Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

«Нижегородский государственный университет

им. Н.И. Лобачевского»

Институт информационных технологий, математики и механики

Отчет по лабораторной работе

**Стеки и Очереди**

**Выполнил:**

студент ИИТММ гр. 0826-3

Баранов Е.В.

**Проверил:**

Доцент кафедры Баркалов Константин Александрович

Нижний Новгород

2015г.

**Содержание**

1. Введение 3

2. Постановка задачи 5

3. Руководство пользователя 6

3.1 Stack\_Task 6

3.2 Queue\_Task 6

4. Руководство программиста 7

А) Описание структуры программы 6

Б) Описание алгоритмов 7

I) Stack\_Task 7

a) TStack 8

b) TParser 10

c) Основная программа 12

II) Queue\_Task 13

a) TQueue 14

b) Основная программа 16

5. Заключение 17

6. Литература 18

7. Приложение 19

I) Stack\_Task 19

II) Queue\_Task 28

**Введение**

Стек -  абстрактный тип данных, представляющий собой список элементов, организованных по принципу *LIFO* (англ. *last in — first out*, «последним пришёл — первым вышел»). Иными словами, стеком называется хранилище данных, в котором можно работать только с одним элементом: тем, который был добавлен в стек последним

Программный вид стека используется для обхода структур данных, например, дерево. Также стек используется для организации стековой машины.

Стековая машина – алгоритм, проводящий вычисления выражений по обратной польской записи.

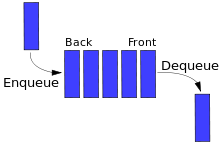
Приведенная ниже иллюстрация, демонстрирует работу двух методов Push (вставка элемента в стек) и Pop (изъятие верхнего элемента из стека).



Очередь – структура данных типа «список», позволяющая добавлять элементы лишь в конец списка, и извлекать их из его начала. Она функционирует по принципу FIFO (First In, First Out — «первым пришёл — первым вышел»), для которого характерно, что все элементы a1, a2, …, an-1, an, добавленные раньше элемента an+1, должны быть удалены прежде, чем будет удален элемент an+1. Также очередь может быть определена как частный случай односвязного списка, который обслуживает элементы в порядке их поступления. Как и в «живой» очереди, здесь первым будет обслужен тот, кто пришел первым.

Очередь в программировании используется, как и в реальной жизни, когда нужно совершить какие-то действия в порядке их поступления, выполнив их последовательно. Примером может служить организация событий в Windows. Когда пользователь оказывает какое-то действие на приложение, то в приложении не вызывается соответствующая процедура (ведь в этот момент приложение может совершать другие действия), а ему присылается сообщение, содержащее информацию о совершенном действии, это сообщение ставится в очередь, и только когда будут обработаны сообщения, пришедшие ранее, приложение выполнит необходимое действие.

Приведенная ниже иллюстрация, демонстрирует работу двух методов Push (Enqueue – вставка в конец) и Pop (Dequeue – изъятие из начала).



**Постановка задачи**

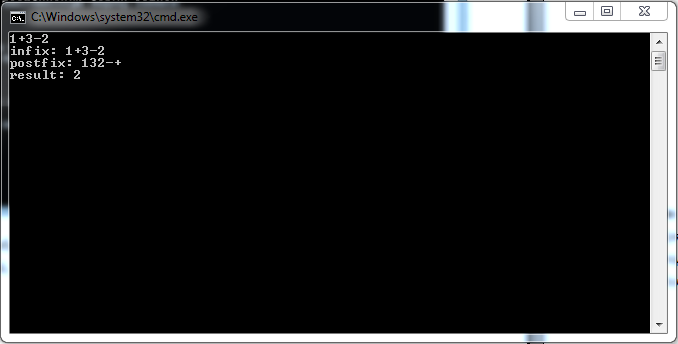
Основные задачи:

1. Реализация класса TStack: тип данных, представляющих собой множество, состоящее из элементов, которые можно добавлять и доставать только с вершины этого множества.
2. Реализация класса TParser с помощью наследования класса TStack: представляет собой инфиксную строку выражения, постфиксную строку выражения и два стека (для вещественных чисел и символов).
3. Реализация вычисления выражений с помощью класса TParser: ввод строки, возможность перевода инфиксной строки в постфиксную, с последующим выводом на экран. Вычисление вводимого выражения, вывод результата.
4. Реализовать класс очередь: Тип данных, представляющих собой список элементов, элементы которого можно добавлять только в конец списка, а доставать только с начала.
5. Реализация программы, принимающей две исходные вероятности: поступление новой задачи (фактически, добавление элемента в очередь) и выполнение задачи (удаление элемента из множества класса «очередь»), стоящей в начале очереди. Далее генерируются две случайные вероятности, которые сравниваются с исходными. Первая вероятность сравнивается с вероятностью поступления новой задачи: если сгенерированная величина меньше исходной, то в очередь добавляется новая задача. Вторая вероятность сравнивается с вероятностью выполнения задачи из начала очереди: если сгенерированная величина меньше исходной, то задача, стоящая в начале очереди, выполнена.

