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. 3342 _____ Роднов И.С.

Преподаватель _____ Глазунов С.А.

Санкт-Петербург
2024

Цель работы

Целью данной лабораторной является изучение и работа с языком C++. Целью также является работа с динамическими структурами данных и их создание.

Задание

Вариант 3.

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

В программе создан класс CustomStack, в котором есть приватное поле mSize и mCapacity которые отвечают за текущий размер и общую вместительность стека. Также есть protected поле mData, которое отвечает за данные в стеке. В классе реализованы 6 методов: push, pop, empty, size, top, extend в соответствии с заданием. В main обрабатываются команды cmd_... каждая из которых вызывает определенный метод класса, а так же реализуются проверки.

Тестирование

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные
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

Выводы

Была реализована программа на языке C++, которая работает с динамической структурой данных – стек на массиве. Также созданы методы для работы с этой структурой данных.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
class CustomStack {
public:
    CustomStack() : mSize(0), mData(nullptr), mCapacity(0){}
    CustomStack(const CustomStack& stack): mSize(stack.mSize),
mCapacity(stack.mCapacity), mData(nullptr){
        if(stack.mData){
            mData = new int[mCapacity];
            if (!mData) {
                cerr << "Memory allocation failed" << endl;
                exit(0);
            }
            for(size_t i = 0; i < mSize; ++i){
                mData[i] = stack.mData[i];
            }
        }
    }

    ~CustomStack(){
        delete[] mData;
    }

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

    void pop(){
        mSize--;
    }

    int top(){
        int top_el = mData[mSize - 1];
        return top_el;
    }

    size_t size(){
        return mSize;
    }

    bool empty(){
        return mSize == 0;
    }

    void extend(int n){
        mCapacity = mCapacity + n;
        int* nData = new int[mCapacity];
        if (!nData) {
            cerr << "Memory allocation failed" << endl;
```

```
            exit(0);
        }
        for(size_t i = 0; i < mSize; ++i){
            nData[i] = mData[i];
        }
        delete[] mData;
        mData = nData;
    }

private:
    int mSize;
    int mCapacity;

protected:
    int* mData;
};

int main() {
    CustomStack myStack;
    string cmd;
    int n;

    while(cin >> cmd){
        if(cmd == "cmd_push"){
            cin >> n;
            myStack.push(n);
            cout << "ok" << endl;
        }
        else if(cmd == "cmd_pop"){
            if(myStack.empty()){
                cout << "error" << endl;
                return 0;
            }
            else{
                cout << myStack.top() << endl;
                myStack.pop();
            }
        }
        else if(cmd == "cmd_top"){
            if(myStack.empty()){
                cout << "error" << endl;
                return 0;
            }
            else{
                cout << myStack.top() << endl;
            }
        }
        else if(cmd == "cmd_size"){
            cout << myStack.size() << endl;
        }
        else if(cmd == "cmd_exit"){
            cout << "bye" << endl;
            break;
        }
    }
    return 0;
}
```