**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**

**«Преобразование и вычисление арифметических выражений в постфиксной форме»**

**Выполнил**:  
студент группы 381703-2

Николаев Д.Э.

**Научный руководитель**:

Кандидат технических наук

Сысоев А.В.

Нижний Новгород

2017

**Содержание**

Введение - 2

Постановка задачи - 2

Руководство пользователя - 3

Руководство программиста - 4

Описание структуры программы - 6

Описание алгоритмов - 7

Заключение - 8

Литература - 8

Приложение - 9

**Введение**

Цель данной лабораторной работы - изучение и применение однонаправленных списков стек. Для этого используются различные варианты структуризации стеков, а также создаются методы алгоритмы вычисления и преобразования данных с помощью стеков. В данном случае - вычисление арифметических выражений в постфиксной форме. Существует алгоритм, позволяющий реализовать вычисление произвольного арифметического выражения линейно, без использования дополнительной памяти. Для реализации данного алгоритма выражение должно быть представлено в постфиксной форме.

**Постановка задачи**

В данной лабораторной работе ставится задача преобразования арифметического выражения в постфиксную форму с последующим вычислением результата.

**Руководство пользователя**

Решение разрабатывается на языке C++, стандарта C++17. Для простейшего взаимодействия с пользователем реализовано консольное приложение. Пользователь должен ввести арифметическое выражение в поле. Строка выражения может включать себя целые и вещественные числа, а также основные арифметические операции.

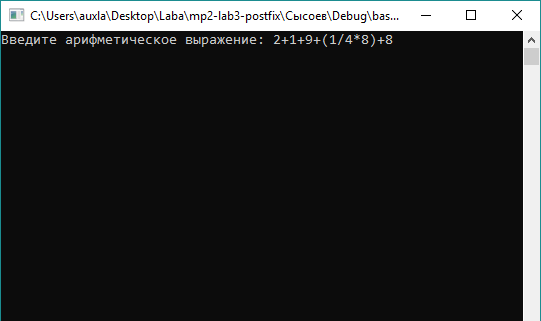


Рис.1. Ввод арифметического выражения

После ввода выражения будет продемонстрирована его постфиксная форма и результат.

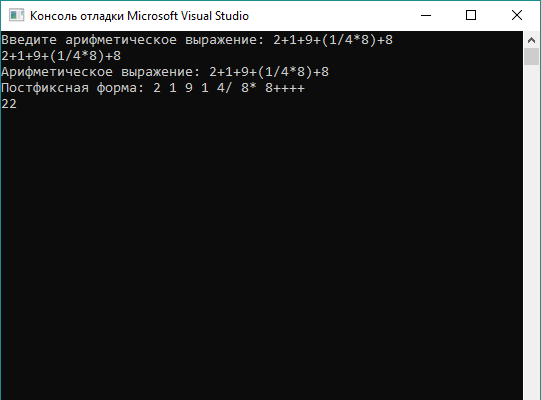


Рис.2. Постфиксная форма

**Руководство программиста**

Программа реализована с помощью двух классов TStack и TPostfix, последний из которых основан на первом. Далее подробно о каждом.

Класс TStack – список элементов, организованных по принципу «последним пришел – первым вышел».

Приватные поля:

1. Type \*pMem – Шаблонный указатель для массива элементов типа <Type>
2. int size – Переменная отвечающая за размер стека.
3. int top – Позиция последнего записанного элемента в стеке, по умолчанию top = -1

Публичные методы:

1. bool IsEmpty() - Возвращает истину, если стек пуст.
2. bool IsFull() - Возвращает истину, если стек полон. Максимальный размер стека задается конструктором класса.
3. Type Pop() - Возвращает и удаляет из очереди последний элемент списка pMem типа Type.
4. void Push(Type&) - Добавляет в конец очереди элемент типа Type. Если стек переполнен, операция будет прервана.
5. int Size() - Возвращает текущий размер стека.
6. int Top() - Возвращает текущую позицию top в стеке.
7. Type GetTop() - Возвращает последний элемент списка pMem без удаления его из очереди.

Класс TPostfix - специальный класс для преобразования и вычисления арифметический выражений.