**Руководство пользователя**

**Stack\_Task**

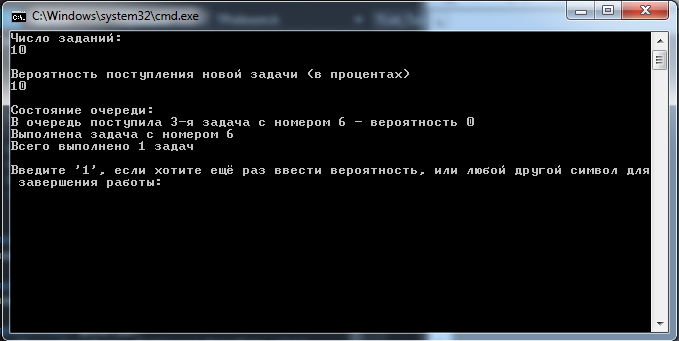
После запуска программы на экране появится консоль с возможностью сразу ввести интересующее вас выражение в инфиксной форме, которое затем будет переведено в постфиксную форму и подсчитано. После чего вы можете ввести другое выражение, либо закрыть программу.



Постфиксная запись (обратная польская запись) – форма записи математических и логических выражений, в которой операнды расположены перед знаком операций.

**Queue\_Task**

После запуска программы на экране появится консоль с предложением ввести сначала число заданий, а затем вероятность поступления новой задачи. После чего на экране будет отображаться состояние очереди и порядок выполнения задач, поступавших в очередь. Затем вы можете выйти из программы или повторить ввод.



**Руководство программиста**

***Описание структуры программы***

Программа представляет собой проектное решение, которое состоит из 4 проектов: gtest, Queue\_Task, Stack\_Task, tests.

gtest: статическая библиотека для поддержки средств тестирования google

Queue\_Task: состоит из 2 модулей: TQueue.h и основной программы.

Stack\_Task: состоит из 3 модулей: TStack.h TParser.h и основной программы.

tests: состоит из 3 модулей: test\_queue.cpp, test\_tstack.cpp, и основной программы.

**Stack\_Task**

**В первом модуле** реализован класс TStack.

Для реализации класса TStack использован шаблон типа данных: template<class ValType>.

Поля класса с доступом protected:

1. int stack\_size – размер стека
2. int top\_index - индекс вершины стека
3. ValType \*data;– указатель на первый элемент массива для хранения стека

Поля класса с доступом public:

* 1. Конструктор по умолчанию
  2. Конструктор копирования
  3. Деструктор
  4. bool IsFull() - проверка полноты стека
  5. bool IsEmpty() - проверка пустоты стека
  6. void push(const ValType &\_elem)– метод вставки элемента в стек
  7. ValType top()– метод, показывающий вершину стека
  8. ValType pop()- метод, позволяющий вытащить верхний элемент стека
  9. void stack\_add\_memory(const int &\_byte) – добавление памяти стеку
  10. void stclear() – очистка стека

**Во втором модуле** реализован класс TParser.

Для использования наследования описанного выше класса необходимо подключить модуль TStack.

Поля класса с доступом private:

1. char infix[MaxLen]– инфиксная строка длины MaxLen
2. char postfix[MaxLen]– постфиксная строка длины MaxLen
3. TStack<double> st\_d– стек вещественных чисел
4. TStack<char> st\_c– стек операций
5. int NumbersLen[NumLen] – массив длин чисел

Поля класса с доступом public:

1. Конструктор
2. Деструктор
3. int Priority(char \_ch) - вычисление приоритета операций
4. bool IsOperation(char \_ch) - проверка является ли символ операцией
5. bool IsNumber(char \_ch) - проверка является ли символ числом
6. bool IsFuction(char \_ch) – проверка является ли символ началом функции
7. void infixtopostfix() - метод перевода инфиксной строки в постфиксную строку
8. double calcPost() - - метод вычисления выражения из постфиксной строки
9. double ExNumber(char \*s, int &len) - перевод из строки в число, нахождение длины числа
10. void printinfix() – вывод на экран строки в инфиксной форме
11. void printpostfix() - вывод на экран строки постфиксной форме
12. void Function(char\* \_ch, int \_k, double &\_rez) – метод определения функции и её вычисления

***Описание алгоритмов Stack\_Task***

*Модуль TStack*

Шаблон: template <class ValType>.

