**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: Линейные структуры данных: стек, очередь, дек.

Вариант 6

Студентка группы 6383 Терещенко В.Н.  
Преподаватель Шолохова О.М.

Санкт-Петербург  
2017

**1. 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.

**2. Задание:** Проверить, является ли содержимое заданного текстового файла F правильной записью формулы следующего вида:    
< формула > ::= < терм > | < терм > + < формула > | < терм > - < формула >   
< терм > ::= < имя > | ( < формула > ) | [ < формула > ] | { < формула > }   
< имя > ::= x | y | z    
Если не является, то указать номер ошибочной позиции.

**3. Основные теоретические сведения.**

Стек, очередь и дек представляют собой разновидности линейного списка.

При задании спецификации линейных списков использовалась модель последовательности [2]. Доступ к каждому элементу последовательности можно получить, продвигаясь по списку от одного элемента к другому. В модели выделяется пройденная часть и текущий элемент, с которого начинается еще не пройденная часть. При этом ранее в курсе лекций и в учебном пособии [2] были определены функции над последовательностями:

* **First** возвращает первый элемент последовательности
* **Last** возвращает последний элемент последовательности
* **Rest** возвращает исходную последовательность, в которой присутствуют все элементы за исключением первого,
* **Lead** возвращает исходную последовательность, в которой присутствуют все элементы за исключением последнего,
* **Prefix** добавляет элемент в начало последовательности
* **Postfix** добавляет элемент в конец последовательности.

Доступ к элементам последовательности (чтение и запись) , осуществляется только через ее начало и конец. Функции First, Rest, Last, Lead − селекторы, а функции Prefix и Postfix − конструкторы.

Если ограничиться только функциями First, Rest, Prefix (или только функциями Last, Lead, Postfix), то получается структура данных, известная как стек (или магазин). Если использовать только набор функций First, Rest, Postfix (или только Last, Lead, Prefix) получим структуру данных, которая называется очередь (англ. queue). Если же используют весь набор функций, то соответствующую структуру данных обычно называют дек (от англ. deq − аббревиатуры сочетания double-ended-queue, т. е. «очередь с двумя концами»). Во все эти структуры данных необходимо добавить еще предикат-индикатор Null, идентифицирующий пустую последовательность, и либо константу, обозначающую пустую последовательность ∆, либо функцию Crtate, порождающую пустую последовательность.

|  |  |
| --- | --- |
| ***Стек Stack*** | *First*, *Rest*, *Prefix* |
| *Last, Lead, Postfix* |
| ***Очередь Queue*** | *First*, *Rest*, *Postfix* |
| *Last, Lead, Prefix* |
| ***Дек Deq*** | *First*, *Rest, Last, Lead, Postfix*, *Prefix* |

Рис. 3.1. Подструктуры последовательности

Исторически сложилось, что при работе со стеком для обозначения функций **First, Rest, Prefix** используют их синонимы **Top, Pop** и **Push** соответственно (Top − верхушка стека, Pop {up} − вытолкнуть (вверх), Push {down} – втолкнуть, вжать). Полезно определить процедуру **Рор2**, совмещающую результат действия функций Top и Pop.

|  |
| --- |
| **procedure** *Pop*2 (**out** *p*: α; **in-out** *s*:  *Stack* (α)); |
| **Begin** |
| *p* := *Top* (*s*); *s* := *Pop* (*s*) |
| **end** |

Рис. 3.2. Процедура, совмещающая результат действия функций Top и Pop

## Реализация стека и очереди

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.

*Стек*:

*Верхушка стека*

*Начало*:

*Конец*:

*Очередь*:

Рис. 3.3. Ссылочное представление стека и очереди

**4. Описание программы.**

*Входные и выходные данные*.

Входные данные представляют собой строку, который считывается из файла input.txt. Критерии ввода: строка должна содержать операнды (x, y или z; регистр важен!), команды ("+" или "-") и скобки ( "(", ")", "[", "]", "{", "}"); не должны вводиться цифры и буквы (и латиница, и кириллица).

Выходные данные представляют собой сообщение с указанием, является введенная строка формулой или нет.

Пример ввода:

*(x+x)*

Пример вывода:

*Это формула!*

*Реализация*

Заносим формулу в стек, начиная с конца строки. Затем вытаскиваем из стека элемент и проверяем, удовлетворяет ли он нашим условиям. Делаем это со всей строкой. Удовлетворяющие нас элементы заносим в стек для вывода на экран. На выходе получаем сообщение, является ли написанная нами формула ФОРМУЛОЙ. Если при написании исходных данных была допущена ошибка, программа нам выведет место ошибки и подскажет, в чем ошибка.

