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

Студент гр. 7383 _____ Медведев И. С.

Преподаватель _____ Размочаева Н. В.

Санкт-Петербург

2018

Содержание

Цель работы

Познакомиться со структурами данных и научиться реализовывать их на языке программирования C++.

Формулировка задачи: перевести выражение, записанное в обычной (инфиксной) форме в заданном текстовом файле infix, в префиксную форму и в таком виде записать его в текстовый файл prefix (вариант 11-е-в).

Реализация задачи

int main ( ) – головная функция, в которой открывается файл infix.txt, из которого считывается строка и начинается основной алгоритм. Функция проверяет строку с конца. Если символ в считанной строке является числом или переменной, то этот символ присоединяется к началу строки *ans*, если символ является оператором, то он проталкивается в стек, если стек пуст, или же сравнивается приоритет операции с приоритетом верхнего элемента в стеке. Если у текущего символа приоритет ниже, то из стека выталкиваются все элементы большего приоритета, иначе оператор просто проталкивается в стек. Если текущим символом оказалась закрывающая скобка, то она проталкивается в стек. Если же ей оказалась открытая скобка, то достаются все операторы из стека, пока не встретится закрытая скобка. Если же символ оказался ни переменной, ни числом, ни оператором и ни скобками, то сообщается об ошибке. Затем выталкивается из стека все операторы, которые остались и записываются в начало строки *ans*, а затем в файл prefix. Так же в функции main проверяется правильность введенной инфиксной записи.

bool isOperation(char x)– функция, которая проверяет, является ли символ оператором.

int prioritet(char ch) – функция, которая возвращает приоритет операции. У ‘+’ и ‘-‘ приоритет равен одному, у ‘*’ и ‘/’ – двум, у ‘^’ – трем.

Методы класса Stack:

Конструктор, который задает начальные параметры для переменных *head* (первый элемент в стеке), *size* (размер стека), *tail* (последний элемент стека), *maxsize* (максимальный размер стека). Так же выделяет память под массив символов.

Push (Type sym) – добавляет элемент в стек, при этом увеличивая переменные *size* и *tail* на единицу.

Pop () – достает последний элемент из стека, уменьшая переменные *size* и *tail* на единицу.

Resize (int new_size) – получает на вход новый размер массива символов, создает новый массив с размером *new_size*, перемещает все элементы из старого массива в новый. Удаляет старый массив. Присваивает указателю на старый массив адрес нового.

Size () – возвращает размер стека.

Top () – возвращает последний элемент стека.

Тестирование

Программа собрана в операционной системе Ubuntu 17.04 с использованием компилятора g++. В других ОС и компиляторах тестирование не проводилось. Результаты тестирования показали, что поставленная цель выполнена. Результаты тестирования представлены в Приложении Б.

Вывод

В ходе выполнения лабораторной работы были изучены основные понятия стеке, был реализован стек на базе массива на языке программирования C++. Также была написана программа для записи выражения из инфиксного в префиксный вид.

ПРИЛОЖЕНИЕ А.

