

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

**ОТЧЕТ по**  
**лабораторной работе №3**  
**по дисциплине «Алгоритмы и структуры данных»**  
**Тема: Линейные структуры данных: стек, очередь**

Студент гр. 8304

Самакаев Д.И.

Преподаватель

Фирсов М.А.

Санкт-Петербург

2019

## **Вариант 7-в**

### **Цель работы.**

Изучить принципы работы структур данных, таких как стек и очередь.

### **Постановка задачи.**

Реализовать вычисление значения выражений на базе вектора при помощи стека.

Программе подаётся строка, состоящая из цифр и функций М и m, функции максимума и минимума соответственно. Программа должна вычислять значение выражения используя стек.

### **Описание алгоритма.**

На стек кладутся поочерёдно символы входной строки, исключая символы скобок, запятые и пробелы. Затем со стека поочерёдно снимаются аргументы функций m и M, символы m и M, означающие применяемые функции, затем на стек кладётся значение, возвращаемое функцией m или M. Функция работает рекурсивно пока длина вектора, на базе которого реализован стек не станет равна единице. При выполнении этого условия на стеке останется единственная цифра, которая и будет значением поданного на вход выражения.

### **Спецификация программы.**

Программа предназначена для вычисления значения выражения методом рекурсии при помощи стека, основанном на векторе.

Программа написана на языке C++. Входные данные подаются в виде строк текстового файла или консольным вводом.

### **Описание функций.**

1. `void fill_stack(Stack<char>& stack, std::string &str)`

Заполняет стек stack, используя введенную строку str.

2. `void calculate(Stack<char>& stack, size_t depth)`

Производит вычисление значения выражения, положенного на стек stack.

### **Вывод.**

Был получен опыт работы со стеком. Была реализована программа вычисления значения выражения с использованием стека, основанном на векторе.

## Приложение.

### 1)Тестирование.

```
M(m(m(m(2, M(3, 6)), 4), 5), 9) = 9
m(M(2,3),m(4,6)) = 3
m(2,3) = 2
M(M(M(2, M(4, 7)), 7), 8) = 8
2 = 2
M(m(7, 4), M(3, 5)) = 5
(7, 4) is incorrect expression!
M() is incorrect expression!
m(2, 7, 8) is incorrect expression!
m is incorrect expression!
```

M(m(m(m(2, M(3, 6)), 4), 5), 9)	9
m(M(2,3),m(4,6))	3
m(2,3)	2
M(M(M(2, M(4, 7)), 7), 8)	8
2	2
M(m(7, 4), M(3, 5))	5
(7, 4)	Is incorrect expression
M()	Is incorrect expression
m(2, 7, 8)	Is incorrect expression
m	Is incorrect expression

### 2)Исходный код.

#### Vector\_stack.h

```
#pragma once

#include <iostream>
#include <fstream>
#include <string>
#include <cctype>
#include <algorithm>

template<typename T>
T* resize(T*& old_data, size_t old_size, size_t new_size) {
    T* new_data = new T[new_size];
```

```

        for (size_t i = 0; i < std::min(old_size, new_size); i++) {
            new_data[i] = old_data[i];
        }
        delete old_data;

        return new_data;
    }

template<typename T>
class vector
{
public:
    vector() = default;

    bool is_empty() {
        return _veclength == 0;
    }

    void erase(size_t start_erasing_inex, int erasing_length = -1)
    {
        if (erasing_length == -1) {
            for (size_t i = start_erasing_inex; i < _veclength;
i++){
                _data[i] = _data[i + 1];
            }
            _veclength -= (_veclength - start_erasing_inex);
        }
        else {
            for (size_t i = start_erasing_inex; i <
start_erasing_inex + erasing_length; i++)
                _data[i] = _data[i + 1];
            _veclength -= erasing_length;
        }
    }

    size_t get_length() {
        return _veclength;
    }

    void operator+(T &element){
        _veclength++;
        if (_veclength > _vector_allocated_size) {
            size_t allocated_size = 10;
            _data = resize(_data, _vector_allocated_size,
_vector_allocated_size + allocated_size);
            _vector_allocated_size += allocated_size;
        }
        _data[_veclength - 1] = element;
    }

    T& operator[](size_t element_index) {
        return _data[element_index];
    }

    friend std::ostream& operator<<(std::ostream& os, vector<T>&
stack) {
        for (size_t i = 0; i < stack.get_length(); i++) {
            os << stack._data[i] << " ";

