Министерство образования и науки РФ

Нижегородский государственный университет имени Н.И. Лобачевского (Национальный исследовательский университет)

Институт Информационных Технологий Математики и Механики

Отчет на тему:

«Перевод арифметического выражения в постфиксную форму записи и вычисления первого за счет последнего»

Выполнил:   
 студент группы 0823-3 Панков Максим

Проверил:   
 Козинов Евгений Александрович

Нижний Новгород

2016

**Содержание**

[**Введение**](#Введение)**.3**

[**Постановка учебно-практической задачи**](#Постанов_учебно_практической_задачи)4

[**Руководство пользователя**](#Руководство_пользователя) **5**

[**Руководство программиста**](#Руководство_программиста)**7**

[**Заключение**](#Заключение)**15**

**[Список литературы](#Список_литературы)17**

[**Приложение**](#приложение)**18**

**Введение**

Обратная польская нотация (ОПН)- форма записи математических и логических выражений, в которой операнды расположены перед знаками операций. Обратная польская запись имеет ряд преимуществ перед инфиксной записью при выражении алгебраических формул. Во-первых, любая формула может быть выражена без скобок. Во-вторых, она удобна для вычисления формул в машинах со стеками. В-третьих, инфиксные операторы имеют приоритеты, которые произвольны и нежелательны. Данное приложение позволяет переводить арифметические выражения из привычной нам инфиксной нотации в ОПН, а так же решать эти самые арифметические выражения при помощи ОПН и тестировать код программы при помощи фреймворка Google Test.

**Постановка учебно-практической задачи**

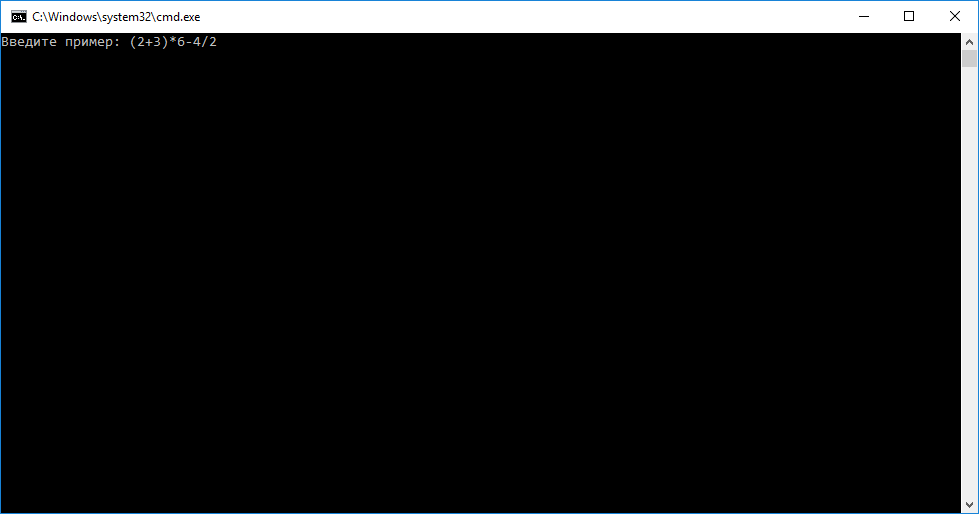
Необходимо создать программу, которая выполняет следующие действия:

1. Принимает арифметическое выражение в инфиксной форме записи (ИФЗ);
2. Переводит выражение из ИФЗ в ОПН при помощи класса TStack и выводит его на экран;
3. Решает арифметическое выражение в форме ОПН выводит результат на экран;
4. Тестирует код программы при помощи фреймворка Google Test.

**Руководство пользователя:**

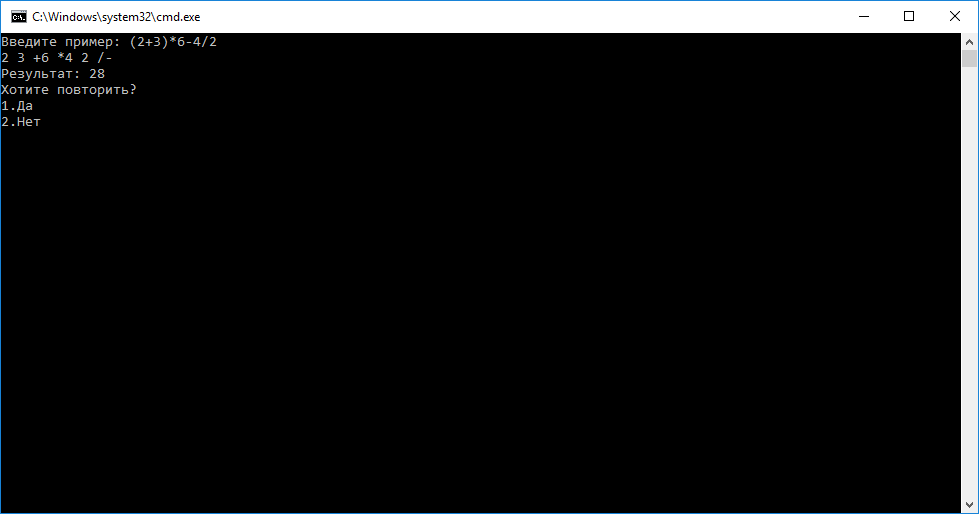
Программа предлагает набор возможностей:

1.Ввод выражения:

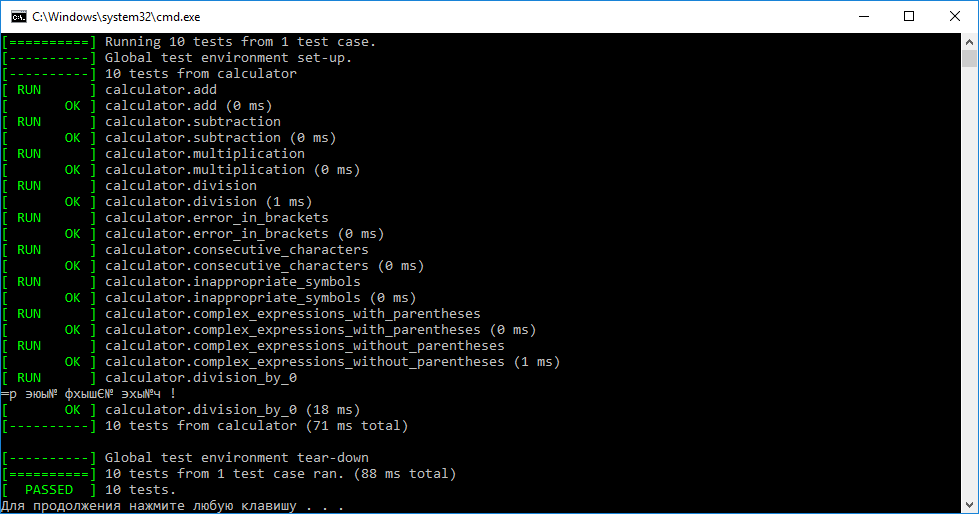
Для этого необходимо назначить проект «sample» автозагружаемым проектом и запустить программу. Далее, нужно ввести желаемое выражение в инфиксной форме записи.

2.Первод в ОПН и решение арифметического выражения:

После того, как вы ввели арифметическое выражение, необходимо нажать клавишу ENTER.После этого на экран выйдет исходное выражение в форме ОПН и ответ к нему. После этого по желанию можно продолжить или прекратить работу.



3.Тестирование приложения:

Для этого необходимо назначить проект «tests» автозагружаемым проектом и запустить программу. 

Как видим, все тесты пройдены.

**Руководство программиста**

***Описание структуры данных:***

Программа работает при помощи структуры данных стек. В программе, он базируется в проекте «stack» в TStack.h и TStack.cpp.

***Описание алгоритмов:***

***1.Стек:***

Стек в программе имеет следующие **методы**:

TStack(const TStack &s)-конструктор копирования

TStack()-инициализирующий конструктор

~TStack(void)-деструктор

TStack(int n)-конструктор преобразования

int IsEmpty(void) const-проверка стека на пустоту

int IsFull(void) const- проверка стека на полноту

void Put(const int val)-добавление элемента в стек

int Get(void)-«взятие» элемента из стека

int Peek(void)-просмотр элемента из стека

***Код:***

**#include "TStack.h"**

**TStack::TStack(const TStack & v)**

**{**

**Size = v.Size;**

**pMem = new int[Size];**

**top = v.top;**

**for (int i = 0; i < Size; i++)**

**{**

**pMem[i] = v.pMem[i];**

**}**

**}**

**TStack::TStack()**

**{**

**top = -1;**

**Size = MemSize;**

**pMem = new int[Size];**

**for (int i = 0; i < Size; i++)**

**{**

**pMem[i] = 0;**

**}**

**}**

**TStack::~TStack(void)**

**{**

**delete[]pMem;**

**}**

**TStack::TStack(int Si)**

**{**

**top = -1;**

**Size = Si;**

**pMem = new int[Size];**

**for (int i = 0; i < Size; i++)**

**{**

**pMem[i] = 0;**

**}**

**}**

**int TStack::IsEmpty(void) const**

**{**

**return top == -1;**

**}**

**int TStack::IsFull(void) const**

**{**

**return top == Size - 1;**

**}**

**void TStack::Put(int val)**

**{**

**pMem[++top] = val;**

**}**

**int TStack::Get(void)**

**{**

**return pMem[top--];**

**}**

**int TStack::Peek(void)**

**{**

**return pMem[top];**

}

***2.Перевод в ОПН:***

Алгоритм перевод в ОПН состоит из двух частей:

1. Вспомогательной функции «priority», в которой для каждой из операций задаются сои приоритеты (так, операции ‘+’ и ‘-‘ имеют приоритет 2, ‘\*’ и ‘/’ - приоритет 3 ,а ‘(‘ и ‘)’ приоритет 0 и 1 соответственно).
2. Функции «polish\_notation».Сначала, программа проходит с помощью цикла по исходному выражению. Если ей попадаются числа(в том числе и двух-,трех- и т.д –значные числа),то программа записывает их результативную строку. Если же ей попадаются знаки операций, то:

a.Если стек пуст, выбранный знак – ‘(‘ или приоритет выбранного знака больше, чем приоритет знака в верхушке стека, то программа поместит этот знак в стек. Если это не так, то программа будет записывать значения стека в результативную строку до тех пор, пока стек не будет пустым и приоритет выбранного знака будет меньше или равен приоритету знака в верхушке стека. После этого, программа поместит выбранный знак в стек.

b.Если выбранный знак – ‘)‘ ,то программа будет записывать значения стека в результативную строку до тех пор, пока её не попадется знак ‘(‘.

После всего, программа будет записывать оставшиеся значения стека в результативную строку до тех пор, пока стек не будет пустым.

***Код:***

**int priority(char str\_s)**

**{**

**if (str\_s == '(')**

**{**

**return 0;**

**}**

**if (str\_s == ')')**

**{**

**return 1;**

**}**

**if (str\_s == '+')**

**{**

**return 2;**

**}**

**if (str\_s == '-')**

**{**

**return 2;**

**}**

**if (str\_s == '\*')**

**{**

**return 3;**

**}**

**if (str\_s == '/')**

**{**

**return 3;**

**}**

**else**

**{**

**return -1;**

**}**

**}**

**void polish\_notation(char \* str, char \* result)**

**{**

**TStack Stack;**

**int str\_i = 0, i = 0;**

**while (str[i] != NULL)**

**{**

**while ((str[i] <= '9') && (str[i] >= '0'))**

**{**

**result[str\_i++] = str[i++];**

**if ((str[i] > '9') || (str[i] < '0'))**

**{**

**result[str\_i++] = ' ';**

**}**

**}**

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

**{**

**if ((Stack.IsEmpty()) || (str[i] == '(') || (priority(str[i]) > priority(Stack.Peek())))**

**{**

**Stack.Put(str[i]);**

**}**

**else**

**{**

**while ((!Stack.IsEmpty()) && (priority(str[i]) <= priority(Stack.Peek())))**

**{**

**result[str\_i++] = Stack.Get();**

**}**

**Stack.Put(str[i]);**

**}**

**i++;**

**}**

**if (str[i] == ')')**

**{**

**while (Stack.Peek() != '(')**

**{**

**result[str\_i++] = Stack.Get();**

**}**

**Stack.Get();**

**i++;**

**}**

**}**

**while (!Stack.IsEmpty())**

**{**

**result[str\_i++] = Stack.Get();**

**}**

**}**

**3**.***Калькулятор***:

Алгоритм вычисления операций из ОПН:

1. Сначала, программа проходит с помощью цикла по исходному выражению. Если ей попадаются числа(в том числе и двух-,трех- и т.д –значные числа),то программа записывает их в стек.
2. Если ей попадается один из допустимых знаков, то программа возьмет число из вершины стека и проведет необходимую операцию со следующим числом из стека и добавит результат в стек. При всем этом, делить на 0 нельзя.
3. Программа выводит результат.

