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

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Сизов И.И

Подпись

**Научный руководитель**:

Доцент каф.МОСТ

Кандидат технических наук

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

Подпись

Нижний Новгород

2017

**Содержание**

Введение2

Постановка задачи3

Руководство пользователя4

Руководство программиста6

Описание структуры программы7

Описание алгоритмов7

Заключение11

Список литературы12

Приложение13

**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)\*(1-3/3\*6-4)")

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

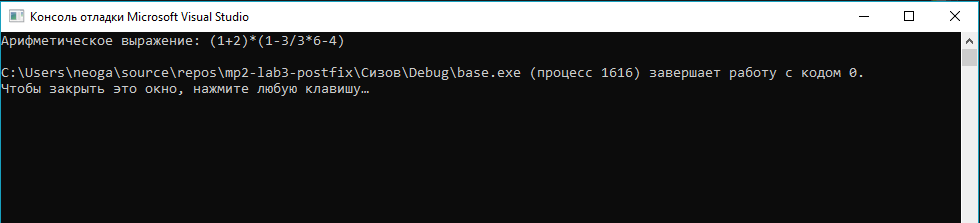


Рис.1 Ввод арифметического выражения.

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

Пример: postfix.ToPostfix()

Если выражение будет записано неправильно в инфиксной форме, тогда программа выдаст ошибку и перевод в постфиксную формы не произойдет. Для удобства строка постфиксной формы разделена запятой.

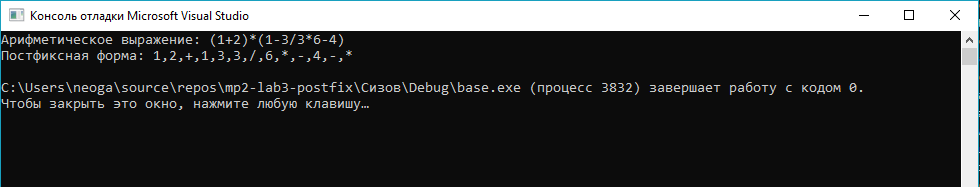


Рис.2 Перевод в постфиксную форму.

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

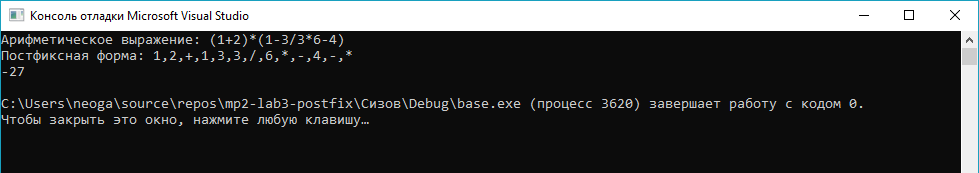


Рис3. Вычисление выражения.

Также данная программа поддерживает вычисления с переменными. При вызове метода Calculate() пользователю будет предложено инициализировать переменные, затем данное выражение будет вычислено, как бы было описано ранее.

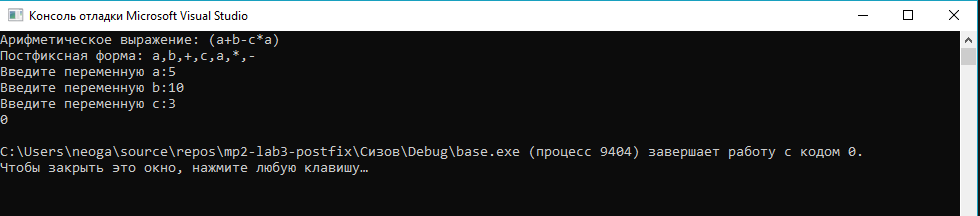


Рис.4 Инициализация переменных.