*Описание функций*

void place\_of\_error(Queue &Q3, char symb, int b) - функция предназначена для указания места и типа ошибки. Функция ничего не возвращает.

bool bracket\_close\_circle\_formula(char symb);

bool bracket\_close\_square\_formula(char symb);

bool bracket\_close\_figure\_formula(char symb); - булевские функции, предназначенные для проверки наличия закрывающей (круглой, квадратной или фигурной) скобки. Функция возвращает true, если данный элемент является закрывающей (круглой, квадратной или фигурной) скобкой, и false в обратном случае.

bool bracket\_open\_circle\_formula(char symb);

bool bracket\_open\_square\_formula(char symb);

bool bracket\_open\_figure\_formula(char symb);­- булевские функции, предназначенные для проверки наличия открывающей (круглой, квадратной или фигурной) скобки. Функция возвращает true, если данный элемент является открывающей (круглой, квадратной или фигурной) скобкой, и false в обратном случае.

bool sign\_formula(char symb) - булевская функция, предназначена для проверки наличия знака. Функция возвращает true, если данный элемент является знаком, и false в обратном случае.

bool name\_formula(char symb) - булевская функция, предназначена для проверки наличия имени. Функция возвращает true, если данный элемент является именем, и false в обратном случае.

void mainformula(ifstream &fin, Stack&Q2) - функция предназначена для определения, является ли введенная строка формулой. Функция ничего не возвращает.

**5. Тестирование.**

|  |  |
| --- | --- |
| Исходная строка | Результат |
| z+x | Это формула! |
| Z+x | Извлеченный элемент [Z] - не удовлетворяет условиям формулы  Строка не является формулой - далее ожидался элемент типа [x], [y], [z] или проблема в скобке |
| (z+x) | Это формула! |
| (z+x} | Извлеченный элемент [}] - не удовлетворяет условиям формулы  Строка не является формулой - далее ожидался элемент типа [+], [-] или проблема в скобке |
| z+x1 | Извлеченный элемент [1] - не удовлетворяет условиям формулы  Строка не является формулой - далее ожидался элемент типа [+], [-] или проблема в скобке |
| (x+[y-z]) | Это формула! |
| x+y-{x+z} | Это формула! |
| ()x+y) | Извлеченный элемент [)] - не удовлетворяет условиям формулы  Строка не является формулой - далее ожидался элемент типа [x], [y], [z] или проблема в скобке |

**Вывод*:*** в данной работе я ознакомилась с часто используемыми на практике линейными структурами данных, обеспечивающими доступ к элементам последовательности только через её начало и конец, и способами реализации этих структур, освоила на практике использование стека и очереди для решения задач.

**Приложение 1. Исходный код.**

--stack.h--

#pragma once

#include <iostream>

using namespace std;

class Stack

{

public:

void init()

{

for (int i = 0; i < size\_1; i++) arr[i] = 0;

work[0] = -1;

work[1] = -1;

}

void push(char a)

{

if (work[1] != size\_1 - 1)

{

if (work[0] == -1)

{

arr[0] = a;

work[1] = 0;

work[0] = 0;

}

else

{

arr[work[0] + 1] = a;

work[0]++;

}

}

else cout << "Стек переполнен." << endl;

}

bool can\_pop()

{

if (work[0] != -1) return 1;

else return 0;

}

char pop()

{

if (work[0] != -1)

{

if (work[0] != work[1])

{

char tmp = arr[work[0]];

arr[work[0]] = 0;

work[0]--;

return tmp;

}

else

{

char tmp = arr[work[0]];

arr[work[0]] = 0;

work[0] = -1;

work[1] = -1;

return tmp;

}

}

else return 0; //т.к. стек пуст

}

Stack()

{

init();

}

private:

const static size\_t size\_1 = 100;

char arr[size\_1];

int work[2];

};

--queue.h--

#pragma once

#include <iostream>

using namespace std;

class Queue

{

public:

void init()

{

for (int i = 0; i < size\_1; i++) arr[i] = 0;

work[0] = -1;

work[1] = -1;

}

void push(char a)

{

if (work[1] != size\_1 - 1)

{

if (work[0] == -1)

{

arr[0] = a;

work[1] = 0;

work[0] = 0;

}

else

{

arr[work[1] + 1] = a;

work[1]++;

}

}

else cout << "Очередь переполнена." << endl;

}

bool can\_pop()

{

if (work[0] != -1) return 1;

else return 0;

}

char pop()

{

if (work[0] != -1)

