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. 0382 _____ Гудов Н.Р.

Преподаватель _____ Берленко Т.А.

Санкт-Петербург
2021

Цель работы.

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

Основные теоретические положения.

Стек - это структура данных, в которой хранятся элементы в виде последовательности, организованной по принципу LIFO (Last In — First Out). Такую структуру данных можно сравнить со стопкой тарелок или магазином автомата. Стек не предполагает прямого доступа к элементам и список основных операций ограничивается операциями помещения элемента в стек и извлечения элемента из стека. Их принято называть PUSH и POP соответственно. Также, обычно есть возможность посмотреть на верхний элемент стека не извлекая его (TOP) и несколько других функций, таких как проверка на пустоту стека и некоторые другие.

Выполнение работы.

Стек реализован на классе CustomStack.

Имеются следующие методы класса:

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

Пользователь вводит команду, смысл которой проверяется в цикле. Бессмысленные команды пропускаются, верные команды выполняются.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	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	Работает верно

Выводы.

В ходе лабораторной работы была разработана программа, реализованная на базе стека. Отработаны основные положения по работе с динамическими структурами данных.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include <iostream>
#include <cstring>

using namespace std;

class CustomStack {
public:
    CustomStack(){
        ssize = 0;
        mData = new int[0];
    }

    ~CustomStack(){
        delete[] mData;
    }

    void push(int val){
        extend(1);
        mData[ssize] = val;
        ssize++;
    }

    void pop(){
        cout << mData[ssize - 1] << endl;
        mData[ssize - 1] = 0;
        ssize--;
    }

    int top(){
        return mData[ssize - 1];
    }

    size_t size(){
        return ssize;
    }

    bool empty(){
        if(ssize == 0)
            return true;
        else
            return false;
    }


    void extend(int n){
        mData = (int*) realloc(mData, n * sizeof(int));
    }

private:
    int ssize;
```

```
protected:
    int* mData;
};

int main(){
    CustomStack stack;
    char com[20];
    while (true){
        cin >> com;
        if (strcmp(com,"cmd_push")==0){
            int val;
            cin >> val;
            stack.push(val);
            cout << "ok" << endl;
            continue;
        }
        if (strcmp(com,"cmd_pop")==0){
            if(stack.empty() == 1){
                cout << "error";
                exit(0);
            }
            stack.pop();
            continue;
        }
        if (strcmp(com,"cmd_top")==0){
            if(stack.empty() == 1){
                cout << "error";
                exit(0);
            }
            cout << stack.top() << endl;
            continue;
        }
        if (strcmp(com,"cmd_size")==0){
            cout << stack.size() << endl;
            continue;
        }
        if (strcmp(com,"cmd_exit")==0){
            cout << "bye" << endl;
            exit(0);
        }
    }
    return 0;
}
```