

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

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

Студент гр.8304

\_\_\_\_\_

Ястребов И.М.

Преподаватель

\_\_\_\_\_

Фирсов М.А.

Санкт-Петербург

2019

## Задание.

### Вариант 11-а-в

Рассматривается выражение следующего вида:

$\langle \text{выражение} \rangle ::= \langle \text{терм} \rangle \mid \langle \text{терм} \rangle + \langle \text{выражение} \rangle \mid$   
 $\langle \text{терм} \rangle - \langle \text{выражение} \rangle$   
 $\langle \text{терм} \rangle ::= \langle \text{множитель} \rangle \mid \langle \text{множитель} \rangle * \langle \text{терм} \rangle$   
 $\langle \text{множитель} \rangle ::= \langle \text{число} \rangle \mid \langle \text{переменная} \rangle \mid ( \langle \text{выражение} \rangle )$   
 $\langle \text{множитель} \rangle ^ \langle \text{число} \rangle$   
 $\langle \text{число} \rangle ::= \langle \text{цифра} \rangle$   
 $\langle \text{переменная} \rangle ::= \langle \text{буква} \rangle$

Требуется:

- а) вычислить как целое число значение выражения (без переменных), записанного в постфиксной форме в заданном текстовом файле postfix;

## Цель работы.

Решить полученную задачу, реализовав стек на основе массива.

## Описание алгоритма.

Программа создает объект реализованного класса стека на массиве — далее, анализируя входную строку, на стеке хранятся и извлекаются текущие значения арифметических подвыражений. Используется стандартный алгоритм вычисления арифметического выражения, записанного в постфиксной форме.

## Описание функций и структур данных.

- `Class Vec_stack;` - класс, реализация стека на основе динамического массива, с методами `push`, `pop`, `isempty`, `top`
- `EvaluateExpression(std::string &expression)` – функция, вычисляющая значения постфиксного выражения
- `IsOperator(char &c)` определяет, является ли символ знаком операции
- `PerformOperation(char&, int&, int&)` совершает арифметическую операцию над двумя операндами

## Выводы.

В ходе выполнения данной работы был написан класс, стек на основе динамического массива.

## Тестирование

```
Expression : 2 5 10 3 - 2 2 1 ^ + ^ * +
Result : 12007

Expression : 23+
wrong input
Failed to calculate

Expression : 69 228 - 420 1337 + * 322 -
Result : -279685

Expression : 2 2 2 ^ ^
Result : 16

Expression : 2 12 3 2 1 + * - ^
Result : 8

Expression : 2 3 4 ^ - 3 3 - 3 + 3 - 3 + 3 + *
Result : -237

Expression : *wrong test here*
wrong input
Failed to calculate

Expression : 200 30 + 3 2 ^ +
Result : 239

C:\Users\vanka\ADS-8304\Yastrebov\lab3>
```

## Пример вывода программы

## Исходный код

```
/*
11 Рассматривается выражение следующего вида:
< выражение > ::= < терм > | < терм > + < выражение > | < терм > - < выражение >
< терм > ::= < множитель > | < множитель > * < терм >
< множитель > ::= < число > | ( < выражение > ) | < множитель > ^ < число >
< число > ::= < цифра >
*/
```

```
#include "pch.h"
#include <iostream>
#include <string>
#include <fstream>
#include "Vec_Stack.hpp"
```

```
nInt EvaluatePostfix(std::string &expression);
```

```
nInt PerformOperation(char &operation, int &operand1, int &operand2);
```

```

bool IsOperator(char &C);

int main(int argc, char* argv[])
{
    std::ifstream input;

    if(argc == 1)
        input.open("postfix.txt");

    else
        input.open(argv[1]);

    if (!input)
    {
        std::cout << "Couldn't open source file" << std::endl;

        return 0;
    }

    std::string expression;

    while (std::getline(input, expression)) {

        nInt result = EvaluatePostfix(expression);

        if (result != std::nullopt)
            std::cout << "_____\\nExpression : " << expression << "\\nResult : " <<
            result.value() << std::endl;

        else
            std::cout << "Failed to calculate" << std::endl;
    }

    return 0;
}

nInt EvaluatePostfix(std::string &expression)
{
    Vec_Stack vecStack;

    for (size_t i = 0; i < expression.size(); ++i)

```

```

{
if (expression[i] == ' ')
    continue;

if (IsOperator(expression[i])) {
    nInt rightOperand = vecStack.pop();

    if (rightOperand == std::nullopt) {
        std::cout << "YOU DIED" << std::endl;

        return std::nullopt;
    }

    nInt leftOperand = vecStack.pop();

    if (leftOperand == std::nullopt) {
        std::cout << "YOU DIED" << std::endl;

        return std::nullopt;
    }

    nInt result = PerformOperation(expression[i], leftOperand.value(),
rightOperand.value());

    if (result == std::nullopt)
        return std::nullopt;

    else
        vecStack.push(result.value());
}

else if (isdigit(expression[i])) {
    int operand(0);

    while (i < expression.size() && isdigit(expression[i])) {
        operand = operand * 10 + (expression[i] - '0');

        ++i;
    }

    --i;

    vecStack.push(operand);
}

```

```

}

else {
    std::cout << "Incorrect input" << std::endl;

    return std::nullopt;
}

}

return vecStack.pop();
}

bool IsOperator(char &c)
{
return (c == '+' ||
c == '-' ||
c == '*' ||
c == '^');
}

nInt PerformOperation(char &operation, int &operand1, int &operand2)
{
if (operation == '+')
return operand1 + operand2;

else if (operation == '-')
return operand1 - operand2;

else if (operation == '*')
return operand1 * operand2;

else if (operation == '^')
return (int)pow(operand1, operand2);

else
return std::nullopt;
}

```

---

**Vec\_Stack.hpp :**

**#pragma once**

**#include <optional>**

**constexpr auto** DEFAULT\_SIZE = 1<<8;

```

typedef std::optional<int> nInt;

class Vec_Stack {
private:
    int* stack;
    size_t index;
    size_t size;

public:
    Vec_Stack(size_t startSize = DEFAULT_SIZE) {
        stack = new int[startSize];
        size = startSize;
        index = -1;
    }

    void push(int element) {

        if (index == (size - 1))
        {
            int* newStack = new int[size + DEFAULT_SIZE];

            std::copy(&stack[0], &stack[size], &newStack[0]);

            delete[] stack;

            stack = newStack;

            size += DEFAULT_SIZE;

            stack[++index] = element;
        }

        else {
            stack[++index] = element;
        }
    }
}

```

```

nInt pop() {
    if (index == -1)
        return std::nullopt;
    else
        return stack[index--];
}

bool isEmpty() {
    return (index == -1);
}

nInt top() {
    if (index != -1)
        return stack[index];
    else
        return std::nullopt;
}

~Vec_Stack() {
    delete[] stack;
}

};

```