

**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. 3341

Бойцов В.А.

Преподаватель

Глазунов С.А.

Санкт-Петербург

2024

## **Цель работы**

Целью работы является изучение основных механизмов языка C++ путём разработки структур данных стека и очереди на основе динамической памяти.

Для достижения поставленной цели требуется решить следующие задачи:

- Ознакомиться со структурами данных стека и очереди, особенностями их реализации;
- Изучить и использовать базовые механизмы языка C++, необходимые для реализации стека и очереди;
- Реализовать индивидуальный вариант стека в виде C++ класса, его операций в виде функций этого класса, ввод и вывод данных программы.

## 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

Создаётся класс *CustomStack* со следующими полями:

- *int\* mData* – указатель на массив данных (*protected*);
- *size\_t mLength* – количество заполненных данных в массиве (*private*);
- *size\_t mSize* – размер выделенной под массив данных памяти (*private*);

В классе реализованы конструктор *CustomStack()*, инициализирующий поля класса, и следующие методы:

*void push(int val)* – метод добавляет в стек данные из *val*, при необходимости расширяя массив *mData*;

*void pop()* – метод удаляет из стека последний элемент, изменяя размер массива. Если стек пуст, выбрасывается исключение *logic\_error*;

*int top()* – метод возвращает последний элемент стека. Если стек пуст, выбрасывается исключение *logic\_error*;

*size\_t size()* – возвращает количество элементов в стеке;

*bool empty()* – проверяет, является ли стек пустым. Возвращает *false*, если стек пуст, иначе *true*;

*void extend(int n)* – расширяет массив на *n* элементов в памяти.

Для каждой пользовательской команды реализована своя одноименная функция. В функциях *cmd\_pop()* и *cmd\_top()* с помощью *try - catch* происходит обработка исключений, вызванных, например, применением функций при пустом стеке. В такой случае программа выводит сообщение об ошибке и завершает свою работу.

Для последовательного считывания команд и вызова функций реализована функция *void cmdExecution(CustomStack &stack)*. Она принимает на вход ссылку на стек. Внутри функции в цикле *while()* считываются команды и вызываются соответствующие им функции. Если вызвана функция *cmd\_exit()*, программа завершает свою работу.

Разработанный программный код см. в приложении А.

## Тестирование

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	cmd_push 864 cmd_push 998 cmd_push 833 cmd_pop cmd_push -7 cmd_push 854 cmd_push 654 cmd_size cmd_pop cmd_pop cmd_exit	ok ok ok 833 ok ok ok 5 654 854 bye	Стандартный тест на общую работоспособность программы
2.	cmd_push 864 cmd_size cmd_pop cmd_pop cmd_exit	ok 1 864 error	Тест на ситуацию, вызывающую ошибку (программа корректно завершает свою работу)
3.	cmd_exit	bye	Тест на единственную команду завершения работы

## **Выводы**

В ходе выполнения работы были изучены основные механизмы языка C++. Были изучены такие структуры данных, как стек и очередь, освоена работа с их функционалом. Для реализации стека в виде класса на языке C++ были изучены базовые механизмы языка C++. Была написана программа, реализующая вариант стека в виде класса с функциями этого класса, был настроен ввод и вывод программы.

## ПРИЛОЖЕНИЕ А

### ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
class CustomStack {
public:
    CustomStack()
    {
        mData=new int[1];
        mLength=0;
        mSize=1;
    }
    void push(int val)
    {
        if(mLength>=mSize)
        {
            int* newData = new int[mSize+1];
            memcpy(newData, mData, sizeof(int)*mSize);
            mSize++;
            delete[] mData;
            mData=newData;
        }
        mData[mLength++]=val;
    }
    void pop()
    {
        if(mLength<=0)
            throw std::logic_error("pop() called on empty stack!");
        mData[--mLength]=0;
        int* newData=new int[--mSize];
        memcpy(newData, mData, sizeof(int)*mSize);
        delete[] mData;
        mData=newData;
    }
    int top()
    {
        if(mLength<=0)
            throw std::logic_error("top() called on empty stack!");
        return mData[mLength-1];
    }
    size_t size()
    {
        return mLength;
    }
    bool empty()
    {
        return mLength==0;
    }
    void extend(int n)
    {
        int* newData = new int[mSize+n];
        memcpy(newData, mData, sizeof(int)*mSize);
        mSize+=n;
        delete[] mData;
        mData=newData;
    }
}
```



```

    ~CustomStack()
    {
        delete[] mData;
        mSize=0;
        mLength=0;
    }
protected:
    int* mData;
private:
    size_t mLength;
    size_t mSize;
};

void cmd_push(CustomStack &stack, int n)
{
    stack.push(n);
    std::cout<<"ok"<<std::endl;
}
void cmd_pop(CustomStack &stack)
{
    int val;
    try
    {
        val=stack.top();
        stack.pop();
    }
    catch(const std::exception& e)
    {
        std::cout<<"error"<<std::endl;
        exit(0);
    }
    std::cout<<val<<std::endl;
}
void cmd_top(CustomStack &stack)
{
    int val;
    try
    {
        val=stack.top();
    }
    catch(const std::exception& e)
    {
        std::cout<<"error"<<std::endl;
        exit(0);
    }
    std::cout<<val<<std::endl;
}
void cmd_size(CustomStack &stack)
{
    std::cout<<stack.size()<<std::endl;
}
void cmd_exit()
{
    std::cout<<"bye";
    exit(0);
}

void cmdExecution(CustomStack &stack)

```

```

{
    std::string newCommand;
    while(true)
    {
        std::cin>>newCommand;
        if (newCommand=="cmd_push")
        {
            int data;
            std::cin>>data;
            cmd_push(stack, data);
        }
        if(newCommand=="cmd_pop")
            cmd_pop(stack);
        if(newCommand=="cmd_top")
            cmd_top(stack);
        if(newCommand=="cmd_size")
            cmd_size(stack);
        if(newCommand=="cmd_exit")
            cmd_exit();
    };
}

int main() {
    CustomStack stack;
    cmdExecution(stack);
    return 0;
}

```