**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. Если стек пуст, 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 Back() – Возвращает значение последнего элемента стека.
7. void Put(T elem) – Метод кладет в стек элемент, если он не полный, если стек заполнен, программа выдаст исключение.
8. T Get() – Программа вытащит из стека верхний элемент, если стек пустой, то программа выдаст исключение.
9. bool IsEmpty() – Проверка на пустоту стека.
10. bool IsFull() - Проверка на полноту стека.
11. int Size() – Возвращает размера стека.
12. int Top() – Возвращает переменную top.
13. 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
3. string Compare(char a, char b) – Функция для сравнения приоритетности операции, принимает на вход два операнда, на выходе выдает результат в виде строки.
4. bool Operand(char p) – Функция определяющая является ли символ операндом.
5. bool Brackets() – Проверка на расстановку скобок, используется в методе ToPostfix
6. bool Term() – Проверка на количество операндов и операций, используется для проверки в методе Calculate()
7. void Split(string &str) – Разделяет строку постфикса запятыми для удобства.
8. void Input(string &v) – Данный метод реализует ввод переменных(если переменные есть в выражении).

Класс TPostfix содержит следующие методы в модификаторе доступа public:

1. TPostfix(string \_inf="") – Конструктор класса, на вход которого передается арифметическое выражение
2. string GetInfix() – Возвращает инфиксное выражение
3. string GetPostfix() - Возвращает постфиксное выражение
4. string ToPostfix() – Производит перевод инфиксного выражения в постфиксную форму
5. 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**

**Опишем алгоритмы класса TStack**

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

T Back() – Функция возвращает значение последнего элемента из стека, так как функция Get() не только возвращает значение, а так же вынимает элемент, не всегда удобно пользоваться данным методом. Для этого была реализована функция Back().

void Put(T elem) – Метод который кладет в стек элемент, на верх стека. В функции есть проверка, если стек полный, то программа выдаст исключение, соответственно положить значение будет невозможно. Переменная top инкрементируется.

T Get() – Функция реализует получение элемента из стека. В данном методе присутствует проверка, если стек пуст, то произойдет исключение, так как вынимать их стека будет нечего. Переменная top декрементируется.

bool IsEmpty() – Сравнивает значение переменной top. Если данная переменная равна -1 то функция вернет true. В противном случае, false.

bool IsFull() - Сравнивает значение переменной top. Если данная переменная равна size-1 то функция вернет true. В противном случае, false.

int Size() int Top() – Возвращают значения соответствующих переменных.

**Опишем алгоритмы класса TPostfix.**

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

bool Operand(char p) – Принимает на входи элемент типа char и сравнивает его со списком операндов, если p – операнд, метод вернет true в противно случае false.В списке оперндов находятся операции: + - / \* ( ).

string Compare(char a, char b) – На вход метод принимает два операнда, которые сравниваются между собой, а затем функция выдаст значение. Если приоритет операции а совпадает с приоритетом операции b, то тогда функция вернет compare. Если приоритет а больше чем b тогда функция вернет greater, если меньше то less соответственно.

void Input(string &v) – Метод позволяет инициализировать в выражении переменные. На вход она принимает строку сформированного постфикса, затем, если в строке встречены операнды, то предлагает ввести их пользователю. Если один и тот же операнд встречен в строке несколько раз, то они все будут заменены первоначальным введённым значением.

bool Brackets() – Функция определяющая, правильно ли расставлены внутри инфиксного выражения скобки, и совпадает ли количество левых скобок, с количеством правых. Вернет true если выражение прошло все проверки.

bool Term() – Данный метод осуществляет проверку выражения на количество операндов и операций. Если выражение прошло проверку, вернет true.

void Split(string &str) – Следующий метод реализует разделение всех символов запятыми, для удобного зрительного восприятия. Там, где необходимо, вызывается данный метод. Функция проверяет стоит ли на последнем месте в строке postfix запятая, если нет, то данный метод добавит ее в строку.

string ToPostfix() – Метод реализующий перевод строки инфикса в постфикс. Перед реализацией метода, происходит проверка на корректную расстановку скобок. Строка рассматривается посимвольно слева на право. Если в строке встречен операнд, он сразу же записывается в строку постфикса, если встречен операция, то возможны следующие исходы:

* Если стек пуст или элемент строки равен “(“ , то операция кладется в стек, к строке постфикса. Происходит следующая итерация.
* Если встречен операнд, который равен “)” то происходит освобождение стека и запись операций в строку постфикса, до тех пор, пока не встречена открывающая скобка. От скобок в стеке нужно избавится.
* Если последний элемент стека равен “(” или функция сравнения операций вернула значение greater, тогда происходит запись данной операции в строку постфикса, происходит следующая итерация.
* Если функция compare при сравнении последнего элемента стека и пришедшей операции вернула значение less, тогда происходит освобождение стека и запись всех операций в строку постфикса, до тех пор, пока не будет пуст стек, или не встретится открывающаяся скобка. Операцию, которая пришла на вход нужно положить в стек.
* Если функция compare при сравнении последнего элемента стека и пришедшей операции вернула значение compare тогда нужно вытащить последний элемент стека, и положить пришедшую операцию.

При необходимости вызывается функция Split для разделения строки.

double Calculate() – Метод реализующий вычисление арифметического выражения. Вначале происходит проверка на составление выражения, если выражение записано правильно, происходит ввод переменных функцией Input(). Если переменные отсутствуют, то программа продолжит выполнение без запроса ввода. Строка сформированного постфикса рассматривается слева на право. Возможны следующие ситуации:

* Если встречен разделить, 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 наш результат вычисления.

**Заключение**

Данная лабораторная работа помогает реализовывать эффективное вычисление выражений. Программа позволяет работать с памятью наилучшим образом, без потери данных. Реализация стека помогает реализовывать алгоритмы более наилучшим способом.

**Список литературы**

1. 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.
2. Брайан Керниган, Деннис Ритчи «Язык программирования Си».
3. Герберт Шилдт - Полный справочник по C

**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;//размер стека

int top;//позиция последнего записанного элемента

public:

TStack(int \_size)

{

size = \_size;

top = -1;

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

throw size;

pMem = new T[size];

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

pMem[i] = {};

}

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

{

if (this != &s)

{

if (size != s.size)

{

delete[] pMem;

pMem = new T[s.size];

size = s.size;

top = s.top;

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

pMem[i] = s.pMem[i];

}

else

{

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

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

top = s.top;

}

}

else return \*this;

}

TStack(const TStack<T> &s)

{

pMem = new T[s.size];

size = s.size;

top = s.top;

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

pMem[i] = s.pMem[i];

}

bool operator==(const TStack &st) const

{

if (this->size != st.size)

return false;

else

{

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

{

if (this->pMem[i] != st.pMem[i])

return false;

}

return true;

}

}

bool operator!=(const TStack &st) const

{

return!(\*this == st);

}

T Back()//чему равен последний элемент

{

return pMem[top];

}

void Put(T elem)

{

if (IsFull())

throw length\_error("Стек полный");

top++;

pMem[top] = elem;

}

T Get()//нумерация с нуля

{

if (IsEmpty())

throw length\_error("Стек пустой");

T tmp = pMem[top];

pMem[top] = {};

top--;

return tmp;

}

bool IsEmpty()

{

if (top == -1)

return true;

else return false;

}

bool IsFull()

{

if (top == size - 1)

return true;

else return false;

}

int Size()

{

return size;

}

int Top()

{

return top;

}

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;

string Compare(char a, char b);

bool Operand(char p);

bool Brackets();

bool Term();

void Split(string &str);

void Input(string &v);

public:

TPostfix(string \_inf="")

{

infix = \_inf;

}

string GetInfix() { return infix; }

string GetPostfix() { return postfix; }

string ToPostfix();

double Calculate();

};

#endif

Модуль postfix.cpp

#include "postfix.h"

#include "stack.h"

bool TPostfix::Operand(char p)

{

if (p == '+' || p == '-' || p == '\*' || p == '/' || p == '(' || p == ')')

return true;

else return false;

