

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. Очередь. Дек

Студентка гр. 7383	_____	Маркова А. В.
Преподаватель	_____	Размочаева Н.В.

Санкт-Петербург

2018

Содержание

Цель работы	3
Реализация задачи	4
Тестирование	6
Выводы	7
Приложение А. Тестовые случаи	8
Приложение Б. Код программы.....	9

Цель работы

Цель работы: ознакомиться со структурой стека, создать его реализацию и изучить использование в практических задачах на языке программирования C++.

Формулировка задачи: перевести выражение, записанное в обычной (инфиксной) форме, в постфиксную. Вариант 11-в-д.

Реализация задачи

Для реализации стека на базе списка было принято создать класс с базовыми функциями стека. Структура класса приведена ниже.

```
class Stack {
private:
    struct mstack;
    mstack *head;
public:
    Stack() {
        head=NULL;
    };
    void pop();
    void push(T a);
    T top();
    bool stempty();
    void clear();
    ~Stack();
};
```

Метод `void pop()` убирает верхний элемент из стека, метод `void push(T a)` добавляет в стек один элемент, `T top()` возвращает последний элемент, а `bool stempty()` показывает пуст ли список. Конструктор инициализирует стек указателем на голову списка. Деструктор очищает память, выделенную под элементы стека.

В главной функции `main()` было реализовано пользовательское меню, где можно выбрать способ ввода данных: из файла или из терминала. Считывается целое число и при помощи оператора `switch()`, выполняется необходимое действие. При нажатие «1», программа копирует строку с файла, при нажатие «2» пользователь сам вводит нужную строку. При выборе «3» функция завершает свою работу. Если было введено другое значение, отличное от стандартных, то программа выведет сообщение об ошибке. Программа завершается при выборе «3», в противном случае – ожидает дальнейших указаний.

Функция `InfixToPostfix(string expression)` принимает на вход введенную строку. Пока строка не закончится, из нее считываются символы. Если встречается буква или цифра, то мы записываем ее в выражение, выделенное для постфиксной записи, а если оператор, то в стек, проверяя приоритеты операций.

Разберём для примера работы программы строку `a+b-c`:


1 шаг: <u>a</u> +b-c	
postfix="a"	stack[]
2 шаг: <u>+</u> b-c	
postfix="a"	stack[+]
3 шаг: <u>b</u> -c	
postfix="ab"	stack[+]
4 шаг: <u>-</u> c	
postfix="ab+"	stack[-]
5 шаг: <u>c</u>	
postfix="ab+c"	stack[-]
Ответ: postfix="ab+c-"	

Рисунок 1 – пример работы программы

Программа записывает в преобразованную строку буквы и цифры, а встречая оператор кладет его в стек, в следующий раз, встречая другой оператор, идет проверка приоритетов, если в стеке лежит знак больший по «весу», то он достается из стека и записывается в постфиксную форму, а текущий знак заносится в стек. Так произошло на 4 шаге программы (см. рис. 1).

Тестирование

Программа собрана в операционной системе Ubuntu 17.04, с использованием компилятора g++ версии 5.4.0 20160609. В других ОС и компиляторах тестирование не проводилось.

Программа может быть скомпилирована с помощью команды:

```
g++ -Wall <имя файла>.c
```

Тестовые случаи представлены в Приложении А.

Исходя из тестовых случаев можно заметить, что в первом тесте программа ведет себя неверно, поэтому была исправлена программа: учтено левое ассоциативное свойство.

После, тестовые случаи не выявили неправильного поведения программы, что говорит о том, что по результатам тестирования было показано, поставленная задача была выполнена.

Выводы

В ходе лабораторной работы были изучены и реализованы шаблоны классов. Получены навыки работы со стеком на основе списка. А так же была написана программа на языке C++, записывающая инфиксную форму в постфиксную.

ПРИЛОЖЕНИЕ А

Тестовые случаи

Ввод	Вывод	Верно?
2 a+b-c/k	Output = abck/-+	Нет
1 В файле: «a+b*(c-d/v)-s»	Output = a b c d v / - * + s -	Да
8	Incorrect pick!	Да
1 (файл не был создан)	Входной файл не может быть открыт!	Да
2 a+b-c/k 2 f*-	Output = a b + c k / - Error: incorrect input	Да
2 a*s+(w/j*s+f)-x 3	Output = as*wj/s*f++x- End!	Да
2 6-5/3*(3-1)+4*2	Output = 6 5 3 / 3 1 - * - 4 2 * +	Да

ПРИЛОЖЕНИЕ Б

Код программы

```
#include<iostream>
#include<stack>
#include<string>
#include <cstdlib>
#include <fstream>

using namespace std;

namespace STACK {
typedef char T;

class Stack { // определение класса
private:
    struct mstack;
    mstack *head;
public:
    Stack() {
        head=NULL;
    };
    void pop();
    void push(T a);
    T top();
    bool stempty();
    void clear();
    ~Stack();
};
}

// Функция сравнения приоритета операторов
bool HasHigherPrecedence(char operator1, char operator2);
// Функция проверки является ли символ оператором
bool IsOperator(char C);
// Функция проверки является ли символ буквой или цифрой
bool IsOperand(char C);
int GetOperatorWeight(char op);
bool HasHigherPrecedence(char op1, char op2);
// Функция преобразования
string InfixToPostfix(const string expression);
```

```

#include "fun.h"

using namespace std;
using namespace STACK;

namespace STACK { // определим пространство имен
struct Stack::mstack { // элемент стека
    char oprt;
    struct mstack*prev;

    mstack() { // инициализируем структуру
        oprt='\0';
        prev=NULL;
    }
};

void Stack::pop() { // удаляем элемент
    if(head==NULL) {
        cerr<<"stack is empty\n";
        exit(1);
    }
    mstack *temp=head;
    head=head->prev;
    delete temp;
}

void Stack::push(char a) { // добавляем новый элемент
    mstack*temp;
    temp = new mstack;
    temp->oprt=a;
    temp->prev=head;
    head=temp;
}

char Stack::top() { // достаем верхний элемент
    if(head==NULL) {
        cerr<<"stack is empty\n";
        exit(1);
    }
    return head->oprt;
}

bool Stack::stempty() { // проверка на пустоту
    if(head==NULL)
        return true;
    else
        return false;
}