Класс: Tstack:

*Конструктор по умолчанию.*

TStack(int \_size = MaxStackSize) Исходные данные:

Int \_size– длина стека

Описание алгоритма:

Максимальному размеру стека stack\_size присвоить значение переменной \_size (в случае, если значение не указано – создать стек максимального размера);

Индексу вершины стека Index присвоить значение -1 (в данном состоянии стек пуст);

Выделить память типа ValType для массива data размера stack\_size.

*Конструктор копирования*

TStack(const TStack & \_stack) Исходные данные:

const TStack & \_stack – константная ссылка на переменную класса TStack

Описание алгоритма:

Максимальному размеру стека присвоить значение максимального размера исходного стека;

Выделить память типа ValType для массива data размера stack\_size;

Индексу вершины стека присвоить значение индекса вершины исходного стека;

Скопировать массив значений исходного стека в массив значений текущего.

*Деструктор*

~Tstack()

Описание алгоритма:

Освободить память, выделенную в конструкторе. Установить указатель data на NULL.

*Проверка полноты стека*

bool IsFull()

Описание алгоритма:

Сравнить индекс текущего элемента с индексом конца стека.

*Проверка пустоты стека*

bool IsEmpty()

Описание алгоритма:

Сравнить индекс текущего элемента с -1.

*Метод, позволяющий положить элемент в стек*

void TStack<ValType>::push(const ValType \_elem)

Исходные данные:

const ValType \_elem – элемент типа ValType, который нужно положить в стек

Описание алгоритма:

Если стек полон, вызвать исключение «стек полон»;

Увеличить индекс текущего элемента на 1;

Записать в массив data по текущему индексу исходный элемент.

*Метод, позволяющий посмотреть, что на вершине стека*

ValType top ()

Возвращаемый результат:

Верхний элемент стека

Описание алгоритма:

Если стек пуст, то вызвать исключение «стек пуст»;

Возвратить значение последнего добавленного в стек элемента.

*Метод, позволяющий вытащить верхний элемент стека*

ValType pop()

Возвращаемый результат:

Верхний элемент стека

Описание алгоритма:

Если стек пуст, то вызвать исключение «стек пуст»;

Возвратить значение последнего добавленного в стек элемента. Уменьшить текущий индекс на единицу (удалить элемент из стека)

*Метод, позволяющий добавить памяти стеку*

void stack\_add\_memory(const int \_byte)

Исходные данные:

const int \_byte – количество дополнительной памяти (количество дополнительных элементов)

Описание алгоритма:

Если в стеке ничего нет, то удалить память и выделить память на \_byte больше

Если стек не пуст, создать его временную копию, удалить память и выделить память на \_byte больше. Скопировать элементы из временной копии в увеличенный стек

*Метод, позволяющий добавить памяти стеку*

void stclear()

Описание алгоритма:

Установить текущий индекс на -1.

*Модуль Tparser*

Для наследования класса TStack подключаем модуль TStack.h.

Подключаем модуль math.h для использования математических функций.

Класс: TParser:

*Конструктор*

TParser::TParser(char \*\_str) :st\_d(100), st\_c(100)

Исходные данные:

char \*\_str – указатель на начало строки

Описание алгоритма:

Если строка пуста, то записываем символ конца строки в инфиксную строку;

Иначе копируем символы из \_str в infix функцией strcpy\_s;

*Вычисление приоритета операций*

int Priority(char \_ch)

Исходные данные:

char \_ch – операция, приоритет которой необходимо определить

Возвращаемый результат:

int n – приоритет операции

Описание алгоритма:

С помощью оператора множественного выбора определить приоритет операции \_ch:

Если \_ch равно ‘(‘ или ‘)’, то переменной n присвоить значение 0;

Если \_ch равно ‘+‘ или ‘-’, то переменной n присвоить значение 1;

Если \_ch равно ‘\*‘ или ‘/’, то переменной n присвоить значение 2;

Если \_ch равно ‘^‘, то переменной n присвоить значение 3;

По умолчанию: переменной n присвоить значение -1;

Возвратить значение n.

