

Zadanie C

Stos i kolejka z minimum

Celem zadania jest zaimplementowanie szablonu klasy `stackMin<C>` realizującej stos z minimum dla typu `C` oraz szablonu klasy `queueMin<C>` realizującej kolejkę z minimum dla typu `C`. Zakładamy, że dla typu `C` zdefiniowany jest operator porównania `<`.

Klasa `stackMin<C>`

1. Posiada konstruktor:
 - z jednym parametrem: `int a`, domyślnie ustawionym na 4 – tworzy stos przygotowany do przechowania maksymalnie a wartości. Podczas działania programu pamięć dla stosu jest sukcesywnie zwiększana.
2. Udostępnia następujące metody (działające w czasie $O(1)$, z wyjątkiem sytuacji gdy konieczne jest zwiększenie tablicy przechowującej stos):
 - `void push(C x)` – dodaje element x do stosu.
 - `bool pop()` – zdejmuje ostatni element ze stosu. Zwraca `true` jeśli operacja się powiodła oraz `false` jeśli stos jest pusty.
 - `C top()` – zwraca ostatni element ze stosu (nie zdejmuje go). Funkcja zakłada, że stos nie jest pusty.
 - `C min()` – zwraca element minimalny spośród elementów przechowywanych aktualnie na stosie. Funkcja zakłada, że stos nie jest pusty.
 - `void clear()` – czyści stos.
 - `bool empty()` – zwraca `true`, jeśli stos jest pusty oraz `false`, jeśli nie jest pusty.

Klasa `queueMin<C>`

1. Posiada konstruktor:
 - z jednym parametrem: `int a`, domyślnie ustawionym na 4 – tworzy kolejkę przygotowaną do przechowania maksymalnie a wartości.
2. Udostępnia następujące metody (działające w czasie amortyzowanym $O(1)$):
 - `void enqueue(C x)` – dodaje element x do kolejki.
 - `bool dequeue()` – zdejmuje ostatni element z kolejki. Zwraca `true` jeśli operacja się powiodła oraz `false` jeśli kolejka jest pusta.
 - `C front()` – zwraca ostatni element z kolejki (nie zdejmuje go). Funkcja zakłada, że kolejka nie jest pusta.

- `C min()` – zwraca element minimalny spośród elementów przechowywanych aktualnie w kolejce. Funkcja zakłada, że kolejka nie jest pusta.
- `void clear()` – czyści kolejkę.
- `bool empty()` – zwraca `true`, jeśli kolejka jest pusta oraz `false`, jeśli nie jest pusta.

Dodatkowo przygotowany program powinien zawierać

- szablon klasy `myPair<C>` zawierającej dwa publiczne pola typu `C` o nazwach `first` i `second`.
- szablon funkcji `void solveStack(stackMin<C> & S, int n)`, która dla otrzymanego stosu z minimum wykonuje n wczytanych z wejścia instrukcji.
- szablon funkcji `void solveQueue(queueMin<C> & S, int n)`, która dla otrzymanej kolejki z minimum wykonuje n wczytanych z wejścia instrukcji.

Definicję wymienionych szablonów wraz z definicjami wszystkich funkcji należy umieścić w pliku z rozszerzeniem `.h`. Tak przygotowany plik należy wysłać na Satori. Zostanie on skompilowany wraz z plikiem zawierającym funkcję `main`. W zadaniu nie należy korzystać z gotowych szablonów z biblioteki standardowej, w szczególności z szablonu `vector`.

Wejście

Pierwsza linia wejścia zawiera liczbę całkowitą z ($1 \leq z \leq 2 \cdot 10^9$) – liczbę zestawów danych, których opisy występują kolejno po sobie. Opis jednego zestawu jest następujący:

Pierwsza linia zestawu zawiera liczbę całkowitą n ($1 \leq n \leq 10^6$) oznaczającą liczbę instrukcji do wykonania na stosie z minimum lub kolejce z minimum, napis `s` oznaczający typ danych (np. `INT`, `DOUBLE`) oraz napis `t` oznaczający typ struktury danych, którą należy użyć (`STACK` lub `QUEUE`). W kolejnych liniach znajdują się instrukcje, które należy wykonać dla stosu lub kolejki z minimum.