Приватные поля:

1. string infix - хранит в себе инфиксную (исходную) форму выражения
2. string postfix - хранит в себе постфиксную форму выражения
3. TStack<string> \_T - хранит в себе промежуточные преобразования.

Приватные методы:

1. string trimString(string) - удаляет лишние символы в постфиксной форме, выполняет постобработку строки.
2. string convertString(string) - удаляет лишние символы в инфиксной форме, выполняет предобработку строки.
3. double decodeVal(string) - переводит число из строкового типа в вещественный тип.
4. string getVal(string&,size\_t) - возвращает первое встречное с позиции start вещественное число и удаляет его из исходной строки.
5. void ToPostfixIteration(TStack<string>&,string) - Производит рекурсивную обработку выражения.

Публичные методы:

1. string GetInfix() - возвращает инфиксную форму выражения
2. string GetPostfix() - возвращает постфиксную форму выражения
3. string ToPostfix(string=”\0”) - возвращает преобразование строки инфиксного вида в постфиксную. Вызов функции без аргументов вернет преобразование строки инфиксного вида текущего объекта класса.
4. double Calculate(string=”\0”) - возвращает результат вычисления постфиксной строки. Вызов функции без аргументов вернет результат вычисления строки постфиксного вида текущего объекта класса.
5. void SetInfix(string) - задает текущему объекту класса строку инфиксного вида.

**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 switch. В зависимости от выбора пользователем действия, очередной раз повторяется то или иное действие.

**Описание алгоритмов**

**Алгоритм класса** **TStack.**

Принцип работы этого класса достаточно прост. Изначально происходит инициализация объекта, с последующим выделением необходимого объема памяти для хранения всех элементов шаблонного типа Type.

После добавления элемента командой push происходит копирование значений в свободную ячейку списка pMem, инкрементируется счетчик списка top. Если свободных ячеек нет, то операция прерывается.

После вызова команды pop возвращается последний элемент списка pMem, декрементируется счетчик списка top. Если список пуст, будет вызвано исключение.

Остальные методы были описаны выше.

**Алгоритм класса TPostfix.**

При инициализации объекта класса или установления инфиксной формы методом SetInfix(string) происходит первичная обработка строки, знак вычитания преобразуется в сложение отрицательных чисел, удаляются лишние пробелы.   
Обработка стандартизированной строки происходит в 3 этапа:

1. В выражении происходит поиск подвыражений в скобках. В исходной строке ставится “x”. Выражение в скобках отправляется в рекурсивную обработку, пока данный этап не будет пройден окончательно.
2. Происходит подсчет и запись операций в дополнительный стек. Операции с наибольшим приоритетом будут последними в списке.
3. Проходя по строке, числа записываются в дополнительный стек, а операции сравниваются с последней операцией из дополнительного стека. Если происходит совпадение, тогда вынимаются 2 последних числа и последняя операция из дополнительных стеков. Числа записываются последовательно в результирующую строку, а операция отправляется в буфер. При следующем совпадении сначала будет проверено, совпадает ли операция из буфера. Если да, тогда буфер пополняется новой операцией, иначе буферные операции записываются в результирующую строку, потом новые числа. Буфер обновляется новой операцией.

Третий пункт повторяется до тех пор, пока строка не будет пуста. Результирующая строка записывается в стек промежуточных вычислений \_T.

После завершения каждый “x” в строке будет последовательно заменен на последний элемент стека \_T. Строка проходит постобработку и возвращается.

**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.

**Литература**

1. Столлингс В. «Структурная организация и архитектура компьютерных систем, 5-е изд.»
2. Брайан Керниган, Деннис Ритчи «Язык программирования Си».
3. Герберт Шилдт - Полный справочник по C

**Приложение**

**Модуль stack.h**

const int MaxStackSize = 100;

template <class Type>

class TStack{

Type \*pMem;

int size;

int top;

public:

TStack(int \_size): size(\_size){

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

throw size;

}

top = -1;

pMem = new Type[size];

}

TStack(TStack& c): size(c.size),top(c.top) {

pMem = new Type[size];

for (size\_t i = 0; i < size; i++) {

pMem[i] = c.pMem[i];

}

}

bool IsEmpty() {

return (top == -1);

}

bool IsFull() {

return (top == size - 1);

}

Type Pop() {

return pMem[top--];

