Министерство образования и науки Российской Федерации

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 № 3**

**«Вычисление арифметических выражений (стеки)»**

**Выполнил**:студент группы 381703-2

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Илюхин Илья Игоревич

Подпись

**Научный руководитель**:

Доцент каф.МОСТ

Кандидат технических наук

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Сысоев А.В

Подпись

Нижний Новгород

2018

**Содержание**

Введение 3

Постановка задачи 4

Руководство пользователя 5

Руководство программиста 7

Описание структуры программы 8

Описание алгоритмов 8

Заключение 9

Литература 10

Приложение 11

**Введение**

Лабораторная работа направлена на практическое освоение динамической структуры данных Стек. С этой целью в лабораторной работе изучаются различные варианты структуры хранения стеков и разрабатываются методы и программы решения ряда задач с использованием стеков. В качестве области приложений выбрана тема вычисления арифметических выражений, возникающей при трансляции программ на языке программирования высокого уровня в исполняемые программы. При вычислении произвольных арифметических выражений возникают две основные задачи: проверка корректности введённого выражения и выполнение операций в порядке, определяемом их приоритетами и расстановкой скобок. Существует алгоритм, позволяющий реализовать вычисление произвольного арифметического выражения за один просмотр без хранения промежуточных результатов. Для реализации данного алгоритма выражение должно быть представлено в постфиксной форме. Рассматриваемые в данной лабораторной работе алгоритмы являются начальным введением в область машинных вычислений.**Постановка задачи**

В рамках лабораторной работы ставится задача реализации программ, обеспечивающих поддержку стеков, и разработки программных средств, производящих обработку арифметических выражений, включая проверку правильности записи выражения, перевод в постфиксную форму и вычисление результата. В начальной – самой простой постановке – можно предполагать, что проверка записи выражения состоит в контроле правильности расстановки скобок, перевод в постфиксную форму производится только для корректных выражений, а вычисление – для корректных выражений, содержащих только числовые операнды и допустимые знаки операций.

**Руководство пользователя**

Данная программа написана в среде разработки Microsoft Visual Studio 2017 на языке С++ и является консольным приложением. На первом этапе пользователю предлагается создать переменную класса TPostfix передав параметром строку арифметического выражения.

Пример: TPostfix postfix("1 + 2 \* 3 / 4 - 5")

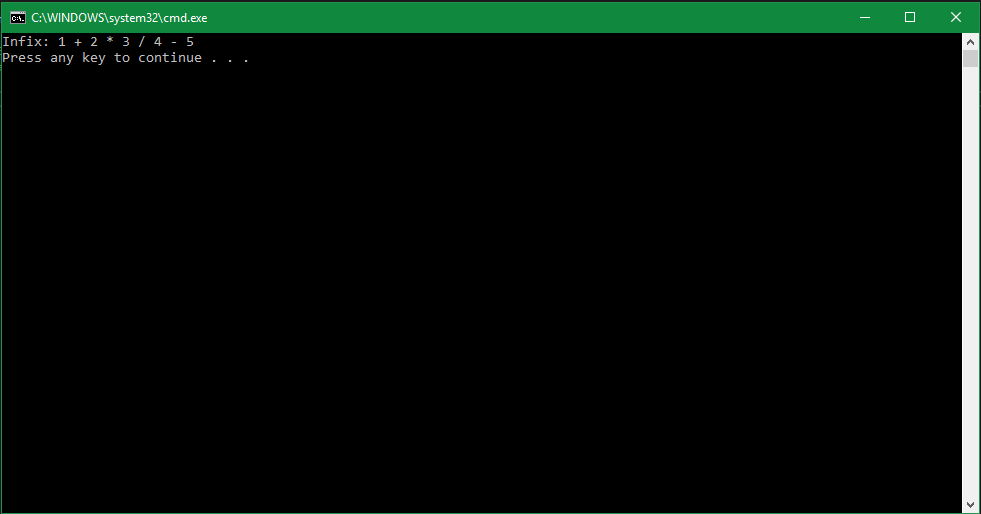
Строка выражения может включать себя вещественные числа, переменные, а также основные арифметические операции.

Рис.1. Ввод арифметического выражения

На следующем этапе пользователю предлагается произвести перевод данного арифметического выражения в постфиксную форму.

Пример: postfix.GetPostfix()

Для удобства строка постфиксной формы разделена подчеркиванием.

