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

Кафедра Теорії оптимальних процесів

Лабораторна робота №2

РЕАЛІЗАЦІЯ СТЕКУ НА БАЗІ ДИНАМІЧНОГО МАСИВУ

з курсу “Алгоритми обчислювальних процесів ”

Виконав:
студент групи ПМА-11
Ковтун Віталій Олегович

Львів - 2024

Реалізація Стеку (Stack)

Стек – впорядкована структура даних, яка працює за принципом **LIFO (L**ast **I**n **F**irst **O**ut – Останній Зайшов, Перший Вийшов).

Елементи додаються та видаляються лише з одного кінця – вершини стеку.

Перший елемент у стеку називають дном або низом стека. Зазвичай на дно поміщають певний елемент, який враховують алгоритми обробки стека.

Розрізняють стеки з проштовхуванням (push down) та без проштовхування. У стеках першого типу закріплена вершина. У стеках без проштовхування закріплене дно, а вершина рухома.

Над стеком визначено дві основні операції:

- push() – включення елемента в стек:

```
void push(T element) {
		if (size < capacity) {
			stack[size++] = element;
		}
		else {
			cerr << "Stack overflow occured. Couldn't add new
			element." << endl;
		}
}
```

- pop() – виключення елемента з стеку:

```
void pop() {
	if (!isEmpty()) {
		--size;

	}
	else {
		cerr << "Stack is empty. " << endl;
	}
}
```

Та додаткові:

- isEmpty() – перевірка наявності елементів у стеку:

```
void topDel() {
	if (!isEmpty()) {
		--size;
	}
	else {
		cerr << "Stack is empty. " << endl;
	}
}
```

- top() – отримання верхнього елемента стеку:

```
T top() {
	if (!isEmpty()) {
		return stack[size - 1];
	}
	else {
		cerr << "Stack is empty. Couldn't get top element. " << endl;
		return T();
	}
}
```

- getSize() – отримання кількості елементів (розміру) стеку:

```
int getSize() {
	return size;
	}
```

Основною перевагою стеку над іншими способами організації даних є те, що адресація елементів у стеці не потрібна. Тому стек може бути базовою структурою у безадресних машинах.

Демонстрація методів top(), getSize(), pop(), isEmpty(), push():

```
Stack<int> newStack;
	newStack.push(1);
	newStack.push(2);
	newStack.push(3);
for (int i = 0; i < 3; i++) {
		cout << "Stack top element: " << newStack.top() << endl;
		cout << "Stack size: " << newStack.getSize() << endl;
		newStack.pop();
	}
	if (newStack.isEmpty()) {
		cout << "Stack is empty." << endl;
		return 0;
	}
```

Результат:

```
Stack top element: 3
Stack size: 3
Stack top element: 2
Stack size: 2
Stack top element: 1
Stack size: 1
Stack is empty.
```

Висновок: стек та його базові методи було успішно реалізовано, роботу методів продемонстровано на знімках екрану.

Тестування програми

Для тестування стеку було використано бібліотеку **cassert** (`#include <cassert>`) та створено функцію runTest()**:**

```
void runTest() {

	Stack<int> newStack;
	newStack.push(1);
	newStack.push(2);
	newStack.push(3);

	assert(newStack.getSize() == 3);
	assert(newStack.top() == 3);

	newStack.pop();
	assert(newStack.top() == 2);

	assert(!newStack.isEmpty());

	newStack.pop();
	newStack.pop();
	assert(newStack.isEmpty());

	assert(newStack.getSize() == 0);

	cout << "All tests passed. " << endl;

}
```

Результат:

```
All tests passed.
```

Висновок: цей спосіб тестування порівнює бажане значення з явним. Якщо програма ‘assert’ не виконується, то програма завершується, а на етапі відладки виводиться повідомлення про помилку. Якщо тестування проходить успішно, програма виводить відповідне повідомлення.