*Проверка является ли символ операцией*

bool isOperation(char \_ch)

Исходные данные:

char \_ch – символ, который необходимо проверить

Возвращаемый результат:

Выражение типа bool (true или false)

Описание алгоритма:

Если код символа \_ch равен коду символа ‘+’, или ‘-‘, или ‘\*’, или ‘/’, или ‘^’, то возвратить значение true;

Иначе возвратить значение false;

*Проверка является ли символ числом*

bool IsNumber(char \_ch)

Исходные данные:

char \_ch – символ, который необходимо проверить

Возвращаемый результат:

Выражение типа bool (true или false)

Описание алгоритма:

Если \_ ch это «.» «,» {0..9} «p» «e» вернуть true;

Иначе false;

*Проверка является ли символ началом функции*

bool IsFuction(char \_ch)

Исходные данные:

char \_ch – символ, который необходимо проверить

Возвращаемый результат:

Выражение типа bool (true или false)

Описание алгоритма:

Если \_ ch это «s» «l» «c» «t» «a» вернуть true;

Иначе false;

*Метод перевода инфиксной строки в постфиксную строку*

void infixtopostfix()

Описание алгоритма:

Очистить стеки

Считывать до конца строки

Встречая операцию – класть её в стек, если приоритет предыдущей операции ниже и выталкивать все операции в постфиксную строку до меньшей по приоритету в обратном случае. Если же встречена операция «)» то выталкивать все элементы из стека операций в постфиксную строку до тех пор, пока не встречена «(».

Встречая число - считывать его и записывать его длину в массив длин NumbersLen записывать посимвольно в постфиксную строку.

Встречая функцию – вычислять то, что находится внутри неё с помощью функции void Function(char\* \_ch, int \_k, double &\_rez) и записывать это в постфиксную строку.

В конце вытолкнуть оставшиеся операции из стека в постфиксную строку.

*Метод вычисления выражения из постфиксной строки*

double calcPost()

Возвращаемый результат:

Последний элемент стека вещественных чисел;

Описание алгоритма:

Очистить стеки

Если встречено число – считать. Если это p (pi) или e (экспонента) - заменить на математические константы. Положить в стек вещественных чисел.

Если встречена операция – применить её к двум последним элементам из стека.

*Методы вывода на экран строк*

*void printinfix()*

*void printpostfix()*

Описание алгоритма:

Вывести строку infix и postfix соответственно на экран.

*Основная программа*

Подключить модуль Tparser.h

Описание алгоритма:

Создаётся пустая строка длины MaxLen=201. Далее в бесконечном цикле вводится выражение в инфиксной форме, которое затем будет преобразовано в постфиксную, выведено в инфиксной и постфиксной форме. Затем будет выведен результат вычисления постфиксной записи.

**Queue\_Task**

**В модуле TQueue.h** реализован класс TQueue (очередь).

Один из самых простых и эффективных способов организовать такую структуру данных – представление очереди в виде кольцевого буфера. Это можно обеспечить с помощью перехода индексов конца и начала очереди на индекс первого элемента при достижении индексами максимальной длины очереди.

Для реализации класса очередь использован шаблон типа данных: template <class QueueType>

Ниже приведены поля класса с доступом private:

1. int begin – индекс начала очереди
2. int end – индекс конца очереди
3. int queuelen – текущая длина очереди
4. int MaxSize – максимальная длина очереди
5. QueueType \*queue– указатель на первый элемент массива для хранения очереди

Поля класса с доступом public:

1. Конструктор по умолчанию (\_maxsize = 10)
2. Деструктор
3. Конструктор копирования
4. bool IsEmpty() – проверка пустоты очереди
5. bool IsFull() – проверка полноты очереди
6. void Push(const QueueType \_elem) – добавление элемента в конец очереди
7. QueueType Pop()– изъятие элемента из начала очереди
8. QueueType Top()– просмотр элемента, стоящего в начале очереди
9. QueueType Bot() - просмотр элемента, стоящего в конце очереди

**В основной программе** (модуль **TQueue.cpp**)реализован алгоритм, в котором на вход принимаются:

1. tmp – число заданий
2. chance – заданная пользователем вероятность поступления новой задачи
3. mass – массив случайных вероятностей выполнения задач

***Описание алгоритмов Queue\_Task***

*Модуль TQueue.h*

Шаблон: template <class QueueType>

Класс: TQueue:

*Конструктор*

TQueue(int \_maxsize = 10)

Исходные данные:

int \_maxsize – длина очереди

Описание алгоритма:

Если \_maxsize < = 0 вызвать исключение;

Максимальной длине очереди MaxSize присвоить значение переменной \_maxsize;

Выделить память типа QueueType для массива queue размера MaxSize;

Индексу начала очереди begin присвоить значение 0;

Индексу конца очереди end присвоить значение -1;

Текущей длине очереди queuelen присвоить значение 0 (очередь пуста);

*Деструктор*

~Tqueue()

Описание алгоритма:

Освободить память, выделенную в конструкторе.

Присвоить указателю queue значение NULL.