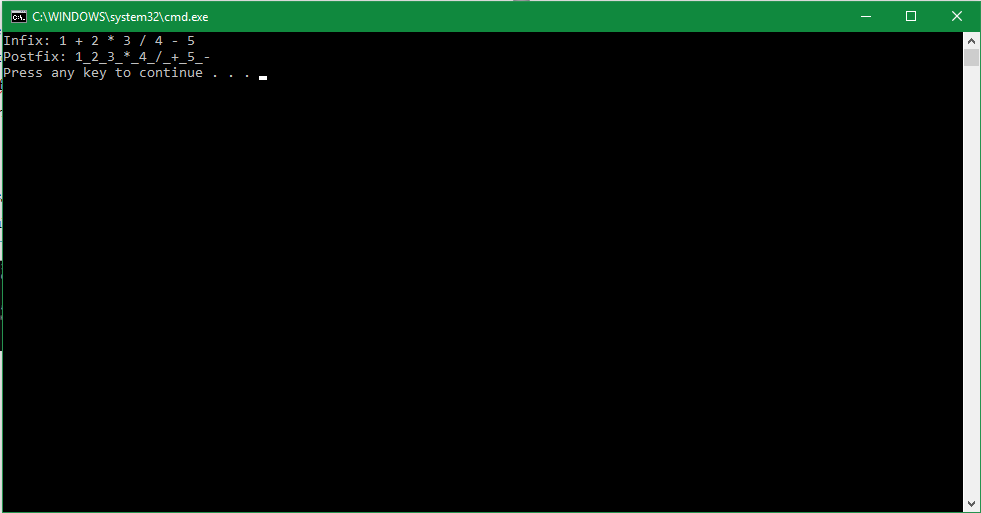


Рис.2. Перевод в постфиксную форму

Далее программисту предлагается осуществить вычисление арифметического выражения с помощью метода Calculate().

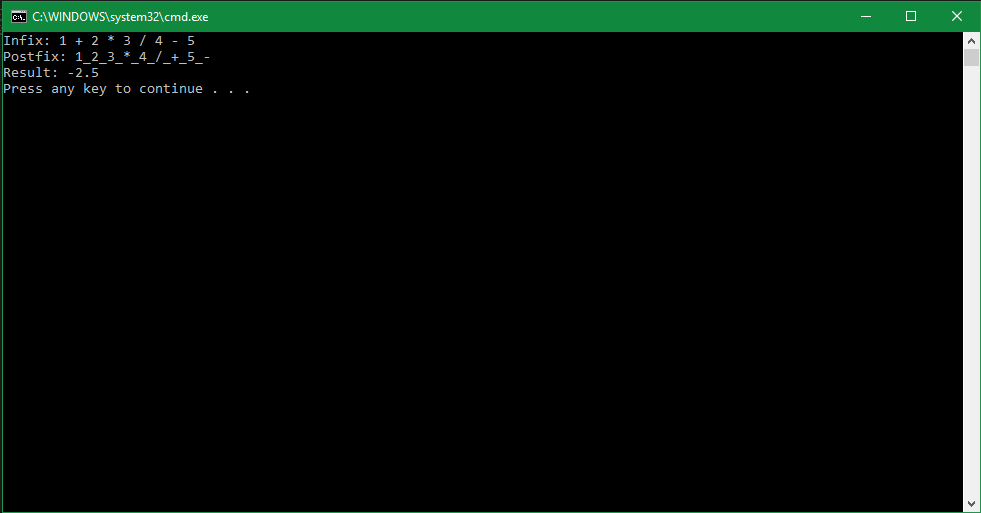


Рис.3. Вычисление выражения

**Руководство программиста**

Программа написана при помощи двух классов TStack и TPostfix.

Класс TStack содержит три приватных поля:

1. T \*pMem – Указатель шаблонного типа, отвечающий за хранения элементов стека.
2. int size – Переменная отвечающая за размер стека.
3. int top - Позиция последнего записанного элемента в стеке. По умолчанию top=-1

Класс TStack содержит следующие методы:

1. TStack(int Size) – Конструктор класса.
2. TStack<T> & operator=(const TStack<T> &s) – Оператор присваивания
3. TStack(const TStack<T> &s) – Конструктор копирования
4. bool operator==(const TStack &st) const – Оператор сравнения(равенство)
5. bool operator!=(const TStack &st) const – Оператор сравнения(неравенство)
6. T watch\_top() – Возвращает значение последнего элемента стека.
7. void push\_back(T elem) – Метод кладет в стек элемент, если он не полный, если стек заполнен, программа выдаст исключение.
8. T pop\_back() – Программа вытащит из стека верхний элемент, если стек пустой, то программа выдаст исключение.
9. bool empty() – Проверка на пустоту стека.
10. bool full () - Проверка на полноту стека.
11. int Size() – Возвращает размера стека.
12. friend ostream& operator<<(ostream &os, TStack<T> &st) -Вывод стека на экран.