{

if (work[0] != work[1])

{

char tmp = arr[work[0]];

arr[work[0]] = 0;

work[0]++;

return tmp;

}

else

{

char tmp = arr[work[0]];

arr[work[0]] = 0;

work[0] = -1;

work[1] = -1;

return tmp;

}

}

else return 0; //т.к. очередь пуста

}

Queue()

{

init();

}

private:

const static size\_t size\_1 = 100;

char arr[size\_1];

int work[2];

};

--methods.h--

#include "queue.h"

#include "stack.h"

void place\_of\_error(Queue &Q3, char symb, int b); //место ошибки

bool bracket\_close\_circle\_formula(char symb); //проверка на закрывающую круглую скобку

bool bracket\_close\_square\_formula(char symb); //проверка на закрывающую квадратную скобку

bool bracket\_close\_figure\_formula(char symb); //проверка на закрывающую фигурную скобку

bool bracket\_open\_circle\_formula(char symb); //проверка на открывающую круглую скобку

bool bracket\_open\_square\_formula(char symb); //проверка на открывающую квадратную скобку

bool bracket\_open\_figure\_formula(char symb); //проверка на открывающую фигурную скобку

bool sign\_formula(char symb); //проверка, является ли элемент знаком

bool name\_formula(char symb); //проверка, является ли элемент именем

bool is\_it\_formula(Stack Q1, Queue &Q3); //проверка, является ли строка формулой

void print\_formula(Queue &Q3); //вывод формулы

void mainformula(ifstream &fin, Stack &Q2); //проверка, является ли формулой

--methods.cpp--

#include "methods.h"

#include <iostream>

#include <fstream>

#include "stack.h"

#include "queue.h"

using namespace std;

//место ошибки

void place\_of\_error(Queue &Q3, char symb, int b)

{

cout << "- не удовлетворяет условиям формулы\n";

cout << "\nСтрока не является формулой - ";

switch (b)

{

case 1:

{

cout << "далее ожидался элемент типа [x],[y],[z]\nили проблема в скобке\n\n";

break;

}

case 2:

{

cout << "далее ожидался элемент типа [+],[-]\nили проблема в скобке\n\n";

break;

}

default:

break;

}

system("pause");

exit(1);

}

//проверка на открывающую круглую скобку

bool bracket\_open\_circle\_formula(char symb)

{

if (symb == '(')

return true;

else return false;

}

//проверка на открывающую квадратную скобку

bool bracket\_open\_square\_formula(char symb)

{

if (symb == '[')

return true;

else return false;

}

//проверка на открывающую фигурную скобку

bool bracket\_open\_figure\_formula(char symb)

{

if (symb == '{')

return true;

else return false;

}

//проверка на закрывающую круглую скобку

bool bracket\_close\_circle\_formula(char symb)

{

if (symb == ')') return true;

else return false;

}

//проверка на закрывающую квадратную скобку

bool bracket\_close\_square\_formula(char symb)

{

if (symb == ']') return true;

else return false;

}

//проверка на закрывающую фигурную скобку

bool bracket\_close\_figure\_formula(char symb)

{

if (symb == '}') return true;

else return false;

}

//проерка, является ли элемент знаком

bool sign\_formula(char symb)

{

if (symb == '+' || symb == '-') return true;

else return false;

}

//проверка, является ли элемент именем

bool name\_formula(char symb)

{

if (symb == 'x' || symb == 'y' || symb == 'z') return true;

else return false;

}

//проверка, является ли строка формулой

bool is\_it\_formula(Stack Q1, Queue &Q3)

{

char symb;

bool name = true;

bool sign = false;

bool bracket\_circle\_close = false;

bool bracket\_square\_close = false;

bool bracket\_figure\_close = false;

bool bracket\_circle\_open = true;

bool bracket\_square\_open = true;

bool bracket\_figure\_open = true;

bool is\_elem = false;

int open\_cir = 0, open\_squ = 0, open\_fig = 0;

int close\_cir = 0, close\_squ = 0, close\_fig = 0;

while (Q1.can\_pop())

{

symb = Q1.pop();

cout << "Извлеченный элемент [" << symb << "] ";

is\_elem = false;

if (!is\_elem && name && name\_formula(symb))

{

cout << "- имя\n";

is\_elem = true;

name = false;

sign = true;

bracket\_circle\_close = true;

bracket\_square\_close = true;

bracket\_figure\_close = true;

bracket\_circle\_open = false;

bracket\_square\_open = false;

bracket\_figure\_open = false;

Q3.push(symb);

}

if (!is\_elem && sign && sign\_formula(symb))

