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. 6382 _____ Вайгачёв А.О.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург
2017

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 .

Задание № 11в-д

.Рассматривается выражение следующего вида:

< выражение >::=< терм> | < терм > + < выражение > |

< терм > - < выражение >

< терм > ::= < множитель > | < множитель > * < терм >

< множитель > ::= < число > | < переменная > | (<выражение>) |

< множитель > ^ < число >

< число > ::= < цифра >

< переменная > ::= < буква >

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:

в) 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;

Основные теоретические положения.

Ссылочная реализация стека и очереди в динамической памяти в основном аналогична ссылочной реализации линейных списков. Упрощение связано с отсутствием необходимости работать с текущим элементом списка. Идеи такой реализации ясны из рисунка 3.3. Для ссылочной реализации дека естественно использовать двунаправленный список.

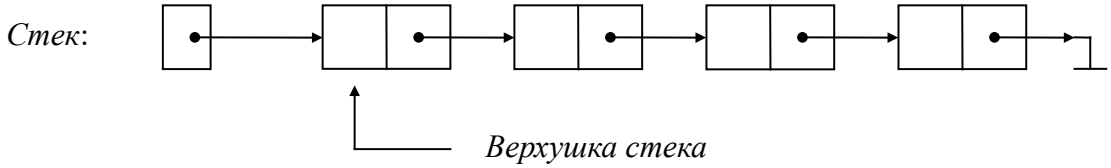
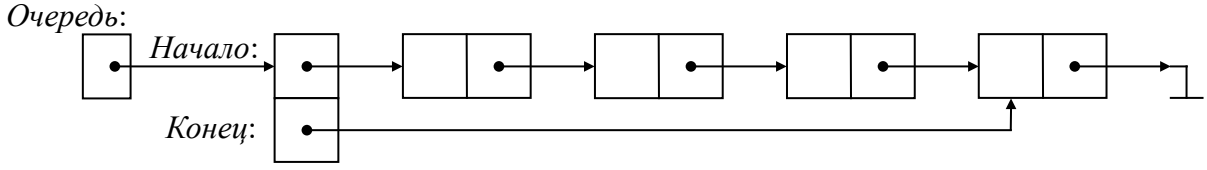


Рис. 3.3



Решение задачи.

На протяжении всей работы программы выражение, записанное в infix-форме, держится в памяти, начало которого хранит указатель cPtr. Этот указатель меняется в ходе процедуры. Конечная строка (string postFixString) будет содержать выражение в postfix-форме.

Основная логика такова: если перед нами операнд, то она пишется без изменений в конечную строку, если перед нами оператор (« + » , « - » , « * » , « / » , «(»), то в зависимости от того, какой приоритет оператора в стэке, добавим текущий к остальным операторам в стэке, либо в конечную строку. Если перед нами закрывающая скобка («) »), тогда мы добавляем все операторы из стэка в конечную строку, до тех пор пока не закончится стэк или встретим открывающую строку. Это можно сделать потому что при таком порядке действий в стэке образуются упорядоченные по убыванию приоритета операторы.

Описание функций и глобальных переменных (Кроме стандартных функций стэка).

bool isOperator(char character) — Просто определяет, является ли символ одним из пяти стандартных операторов. Возвращает True, если да иначе False. На вход — символ.

bool isOperand(char character) - Если символ не является оператором или скобкой, то предполагается, что он является операндом. Возвращает True, если да иначе False. На вход — символ.

int compareOperators(char op1, char op2) - Сравнивает приоритет основных операторов. Возвращет 0, если они равны, -1, если OP2 меньше OP1, и 1, если OP2 больше OP1

Тестирование.

№	Входные данные из консоли и источников (infix.txt)	Выход (postfix.txt)
1	a-b	a-b Current character = a Current character = - push -> - Current character = b pop -> - Postfix is: ab-

№	Входные данные из консоли и источников (infix.txt)	Выход (postfix.txt)
2	a*b+c	a*b+c Current character = a Current character = * push -> * Current character = b Current character = + pop -> * push -> + Current character = c pop -> + Postfix is: ab*c+
3	a*(b+c)	a*(b+c) Current character = a Current character = * push -> * Current character = (push -> (Current character = b Current character = + push -> + Current character = c Current character =) pop -> + pop -> (pop -> * Postfix is: abc+*
4	a+b^c^d*e	a+b^c^d*e Current character = a Current character = + push -> + Current character = b Current character = ^ push -> ^ Current character = c Current character = ^ pop -> ^ push -> ^ Current character = d Current character = * pop -> ^ push -> * Current character = e pop -> * pop -> + Postfix is: abc^d^e*+

№	Входные данные из консоли и источников (infix.txt)	Выход (postfix.txt)
5	(a+b*c^d)^e	(a+b*c^d)^e Current character = (push -> (Current character = a Current character = + push -> + Current character = b Current character = * push -> * Current character = c Current character = ^ push -> ^ Current character = d Current character =) pop -> ^ pop -> * pop -> + pop -> (Current character = ^ push -> ^ Current character = e pop -> ^ Postfix is: abcd^*+e^

Вывод:

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

Приложение А. (Код программы)

