

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: Стеки и очереди

Студент гр. 7381

Дорох С.В.

Преподаватель

Фирсов М.А.

Санкт-Петербург

2018

Задание

Вариант 11-в-д:

Рассматривается выражение следующего вида:

$$\begin{aligned} < \text{выражение} > ::= < \text{терм} > \mid < \text{терм} > + < \text{выражение} > \mid \\ & \qquad \qquad \qquad < \text{терм} > - < \text{выражение} > \\ < \text{терм} > ::= < \text{множитель} > \mid < \text{множитель} > * < \text{терм} > \\ < \text{множитель} > ::= < \text{число} > \mid < \text{переменная} > \mid (< \text{выражение} >) \mid \\ & \qquad \qquad \qquad < \text{множитель} > ^ < \text{число} > \\ < \text{число} > ::= < \text{цифра} > \\ < \text{переменная} > ::= < \text{буква} > \end{aligned}$$

Такая форма записи выражения называется **инфиксной**.

Постфиксной (префиксной) формой записи выражения aDb называется запись, в которой знак операции размещен за (перед) операндами: abD (Dab).

Примеры:

Инфиксная	Постфиксная	Префиксная
$a-b$	$ab-$	$-ab$
$a*b+c$	$ab*c+$	$+*abc$
$a*(b+c)$	$abc+*$	$*a+bc$
$a+b^c^d*e$	abc^d^e*+	$+a*^b^cde$

Отметим, что постфиксная и префиксная формы записи выражений не содержат скобок.

Требуется:

Перевести выражение, записанное в обычной (инфиксной) форме в заданном текстовом файле `infix`, в постфиксную форму и в таком виде записать его в текстовый файл `postfix`.

Пояснение задания

Постфиксная нотация (англ. Postfix notation) — форма записи математических и логических выражений, в которой операнды расположены

перед знаками операций. Также именуется как обратная польская запись, обратная бесскобочная запись.

Описание алгоритма

Из потока ввода считывается исходное выражение в инфиксной форме. Далее проверяется, соответствие вида начального вида с формой выражения инфиксной формой (см. Задание), если выражение корректное то вызывается функция, которая переводит выражение из инфиксной формы в префиксную, иначе выводится сообщение об соответствующей ошибке. Функция перевода выражения принимает начальное выражение и дописывает в строку символы открывающейся и закрывающейся скобки в начало и конец строки соответственно. Далее создаётся строка вывода, в которой будет храниться конечной выражение. Символ начальной строки является буквой или цифрой – он записывается в строку вывода; символ строки является открывающейся скобкой – она добавляется в стек; символ строки является закрывающейся скобкой – из стека символы записываются в строку вывода и удаляются до тех пор, пока не будет найден символ открывающейся строки, он также удаляется. Когда вышеперечисленные условия не выполняются, то символ строки является оператором, после этого из стека записываются символы в строку вывода в соответствии с приоритетом, когда условие перестаёт выполняться, то оператор добавляется в стек. Вышеперечисленные действия повторяются до тех пор, пока не будет обработана начальная строка, после этого функция заканчивает работу и возвращает конечную строку.

Описание функций и структур данных.

`bool isOperator(char c);` – функция, определяющая является ли символ оператором.

`char operation` – символ проверяемой операции.

Возвращаемое значение: true если символ оператор и false если наоборот.

`int getPriority(char symbol);` – функция, выставяющая приоритет операторам.

`char c` – проверяемый символ.

Возвращаемое значение: цифру, которая характеризует приоритет оператора.

`std::string infixToPostfix(std::string infix)` – функция, переводящая данное выражение из инфиксной формы в постфиксную.

`std::string infix` – входная строка.

Возвращаемое значение: функция возвращает строку в постфиксной форме.

`bool isValid (std::string infix);` – функция, проверяющая выражение, записанное в инфиксной форме на корректность.

`std::string infix` – входная строка.

