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: Стек

Студент гр. 7383 _____ Лосев М. Л.

Преподаватель _____ Размочаева Н. В.

Санкт-Петербург
2018

# СОДЕРЖАНИЕ

# 1. ЦЕЛЬ РАБОТЫ

Цель работы: Познакомиться с часто используемыми на практике линейными структурами данных, обеспечивающими доступ к элементам последовательности только через её начало и конец, и способами реализации этих структур, освоить на практике использование стека, очереди и дека для решения задач.

Формулировка задачи: вычислить как целое число значение выражения (без переменных), записанного в префиксной форме.

## 2. РЕАЛИЗАЦИЯ ЗАДАЧИ

### 2.1 ИСПОЛЬЗУЕМЫЕ СТРУКТУРЫ ДАННЫХ

*Stack* – шаблонный класс с шаблонным параметром *Elem* (тип хранимых элементов). Представляет из себя стек в ссылочной реализации.

Поля класса *Stack:*

- *node* – структура, содержащая два поля: указатель *hd* на элемент типа *Elem,* указатель *tl* на следующий элемент типа *node* – «хвост»; и конструктор *node*, обнуляющий оба поля.
- *topOfStack* – указатель на вершину стека.

Методы класса *Stack:*

- Конструктор *Stack,* обнуляющий указатель *topOfStack* на вершину стека.
- *top* – если стек не пуст, возвращает элемент, который хранит вершина сткеа.
- *pop* – если стек не пуст, удаляет его вершину (при этом вершиной становится следующий за ней элемент).
- *pop2* – если стек не пуст, удаляет его вершину (при этом вершиной становится следующий за ней элемент), возвращает значения элемента, который хранила удаляемая вершина.
- *push* – добавляет в стек элемент. Бывшая вершина после добавления нового элемента идет за ним, а сам новый элемент становится вершиной.
- *isNull* – возвращает *true*, если стек пуст, или *false* во всех остальных случаях.
- d*estroy* – освобождает место, занятое стеком.

### 2.2 ИСПОЛЬЗУЕМЫЕ ФУНКЦИИ

*IsDigit* – принимает символ, возвращает *true*, если этот символ является цифрой, или *false*, если не является.

*IsOperator* – принимает символ, возвращает *true*, если этот символ является оператором, или *false*, если не является.

*CalcPrefixExp* – приимает строку (выражение в префиксной форме), вычисляет значение выражения как целое число и возвращает его.

# 3. ТЕСТИРОВАНИЕ

## 3.1 ПРОЦЕСС ТЕСТИРОВАНИЯ

Программа собрана в операционной системе Ubuntu 18.04 компилятором gcc. В других ОС тестирование не проводилось.

## 3.1 Результаты тестирования

Тестовые случаи представлены в приложении Б. Ошибки выявить не удалось. По результатам тестирования было показано, что задача выполнена.

# 4 . ВЫВОД

В ходе работы была написана программа на языке C++, вычисляющая значение выражения, записанного в префиксной форме. Был получен опыт в использовании стека в ссылочной реализации.

# ПРИЛОЖЕНИЕ А
# ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл MakeFile:

```
all: main2.o
	g++ main2.o -o main2
main2.o: main2.cpp st_interf2.h
	g++ -c main2.cpp
clean:
	rm *.o main2
```

Файл st_interf2.cpp:

```
#ifndef ST_INTERF2_H_INCLUDED
#define ST_INTERF2_H_INCLUDED

// Интерфейс АТД "Стек" (ссылочная реализация в динамической памяти)с шаблоном класса
// И интерфейс, и реализация в одном заголовочном файле

#include <iostream>
#include <cstdlib>

using namespace std;

namespace st_modul2
{

//-------------------------------------
	template <class Elem>
	class Stack
	{
		private:
		struct node
		{ //
			Elem *hd;
			node *tl;
			// constructor
			node ()
				{
					hd = NULL;
```

```
                        tl = NULL;
                    };
        };// end node

        node *topOfStack;

        public:
        Stack ()
            { topOfStack = NULL;
            }//;

//-------------------------------------
    Elem top (void) //
    {// PreCondition: not null
        if (topOfStack == NULL) { cerr << "Error: top(null) \n"; exit(1); }
        else return *topOfStack->hd;
    }
//-------------------------------------
    void pop (void)//
    {// PreCondition: not null
        if (topOfStack == NULL) { cerr << "Error: pop(null) \n"; exit(1); }
        else
        {   node *oldTop = topOfStack;
            topOfStack = topOfStack->tl;
            delete oldTop->hd;
            delete oldTop;
        }
    }
//-------------------------------------
    Elem pop2(void)//
    {// PreCondition: not null
        if (topOfStack == NULL) { cerr << "Error: pop(null) \n"; exit(1); }
        else
        {   node *oldTop = topOfStack;
            Elem r = *topOfStack->hd;
            topOfStack = topOfStack->tl;
            delete oldTop->hd;
            delete oldTop;
            return r;
        }
    }
```

```
//-------------------------------------
      void push (const Elem &x)//
      {     node *p;
            p = topOfStack;
            topOfStack = new node;
            if ( topOfStack != NULL) {
                  topOfStack->hd = new Elem;
                  *topOfStack->hd = x;

                  topOfStack->tl = p;
            }
            else {cerr << "Memory not enough\n"; exit(1);}
      }
//-------------------------------------
      bool isNull(void)//
      {     return (topOfStack == NULL) ;
      }
//-------------------------------------
      void destroy (void)//
      {     while ( topOfStack != NULL) {
                  pop();
            }
      }
      };
}

#endif
```