`main.cpp`

```
#include <iostream>
#include <string>
#include <fstream>
#include "st_interf.h"
using namespace std;
using namespace st_modul1;

// Simply determine if character is one of the four standard operators.
bool isOperator(char character) {
    if (character == '+' || character == '-' || character == '*' || character == '/' ||
character == '^') {
        return true;
    }
    return false;
}

// If the character is not an operator or a parenthesis, then it is assumed to be an operand.
bool isOperand(char character) {
    if (!isOperator(character) && character != '(' && character != ')') {
        return true;
    }
```

```
    return false;
}

// Compare operator precedence of main operators.
// Return 0 if equal, -1 if op2 is less than op1, and 1 if op2 is greater than op1.
int compareOperators(char op1, char op2) {
    if (op1 == '^') {return -1;}
    else if (op2 == '^') {return 1;};
    if ((op1 == '*' || op1 == '/') && (op2 == '+' || op2 == '-')) { return -1; }
    else if ((op1 == '+' || op1 == '-') && (op2 == '*' || op2 == '/')) { return 1; }
    return 0;
}

int main()
{
    // Empty character stack and blank postfix string.
    Stack opStack;
    string postFixString = "";

    char input[100];

    // Collect input
    char ans;
    do{
        cout << "Use file? (y/n): ";
        cin >> ans;
    }while (ans != 'y' && ans != 'n');
    if (ans == 'n') cin >> input;
    else {
        ifstream infile ("infix.txt");
        if(!infile) {cerr << "File not found!"; exit(EXIT_FAILURE);}
        infile >> input;
        cout << input << endl;
    }

    // Get a pointer to our character array.
    char *cPtr = input;

    // Loop through the array (one character at a time) until we reach the end of the string.
    while (*cPtr != '\0') {
        cout << "Current character = " << *cPtr << endl;
        // If operand, simply add it to our postfix string.
        // If it is an operator, pop operators off our stack until it is empty, an open
parenthesis or an operator with less than or equal precedence.
        if (isOperand(*cPtr)) { postFixString += *cPtr; }
        else if (isOperator(*cPtr)) {
            while (!opStack.empty() && opStack.top() != '(' &&
compareOperators(opStack.top(),*cPtr) <= 0) {
                postFixString += opStack.top();
                opStack.pop();
            }
            opStack.push(*cPtr);
        }
        // Simply push all open parenthesis onto our stack
        // When we reach a closing one, start popping off operators until we run into the
opening parenthesis.
        else if (*cPtr == '(') { opStack.push(*cPtr); }
        else if (*cPtr == ')') {
            while (!opStack.empty()) {
                if (opStack.top() == '(') { opStack.pop(); break; }
                postFixString += opStack.top();
                opStack.pop();
            }
        }

        // Advance our pointer to next character in string.
        cPtr++;
    }
```

```
    // After the input expression has been ran through, if there is any remaining operators
left on the stack
    // pop them off and put them onto the postfix string.
    while (!opStack.empty()) {
        postFixString += opStack.top();
        opStack.pop();
    }

    // Show the postfix string at the end.
    ofstream postfix ("postfix.txt");
    postfix << postFixString << endl;
    cout << "Postfix is: " << postFixString << endl;
    return 0;
}
```

st_impl.cpp

```
// Implementation - Реализация АТД "Стек"(ссылочная реализация в динамической памяти)

#include <iostream>
#include <cstdlib>
#include "st_interf.h"
using namespace std ;

namespace st_modul1
{
        struct Stack::node {//
        base *hd;
        node *tl;
        // constructor
                node ()
                        {hd = NULL; tl = NULL;
                }
        };// end node

//--------------------------------------
        base Stack::top (void)
        {// PreCondition: not null
                if (topOfStack == NULL) { cerr << "Error: top(null) \n"; exit(1); }
                else return *topOfStack->hd;
        }
//--------------------------------------
        void Stack::pop (void)
        {// PreCondition: not null
                if (topOfStack == NULL) { cerr << "Error: pop(null) \n"; exit(1); }
                else
                {       node *oldTop = topOfStack;
            cout << " pop -> " << top() << endl;               // Demo
            topOfStack = topOfStack->tl;
                        delete oldTop->hd;
                        delete oldTop;
                }
        }
//--------------------------------------
        base Stack::pop2(void)
        {// PreCondition: not null
                if (topOfStack == NULL) { cerr << "Error: pop(null) \n"; exit(1); }
                else
                {       node *oldTop = topOfStack;
                        base r = *topOfStack->hd;
                        topOfStack = topOfStack->tl;
           // cout << " pop -> " << r << endl;          // Demo
                        delete oldTop->hd;
                        delete oldTop;
                        return r;
                }
        }
//--------------------------------------
        void Stack::push (const base &x)
```

```
        {       node *p;
                p = topOfStack;
                topOfStack = new node;
                if ( topOfStack != NULL)        {
                        topOfStack->hd = new base;
                        *topOfStack->hd = x;
                        cout << "push -> " << x << endl;                // Demo
                        topOfStack->tl = p;
                }
                else {cerr << "Memory not enough\n"; exit(1);}
        }
//-------------------------------------
    bool Stack::empty(void)
        {       return (topOfStack == NULL) ;
        }
//-------------------------------------
        void Stack::destroy (void)
        {       while ( topOfStack != NULL) {
                        pop();
                }
        }
} // end of namespace st_modul1
```

st_interf.h

```
// интерфейс АТД "Стек" (ссылочная реализация в динамической памяти)

namespace st_modul1
{

//-------------------------------------
    typedef char base;

        class Stack {
        private:
                struct node;
/*      определение структуры будет дано в другом файле (продолжении namespace st_modul) - в
файле Implementation,
        а здесь достаточно объявления "struct node;"
*/
        node *topOfStack;

        public:
                Stack ()
                        { topOfStack = NULL;
                        };
                base top (void);
                void pop (void);
                base pop2(void);
                void push (const base &x);
        bool empty(void);
                void destroy (void);
        };
}
```