Возвращаемое выражение: возвращает `false`, если выражение некорректно и `true` если корректно.

`class Stack` – шаблонный класс, реализующий абстрактный тип данных под названием стек. Стек реализован на основе связанного списка. Экземпляр класса хранит в себе:

`int size;` – количество элементов, находящихся в стеке.

`struct StackNode{ } *head;` – структура, представляющая из себя шаблон узла стека, содержит в себе поле `Type value`, которое хранит значение элемента стека и указатель на предыдущий элемент стека.

Методы для работы со стеком:

`Stack();` – стандартный конструктор стека. Инициализирует все поля объекта стандартными значениями и выделяет память для одного элемента.

`Type top();` – метод, возвращающий элемент с вершины стека.

Возвращаемое значение: функция возвращает копию объекта на вершине стека.

`void pop();` – метод, удаляющий верхний элемент стека.

Возвращаемое значение: функция ничего не возвращает.

`void push(const Type& value);` – метод, вставляющий в вершину стека новый элемент.

Возвращаемое значение: функция ничего не возвращает.

`int size_s() const;` – метод, определяющий количество элементов в стеке.

Возвращаемое значение: целое число: количество элементов в коллекции.

`bool empty() const;` – Метод, определяющий, пуст ли стек.

Возвращаемое значение: логический тип: истина, если стек пуст.

`~Stack();` – Деструктор класса, освобождает выделенную память под элементы стека.

Тестирование

Для проверки работоспособности программы был создан скрипт (см. ПРИЛОЖЕНИЕ В) для автоматического ввода и вывода тестовых данных. Результаты тестирования сохраняются в файл `postfix.txt`.

Ниже представлена таблица тестирования:

Входные данные	Выходные данные
$((a*2)/(f-5))/((a/2)+(f*5))$	Expression in postfix notation: $a2*f5-/a2/f5*+/-$
$(q*(w/(e-(r+(2))))))$	Expression in postfix notation: $qwer2+-/*$
$q/w*e-r+t/r*e-3+4*5+3-2$	Expression in postfix notation: $qw/e*r-tr/e*+3-45*+3+2-$
$a+b+c/$	Invalid input! The number of operators greater than or equal to number of letters(numbers)
	Is empty!
$)d+x)$	Incorrect input! Expression started with ')'
$(((((d+9)+e)-8)/3)$	Different number of brackets!
$a\%c$	Operator is not valid!
$m-n*i/kk$	Two letters or numbers in a row!
$a+b*(c--d)$	Two operators in a row!

Выводы

В процессе выполнения лабораторной работы были получены знания и навыки по реализации такой структуры данных, как стек, были изучены различные способы записи математических выражений и методы перевода выражений из инфиксной в постфиксную нотацию. Закреплены навыки работы с системой контроля версий, bash-скриптами.

ПРИЛОЖЕНИЕ А

КОД MAIN.CPP

```
#include <iostream>
#include <string>
#include <algorithm>
#include "stack.hpp"

bool isOperator(char c) {
    return (!isalpha(c) && !isdigit(c));
}

int getPriority(char c) {
    if (c == '-' || c == '+')
        return 1;
    else if (c == '*' || c == '/')
        return 2;
    else if (c == '^')
        return 3;
    return 0;
}

std::string infixToPostfix(std::string infix) {
    infix = '(' + infix + ')';
    int length = infix.size();
    Stack<char> stack;
    std::string output;

    for (int i = 0; i < length; i++) {

        if (isalpha(infix[i]) || isdigit(infix[i])) //Если символ
является операндом, добавьте его в вывод
            output += infix[i];
        else if (infix[i] == '(') //Если символ
равен '(', закидываем его в стек
            stack.push('(');
        else if (infix[i] == ')') { //Если
отсканированный символ является ')', добавляем в вывод и удаляем символы,
пока не будет встречен символ '('.
            while (stack.top() != '(') {
                output += stack.top();
                stack.pop();
            }
            stack.pop();
        }
        else {