```

```

}

Stack::~Stack() { // очистка
    while(head!=NULL) {
        pop();
    }
}
}

Stack St;

// Функция для проверки считанного символа: является ли он буквой или цифрой
bool IsOperand(char C) {
    if(isalnum(C)||isalpha(C)) return true;
    return false;
}
// Функция для проверки считанного символа: является ли оператором
bool IsOperator(char C) {
    if(C == '+' || C == '-' || C == '*' || C == '/' )
        return true;
    return false;
}
// Функция для проверки приоритетов, чем больше "вес" оператора, тем раньше
он выполняется
int GetOperatorWeight(char op) {
    int weight = -1;
    if(op == '+' || op == '-' )weight = 1;
    if(op == '/' || op == '*' )weight = 2;
    return weight;
}
// Функция, которая определяет какая операции будет выполнена раньше,
используя определение "веса"
bool HasHigherPrecedence(char op1, char op2) {
    int op1Weight = GetOperatorWeight(op1);
    int op2Weight = GetOperatorWeight(op2);
    // Если операторы имеют одинаковый "вес", а значит и приорет, то вернется
true (ассоциативность
    // Так же используем левую ассоциативность, т.е в выражение A+B-C, "+"
больше, чем "-"
    if(op1Weight < op2Weight)
        return false;
    else
        return true;
}
// Функция, которая преобразовывает запись
string InfixToPostfix(string expression) {
    string postfix = ""; // Инициализируем пустую строку

```

```

        for(int i = 0; i < expression.length(); i++) { // Обрабатываем строку до
ее конца
// Сканируем посимвольно
// Если символ является разделительным, то пропускаем его
        if(IsOperator(expression[i])) { // проверка на оператор
            int test=i;
            test++;
            while (expression[test]==' ')
                test++;
            if(IsOperator(expression[test])||expression[test]=='\0') {
                cerr << "Error: incorrect input\n"; // два оператора подряд
идут
                exit(1);
            }
        }

        if(expression[i] == ' ' || expression[i] == ',') continue; // пропуск
разделительных символов
// Если оператор, то проверяем есть ли что в стеке, если да, то смотрим на
приоритеты
// Если оператор в стеке больше "весит", то он записывается в преобразованную
строку, иначе в стек кладется текущий оператор
        else if(IsOperator(expression[i]))
        {
            while(!St.stempty() && St.top() != '(' &&
HasHigherPrecedence(St.top(),expression[i])) {
                postfix+= St.top();
                postfix+=" ";
                St.pop();
            }
            St.push(expression[i]);
        }
// Иначе если символ является буквой или цифрой, записываем в преобразованную
строку
        else if(IsOperand(expression[i])) {
            postfix +=expression[i];
            postfix+=" ";
        }
        else if (expression[i] == '(') {
            St.push(expression[i]);
        }
        else if(expression[i] == ')') { // Записываем выражение, которое было
в скобках
            while(!St.stempty() && St.top() != '(') {
                postfix += St.top();
                postfix+=" ";
            }
        }
    }
}

```

```

        St.pop();
    }
    St.pop(); //убираем из стека "("
}
}
while(!St.stempty()) { // Оставшиеся операторы достаем из стека
    postfix += St.top();
    postfix+=" ";
    St.pop();
}
St.~Stack(); // Очистка стека
return postfix;
}

#include "fun.h"
using namespace STACK;

int main() {
    string expression, postfix;
    int run = 1, check;
    cout << "\033[34m\tЗдравствуй! Выберите что вы хотите:\033[0m\n 1)
Нажмите 1, чтобы считать с файла.\n 2) Нажмите 2, чтобы считать с консоли.\n
3) Нажмите 3, чтобы выйти из программы.\n" << endl;
    while(run) {
        cin >> check;
        cin.ignore();
        switch(check) {
            case (1): {
                ifstream infile("test.txt");
                if(!infile)cout<<"Входной файл не может быть открыт!"<<endl;
                else {
                    getline(infile, expression);
                    postfix = InfixToPostfix(expression);
                    cout<<"Output = "<<postfix<<"\n";
                }
                break;
            }
            case(2): {
                cout<<"Enter Infix Expression \n";
                getline(cin,expression);
                postfix = InfixToPostfix(expression);
                cout<<"Output = "<<postfix<<"\n";
                break;
            }
            case(3): {
                cout<<"End!"<<endl;
                run = 0;
            }
        }
    }
}

```

```
        break;
    }
    default: {
        cout<<"Incorrect pick!"<<endl;
        run = 0;
        break;
    }
}
}
return 0;
}
```