Файл main2.cpp:

```
// Программа клиент вычисляет арифметическое выражение, заданное в префиксной форме
// Ссылочная реализация в динамической (связанной) памяти
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <cmath>
#include <cstring>

#include "st_interf2.h"
```

```
using namespace std;
using namespace st_modul2;

bool IsDigit( char c )
{
	return ( (c <= '9') && (c >= '0'));
}

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

int CalcPrefixExp(string expr)
{
	Stack <int> numbers;
	int n = expr.length();

	for (int i = n - 1; i >= 0; i--)
	{	// бежим от конца выражения к началу
		if (IsOperator(expr[i]))
		{	// если попался оператор
			int x = numbers.pop2();
			int y = numbers.pop2();
			if (expr[i] == '+')
				numbers.push(x + y);
			if (expr[i] == '*')
				numbers.push(x * y);
			if (expr[i] == '-')
				numbers.push(x - y);
			if (expr[i] == '/')
				numbers.push(x / y);
			if (expr[i] == '^')
				numbers.push(pow(x, y));
		}
		if (IsDigit(expr[i])) //
		{	// если попалось число
			int dec_exp = 10;
			numbers.push(expr[i--] - '0');  // пушим младшую цифру
			while (IsDigit(expr[i]))
```

```
                {     // пока идут цифры числа, продолжаем его ввод
                      numbers.push(numbers.pop2() + dec_exp * (expr[i--] - '0'));
                      dec_exp *= 10;
                }
                if (expr[i] == '-') // если число отрицательное
                      numbers.push(-1 * numbers.pop2());
          }

    }

    int answ = numbers.pop2(); // сохраение ответа

    numbers.destroy();

    return answ;
}

void UserInterface ()
{
    int key;
      string expr;

    while (true) {
        cout << "Выбери действие:" << endl;
        cout << "0. Завершить выполнение;" << endl;
         cout << "1. Загрузить префиксное выражение из пользовательского файла;" <<
endl;
              cout << "2. Загрузить префиксное выражение из файла по умолчанию
(prefix.txt);" << endl;
        cout << "3. Ввести префиксное выражение с клавиатуры;" << endl;
        cin >> key;
        switch (key) {
            case 0:
                return;
                break;
            case 1:
                  {
                      string filename;
                      cout << "Введите имя файла: ";
```

```
                        cin.ignore(256, '\n');// игнорируем оставшиеся в cin после
ввода '1' символы
                        getline(cin, filename);
                        ifstream fin(filename);
                        fin >> noskipws;   // не пропускать пробелы
                        if (fin){
                              getline(fin, expr);
                              cout << expr << " = " << CalcPrefixExp(expr) << endl;
                        }
                        else cout << "Файл не открыт\n";
                  }
            break;
            case 2:
            {
                        ifstream fin("prefix.txt");
                        fin >> noskipws;   // не пропускать пробелы
                        if (fin){
                              getline(fin, expr);
                              cout << expr << " = " << CalcPrefixExp(expr) << endl;
                        }
                        else cout << "Файл не открыт\n";
            }
            break;

            case 3:
            {
             cout << "Введите выражение: ";
             cin.ignore(256, '\n');// игнорируем оставшиеся в cin после ввода '3'
символы
                        getline(cin, expr);
                        cout << expr << " = " << CalcPrefixExp(expr) << endl;
            }
            break;

            default : cout << "! - ...";
            break;
        };
    }
}

int main ()
```

```
{
	UserInterface();
	return (0);
}
```

# ПРИЛОЖЕНИЕ Б
# ТЕСТОВЫЕ СЛУЧАИ

| Входное выражение | Вывод программы | Корректность выполнения |
|---|---|---|
| + 100 500 | + 100 500 = 600 | да |
| + 100 / 100 4 | + 100 / 100 4 = 125 | да |
| - 169 ^ 13 2 = 0 | - 169 ^ 13 2 = 0 | да |
| - 100 500 | - 100 500 = -400 | да |
| * -3 14 | * -3 14 = -42 | да |
| + + ^ 10 2 10 * 2 * 3 10 | + + ^ 10 2 10 * 2 * 3 10  = 170 | да |