Министерство науки и высшего образования Российской Федерации
Российский химико-технологический университет имени Д. И. Менделеева

Факультет информационных технологий и управления
Кафедра информационных компьютерных технологий

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №3

«Базовые структуры»

Выполнил студент группы КС-30 Колесников Артем Максимович

Ссылка на репозиторий: https://github.com/MUCTR-IKT-CPP/AMKolesnikov_30_ALG

Приняли: аспирант кафедры ИКТ Пысин Максим Дмитриевич
аспирант кафедры ИКТ Краснов Дмитрий Олегович

Дата сдачи: 28.03.2022

Москва
2022

Содержание

Описание задачи

В рамках лабораторной работы необходимо изучить и реализовать структуру «стек», которая должна содержать:

- Использовать шаблонный подход, обеспечивая работу контейнера с произвольными данными.
- Реализовывать свой итератор с реализацией операторов ++ и !=
- Обеспечивать работу стандартных библиотек и конструкции for each если она есть в языке.
- Проверку на пустоту и подсчет количества элементов.
- Операцию сортировки с использованием стандартной библиотеки.
- добавление в начало
- взятие из начала

Для демонстрации работы структуры необходимо создать набор тестов(под тестом понимается функция, которая создаёт структуру, проводит операцию или операции над структурой и удаляет структуру):

1. Заполнение контейнера 1000 целыми числами в диапазоне от -1000 до 1000 и подсчет их суммы, среднего, минимального и максимального.
2. Провести проверку работы операций вставки и изъятия элементов на коллекции из 10 строковых элементов.
3. Заполнение контейнера 100 структур, содержащих фамилию, имя, отчество и дату рождения (от 01.01.1980 до 01.01.2020) значения каждого поля генерируются случайно из набора заранее заданных. После заполнение необходимо найти всех людей младше 20 лет и старше 30 и создать новые структуры, содержащие результат фильтрации, проверить выполнение на правильность подсчётом кол-ва элементов, не подходящих под условие в новых структурах.
4. Заполнить структуру 1000 элементов и отсортировать ее, проверить правильность использую структуру из стандартной библиотеки и сравнив результат.
5. Инверсировать содержимое контейнера, заполненного отсортированными по возрастанию элементами не используя операцию перемещения при помощи итератора, а только операторы изъятия и вставки.

Описание структуры

Стек (англ. stack — стопка) абстрактный тип данных, представляющий собой список элементов, организованных по принципу LIFO (англ. last in — first out, «последним пришёл — первым вышел»).

Чаще всего принцип работы стека сравнивают со стопкой тарелок: чтобы взять вторую сверху, нужно снять верхнюю.

Выполнение задачи

Я реализовал стек на языке C++. Моя программа состоит из 5 функций-тестов и шаблонной структуры «MyStack», которая содержит:

- структуру «Node»
- класс «Iterator», в котором я перегрузил операторы ++, !=, *, Begin(), End()
- 7 функций ~ push, pop, top, size, empty, printStack, sorting.

Структура «MyStack»:

```
template <typename T>
struct MyStack
{
    MyStack() : _top(NULL), _size(0){}

        struct Node
        {
                Node(T el) : data(el), next(NULL) {}
                Node(T el, Node* node) : data(el), next(node) {}
                Node* next;
                T data;
        };

        void push(const T);
        void pop();
        T& top();
        int size();
        bool empty();
        void printStack();
        void sorting();

        class Iterator{…};

        Iterator Begin() { return Iterator(_top); }
        Iterator End() { return Iterator(); }

private:
        Node* _top;
        int _size;
};
```

Функция «push» ~ добавляет элемент в начало стека.

```
template<typename T>
void MyStack<T>::push(const T el)
{
        try
        {
                if (_top != NULL) {
                        Node* tmp = new Node(el, _top);
                        _top = tmp;
                }
                else
                        _top = new Node(el);
```

```
			_size++;
		}
		catch (bad_alloc)
		{
			cout << "bad_alloc" << endl;
		}

		 }
```

