

Міністерство освіти і науки України
Львівський національний університет імені Івана Франка
Факультет прикладної математики та інформатики

Кафедра програмування

Лабораторна робота №2
Стек та стек-STL
з курсу “Алгоритми та структури даних”

Виконав:
студент групи ПМІ-13
Лук'янчук Денис
Євгенійович

Львів – 2024

Стек + стекSTL

Стек – це динамічна структура зберігання даних, яка працює за принципом “останній прийшов – перший вийшов” (Last-In First-Out). У стеку додавання нових елементів та видалення існуючих відбувається з одного кінця, який називається вершиною стеку.

Застосування стеку (не використовуючи бібліотеку STL)

Хід роботи

1. Створення класу Stack:

Визначається клас Stack, який має приватні поля: array (динамічний масив), capacity (максимальна ємність стеку), і size (поточний розмір стеку).

2. Конструктор:

В конструкторі відбувається ініціалізація динамічного масиву зазначеної ємності та початкового розміру 0.

3. Деструктор:

У деструкторі відбувається звільнення пам'яті, виділеної для динамічного масиву.

4. Метод push:

Перевіряється, чи стек не заповнений. Якщо ні, елемент додається до стеку, інкрементується розмір стеку, і виводиться повідомлення. Якщо стек вже заповнений, виводиться відповідне повідомлення.

5. Метод pop:

Перевіряється, чи стек не порожній. Якщо ні, виводиться верхній елемент стеку, декрементується розмір стеку, і виводиться повідомлення. Якщо стек порожній, виводиться відповідне повідомлення.

6. Метод top:

Перевіряється, чи стек не порожній. Якщо ні, виводиться верхній елемент стеку. Якщо стек порожній, виводиться відповідне повідомлення.

7. Методи isEmpty і isFull:

isEmpty повертає true, якщо розмір стеку дорівнює 0, інакше false. isFull

повертає true, якщо розмір стеку дорівнює ємності, інакше false.

8. Функція main:

Створюється об'єкт myStack класу Stack з максимальною ємністю 5.

Викликаються різні методи для додавання, видалення та виведення елементів стеку.

9. Вивід результатів:

Під час виконання програми виводяться повідомлення про додавання, видалення та стан стеку.

Приклад #1

Дано: size = 5, elements = {1, 2, 3}

```
Top element: 3  
Popped: 3  
Top element: 2
```

Приклад #2

Дано: size = 4, elements = {5, 0, 8, -3, 6}

```
Stack overflow  
Top element: -3  
Popped: -3  
Top element: 8
```

Застосування стеку STL

Хід роботи

1) Головна функція:

Створення об'єкту стеку з використанням `stack<int>`.

2) Додавання елементів до стеку:

Використання методу `push` для додавання елементів 1, 2, та 3 до стеку.

3) Виведення верхнього елемента стеку:

Використання методу top для виведення верхнього елемента стеку.

4) Видалення верхнього елемента стеку:

Використання методу pop для видалення верхнього елемента стеку.

5) Повторне виведення верхнього елемента після видалення:

Використання методу top для виведення верхнього елемента стеку після видалення.

6) Перевірка, чи стек порожній:

Використання методу empty для перевірки, чи стек порожній.

7) Вивід результатів:

Виведення відповідних повідомлень залежно від стану стеку.

Приклад #1

Дано: Elements = {1, 2, 3}

```
Top element of the stack: 3
Top element of the stack after removal: 2
The stack is not empty.
```

Приклад #2

Дано: Elements = {6, 2, 8, -7}

```
Top element of the stack: -7
Top element of the stack after removal: 8
The stack is not empty.
```

Приклад Unit-теста

```
[=====] Running 5 tests from 1 test suite.
[-----] Global test environment set-up.
[-----] 5 tests from StackTest
[ RUN      ] StackTest.PushAndPeek
[          OK ] StackTest.PushAndPeek (0 ms)
[ RUN      ] StackTest.PopAndPeek
[          OK ] StackTest.PopAndPeek (0 ms)
[ RUN      ] StackTest.PopUntilEmpty
[          OK ] StackTest.PopUntilEmpty (0 ms)
[ RUN      ] StackTest.Overflow
Stack overflow
Stack overflow
Stack overflow
[          OK ] StackTest.Overflow (0 ms)
[ RUN      ] StackTest.Underflow
[          OK ] StackTest.Underflow (0 ms)
[-----] 5 tests from StackTest (1 ms total)

[-----] Global test environment tear-down
[=====] 5 tests from 1 test suite ran. (2 ms total)
[ PASSED ] 5 tests.
```

Висновок: Використання бібліотеки STL, зокрема класу `std::stack`, виявляється більш зручним та ефективним способом роботи із стеком у мові програмування C++. Цей підхід надає готовий інтерфейс для операцій над стеком, таких як додавання (`push`), видалення (`pop`), отримання верхнього елемента (`top`), і перевірка на порожнечу (`empty`). Крім того, використання STL спрощує код, робить його більш читабельним та компактним.

З іншого боку, реалізація стеку без використання STL може бути корисною для вивчення базових принципів динамічних масивів та управління пам'яттю. Такий підхід може забезпечити більший контроль над споживанням пам'яті та оптимізацією, але водночас вимагає більше коду та зусиль розробника.

В цілому, вибір між двома підходами залежить від конкретних потреб проекту, але в більшості випадків використання STL виявляється більш зручним та ефективним.