Zadanie: ROZ
Rozliczenia

Olimpiada Informatyczna

XXVI OI, etap II, dzień próbny. Plik źródłowy `roz.*` Dostępna pamięć: 10 MB. *12.02.2019*

Bajtazar został księgowym w firmie Bajtkom i jest odpowiedzialny za obsługę faktur i przygotowywanie rozliczeń. Aby usprawnić swoją pracę, postanowił zamówić u Ciebie system komputerowy do obsługi księgowości. System musi umożliwiać wykonywanie następujących operacji na liście faktur:

- dodanie nowej faktury na koniec listy (dla uproszczenia jesteśmy zainteresowani jedynie sumaryczną kwotą na tej fakturze),
- skorygowanie kwoty i-tej faktury, licząc od końca listy,
- podanie sumy kwot dla ostatnich i faktur na liście.

System musi umożliwiać przeprowadzenie dowolnej liczby operacji, przy czym operacje korekty i sumy są ograniczone do m ostatnio dodanych faktur (zakładamy, że wcześniejsze faktury zostały już zaksięgowane i nie można w nich nic zmieniać, ani o nie pytać).

Napisz moduł, który będzie udostępniał funkcje do obsługi księgowości. W tym zadaniu zwróć szczególną uwagę na **limit dostępnej pamięci** oraz to, że nie tworzysz funkcji `main`. Do limitu dostępnej pamięci **nie** wliczamy pamięci wykorzystywanej przez program sprawdzający.

Komunikacja

Twój moduł musi udostępniać następujące funkcje:

- `inicjuj`(m)
  Ta funkcja zostanie wywołana tylko raz, na początku działania programu. Możesz jej użyć, aby poznać wartość m ($1 \leq m$) i zainicjować swoje struktury danych.
  - **C++:** `void inicjuj(int m);`
  - **Python:** `def inicjuj(m)`

- `dodaj`(k)
  Funkcja powinna dodać nową fakturę o kwocie k ($-10^9 \leq k \leq 10^9$) na koniec listy faktur.
  - **C++:** `void dodaj(int k);`
  - **Python:** `def dodaj(k)`

- `koryguj`(i, k)
  Funkcja powinna skorygować kwotę na i-tej fakturze od końca listy faktur ($1 \leq i \leq m$), dodając do niej wartość k ($-10^9 \leq k \leq 10^9$). W szczególności $i = 1$ oznacza korektę kwoty na ostatnio dodanej fakturze. Możesz założyć, że podczas wywołania tej funkcji na liście będzie co najmniej i faktur.
  - **C++:** `void koryguj(int i, int k);`
  - **Python:** `def koryguj(i, k)`

- `suma`(i)
  Funkcja powinna dać w wyniku sumę kwot na i ($1 \leq i \leq m$) ostatnich fakturach na liście faktur. Jeśli do systemu było dodanych mniej niż i faktur, funkcja powinna dać w wyniku sumę wszystkich faktur.
  - **C++:** `long long suma(int i);`
  - **Python:** `def suma(i)`

Twój program **nie może** czytać żadnych danych (ani ze standardowego wejścia, ani z plików). **Nie może** również nic wypisywać do plików ani na standardowe wyjście. Może pisać na standardowe wyjście diagnostyczne (`stderr`) – pamiętaj jednak, że zużywa to cenny czas.

Ocenianie

Zestaw testów dzieli się na następujące podzadania. Testy do każdego podzadania składają się z jednej lub większej liczby osobnych grup testów. Liczba operacji, które zostaną wykonane w pojedynczym teście, nie przekroczy 10 000 000.

Limity czasowe obowiązujące w poszczególnych podzadaniach będą opublikowane w SIO.