*Конструктор копирования*

TQueue(const TQueue& \_queue)

Исходные данные:

const TQueue& \_queue – константная ссылка на переменную класса TQueue

Описание алгоритма:

Максимальной длине очереди MaxSize присвоить значение максимальной длины исходной очереди \_queue.MaxSize;

Выделить память типа QueueType для массива queue размера MaxSize;

Индексу начала очереди begin присвоить значение индекса начала исходной очереди \_queue.begin;

Индексу конца очереди end присвоить значение индекса конца исходной очереди \_queue.end;

Текущей длине очереди queuelen присвоить значение длины исходной очереди \_queue. queuelen;

Скопировать значения массива queue из исходной очереди в текущую очередь

*Проверка пустоты очереди*

bool IsEmpty()

Описание алгоритма:

Сравнить текущую длину очереди с нулем

*Проверка полноты очереди*

bool IsFull()

Описание алгоритма:

Сравнить текущую длину очереди с максимальной длиной очереди

*Добавление элемента в конец очереди*

void Push(const QueueType \_elem)

Исходные данные:

\_elem – элемент типа QueueType, который необходимо добавить в очередь

Возвращаемый результат:

Очередь с измененной длиной на 1 и новым элементом в конце очереди

Описание алгоритма:

Если очередь полна – вызвать исключение .

Если очередь не полна – перемещаем указатель конца очереди, увеличиваем длину очереди и добавляем элемент.

Иначе конец очереди = 0;

*Изъятие элемента из начала очереди*

QueueType Pop()

Требуемый результат:

T tmp – значение элемента, стоящего в начале очереди

Описание алгоритма:

Если очередь пуста – вызвать исключение

Иначе вернуть копию элемента из начала очереди

Сдвинуть указатель на начало очереди

Уменьшить длину очереди

*Просмотр элемента, стоящего в начале очереди*

QueueType Top()

Возвращаемый результат:

Значение из начала очереди типа QueueType

Описание алгоритма:

Если очередь пуста, то выбросить исключение;

Иначе вернуть копию значения из начала очереди;

*Просмотр элемента, стоящего в конце очереди*

Возвращаемый результат:

Значение из начала очереди типа QueueType

Описание алгоритма:

Если очередь пуста, то выбросить исключение;

Иначе вернуть копию значения из конца очереди;  
*Очистка очереди*

Описание алгоритма:

Обнулить указатели и длину очереди.

*Основная программа Queue\_Task.cpp*

Подключить модуль TQueue.h.

Описание алгоритма:

Пока не введена команда выхода из программы

Ввести число заданий;

Сформировать массив размера числа заданий и заполнить его вероятностями выполнения задачи

Ввести вероятность поступления новой задачи (в процентах);

Вывод состояния очереди

**Заключение**

В данном проекте была реализована стековая машина, позволяющая вычислять выражения, записанные в постфиксной форме, а так же класс очереди, с помощью которой можно ставить задачи (элементы) в начало очереди, доставать их, если данная задача выполнена, т.е. реализован алгоритм, с помощью которого можно выполнять задачи в порядке их поступления.

**Литература**

1. <http://aliev.me/runestone/BasicDS/TheStackAbstractDataType.html>
2. <https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D0%B5%D0%BA>
3. Уильям Форд “Структуры данных в C++”;
4. Бьерн Страуструп “Язык программирования C++”;
5. Стенли Б. Липпман “Язык программирования C++”

**Приложение Stack\_Task**

*Основная программа*

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "../../../include/Parser.h"

#include <iostream>

#define MaxLen 201

using namespace std;

int main()

{

char tmpch[MaxLen];

double rez;

while (1)

{

cin >> tmpch;

if (tmpch[0] == '\0')

break;

TParser tmp(tmpch);

tmp.infixtopostfix();

tmp.printinfix();

tmp.printpostfix();

rez=tmp.calcPost();

cout << "result: " <<rez<< endl;

}

return 0;

}

*Модуль TStack.h*

#pragma once

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#define MaxStackSize 100000

template <class ValType>

class TStack

{

protected:

ValType \*data;

int top\_index;

int stack\_size;

public:

TStack(int \_size = MaxStackSize);

TStack(const TStack & \_stack);

~TStack();

bool IsFull();

bool IsEmpty();

void push(const ValType \_elem);

ValType top();

ValType pop();

void stack\_add\_memory(const int \_byte);

void stclear();

};

template <class ValType>

TStack<ValType>::TStack(int \_size = MaxStackSize)

{

stack\_size = \_size;

top\_index = -1;

data = new ValType[stack\_size];

}

template<class ValType>

TStack<ValType>::TStack(const TStack & \_stack)