Класс TPostfix содержит следующие приватные поля и методы:

1. string infix – Переменная содержащая в себе инфиксную форму выражения
2. string postfix - Переменная содержащая в себе постфиксную форму выражения

Класс TPostfix содержит следующие методы в модификаторе доступа public:

1. TPostfix() – Конструктор по умолчанию(пустой)
2. TPostfix(const string &\_inf="") – Конструктор класса, на вход которого передается арифметическое выражение
3. Tpostfix operator=(const TPostfix &p) – Перегруженный оператор присваивания
4. Void Set(const string &p) – Функция , устанавливающая инфикс
5. String GetInfix() – Возвращает инфикс
6. String GetPostfix() – Производит и возвращает постфикс
7. double Calculate() – Вычисляет арифметическое выражение

**Описание структуры программы**

Программа содержит три основных проекта:

1. base,
2. base\_test,
3. gtest.

Проект base содержит модули “stack.h” с объявлением и реализацией класса «Стек», “postfix.h” с объявлением класса «Постфикс», “postfix.cpp” с реализацией методов класса «Постфикс» и с вспомогательными функциями и “sample\_postfix.cpp” для составления пользовательского меню.

Проект base\_test содержит набор необходимых тестов Google Test, проверяющих правильность реализации основных классов.

Проект gtest содержит необходимую структуру для работы тестов Google Test.

В целом, программа построена на использовании интуитивно понятного пользователю меню. В основной части главной функции программы main создается объект типа «Постфикс», проверяется корректность арифметического выражения, считываются аргументы. Главная функция представляет собой оператор множественного выбора swich. В зависимости от выбора пользователем действия, очередной раз повторяется то или иное действие.

**Описание алгоритмов**

**TStack**

TStack(int Size) – Конструктор принимающий значение длинны стека. В данном методе произойдет выделение динамической памяти и заполнение массива стека нулями.

T watch\_top() – Функция возвращает значение последнего элемента из стека, не изменяя размер стека.

void push\_back(T elem) – Метод кладет элемент в конец стека если стек не полон. В противном случае бросается исключение.

T pop\_back() – Метод возвращает последний элемент стека, если стек не пуст. В противном случае бросается исключение. Размер стека уменьшается.

bool empty() – Метод возвращает true, если стек пустой, false – если нет.

bool full() - Метод возвращает true, если стек полон, false – если нет.

int Size() – Метод возвращает размер стека.

**TPostfix.**

GetInfix и GetPostfix: методы, возвращающие поля infix и postfix соответственно

string GetPostfix() – Метод реализующий перевод строки инфикса в постфикс. Если постфикс уже существует , метод просто вернет его. Строка рассматривается слева на право. Если встречено число, оно помещается в стек. Если оператор , то возможны следующие варианты:

* Если встречена «(» она кладется в стек.
* Если встречена «)» из стека вынимаются все операторы до «(»
* При встрече математического оператора проводится проверка на эквивалентность оператору, находящемуся в стеке. Если они эквивалентны (+, -), то сначала оператор вынимается из стека и добавляется в постфикс, затем текущий. Если более высокого порядка (+, \*), текущий оператор добавляется в стек. В других случаях из стека вынимаюстя все элементы (или до «(») и добавляются в постфикс.

По окончании цикла в постфикс добавляются оставшиеся операторы из стека.

void Set(const string &p) – Метод устанавливает значение инфикса. Также удаляет значение постфикса.

double Calculate() – Метод реализующий вычисление арифметического выражения. Сначала выпослняетя проверка на существование постфикса, если он не существует, вызывается функция GetPostfix(). Строка сформированного постфикса рассматривается слева на право. Возможны следующие ситуации:

* Если встречен разделитель, то происходит следующая итерация.
* Если встречен операнд, происходит его считывание, перевод в тип double и записывание в стек.
* Если встречена операция, из стека вытаскиваются два операнда и происходит соответствующее вычисление. Результат кладется снова в стек.

В итоге получится стек с одним элементом, это и будет результат вычисления.

**Заключение**

Это лабораторная работа поможет многим людям эффективно реализовывать вычисление арифметических выражений. Программа хорошо работает с памятью, в ней нет потери данных. Реализация стека помогает реализовывать алгоритмы более рациональным и удобным способом. Без стека программа работала бы дольше.

**Литература**

1. Герберт Шилдт - Полный справочник по C, 2007 год, издательство: «Вильямс».
2. Брайан Керниган, Деннис Ритчи «Язык программирования Си», 1978 год, издательство: «Финансы и статистика» в 1992 г с переводом на русский язык.
3. Столлингс, В. Структурная организация и архитектура компьютерных систем, 5-е изд.: Пер. с англ. — М.: Издательский дом «Вильямс», 2002. — 896 с.: ил. — Парал. тит. Англ., 2002 год.
4. Литвиненко Н. А. - Технология программирования на С++, 2014 год, издательство: «БХВ-Петербург».