}

void Push(Type v) {

if (IsFull()) {

return;

}

pMem[++top] = v;

}

int Size() {

return size;

}

int Top() {

return top;

}

Type GetTop() {

return pMem[top];

}

TStack& operator=(TStack &c) {

if (size != c.size) {

size = c.size;

delete[] pMem;

pMem = new Type[size];

}

top = c.top;

for (size\_t i = 0; i < size; i++) {

pMem[i] = c.pMem[i];

}

return (\*this);

}

bool operator==(const TStack &c) const {

if (size != c.size) {

return false;

}

if (top != c.top) {

return false;

}

for (size\_t i = 0; i < size; i++) {

if (pMem[i] != c.pMem[i]) {

return false;

}

}

return true;

}

bool operator!=(const TStack &c) const {

return !(\*this == c);

}

~TStack(){

delete [] pMem;

}

};

**Модуль postfix.h**

#include <string>

#include "stack.h"

using namespace std;

class TPostfix {

string infix;

string postfix;

TStack<string> \_T;

string trimString(string source);

string convertString(string source);

double decodeVal(string value);

string getVal(string& source, size\_t start);

void ToPostfixIteration(TStack<string>& \_T, string source);

public:

TPostfix(string base = ""): \_T(100) {

infix = convertString(base);

postfix = ToPostfix();

}

TPostfix(TPostfix& c) : \_T(100) {

infix = c.infix;

postfix = ToPostfix();

}

string GetInfix() { return infix; }

string GetPostfix() { return postfix; }

string ToPostfix(string source = "\0");

double Calculate(string source = "\0");

void SetInfix(string base);

};

**Модуль postfix.cpp**

#include "postfix.h"

#include "stack.h"

string TPostfix::trimString(string source) {

if (source.length() < 2) {

return source;

}

if (source[0] == ' ') {

source.erase(0, 1);

}

if (source[source.length() - 1] == ' ') {

source.erase(source.length() - 1, 1);

}

for (int i = 0; i < source.length(); i++) {

char c = source[i];

if (c == ' ') {

if (i < source.length() - 1) {

char \_c = source[i + 1];

if (\_c == '+' || \_c == '/' || \_c == '\*') {

source.erase(i, 1);

i--;

}

}

}

}

return source;

}

string TPostfix::convertString(string source) {

for (size\_t i = 0; i < source.length(); i++) {

switch (source[i]) {

case ' ':

source.erase(i, 1);

i--;

continue;

case '-':

source.replace(i, 1, "+-");

i++;

continue;

default:

if ((int)source[i] >= (int)'a' && (int)source[i] <= (int)'z') {

throw source[i];

}

break;

}

}

return source;

}

double TPostfix::decodeVal(string value) {

double div = 1;

double result = 0;

bool negative = false;

const int sta = (int)('0');

const int end = (int)('9');

const int exc\_1 = (int)('.');

const int exc\_2 = (int)('-');

for (auto c : value) {

int ch = (int)c;

switch (ch) {

case exc\_1:

div = 0.1;

continue;

case exc\_2:

negative = !negative;

break;

default:

if (ch >= sta && ch <= end) {

if (div > 0.5) {

result \*= 10;

}

result += (ch - sta)\*div;

}

}

if (div < 1) {

div \*= 0.1;

}

}

if (negative) {

result \*= -1;

}

return result;

}

string TPostfix::getVal(string& source, size\_t start) {

string val;

const int sta = (int)('0');

const int end = (int)('9');

const int exc\_1 = (int)('.');

const int exc\_2 = (int)('-');

const int exc\_3 = (int)('x');

for (int i = 0; start + i < source.length();) {

int c = (int)source[start + i];

switch (c) {

case '+':

case '\*':

case '/':

if (val.length() == 0) {

i++;

continue;

}

default:

if (c < sta || end < c) { return val; }

case exc\_1:

case exc\_2:

case exc\_3:

val += source[start + i];

source.erase(start + i, 1);

}

}

return val;

}

void TPostfix::ToPostfixIteration(TStack<string>& \_T, string source) {

TStack<char> O(100);

TStack<string> N(100);

string result = "";

char last = '\0';

size\_t lastPos;

for (int i = 0; i < source.length(); i++) {

char c = source[i];

if (c == '(') {

for (size\_t j = i, off = 0; j < source.length(); j++) {

char \_c = source[j];

if (\_c == '(') {

off++;

