Zestaw 1

Stos

Algorytmy i strukury danych I

Stos

> ***Stos*** *(ang.* stack*) to podstawowa struktura danych (abstrakcyjny typ danych), która implementuje zbiory dynamiczne. Elementy są usuwane ze stosu w kolejności od* najpóźniej *dodanego (strategia last-in, first-out -* ***LIFO****).*

Proszę najpierw przeczytać wskazówki w folderze `Materiały` na stronie ćwiczeń.

Interfejs

```
class Stack {
  Stack(int capacity); // Konstruktor. Argumentem jest rozmiar tablicy.
  void push(int x);    // Wstawia element na stos
  int pop();           // Usuwa element ze stosu i zwraca jego wartość
  int size();          // Zwraca liczbę elementów na stosie
  bool empty();        // Sprawdza czy stos jest pusty
};
```

Uwagi

Operacje mają mieć złożoność $O(1)$. Złożoność obliczeniowa programów powinna być *optymalna* dla danej implementacji.

W przypadku wystąpienia błędu **niedomiaru** lub **przepełnienia** operacje powinny wyrzucać wyjątek `std::out_of_range`.

Zadanie 1. Implementacja tablicowa stosu (`ArrayStack.hpp`)

Napisać implementację tablicową stosu zgodnie z podanym interfejsem. Proszę nie używać klasy `std::vector`.

Zadanie 2. Implementacja wskaźnikowa stosu (`LinkedStack.hpp`)

Napisać implementację wskaźnikową stosu zgodnie z podanym interfejsem.

Zadanie 3. Stos (`Stack.cpp`)

Program `Stack.x` ma wczytać ze standardowego wejścia dane wg poniższego formatu wygenerowane przez program opisany w zadaniu **Generator**. Wynik działania odpowiednich operacji na stosie wypisać na standardowe wyjście. Stos przechowuje elementy typu `int`. Założyć, że na stosie może się naraz znajdować maksymalnie 10^6 elementów.

Program musi przechodzić testy.

Format danych: W pierwszej linii podana jest liczba $n \leq 10^6$ wskazującą na liczbę operacji do wykonania oraz n linii poleceń. Operacje mogą być następującego typu:

- `A x` - połóż na stos liczbę $0 \leq x \leq 10^6$
- `D` - zdejmij element ze stosu i go wypisz, jeśli stos/kolejka jest pusta wypisz "EMPTY"
- `S` - wypisz liczbę elementów na stosie lub w kolejce

> ***Uwaga:*** *Programy* ***muszą*** *wczytywać dane wejściowe ze standardowego wejścia i wypisać rezultat na standardowe wyjście.*

Zadanie 4. Generator (`Generator.cpp`)

Napisać program `Generator.x`, który generuje dane wejściowe dla programu `Stack.x`. Dane powinny być generowane losowo (inne przy każdym uruchomieniu) i muszą być zgodne z powyższym formatem. Liczbę operacji podać jako argument linii komend.

Zadanie 5. Odwrotna notacja polska (`ONP.cpp`)

Napisz program, który za pomocą **stosu** zamienia wyrażenie arytmetyczne w *zapisie klasycznym* na *odwrotną notację polską*.

- W wyrażeniu występują jedynie nawiasy okrągłe `( )`, operatory binarne `+ - * /` i dodatnie liczby całkowite. Każdy z w.w. elementów jest oddzielony spacją.
- Każde wyrażenie ma składnię *nawias_otwierający lewy_argument operator prawy_argument nawias_zamykający*. Każda z operacji wraz z argumentami objęta jest nawiasem, w związku z czym można pominąć kolejność wykonywania działań.

Przykładowe dane wejściowe i wyjściowe:

```
( 11 + ( ( ( ( 1 + 2 ) * ( 4 - 3 ) ) + ( 4 / 2 ) ) * ( 8 - 6 ) ) )
11 1 2 + 4 3 - * 4 2 / + 8 6 - * +
```

Uwagi

- Na platformę Pegaz należy wysłać spakowany katalog w formacie `.tar.gz` lub `zip`.
- Katalog musi się nazywać `Zestaw01` i zawierać tylko pliki źródłowe i `Makefile`.
- Pliki źródłowe muszą mieć podaną nazwę, a programy wykonywalne muszą mieć *rozszerzenie* `.x`.
- Wywołanie komendy `make` w tym katalogu powinno kompilować wszystkie programy i tylko kompilować.
- Kompilacja musi przebiegać bez błędów ani ostrzeżeń.
- Należy używać własnych implementacji typów danych w programach.
- Programy nie powinny wypisywać niczego ponad to co opisano w instrukcji. Proszę dokładnie czytać opis formatu danych wejściowych i wyjściowych.
- Implementacje klas mogą znajdować się w pliku nagłówkowym. Taka konstrukcja jest konieczna w przypadku szablonów klas.

Pytania

- Co to jest **stos**?
- Omów przykłady zastosowania **stosu**?
- Dlaczego operacja `pop()` z `std::stack` **nie** zwraca wartości elementu?
- Dlaczego operacja `pop()` z `std::stack` **nie** zwraca referencji do elementu?
- Do czego służy funkcja `std::queue::emplace`?

Dodatkowe punkty

Dodatkowe punkty (po 1 pkt) można zdobyć za:

- Napisanie szablonu klas
- Wykorzystanie referencji do r-wartości, semantyki przenoszenia, uniwersalnych referencji, doskonałego przekazywanie.
- Napisanie testera

Napisać szablon klas wg poniższego schematu, który implementuje stos przechowujący obiekty typu `T` w oparciu o tablicę o rozmiarze `N`.

```
template<class T, int N>
class Stack {
  template<class U>   // Uniwersalne referencje
  void push(U&& x);   // Wstawia element x na stos
  T pop();            // Usuwa element ze stosu i zwraca jego wartość
  T& top();           // Zwraca referencję do najmłodszego elementu
  int size();         // Zwraca liczbę elementów na stosie
  bool empty();       // Sprawdza czy stos jest pusty
};
```

Andrzej Görlich
a.goerlich@outlook.com