**Приложение**

**Модуль stack.h**

#include <iostream>

#ifndef \_\_STACK\_H\_\_

#define \_\_STACK\_H\_\_

using namespace std;

const int MaxStackSize = 100;

template <class T>

class TStack

{

T \*pMem;

int size;

int top;

public:

TStack(int Size)

{

size = Size;

top = -1;

if ((size < 1) || (size > MaxStackSize))

throw size;

pMem = new T[size];

}

TStack<T> & operator=(const TStack<T> &s)

{

if (this == &s)

return \*this;

if (size != s.size)

{

delete[] pMem;

pMem = new T[s.size];

size = s.size;

}

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

this->pMem[i] = s.pMem[i];

top = s.top;

return \*this;

}

bool operator==(const TStack &st)

{

if (size != st.size)

return false;

if (\*this == &st)

return true;

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

if (pMem[i] != st.pMem[i])

return false;

return true;

}

int Size()

{

return size;

}

TStack(const TStack &s)

{

pMem = new T[s.size];

size = s.size;

top = s.top;

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

this->pMem[i] = s.pMem[i];

}

T watch\_top()

{

return pMem[top];

}

void push\_back(T elem)

{

if (full())

throw "Full";

pMem[++top] = elem;

}

T pop\_back()

{

if (empty())

throw "Empty";

return pMem[--top + 1];

}

bool empty()

{

return top == -1;

}

bool full()

{

return top == size - 1;

}

friend ostream& operator<<(ostream &os, TStack<T> &st)

{

for (int i = 0; i <= st.top; i++)

os << st.pMem[i] << endl;

return os;

}

~TStack()

{

delete[] pMem;

}

};

#endif

Модуль postfix.h

#ifndef \_\_POSTFIX\_H\_\_

#define \_\_POSTFIX\_H\_\_

#include <string>

#include <sstream>

using namespace std;

class TPostfix {

string infix;

string postfix;

public:

TPostfix() {}

TPostfix(const string &Inf)

{

infix = Inf;

postfix = "";

}

TPostfix(const TPostfix &p)

{

infix = p.infix;

postfix = p.postfix;

}

TPostfix operator=(const TPostfix &p)

{

infix = p.infix;

postfix = p.postfix;

return \*this;

}

void Set(const string &Inf)

{

infix = Inf;

postfix = "";

}

string GetInfix() { return infix; }

string GetPostfix();

double Calculate();

};

#endif

Модуль postfix.cpp

#include "postfix.h"

#include "stack.h"

string TPostfix::GetPostfix()

{

if (postfix != "")

return postfix;

int count = -1;

TStack<char> stack(infix.size());

auto isOperator = [](char p)

{

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

};

auto split = [&]()

{

postfix += '\_';

};

auto orderCmp = [](char a, char b)

{

return ((a == '\*' || a == '/') && (b == '+' || b == '-'));

};

auto eqCmp = [](char a, char b)

{

if ((a == '\*' || a == '/') && (b == '\*' || b == '/'))

return true;

if ((a == '-' || a == '+') && (b == '-' || b == '+'))

return true;

return false;

};

for (char part : infix)

{

if (part == ' ')

continue;

if (!isOperator(part))

{

postfix += part;

continue;

}

switch (part)

{

case '(':

stack.push\_back(part);

++count;

break;

case ')':

split();

while (stack.watch\_top() != '(')

{

postfix += stack.pop\_back();

--count;

}

stack.pop\_back();

--count;

break;

default:

split();

if (orderCmp(part, stack.watch\_top()))

{

stack.push\_back(part);

break;

}

else if (eqCmp(part, stack.watch\_top()))

{

postfix += stack.pop\_back();

stack.push\_back(part);

++count;

split();

break;

}

while (!stack.empty())

{

if (stack.watch\_top() == '(') break;

postfix += stack.pop\_back();

split();

--count;

}

stack.push\_back(part);

++count;

break;

}

}

while (count >= 0)

{

split();

postfix += stack.pop\_back();

--count;

}

return postfix;

}

double TPostfix::Calculate()

{

if (postfix == "")

GetPostfix();

TStack<double> stack(infix.size());

string temp;

auto isOperator = [](char p)

{

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

};

auto push = [&stack](string &temp)

{

if (!temp.empty())

{

for (auto i : temp)

if (!isdigit(i) && i != '.')

throw "Incorrect";

stack.push\_back(atof(temp.c\_str()));

temp.clear();

}

};

for (char part : postfix)

{

if (part == '\_')

{

push(temp);

continue;

}

if (!isOperator(part))

{

temp += part;

continue;

}

double a = stack.pop\_back();

double b = stack.pop\_back();

switch (part)

{

case '+':

stack.push\_back(b + a);

break;

case '-':

stack.push\_back(b - a);

break;

case '\*':

stack.push\_back(b \* a);

break;

case '/':

if (a == 0)

throw "Divide by null";

stack.push\_back(b / a);

break;

}

}

return stack.pop\_back();

}