}

string TPostfix::Compare(char a, char b)

{

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

return "compare";

else if ((a == '+' || a == '-') && (b == '+' || b == '-'))//одного порядка

return "compare";

else if ((a == '\*' || a == '/') && (b == '+' || b == '-'))//большего порядка

return "greater";

else if ((a == '+' || a == '-') && (b == '\*' || b == '/'))//меньшего порядка

return "less";

//else return"last elem if bracket";

}

void TPostfix::Input(string &v)

{

string tmp;

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

{

if((int(v[i])>=65&&int(v[i])<=90)||(int(v[i])>=97&&int(v[i])<=122))//значения переменных в диапазоне

{

cout << "Введите переменную " << v[i]<<':';

cin >> tmp;

char var = v[i];

int j = i;

while (j < v.size())//замена всех одинаковых переменных в строке

{

if (var == v[j])

{

v.erase(j, 1);

v.insert(j, tmp);

j += tmp.size();

}

else j++;

}

}

}

}

bool TPostfix::Brackets()//проверка на расстановку скобок

{

int left = 0, right = 0;

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

{

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

left++;

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

right++;

if (right > left)

throw"ошибка в скобках";

}

if (left == right)

return true;

else return false;

}

bool TPostfix::Term()//проверка на колличество операндов и операций

{

int operands = 0, operations = 0;

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

{

if (infix[i] == '+' || infix[i] == '-' || infix[i] == '\*' || infix[i] == '/')

operations++;

else if (!Operand(infix[i]))

{

while (infix[i] != '+' && infix[i] != '-' && infix[i] != '\*' && infix[i] != '/')

{

i++;

if (i == infix.size())

break;

}

i--;

operands++;

}

}

if (operands == operations + 1)

return true;

else return false;

}

void TPostfix::Split(string &str)//разделить запятыми строчку

{

if (str.back() != ',')

str += ',';

}

string TPostfix::ToPostfix()

{

if (!Brackets())

throw"ошибка в расстановке скобок";

int count = -1;//счетчик операндов

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

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

{

if (Operand(infix[i]))

{

if (op.IsEmpty() || infix[i] == '(')

{

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

Split(postfix);

op.Put(infix[i]);

count++;

continue;

}

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

{

while (op.Back() != '(')

{

Split(postfix);

postfix += op.Get();

count--;

}

op.Get();

count--;

}

else if ((op.Back()) == '(' || Compare(infix[i], op.Back()) == "greater")

{

op.Put(infix[i]);

Split(postfix);

count++;

}

else if (Compare(infix[i], op.Back()) == "less")

{

Split(postfix);

while (!(op.IsEmpty()))

{

if (op.Back() == '(')

break;

postfix += op.Get();

Split(postfix);

count--;

}

op.Put(infix[i]);

count++;

}

else if (Compare(infix[i], op.Back()) == "compare")

{

Split(postfix);

postfix += op.Get();

op.Put(infix[i]);

Split(postfix);

}

}

else

postfix += infix[i];

if (i == infix.size() - 1)

{

for (count; count >= 0; count--)

{

Split(postfix);

postfix += op.Get();

}

}

}

return postfix;

}

double TPostfix::Calculate()

{

if (!Term()||postfix=="")

throw "Недостаточно операндов";

Input(postfix);

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

string tmp;

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

{

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

continue;

if (!Operand(postfix[i]))

{

while (postfix[i] != ',')

{

tmp += postfix[i];

i++;

}

result.Put(atof(tmp.c\_str()));

tmp = "";

}

else

{

double t2 = result.Get(), t1 = result.Get();

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

result.Put(t1 + t2);

else if (postfix[i] == '-')

result.Put(t1 - t2);

else if (postfix[i] == '\*')

result.Put(t1 \* t2);

else if (postfix[i] == '/')

{

if (t2 == 0)

throw "Деление на ноль невозможно";

result.Put(t1 / t2);

}

}

}

return result.Get();

}