Функция «pop» ~ удаляет элемент из начала стека

```
template<typename T>
void MyStack<T>::pop()
{
		try
		{
			if (empty())
				throw out_of_range("Error: Out of range");

			Node* tmp = _top;
			_top = _top->next;
			delete tmp;
			_size--;
		}
		catch (std::exception& e)
		{
			cout << e.what() << endl;
		}

		 }
```

Функция «top» ~ возвращает элемент из начала стека.

```
template<typename T>
T& MyStack<T>::top()
{
		try
		{
			if (empty())
				throw out_of_range("Error: MyStack is empty");

			return _top->data;

		}
		catch (const std::exception& e)
		{
			cout << e.what() << endl;
		}
}
```

Функция «size» ~ возвращает размер стека.

```
template<typename T>
int MyStack<T>::size()
{
		return _size;
}
```

Функция «empty» ~ возвращает true, если стек пуст или false, если в стеке есть хотя бы 1 элемент.

```
template<typename T>
bool MyStack<T>::empty()
{
	return _size == 0 ? true : false;
}
```

Функция «printStack» ~ выводит все элементы стека в консоль.

```
template<typename T>
void MyStack<T>::printStack()
{
	//for (Iterator it = my_stack.Begin(); it != my_stack.End(); ++it) {
	//	cout << *it << endl;
	//}
	auto print = [](const T& n) { cout << n << endl; };
	for_each(Begin(), End(), print);

}
```

Функция «sorting» ~ сортирует стек по возрастанию.

```
template<typename T>
void MyStack<T>::sorting()
{
	vector<T> v;
	while (!empty())
	{
		v.push_back(top());
		pop();
	}

	sort(v.begin(), v.end());

	for (int i = 0; i < v.size(); i++) {
		push(v[i]);
	}
}
```

Класс «Iterator» ~ содержит перегрузку операторов.

```
	class Iterator
	{
	public:
		Iterator(Node* current_node = NULL) : current_node(current_node) {};
		~Iterator() {};

		Iterator& operator++() {
			current_node = current_node->next;
			return *this;
		}

		T& operator*() const {
			return current_node->data;
		}

		bool operator!=(const Iterator& other) const {
			return current_node != other.current_node;
		}
```

```
	private:
		Node* current_node;
	};

	Iterator Begin() { return Iterator(_top); }
	Iterator End() { return Iterator(); }
```

Функция «test1» ~ заполнение контейнера 1000 целыми числами в диапазоне от -1000 до 1000 и подсчет их суммы, среднего, минимального и максимального.

```
void test1() {
	MyStack<int> my_stack;
	int N = 1000;
	int MIN = -1000;
	int MAX = 1000;

	int sum = 0, min_el = MAX, max_el = MIN;
	double avg;

	cout << "Тест 1" << endl << endl;

	for (int i = 0; i < N; i++)
	{
		int el = numGenerator(MIN, MAX);
		my_stack.push(el);
		sum += el;
		if (el > max_el) max_el = el;
		if (el < min_el) min_el = el;
	}

	avg = (double)sum / N;

	cout << "Размер стэка: " << my_stack.size() << endl;
	cout << "Минимальный элемент: " << min_el << endl;
	cout << "Максимальный элемент: " << max_el << endl;
	cout << "Среднее арифметическое: " << avg << endl;
	cout << "Сумма элементов: " << sum << endl << endl << endl;
}
```