Instrukcje dla stosu:

- `push x` – dodaje do stosu element `x`,
- `pop` – zdejmuje ostatni element ze stosu. Wypisuje wartość zdjętego elementu, jeśli operacja się powiodła oraz `ERROR` jeśli stos jest pusty.
- `top` – wypisuje ostatni element z stosu (nie zdejmuje go) lub `EMPTY`, jeśli stos jest pusty.
- `min` – wypisuje element minimalny spośród elementów przechowywanych aktualnie na stosie lub `EMPTY`, jeśli stos jest pusty.
- `clear` – ustawia liczbę elementów przechowywanych w stosie na 0.

- **empty** – wypisuje YES jeśli stos jest pusty oraz NO, jeśli nie jest pusty.

Instrukcje dla kolejki:

- **enqueue x** – dodaje do kolejki element **x**,
- **dequeue** – zdejmuje ostatni element z kolejki. Wypisuje wartość zdjętego elementu, jeśli operacja się powiodła oraz **ERROR** jeśli kolejka jest pusta.
- **front** – wypisuje ostatni element z kolejki (nie zdejmuje go) lub **EMPTY**, jeśli kolejka jest pusta.
- **min** – wypisuje element minimalny spośród elementów przechowywanych aktualnie w kolejce lub **EMPTY**, jeśli kolejka jest pusta.
- **clear** – ustawia liczbę elementów przechowywanych w kolejce na 0.
- **empty** – wypisuje YES jeśli kolejka jest pusta oraz NO, jeśli nie jest pusta.

Wyjście

Dla każdego zestawu danych, wykonaj wczytany ciąg instrukcji na stosie lub kolejce z minimum.

Dostępna pamięć: 24MB

Przykład

Przykładowy plik z funkcją main:

```
#include<iostream>
#include<string>
using namespace std;
#include "solution.h"

int main() {
    ios_base::sync_with_stdio(false);
    int z, n;
    string s, t;

    myPair<int> x;
    myPair< stackMin<int> > p;
    p.first.push(5);
    p.second.push(7);
    p.first.push(15);
    p.second.push(3);
    cout << p.first.top() << " " << p.second.top() << endl;
    cout << p.first.min() << " " << p.second.min() << endl;

    cin >> z;
    while (z--> {
        cin >> n >> s >> t;

        if (t == "STACK") {
            switch(s[0]) {
                case 'I': { stackMin<int> S;    solveStack(S, n); break; }
                case 'S': { stackMin<string> S; solveStack(S, n); break; }
                case 'D': { stackMin<double> S; solveStack(S, n); break; }
            }
        }
        else {
            //if (t == "QUEUE")
            switch(s[0]) {
                case 'I': { queueMin<int> Q;    solveQueue(Q, n); break; }
                case 'S': { queueMin<string> Q; solveQueue(Q, n); break; }
                case 'D': { queueMin<double> Q; solveQueue(Q, n); break; }
            }
        }
    }
    return 0;
}
```

Dla danych wejściowych:

```
3
17 INT STACK
pop
push 10
push 8
push 11
push 12
top
min
pop
min
empty
pop
pop
min
pop
top
min
empty
9 STRING QUEUE
enqueue ola
front
min
enqueue ala
enqueue kogut
min
clear
enqueue kot
min
8 DOUBLE QUEUE
enqueue 9.7
enqueue 7.3
enqueue 10.11
front
min
dequeue
dequeue
min
```

Poprawną odpowiedzią jest:

```
15 3
5 3
ERROR
12
8
12
8
NO
11
8
10
10
EMPTY
EMPTY
YES
ola
ola
ala
kot
9.7
7.3
9.7
7.3
10.11
```