{

stack\_size = \_stack.stack\_size;

data = new ValType[stack\_size];

top\_index = \_stack.top\_index;

for (int i = 0; i < top\_index + 1; i++)

data[i] = \_stack.data[i];

}

template <class ValType>

TStack<ValType>::~TStack()

{

delete[] data;

data = NULL;

}

template<class ValType>

bool TStack<ValType>::IsFull()

{

return top\_index == stack\_size - 1;

}

template<class ValType>

bool TStack<ValType>::IsEmpty()

{

return top\_index == -1;

}

template<class ValType>

void TStack<ValType>::push(const ValType \_elem)

{

if (top\_index == stack\_size - 1)

throw "Stack is Full";

data[++top\_index] = \_elem;

}

template<class ValType>

inline ValType TStack<ValType>::top()

{

if (top\_index == -1)

throw "Stack is Empty";

return data[top\_index];

}

template<class ValType>

ValType TStack<ValType>::pop()

{

if (top\_index == -1)

throw "Stack is Emtpy";

return data[top\_index--];

}

template<class ValType>

void TStack<ValType>::stack\_add\_memory(const int \_byte)

{

if (top\_index == -1)

{

delete[] data;

stack\_size += \_byte;

data = new ValType[stack\_size];

}

else

{

TStack <ValType> tmp(\*this);

delete[] data;

stack\_size += \_byte;

data = new ValType[stack\_size];

for (int i = 0; i < tmp.top\_index + 1; i++)

data[i] = tmp.data[i];

top\_index = tmp.top\_index;

}

}

template<class ValType>

void TStack<ValType>::stclear()

{

top\_index = -1;

}

*Модуль Tparser.h*

#pragma once

#define \_USE\_MATH\_DEFINES

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "../include/TStack.h"

#include <stdio.h>

#include <cstring>

#include <math.h>

#include <iostream>

#include <string>

#include <sstream>

#define MaxLen 201

#define NumLen 100

using namespace std;

class TParser

{

private:

char infix[MaxLen];

char postfix[MaxLen];

TStack<double> st\_d;

TStack<char> st\_c;

int NumbersLen[NumLen];

public:

TParser(char \*\_str);

~TParser();

int Priority(char \_ch);

bool IsOperation(char \_ch);

bool IsNumber(char \_ch);

bool IsFuction(char \_ch);

void infixtopostfix();

double calcPost();

void printinfix() { cout << "infix: " << infix << endl; }

void printpostfix() { cout << "postfix: " << postfix << endl; }

void Function(char\* \_ch, int \_k, double &\_rez);

};

TParser::TParser(char \*\_str) :st\_d(100), st\_c(100)

{

if (\_str == NULL)

infix[0] = '\0';

else

strcpy\_s(infix, \_str);

}

TParser :: ~TParser() {}

int TParser::Priority(char \_ch)

{

int n;

switch (\_ch)

{

case '(':

case ')': n = 0; break;

case '+':

case '-': n = 1; break;

case '\*':

case '/': n = 2; break;

case '^': n = 3; break;

default:

n = -1; break;

}

return n;

}

bool TParser::IsOperation(char \_ch)

{

return (\_ch == '+' || \_ch == '-' || \_ch == '\*' || \_ch == '/' || \_ch == '(' || \_ch == ')' || \_ch == '^');

}

bool TParser::IsNumber(char \_ch)

{

return (\_ch == '.' || \_ch == ',' || \_ch == '0' || \_ch == '1' || \_ch == '2' || \_ch == '3' || \_ch == '4' || \_ch == '5' || \_ch == '6' || \_ch == '7' || \_ch == '8' || \_ch == '9' || \_ch=='e'|| \_ch=='p');

}

bool TParser::IsFuction(char \_ch)

{

return (\_ch=='s' || \_ch == 'l' || \_ch == 'c' || \_ch == 't' || \_ch == 'a');

}

void TParser::Function(char\* \_ch, int \_k, double &\_rez)

{

switch (\_ch[\_k])

{

case 's':

if (\_ch[\_k + 3] == 'h')

{

\_rez = sinh(\_rez);

}

else

{

\_rez = sin(\_rez);

}

break;

case 'l':

if (\_ch[\_k + 3] == '1')

{

\_rez = log10(\_rez);

}

else

{

\_rez = log(\_rez);

}

break;

case 'c':

if (\_ch[\_k + 3] == 'h')

{

\_rez = cosh(\_rez);

}

else

{

\_rez = cos(\_rez);

}

break;

/\*case 'a':

if (\_ch[\_k + 1] == 'c')

{

if (\_ch[\_k + 4] == 'h')

{

\_rez = acosh(\_rez);

}

else

{

\_rez = acos(\_rez);

}

}

if (\_ch[\_k + 1] == 's')

