МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Программирование»

Тема: «Динамические структуры данных»

Студент гр. 1304 _____ Шаврин А.П

Преподаватель _____ Чайка К.В.

Санкт-Петербург

2022

Цель работы.

Изучить динамические структуры данных в языке C++

Задание.

Моделирование стека.

Требуется написать программу, моделирующую работу стека на базе **массива**. Для этого необходимо:

1) Реализовать **класс** *CustomStack,* который будет содержать перечисленные ниже методы. Стек должен иметь возможность хранить и работать с типом данных ***int.***

Объявление класса стека:

```
class CustomStack {
public:
// методы push, pop, size, empty, top + конструкторы, деструктор
private:
// поля класса, к которым не должно быть доступа извне
protected:  // в этом блоке должен быть указатель на массив данных
   int* mData;
};
```

Перечень методов класса стека, которые должны быть реализованы:

- ***void push(int val)*** - добавляет новый элемент в стек
- ***void pop()*** - удаляет из стека последний элемент
- ***int top()*** - возвращает верхний элемент
- ***size_t size()*** - возвращает количество элементов в стеке
- ***bool empty()*** - проверяет отсутствие элементов в стеке
- ***extend(int n*)** - расширяет исходный массив на n ячеек

2) Обеспечить в программе считывание из потока ***stdin*** последовательности команд (каждая команда с новой строки), в зависимости от которых программа выполняет ту или иную операцию и выводит результат ее выполнения с новой строки.

Перечень команд, которые подаются на вход программе в ***stdin***:

- ***cmd_push n*** - добавляет целое число n в стек. Программа должна вывести ***"ok"***
- ***cmd_pop*** - удаляет из стека последний элемент и выводит его значение на экран
- ***cmd_top*** - программа должна вывести верхний элемент стека на экран не удаляя его из стека
- ***cmd_size*** - программа должна вывести количество элементов в стеке
- ***cmd_exit*** - программа должна вывести ***"bye"*** и завершить работу

Если в процессе вычисления возникает ошибка (например вызов метода ***pop*** или ***top*** при пустом стеке), программа должна вывести ***"error"*** и завершиться.

Примечания:

1. Указатель на массив должен быть *protected.*
2. Подключать какие-то заголовочные файлы не требуется, всё необходимое подключено.
3. Предполагается, что пространство имен *std* уже доступно.
4. Использование ключевого слова *using* также не требуется.
5. Методы не должны выводить ничего в консоль.

Выполнение работы.

Был реализован **класс** *CustomStack,* который содержит перечисленные выше методы. Стек имет возможность хранить и работать с типом данных ***int.***

Перечень методов класса стека, которые были реализованы:

- ***void push(int val)*** - добавляет новый элемент в стек
- ***void pop()*** - удаляет из стека последний элемент
- ***int top()*** - возвращает верхний элемент
- ***size_t size()*** - возвращает количество элементов в стеке
- ***bool empty()*** - проверяет отсутствие элементов в стеке
- ***extend(int n*)** - расширяет исходный массив на n ячеек

Разработанный программный код см. в приложении А.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

Входные данные	Выходные данные	Комментарии
cmd_push 1 cmd_top cmd_push 2 cmd_top cmd_pop cmd_size cmd_pop cmd_size cmd_exit	ok 1 ok 2 2 1 1 0 bye	Результат корректен

Выводы.

Были изучены основы языка C++, рассмотрена работа динамических структур данных. В качестве практического задания был написан стек на основе массива и продемонстрирована его работа.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include <iostream>
#include <cstring>
#include <cstdlib>
#define BASE_SIZE 100


using std::cout;
using std::endl;


class CustomStack{
	public:
		CustomStack(){
			this->mData = NULL;
			this->buffer_size = 0;
			this->count_elem = 0;
		}

		~CustomStack(){
			delete[] this->mData;
		}

		void push(int val){
			if (this->count_elem == this->buffer_size){
				extend(BASE_SIZE);
			}
			this->mData[this->count_elem++] = val;
		}
```

```
        void pop(){
            this->count_elem--;
        }

        int top(){
            return this->mData[count_elem-1];
        }

        size_t size(){
            return this->count_elem;
        }

        bool empty(){
            return this->count_elem == 0;
        }

    Private:
        void extend(int n){
            this->buffer_size += n;
            int *tmp = new int [this->buffer_size];
            memcpy(tmp, this->mData, sizeof(int)*this->count_elem);
            delete[] this->mData;
            mData = tmp;
        }
        int buffer_size;
        int count_elem;

    protected:
        int *mData;
};
```

```
void free_all(char **cmd_arr, int n){
	for (int i = 0; i < n; i++){
		delete[] cmd_arr[i];
	}
}
int main(){
	CustomStack stack;
	char **cmd_arr = new char* [BASE_SIZE];
	char *inp_str = new char [BASE_SIZE];
	fgets(inp_str, BASE_SIZE, stdin);
	int i = 0;
	while (strcmp(inp_str, "cmd_exit\n\0")){
		cmd_arr[i] = (char *)malloc(BASE_SIZE*sizeof(char));
		strcpy(cmd_arr[i++], inp_str);
		fgets(inp_str, BASE_SIZE, stdin);
	}
	cmd_arr[i] = (char *)malloc(BASE_SIZE*sizeof(char));
	strcpy(cmd_arr[i++], inp_str);
	delete[] inp_str;
	for (int j = 0; j < i; j++){
		char *p = strtok(cmd_arr[j], " \n\0");
		if (!strcmp(p, "cmd_push")){
			char *p = strtok(NULL, " \n\0");
			int n = atoi(p);
			stack.push(n);
			cout << "ok" << endl;
			continue;
		}
		if (!strcmp(cmd_arr[j], "cmd_pop")){
			if (stack.empty()){
```

```
                cout << "error" << endl;
                return 0;
            }
            cout << stack.top() << endl;
            stack.pop();
            continue;
        }
        if (!strcmp(cmd_arr[j], "cmd_top")){
            if (stack.empty()){
                cout << "error" << endl;
                return 0;
            }
            cout << stack.top() << endl;
            continue;
        }
        if (!strcmp(cmd_arr[j], "cmd_size")){
            cout << stack.size() << endl;
            continue;
        }
        if (!strcmp(cmd_arr[j], "cmd_exit")){
            cout << "bye" << endl;
            break;
        }
    }
    free_all(cmd_arr, i);
    return 0;
}
```