КОД ПРОГРАММЫ

```
#include <iostream>
#include <cstring>
#include <fstream>
#include <cctype>

using namespace std;



template <class Type>
class Stack {
Type* arr;
int head,tail, size, maxsize;
public :
   Stack () ;
   void push ( Type sym ) ;
   Type pop();
   int Size();
   void resize(int maxsize);
   Type Top();
} ;
//----------------

template <class Type>
Stack<Type>::Stack()
{
   arr = new char[100];
   head = size = 0 ;
   tail= -1;
   maxsize = 100;
}
//----------------
```

```
template <class Type>
void Stack<Type>::push (Type sym){
    if (size == maxsize){
        resize(2*maxsize);

    }

        tail++;
        size++;
        arr[tail] = sym;
}
//----------------
template <class Type>
Type Stack<Type>::pop(){
    char x;
    x = arr[tail];
        tail--;
        size--;
        return(x);
}

//----------------

template <class Type>
int Stack<Type>::Size(){
    return (size);
}



template <class Type>
void Stack<Type>::resize(int new_size){
    Type* new_arr = new Type[new_size];
    int i;
```

```
    for (i = 0; i < size; i++){
        new_arr[i] = arr[i];
    }
    delete[] arr;
    maxsize = new_size;
    arr = new_arr;

}

template <class Type>
Type Stack<Type>::Top(){
    return(arr[tail]);

}



int prioritet(char ch){
    switch(ch){
        case '+': case'-':{
            return 1;
        }
        case '*': case '/':{
            return 2;
        }
        case '^':{
             return 3;
        }
    default:{
        return 0;
    }

    }
}
```

```
bool isOperation(char x){
    if (x == '^' || x == '*' || x == '/' || x == '+' || x == '-')
        return true;
    return false;
}




int main(){
        int i = 0;
        const char space = ' ';
        string ans;
        string infix;
        Stack <char> operations;
        ifstream infile("infix.txt");
        getline(infile, infix);
        if(!isalpha(infix[i]) && !isdigit(infix[i]) && infix[i] != '('){
            cout<<"Неверное инфиксное выражение - первый символ не переменная и не
число"<<endl;
            return 0;
        }
        infile.close();
        i = infix.length() - 1;
        if(!isalpha(infix[i]) && !isdigit(infix[i]) && infix[i] != ')'){
            cout<<"Неверное инфиксное выражение - операция в конце строки"<<endl;
            return 0;
        }
        while (i >= 0){

            if(isalpha(infix[i]) || isdigit(infix[i])){
                check++;
                ans.insert(0, 1, infix[i]);
                if(!isalpha(infix[i-1]) && !(isdigit(infix[i-1])))
```

```
                ans.insert(0, 1, space);


            }
            else if (isOperation(infix[i])){
                if (isOperation(infix[i-1])){
                    cout<<"Неверное инфиксное выражение - два знака подряд"<<endl;
                    return 0;
                }
                if(!operations.Size())
                    operations.push(infix[i]);
                else{
                    while(prioritet(infix[i]) <= prioritet(operations.Top()) &&
operations.Size()){
                        ans.insert(0, 1, operations.pop());
                        ans.insert(0, 1, space);
                    }
                    operations.push(infix[i]);
                }


            }
          else if(infix[i] ==')'){
                operations.push(infix[i]);
            }
          else if(infix[i] =='('){
               while (prioritet(operations.Top())){


                  ans.insert(0, 1, operations.pop());
                  ans.insert(0, 1, space);


               }
               if (operations.Top() != ')'){
                   cout<<"Нет закрывающей скобки!";
                   return 0;
               }
```

```
            operations.pop();
        }


        else{
            cout<<"Лишние символы"<<endl;
            return 0;
        }
        i--;
    }
    while(operations.Size()){
        ans.insert(0, 1, operations.pop());
        ans.insert(0, 1, space);
    }
    ofstream out("prefix.txt");
    out<<ans;
   cout<<"Выражение записано в префиксном форме"<<endl;
    out.close();
    ans.clear();
    infix.clear();
    return 0;


}
```

ПРИЛОЖЕНИЕ Б.
ТЕСТОВЫЕ СЛУЧАИ

Таблица 1 — Результаты тестов.

Input	Output	True/False
(5-b)*7	* - 5 b 7	True
(q+5	Нет закрывающей скобки!	True
A+B*	Неверное инфиксное выражение - операция в конце строки	True
(a+32)*5-2*b	- * + a 32 5 * 2 b	True
A+B^C^12*D	+ A * ^ B ^ C 12 D	True
(A+B)*C-(D-E)*(F+G)	- * + A B C * - D E + F G	True
*A+C	Неверное инфиксное выражение - первый символ не переменная и не число	True