{

if (\_ch[\_k + 4] == 'h')

{

\_rez = asinh(\_rez);

}

else

{

\_rez = asin(\_rez);

}

}

if (\_ch[\_k + 1] == 't')

{

if (\_ch[\_k + 4] == 'h')

{

\_rez = atanh(\_rez);

}

else

{

\_rez = atan(\_rez);

}

}

break;\*/

case 't':

if (\_ch[\_k + 3] == 'h')

{

\_rez = tanh(\_rez);

}

else

{

\_rez = tan(\_rez);

}

break;

}

}

void TParser::infixtopostfix()

{

st\_c.stclear(); st\_d.stclear(); // очистка стеков

int i = 0, j = 0, k = 0;

// сначала необходимо проверить первый знак на . либо -

// затем считываем пока не '\0'

while (infix[i] != '\0')

{

if (IsOperation(infix[i])) //если встречена операция

{

if (infix[i] == '(')

{

if ( i == 0 || IsOperation(infix [i-1]) )

{

st\_c.push(infix[i]);

i++;

}

else

{

st\_c.push('\*');

st\_c.push(infix[i]);

i++;

}

}

else

{

if (infix[i] == ')')

{

while (st\_c.top() != '(')

{

postfix[j] = st\_c.pop();

j++;

}

int tmp=st\_c.pop();

i++;

}

else

{

if (infix[i] == '-' && (i == 0 || infix[i - 1] == '(')) //особый случай с минусом

{

postfix[j++] = '0'; //записываем ноль в постфиксную строку (для получения выражения 0-x= при встреченном -x=)

NumbersLen[k] = 1;

k++;

i++;

while (IsNumber(infix[i]))

{

NumbersLen[k] += 1;

postfix[j++] = infix[i++];

}

k++;

postfix[j++] = '-';

}

else

{

if (st\_c.IsEmpty() || Priority(infix[i]) > Priority(st\_c.top())) //если стек пуст или если приоритет текущей операции больше чем приоритет предыдущей

{

st\_c.push(infix[i]); //кладём операцию в стек

i++;

}

else //если приоритет текущей операции меньше чем приоритет предыдущей

if (Priority(infix[i]) < Priority(st\_c.top()))

{

while (!st\_c.IsEmpty() && Priority(infix[i]) <= Priority(st\_c.top())) // пока не встречена операция с наименьшим приоритетом

{

postfix[j] = st\_c.pop(); // выталкиваем все операции

j++;

}

st\_c.push(infix[i]);// оставляем знак на верхушке стека

i++;

}

else

{

st\_c.push(infix[i]);

i++;

}

}

}

}

}

else

{

if (IsNumber(infix[i]))

{

while (IsNumber(infix[i]))

{

NumbersLen[k] += 1;

postfix[j] = infix[i];

j++;

i++;

}

k++;

}

}

if (IsFuction(infix[i]))

{

int m=0, n=i, l=0;

char \*len=new char [195];

while (infix[i] != '(')

i++;

i++;

while (infix[i] != '\0')

{

len[m] = infix[i];

m++;

i++;

}

i++;

len[m] = '\0';

TParser tmp(len);

tmp.infixtopostfix();

double reztmp = tmp.calcPost();

Function(infix, n, reztmp);

char len2[30];

sprintf(len2, "%f", reztmp);

if (len2[0] == '-')

{

postfix[j] = '0';

j++;

NumbersLen[k] += 1;

k++;

l++;

while (len2[l]!='\0')

{

NumbersLen[k] += 1;

postfix[j] = len2[l];

j++;

l++;

}

k++;

postfix[j] = '-';

j++;

}

else

{

NumbersLen[k] += 1;

postfix[j] = len2[l];

j++;

l++;

}

k++;

}

}

//выталкиваем оставшиеся операции из стека

while (!st\_c.IsEmpty()) // пока стек не пуст

{

postfix[j] = st\_c.pop(); // выталкиваем все операции до тех пор пока стек не опустеет

j++;

}

}

double TParser::calcPost()

{

int i = 0, k=0;

st\_c.stclear(); st\_d.stclear(); //очистка стеков

while (postfix[i] != '\0')

{

if (IsNumber(postfix[i]))

{

if (postfix[i] == 'e')

{

st\_d.push(M\_E);

i++;

}

else

if (postfix[i] == 'p')

{

st\_d.push(M\_PI);

i++;

}

else

{

double tmp;

char \*len = new char[NumbersLen[k]];

for (int j = 0; j < NumbersLen[k]; j++)

len[j] = postfix[i + j];

tmp = atof(len);

st\_d.push(tmp);

i += NumbersLen[k];

k++;

delete[]len;

