpp` contiene l'implementazione della struttura `Stack`, di alcuni metodi di utilità, e un `main` con alcuni esempi e alcune invocazioni della funzione `stackOperator`. Di seguito è riportato l'output di esecuzione.

```
marco > ./a.out
Original before: 1 2 4 5
Result StackOperator: 12 5 7 4 3 2 1 1
Original after: 1 2 4 5
Original before: 35 50 48 86 6 98 26 7 57 27
Result StackOperator: 440 27 413 57 356 7 349 26 323 98 225 6 219 86 133 48 85 50 35 35
Original after: 35 50 48 86 6 98 26 7 57 27
```

Note:

- Scaricare il file `esercizio3.cpp`, modificarlo per inserire la dichiarazione e la definizione della funzione `stackOperator`, e **caricare il file sorgente risultato delle vostre modifiche a soluzione di questo esercizio** nello spazio apposito.
- All'interno di questo programma **non è ammesso** l'utilizzo di variabili globali o di tipo `static` e di funzioni di libreria al di fuori di quelle definite in `iostream`, `cstdlib`.
- Si ricorda che, gli esempi di esecuzione sono puramente indicativi, e la soluzione proposta NON deve funzionare solo per l'input fornito, ma deve essere robusta a variazioni compatibili con la specifica riportata in questo testo.
- Si ricorda di inserire solo nuovo codice e di **NON MODIFICARE** il resto del programma (pena annullamento dell'esercizio).

esercizio3.cpp

Information for graders:

(4) **Esercizio 3 v4**

ESSAY | marked out of 10 | penalty 0 | File picker

Data una struttura dati `Stack` che rappresenta uno stack di interi (si veda il file `esercizio3.cpp`), e una serie di operazioni su questa struttura (e.g., `initStack`, `isEmpty`, `push`, `pop`, `top`, `printStack`, `deleteStack`), si vuole implementare una nuova funzione `stackOperator` che prende come argomento un puntatore a `Stack s` e ritorna un nuovo puntatore a `Stack r`. Lo stack `r` è ottenuto a partire dallo stack `s` aggiungendo prima di ogni elemento la differenza degli elementi che lo precedono in `s`. Ad esempio se lo stack `s` passato in input è `<1, 2, 4, 5>` (il valore `1` è al top dello stack), allora lo stack restituito in output sarà: `<-12, 5, -7, 4, -3, 2, -1, 1>` (*si noti che l'ordine è invertito rispetto a* `s`.

La funzione `stackOperator` deve **lasciare inalterato il contenuto** di `s` alla fine dell'esecuzione (*ma lo può modificare se ritenuto necessario per la realizzazione*).

La funzione `stackOperator` **non deve creare strutture intermedie** (e.g., array, stack, liste, ...) **dove memorizzare il contenuto dello stack** `s`. Valutare con attenzione le scelte implementative relative alle modalità di passaggio dello stack `s` alla funzione `stackOperator`.

Il file `esercizio3.cpp` contiene l'implementazione della struttura `Stack`, di alcuni metodi di utilità, e un `main` con alcuni esempi e alcune invocazioni della funzione `stackOperator`. Di seguito è riportato l'output di esecuzione.

```
marco > ./a.out
Original before: 1 2 4 5
Result StackOperator: -12 5 -7 4 -3 2 -1 1
Original after: 1 2 4 5
Original before: 35 50 48 86 6 98 26 7 57 27
Result StackOperator: -440 27 -413 57 -356 7 -349 26 -323 98 -225 6 -219 86 -133 48 -85 50 -35 35
Original after: 35 50 48 86 6 98 26 7 57 27
```

Note:

- Scaricare il file `esercizio3.cpp`, modificarlo per inserire la dichiarazione e la definizione della funzione `stackOperator`, e **caricare il file sorgente risultato delle vostre modifiche a soluzione di questo esercizio** nello spazio apposito.
- All'interno di questo programma **non è ammesso** l'utilizzo di variabili globali o di tipo `static` e di funzioni di libreria al di fuori di quelle definite in `iostream`, `cstdlib`.
- Si ricorda che, gli esempi di esecuzione sono puramente indicativi, e la soluzione proposta NON deve funzionare solo per l'input fornito, ma deve essere robusta a variazioni compatibili con la specifica riportata in questo testo.
- Si ricorda di inserire solo nuovo codice e di **NON MODIFICARE** il resto del programma (pena annullamento dell'esercizio).

esercizio3.cpp

Information for graders:

Total of marks: 40