Функция «test2» ~ провести проверку работы операций вставки и изъятия элементов на коллекции из 10 строковых элементов.

```
void test2() {
	MyStack<string> my_stack;
	vector<string> bank = { "Москва" , "Омск" , "Калуга", "Коломна", "Челябинск",
"Балашиха", "Егорьевск", "Воскресенск", "Гжель", "Серпухов" };

	cout << "Тест 2" << endl << endl;

	for (int i = 0; i < bank.size(); i++)
	{
		my_stack.push(bank[i]);
	}

	cout << "Элементы стека:" << endl;
	my_stack.printStack();
	cout << "Размер стэка: " << my_stack.size() << endl;

	cout << "\nУдалили 2 элемента с начала стека:" << endl;
	my_stack.pop();
	my_stack.pop();
```

```
	my_stack.printStack();
	cout << "Размер стэка: " << my_stack.size() << endl;

	cout << "\nДобавили 1 элемент (Воронеж): " << endl;
	my_stack.push("Воронеж");
	my_stack.printStack();

	cout << "Размер стэка: " << my_stack.size() << endl << endl << endl;
}
```

Функция «test3» ~ заполнение контейнера 100 структур, содержащих фамилию, имя, отчество и дату рождения (от 01.01.1980 до 01.01.2020) значения каждого поля генерируются случайно из набора заранее заданных. После заполнение необходимо найти всех людей младше 20 лет и старше 30 и создать новые структуры, содержащие результат фильтрации, проверить выполнение на правильность подсчётом кол-ва элементов, не подходящих под условие в новых структурах.

```
void test3() {

	struct Human
	{
		string fio;
		int year;
	};

	cout << "Тест 3" << endl;
	int N = 100;
	MyStack<Human> humanList;

	for (int i = 0; i < N; i++)
	{
		Human newhuman;
		newhuman.fio = fio();
		newhuman.year = year();
		humanList.push(newhuman);
	}

	MyStack<Human> y20;
	MyStack<Human> o30;
	int ageError = 0;

	while (!humanList.empty())
	{
		auto el = humanList.top();

		if (2020 - el.year < 20) y20.push(el);
		else if (2020 - el.year > 30) o30.push(el);

		humanList.pop();
	}


	while (!y20.empty()) {
		auto el = y20.top();

		if (el.year < 2000) ageError++;

		y20.pop();
	}

	cout << "Число ошибок в стеке людей младше 20: " << ageError << endl;

	ageError = 0;
```

```
	while (!o30.empty()) {
		auto el = o30.top();

		if (el.year > 1990) ageError++;

		o30.pop();
	}
	cout << "Число ошибок в стеке людей старше 30: " << ageError << endl << endl;

}

string fio() {

	const int N = 2;
	string fio;

	string firstName[N] = { "Артем", "Максим" };
	string secondName[N] = { "Петров", "Иванов" };
	string patronymic[N] = { "Андреевич", "Степанович" };

	int randomName[3] = { rand() % ((N - 1) + 1), rand() % ((N - 1) + 1), rand() % ((N -
1) + 1) };
	return fio = secondName[randomName[0]] + " " + firstName[randomName[0]] + " " +
patronymic[randomName[0]];
};

int year() {
	return numGenerator(1990, 2020);
}
```

Функция «test4» ~ заполнить структуру 1000 элементов и отсортировать ее, проверить правильность использую структуру из стандартной библиотеки и сравнив результат.

```
void test4() {
	MyStack<int> my_stack;
	//stack<int> defolt_stack;

	int N = 10;
	int MIN = -10;
	int MAX = 10;

	cout << "Тест 4" << endl;

	for (int i = 0; i < N; i++)
	{
		int el = numGenerator(MIN, MAX);
		my_stack.push(el);
		//defolt_stack.push(el);
	}
	cout << "\nИсходный стек:" << endl;
	my_stack.printStack();
	my_stack.sorting();
	cout << "\nОтсортированный стек:" << endl;
	my_stack.printStack();
	cout << endl << endl;

	}
```