{

cout << "- знак\n";

is\_elem = true;

name = true;

sign = false;

bracket\_circle\_close = false;

bracket\_square\_close = false;

bracket\_figure\_close = false;

bracket\_circle\_open = true;

bracket\_square\_open = true;

bracket\_figure\_open = true;

Q3.push(symb);

}

if(!is\_elem && bracket\_circle\_open && bracket\_open\_circle\_formula(symb))

{

cout << "- открывающая круглая скобка\n";

is\_elem = true;

bracket\_circle\_open = true;

bracket\_circle\_close = false;

name = true;

sign = false;

open\_cir++;

Q3.push(symb);

}

if (!is\_elem && bracket\_square\_open && bracket\_open\_square\_formula(symb))

{

cout << "- открывающая квадратная скобка\n";

is\_elem = true;

bracket\_square\_open = true;

bracket\_square\_close = false;

name = true;

sign = false;

open\_squ++;

Q3.push(symb);

}

if (!is\_elem && bracket\_figure\_open && bracket\_open\_figure\_formula(symb))

{

cout << "- открывающая фигурная скобка\n";

is\_elem = true;

bracket\_figure\_open = true;

bracket\_figure\_close = false;

name = true;

sign = false;

open\_fig++;

Q3.push(symb);

}

if (!is\_elem && bracket\_circle\_close && bracket\_close\_circle\_formula(symb))

{

close\_cir++;

cout << "- закрывающая круглая скобка\n";

if (close\_cir <= open\_cir)

{

close\_cir--;

open\_cir--;

is\_elem = true;

bracket\_circle\_open = false;

bracket\_circle\_close = true;

name = false;

sign = true;

Q3.push(symb);

}

else place\_of\_error(Q3, symb, 2);

}

if (!is\_elem && bracket\_square\_close && bracket\_close\_square\_formula(symb))

{

close\_squ++;

cout << "- закрывающая квадратная скобка\n";

if (close\_squ <= open\_squ)

{

close\_squ--;

open\_squ--;

is\_elem = true;

bracket\_square\_open = false;

bracket\_square\_close = true;

name = false;

sign = true;

Q3.push(symb);

}

else place\_of\_error(Q3, symb, 2);

}

if (!is\_elem && bracket\_figure\_close && bracket\_close\_figure\_formula(symb))

{

close\_fig++;

cout << "- закрывающая фигурная скобка\n";

if (close\_fig <= open\_fig)

{

close\_fig--;

open\_fig--;

is\_elem = true;

bracket\_figure\_open = false;

bracket\_figure\_close = true;

name = false;

sign = true;

Q3.push(symb);

}

else place\_of\_error(Q3, symb, 2);

}

if (!is\_elem && (name && (bracket\_circle\_open || bracket\_square\_open || bracket\_figure\_open)))

place\_of\_error(Q3, symb, 1);

else

if (!is\_elem && (sign && (bracket\_circle\_close || bracket\_square\_close || bracket\_figure\_close)))

place\_of\_error(Q3, symb, 2);

}

if ((open\_cir != close\_cir) || (open\_squ != close\_squ) || (open\_fig != close\_fig))

place\_of\_error(Q3, symb, 2);

return true;

}

--main.cpp--

#include <iostream>

#include <fstream>

#include <locale.h>

#include "stack.h"

#include "queue.h"

#include "methods.h"

#include <Windows.h>

using namespace std;

//проверка, является ли формулой

void mainformula(ifstream &fin, Stack &Q2)

{

Queue Q3;

int n = 1;

char symb;

fin >> symb;

while (!(fin.eof()))

{

Q2.push(symb);

Q3.push(symb);

fin >> symb;

}

Stack Q1;

while (Q2.can\_pop())

{

char a = Q2.pop();

cout << "Заносим в стек элемент [" << a << "]\n";

Q1.push(a);

}

cout << "\nСчитанная строка: ";

print\_formula(Q3);

cout << endl << endl;

bool f = is\_it\_formula(Q1, Q3);

cout << "\nЭто формула!\n\n";

int what\_is\_open\_bracket(char symb, int s);

}

//вывод формулы

void print\_formula(Queue &Q3)

{

while (Q3.can\_pop())

cout << Q3.pop();

}

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

setlocale(0, "Rus");

ifstream fin("input.txt");

if (!fin.is\_open())

{

cout << "Файл не может быть открыт или не создан\n";

system("pause");

return 1;

}

if (fin.eof()) cout << "Файл пуст\n";

else

{

Stack Q2;

mainformula(fin, Q2);

}

fin.close();

system("pause");

}