}

}

else

{

double reg1, reg2;

reg1 = st\_d.pop();

reg2 = st\_d.pop();

switch (postfix[i])

{

case '+': st\_d.push(reg1 + reg2); break;

case '-': st\_d.push(reg2 - reg1); break;

case '\*': st\_d.push(reg1 \* reg2); break;

case '/': st\_d.push(reg2 / reg1); break;

case '^': st\_d.push(exp(reg1\*log(reg2))); break;

}

i++;

}

}

return st\_d.pop();

}

**Приложение Queue\_Task**

*Основная программа*

#include "../../../include/TQueue.h"

#include <stdlib.h>

#include <locale.h>

#include <iostream>

using namespace std;

int main()

{

int tmp;

double chance, \*mass;

setlocale(LC\_CTYPE, "Russian");

char c;

do

{

cout << "Число заданий:" << endl;

do

{

cin >> tmp;

if (tmp <= 0)

cout << "Некорректный ввод" << endl;

} while (tmp <= 0);

TQueue <int> queue(tmp);

mass = new double[tmp];

for (int i = 0; i < tmp; i++)

mass[i] = rand() % 100 / 100.0;

cout << endl << "Вероятность поступления новой задачи (в процентах)" << endl;

do

{

cin >> chance;

if ((chance < 0) || (chance>100))

cout << "Некорректный ввод" << endl;

} while ((chance < 0) || (chance>100));

chance /= 100.0;

cout << endl << "Состояние очереди:" << endl;

int j = 0;

for (int i = 0; i < tmp; i++)

{

if (mass[i] >= chance)

continue;

int tmp = (rand() % 100) + 1; //значение задачи

queue.Push(tmp);

j++;

cout << "В очередь поступила " << i << "-я задача с номером " << tmp << " - вероятность " << mass[i] << endl;

while ((queue.Top() < queue.Bot()) && (!queue.IsEmpty())) //если значение первой задачи в очереди меньше последней, то задача считается выполненной

cout << "Выполнена задача с номером " << queue.Pop() << endl;

}

while (!queue.IsEmpty())

cout << "Выполнена задача с номером " << queue.Pop() << endl;

if (!j)

cout << "В очередь не поступило ни одной задачи" << endl;

else cout << "Всего выполнено " << j << " задач" << endl;

cout << endl << "Введите '1', если хотите ещё раз ввести вероятность, или любой другой символ для завершения работы:" << endl;

cin >> c;

cout << endl;

} while (c == '1');

return 0;

}

*Модуль TQueue.h*

#pragma once

template <class QueueType>

class TQueue

{

private:

int begin, end, queuelen, MaxSize;

QueueType \*queue;

public:

TQueue(int \_maxsize = 10);

~TQueue() { delete[] queue; queue = NULL; }

TQueue(const TQueue& \_queue);

bool IsEmpty(){return queuelen == 0;}

bool IsFull(){return queuelen == MaxSize;}

void Push(const QueueType \_elem);

QueueType Pop();

QueueType Top();

QueueType Bot();

void Clear();

};

template<class QueueType>

TQueue<QueueType>::TQueue(int \_maxsize)

{

if (\_maxsize <= 0)

throw \_maxsize;

MaxSize = \_maxsize;

queue = new QueueType[MaxSize];

begin = 0;

end = -1;

queuelen = 0;

}

template<class QueueType>

TQueue<QueueType>::TQueue(const TQueue & \_queue)

{

MaxSize = \_queue.MaxSize;

begin = \_queue.begin;

end = \_queue.end;

queuelen = \_queue.queuelen;

queue = new QueueType[MaxSize];

for (int i = 0; i < MaxSize; i++)

queue[i] = \_queue.queue[i];

}

template<class QueueType>

void TQueue<QueueType>::Push(const QueueType \_elem)

{

if (queuelen == MaxSize)

throw queuelen;

if (end < MaxSize - 1)

end++;

else

end = 0;

queuelen++;

queue[end] = \_elem;

}

template<class QueueType>

QueueType TQueue<QueueType>::Pop()

{

if (!queuelen)

throw queuelen;

QueueType tmp = queue[begin];

if (begin < MaxSize - 1)

begin++;

else

begin = 0;

queuelen--;

return tmp;

}

template<class QueueType>

QueueType TQueue<QueueType>::Top()

{

if (!queuelen)

throw - 1;

return queue[begin];

}

template<class QueueType>

QueueType TQueue<QueueType>::Bot()

{

if (!queuelen)

throw - 1;

return queue[end];

}

template<class QueueType>

void TQueue<QueueType>::Clear()

{

begin = 0;

end = -1;

queuelen = 0;

}