Функция «test5» ~ инверсировать содержимое контейнера, заполненного отсортированными по возрастанию элементами не используя операцию перемещения при помощи итератора, а только операторы изъятия и вставки.

```
void test5() {
	MyStack<int> my_stack;
	MyStack<int> stack2;
	int N = 10;

	cout << "Тест 5" << endl;

	for (int i = 0; i < N; i++)
	{
		my_stack.push(i+1);
	}
	cout << "\nИсходный стек:" << endl;
	my_stack.printStack();

	while (!my_stack.empty())
	{
		stack2.push(my_stack.top());
		my_stack.pop();
	}

	my_stack = stack2;

	cout << "\nИнверсированный:" << endl;
	my_stack.printStack();
	cout << endl << endl;
}
```

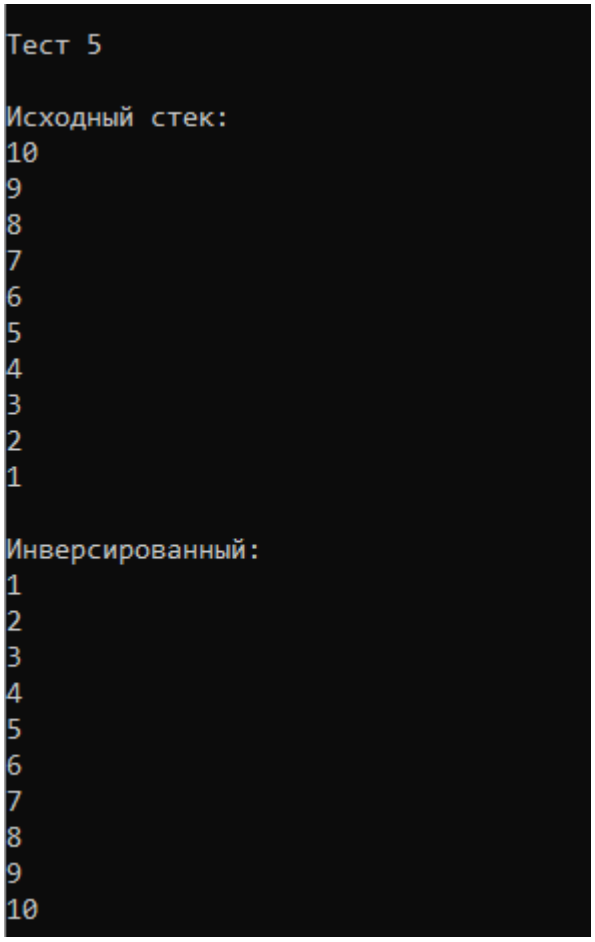
Функция «main» ~ запуск всех тестов моего стека.

```
int main()
{
	srand(time(0));
	setlocale(LC_ALL, "Rus");
	test1();
	test2();
	test4();
	test5();
}
```

Результат работы программы:

```
Тест 1

Размер стэка: 1000
Минимальный элемент: -998
Максимальный элемент: 999
Среднее арифметическое: 3.127
Сумма элементов: 3127


Тест 2

Элементы стека:
Серпухов
Гжель
Воскресенск
Егорьевск
Балашиха
Челябинск
Коломна
Калуга
Омск
Москва
Размер стэка: 10

Удалили 2 элемента с начала стека:
Воскресенск
Егорьевск
Балашиха
Челябинск
Коломна
Калуга
Омск
Москва
Размер стэка: 8

Добавили 1 элемент (Воронеж):
Воронеж
Воскресенск
Егорьевск
Балашиха
Челябинск
Коломна
Калуга
Омск
Москва
Размер стэка: 9
```

```
Тест 3
Число ошибок в стеке людей младше 20: 0
Число ошибок в стеке людей старше 30: 0

Тест 4

Исходный стек:
3
7
-9
9
-2
-5
-7
-2
-3
7

Отсортированный стек:
9
7
7
3
-2
-2
-3
-5
-7
-9
```



Заключение

Мне удалось реализовать свой собственный стек, практически со всеми функциями стека из стандартной библиотеки C++. К тому же я дополнил его 2-мя своими функциями (сортировка, вывод на экран всего стека), которых нет в стандартной библиотеке.