Podzadanie	Warunki	Liczba punktów
1	$m \leq 50$	10
2	$m \leq 100\,000$	33
3	$m \leq 250\,000$	35
4	$m \leq 500\,000$	11
5	$m \leq 1\,000\,000$	11

Przykładowy przebieg programu

Operacja	Wynik	Wyjaśnienie
`inicjuj(3)`		$m = 3$
`dodaj(-6)`		dodaj fakturę z kwotą -6
`suma(1)`	-6	podaj sumę z ostatniej faktury
`dodaj(5)`		dodaj fakturę z kwotą 5
`koryguj(2, 10)`		koryguj drugą od końca fakturę, zmieniając jej kwotę na $-6 + 10 = 4$
`suma(3)`	9	podaj sumę wszystkich faktur
`suma(1)`	5	podaj sumę ostatniej faktury

Eksperymenty

W katalogu `dlazaw` znajdziesz pliki, które pozwolą Ci przetestować poprawność formalną rozwiązania. Możesz tam znaleźć przykładowe programy oceniające (`rozgrader.cpp` i `rozgrader.py`). Zwróć uwagę, że różnią się one od tych, przy pomocy których będzie finalnie oceniane Twoje rozwiązanie, i służą jedynie do sprawdzenia poprawnej interakcji z programem oceniającym. Żeby uruchomić program oceniający z Twoją biblioteką, powinieneś umieścić ją w pliku `roz.cpp` lub `roz.py`. Na początku zawodów znajdziesz tam przykładowe, błędne rozwiązania, które mogą pomóc w zrozumieniu interfejsu. Program oceniający wraz z Twoją biblioteką możesz zamienić na plik wykonywalny za pomocą polecenia:

```
make
```

W wyniku powstanie plik `rozCPP.e` lub `rozPy.e`, w zależności od języka, w którym napisano rozwiązanie. Kompilacja rozwiązania w języku C++ wymaga pliku `rozinc.h`, który również znajduje się w odpowiednich katalogach.

Tak otrzymany program oceniający wczytuje ze standardowego wejścia specjalnie przygotowany opis testu, wywołuje odpowiednie funkcje Twojej biblioteki, a wyniki wypisuje na standardowe wyjście.

Opis testu powinien być w następującym formacie: W pierwszym wierszu znajduje się liczba naturalna m. W drugim wierszu mamy liczbę q oznaczającą liczbę wywołań poszczególnych funkcji. Dalej następuje q wierszy, z których każdy zawiera pojedynczy znak `d`, `k` lub `s` oraz jedną (a) lub dwie (a, b) liczby całkowite. Znak określa, która funkcja powinna być wywołana: `d` dla `dodaj(`a`)`, `k` dla `koryguj(`a`, `b`)` i `s` dla `suma(`a`)`. Funkcja `inicjuj` zostanie wywołana od razu po wczytaniu pierwszego wiersza.

Pamiętaj jednak, że przykładowy program oceniający nie sprawdza, czy dane są sformatowane poprawnie ani czy spełnione są ograniczenia.

W folderze z testami można również znaleźć plik `roz0.in`, który odpowiada przykładowemu wykonaniu programu opisanemu powyżej. Aby uruchomić program oceniający na podanym przykładzie, użyj jednego z następujących poleceń:

- **C++:** `./rozCPP.e < roz0.in`
- **Python:** `./rozPy.e < roz0.in`

Wyniki wywołań funkcji `suma` zostaną wypisane na standardowe wyjście. Poprawny wynik dla powyższego przykładowego wykonania znajdziesz na dysku w pliku `roz0.out`.

Testy „ocen”:

`1ocen:` $m = 250$, łączna liczba wywołań funkcji `dodaj`, `koryguj` i `suma` wynosi 600. Najpierw 500 razy wywołana jest funkcja `dodaj`, z kolejnymi liczbami od -250 do 249. Następnie, dla i od 1 do 100 wywoływane są na przemian funkcje:

- `koryguj(`i`, `$2i$`)` dla i nieparzystych,
- `suma(`i`)` dla i parzystych.

`2ocen:` $m = 1000$, łączna liczba wywołań funkcji `dodaj`, `koryguj` i `suma` wynosi 6000. Najpierw 1000 wywołań funkcji `dodaj` z kolejnymi liczbami od 0 do 999. Następnie, dla i od 1 do 5000 wywoływane są na przemian funkcje:

- `koryguj(`$1 + i \bmod m$`, `$2i$`)` dla $i \bmod 3 = 1$,
- `suma(`$1 + i \bmod m$`)` dla $i \bmod 3 = 2$,
- `dodaj(`i`)` dla $i \bmod 3 = 0$.