```

```

        if (isOperator(stack.top())) { //Если оператор,
добавляем операторы в вывод в соответствии с приоритетом и удаляем из
стека
            while (getPriority(infix[i]) <= getPriority(stack.top()))
{
                output += stack.top();
                stack.pop();
            }
            stack.push(infix[i]);
        }
    }
    return output;
}

int isValid (std::string infix){
    int length = infix.size();
    int signum_count = 0;
    int alpha_count = 0;
    for(int i = 1; i < length; i++) {
        if(isOperator(infix[i-1]) && infix[i-1] != '(' && infix[i-1] !=
')') {
            if(isOperator(infix[i]) && infix[i] != '(' && infix[i-1] !=
')') {
                std::cout << "Two operators in a row!" << std::endl;
                return 0;
            }
        }
        if(isalpha(infix[i-1]) || isdigit(infix[i-1])){
            if(isalpha(infix[i]) || isdigit(infix[i])) {
                std::cout << "Two letters or numbers in a row!" << std::endl;
                return 0;
            }
        }
    }

    unsigned int bracket_count = 0;

    for(int i = 0 ; i < length; i++){
        if(isOperator(infix[i]) && infix[i] != ')' && infix[i] != '(') {
            ++signum_count;
            if(!(infix[i] == '-' || infix[i] == '*' || infix[i] == '+' ||
infix[i] == '/' || infix[i] == '^')) {
                std::cout << "Operator is not valid!" << std::endl;
                return 0;
            }
        }
    }

    if(isalpha(infix[i]) || isdigit(infix[i]))

```



```

        ++alpha_count;

        if (infix[i] == '(')
            ++bracket_count;
        if (infix[i] == ')')
            --bracket_count;
        if(i == length-1 && bracket_count != 0){
            std::cout << "Different number of brackets!" << std::endl;
            return 0;
        }
        if(infix[0] == ')') {
            std::cout << "Incorrect input! Expression started with ')'"
<< std::endl;
            return 0;
        }

    }

    if(signum_count >= alpha_count ) {
        if(signum_count == alpha_count && signum_count == 0 &&
alpha_count == 0){
            std::cout << "Is empty!" << std::endl;
            return 0;
        }
        std::cout << "Invalid input! The number of operators greater than
or equal to number of letters(numbers)" << std::endl;
        return 0;
    }

    return 1;
}

int main()
{
    std::string s;
    std::cout << "Expression in infix notation: ";
    std::getline(std::cin, s);
    std::cout << s << std::endl;
    s.erase(std::remove(s.begin(), s.end(), ' '), s.end()); //удаляет все
пробелы из строки, перемещая их в её конец, а затем стирая
    int flag = isValid(s);
    if(flag){
        std::cout << "Expression in postfix notation: ";
        std::cout << infixToPostfix(s) << std::endl;
    }
    std::cout << std::endl;
    return 0;
}

```

ПРИЛОЖЕНИЕ Б

ФАЙЛ STACK.HPP

```
template <class Type>
class Stack{
public:
    Stack() {
        head = nullptr;
    }

    void push(const Type& value) {
        ++size;
        StackNode *temp = new StackNode(value, head);
        head = temp;
    }

    Type top() const {
        return head->value;
    }

    void pop() {
        --size;
        StackNode *temp = head;
        head = head->next;
        delete temp;
    }

    int size_s() const {
        return (size);
    }

    bool empty() {
        return (!size_s);
    }

    ~Stack() {
        StackNode *temp = head;
        while (head) {
            head = head->next;
            delete temp;
            temp = head;
        }
    }

private:
    struct StackNode {
        Type value;
        StackNode *next;
        StackNode(const Type& value, StackNode *next) : value(value),
next(next) { }
    };
};
```

```
    } *head;  
    int size;  
};
```