}

else if (\_c == ')') {

string h = source.substr(i + off, j - i - off);

ToPostfixIteration(\_T, h);

source.replace(i + off - 1, j - i - off + 2, "x");

i = -1;

}

}

}

}

for (int i = 0; i < source.length(); i++) {

char c = source[i];

if (c == '/') {

size\_t sta;

size\_t end;

for (sta = i; sta > 0; sta--) {

if (source[sta] == '\*' || source[sta] == '+') {

sta++;

break;

}

}

for (end = i; end < source.length(); end++) {

if (source[end] == '\*' || source[end] == '+') {

break;

}

}

if (sta == 0 && end == source.length()) {

break;

}

string h = source.substr(sta, end - sta);

ToPostfixIteration(\_T, h);

source.replace(sta, end - sta, "x");

i = -1;

}

}

string base = source;

while (base.length() > 0) {

last = '\0';

for (int i = 0; i < base.length(); i++) {

switch (char c = base[i]) {

case '+':

if (last == '\0' || last == '\*' || last == '/') {

last = c;

lastPos = i;

}

break;

case '\*':

case '/':

if (last == '\0') {

last = c;

lastPos = i;

}

break;

default:

base.erase(i, 1);

i--;

}

}

if (last != '\0') {

O.Push(last);

base.erase(lastPos, 1);

}

}

string tempO = " ";

for (int i = 0; i < source.length(); i++) {

char c = source[i];

if (c == '+' || c == '\*' || c == '/') {

if (c == O.GetTop()) {

char tr = O.Pop();

char td = tempO[tempO.length() - 1];

if (td == ' ' || td == c && (td == '\*' || td == '+')) {

if (!N.IsEmpty()) {

result += ' ' + N.Pop();

}

tempO += tr;

}

else {

if (!N.IsEmpty()) {

result += ' ' + N.Pop();

}

result += tempO;

tempO = tr;

}

source.erase(i, 1);

i--;// = -1;

}

else {

if (!N.IsEmpty()) {

result += ' ' + N.Pop();

}

}

}

else {

N.Push(getVal(source, i));

i = -1;

}

}

if (!N.IsEmpty()) {

result += ' ' + N.Pop();

}

result += tempO;

\_T.Push(result);

}

string TPostfix::ToPostfix(string source) {

if (source == "\0") {

source = infix;

}

else {

source = convertString(source);

}

if (source.length() < 1) {

return source;

}

ToPostfixIteration(\_T, source);

string temp = \_T.Pop();

while (!(\_T.IsEmpty())) {

for (int i = temp.length(); i >= 0; --i) {

if (temp[i] == ' ') {

if (i + 1 == temp.length()) {

temp.erase(i, 1);

}

else {

if (temp[i + 1] == ' ' || temp[i + 1] == '+' || temp[i + 1] == '/' || temp[i + 1] == '\*') {

temp.erase(i, 1);

}

}

}

if (temp[i] == 'x') {

temp.erase(i, 1);

string h = \_T.Pop();

temp.insert(i, h);

i = temp.length() - 1;

}

}

}

temp = trimString(temp);

return temp;

}

double TPostfix::Calculate(string source) { // Ввод переменных, вычисление по постфиксной форме

TStack<double> \_D(100);

if (source == "\0") {

source = postfix;

}

for (int i = 0; i < source.length(); i++) {

switch (source[i]) {

default: {

\_D.Push(decodeVal(getVal(source, i)));

i = -1;

continue;

}

case ' ': {

break;

}

case '+': {

\_D.Push(\_D.Pop() + \_D.Pop());

break;

}

case '\*': {

\_D.Push(\_D.Pop() \* \_D.Pop());

break;

}

case '/': {

double temp = \_D.Pop();

if (temp == 0) {

throw temp;

}

\_D.Push(\_D.Pop() / temp);

break;

}

}

source.erase(0, 1);

i = -1;

}

return \_D.Pop();

}

void TPostfix::SetInfix(string base) {

infix = convertString(base);

postfix = ToPostfix();

while (!\_T.IsEmpty()) {

\_T.Pop();

}

}