```

```

        }
        return os;
    }

    ~vector() {
        delete[] _data;
    }

private:
    T* _data = nullptr;
    size_t _veclength = 0;
    size_t _vector_allocated_size = 0;
};

template<typename T>
class Stack {
public:
    Stack() = default;

    void pop() {
        if (_stack.is_empty())
            throw std::out_of_range("Stack is empty");
        _stack.erase(_stack.get_length() - 1, 1);
    }

    void push(T argument){
        _stack + argument;
    }

    T& get_high() {
        if (!_stack.is_empty())
            return _stack[_stack.get_length() - 1];
        throw std::out_of_range("Stack is empty");
    }

    bool is_empty() {
        return _stack.is_empty();
    }

    size_t get_length() {
        return _stack.get_length();
    }

    friend std::ostream& operator<<(std::ostream& os, Stack<T>&
stack) {
        return os << stack._stack;
    }

private:
    vector<T> _stack;
};

```

## Lab3.cpp

```

#include "vector_stack.h"

void fill_stack(Stack<char>& stack, std::string &str){
    for (size_t i = 0; i < str.length(); i++) {
        if (str[i] == '(' || str[i] == ')' || str[i] == ' ' ||
str[i] == ',')
            continue;
        else stack.push(str[i]);
    }
}

void calculate(Stack<char>& stack, size_t depth) {
    if (stack.get_length() == 1 || stack.get_length() == 2)
        return;

    int right_arguement = stack.get_high();
    stack.pop();

    int left_arguement = stack.get_high();
    stack.pop();

    if (stack.get_high() == 'm') {
        stack.pop();

        if (stack.is_empty() || isdigit(stack.get_high())) {
            stack.push(std::min(left_arguement,
right_arguement));

            calculate(stack, depth + 1);
        }
        else stack.push(std::min(left_arguement,
right_arguement));
    }
    else if (stack.get_high() == 'M') {
        stack.pop();

        if (stack.get_length() == 0 ||
isdigit(stack.get_high())) {
            stack.push(std::max(left_arguement,
right_arguement));

            calculate(stack, depth + 1);
        }
        else stack.push(std::max(left_arguement,
right_arguement));
    }
    else {
        stack.push(left_arguement);
    }
}

```

```

        calculate(stack, depth + 1);

        stack.push(right_arguement);

        calculate(stack, depth + 1);

    }

}

void file_input(char* &argv) {

    std::ifstream file;
    std::string testfile = argv;

    file.open(testfile);
    if (!file.is_open())
        std::cout << "Error! File isn't open" << std::endl;

    std::string str;

    while (!file.eof()) {

        Stack<char> stack;
        size_t depth = 0;

        getline(file, str);

        fill_stack(stack, str);
        calculate(stack, depth);

        if (stack.get_length() == 1 &&
            isdigit(stack.get_high()))
            std::cout << str << " = " << stack << std::endl;
        else std::cout << str << " is incorrect expression!" <<
std::endl;
    }

}

void console_input()
{
    Stack<char> stack;
    size_t depth = 0;

    std::string str;

    std::cout << "Enter the pattern to calculate" << std::endl;
    std::cin >> str;

    fill_stack(stack, str);
    calculate(stack, depth);

    if (stack.get_length() == 1 && isdigit(stack.get_high()))
        std::cout << str << " = " << stack << std::endl;
    else std::cout << str << " is incorrect expression!" <<
std::endl;
}

```



```
//аргументом передаётся имя тестового файла
int main(int argc, char** argv)
{
    if (argc == 1)
        console_input();
    else if (argc == 2)
        file_input(argv[1]);
    else
        std::cout << "too much arguments" << std::endl;

    return 0;
}
```