***Код:***

**TStack Stack;**

**int result = 0, elem;**

**int a, i = 0;**

**while (str[i] != NULL)**

**{**

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

**{**

**elem = 0;**

**while (str[i] != ' ')**

**{**

**elem = elem \* 10 + (str[i++] - 48);**

**}**

**Stack.Put(elem);**

**i++;**

**}**

**if (str[i] == '\*')**

**{**

**a = Stack.Get();**

**result = Stack.Get() \* a;**

**Stack.Put(result);**

**i++;**

**}**

**if (str[i] == '/')**

**{**

**a = Stack.Get();**

**if (a != 0)**

**{**

**result = Stack.Get() / a;**

**Stack.Put(result);**

**i++;**

**}**

**else**

**{**

**cout << "На ноль делить нельзя!" << endl;**

**result = NULL;**

**return result;**

**}**

**}**

**if (str[i] == '+')**

**{**

**a = Stack.Get();**

**result = Stack.Get() + a;**

**Stack.Put(result);**

**i++;**

**}**

**if (str[i] == '-')**

**{**

**a = Stack.Get();**

**if (Stack.IsEmpty())**

**{**

**Stack.Put(0);**

**}**

**result = Stack.Get() - a;**

**Stack.Put(result);**

**i++;**

**}**

**}**

**return result;**

**4.*Защита от дурака:***

Функция «foolproof»-вспомогательная функция для наложения некоторых ограничений для исходной строки, таких как проверка на неподходящие символы, корректность 1-го и последнего символов, проверка скобок и т.д.

***Код:***

**int foolproof(char \*str)**

**{**

**int ch = 0;**

**int i = 0;**

**int q = 0, p = 0;**

**TStack skob;**

**while (str[i] != NULL) //Неподходящие символы(ASCII)**

**{**

**if ((str[i]<40) || (str[i]>57) || (str[i] == 44) || (str[i] == 46) || (str[i] == 32) )**

**{**

**ch = 1;**

**}**

**i++;**

**}**

**i = 0;**

**if ((str[0] > 57) || (str[0] < 48) && (str[0] != 45) && (str[0] != 40))//Корректность 1-го символа**

**{**

**ch = 1;**

**}**

**if ((str[strlen(str)-1] > 57) || (str[strlen(str) - 1] < 48) && (str[strlen(str) - 1] != 41))//Корректность последнего символа**

**{**

**ch = 1;**

**}**

**while (str[i] != NULL) //Парность/непарность скобок**

**{**

**if (str[i] == 40)**

**{**

**q++;**

**}**

**if (str[i] == 41)**

**{**

**p++;**

**}**

**i++;**

**}**

**if (q != p)**

**{**

**ch = 1;**

**}**

**i = 0; q = 0; p = 0;**

**while (str[i] != NULL) //Порядок скобок**

**{**

**if (str[i] == '(')**

**{**

**skob.Put(str[i]);**

**i++;**

**continue;**

**}**

**if (str[i] == ')')**

**{**

**if (skob.IsEmpty())**

**{**

**ch = 1;**

**}**

**}**

**i++;**

**}**

**i = 0;**

**while (str[i] != NULL) //Повторение символов**

**{**

**if ((str[i] >= 42) && (str[i] <= 47) && (str[i + 1] >= 42) && (str[i + 1] <= 47))**

**{**

**ch = 1;**

**}**

**i++;**

**}**

**i = 0;**

**return ch;**

**}**

*Описание структуры программного комплекса*

**TStack.h** и **TStack.cpp –** заголовочный и .cpp файлы, в которых находится класс стек.

**Calculator.h –** заголовочный файл, в котором находятся основных операций, такие как перевод в ОПН, решение арифметических выражений и т.д.

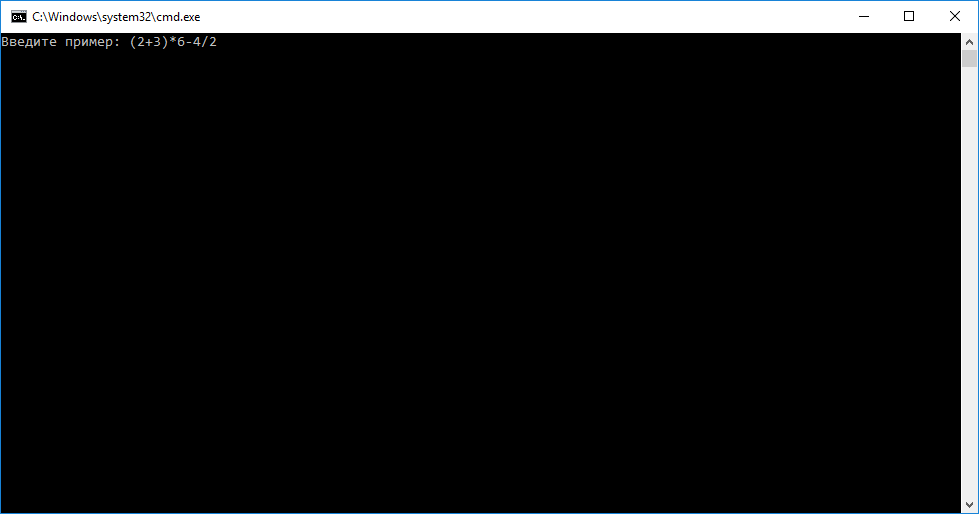
**Main.cpp –** файл, в котором находится функция main, реализующая взаимодействие с пользователем.

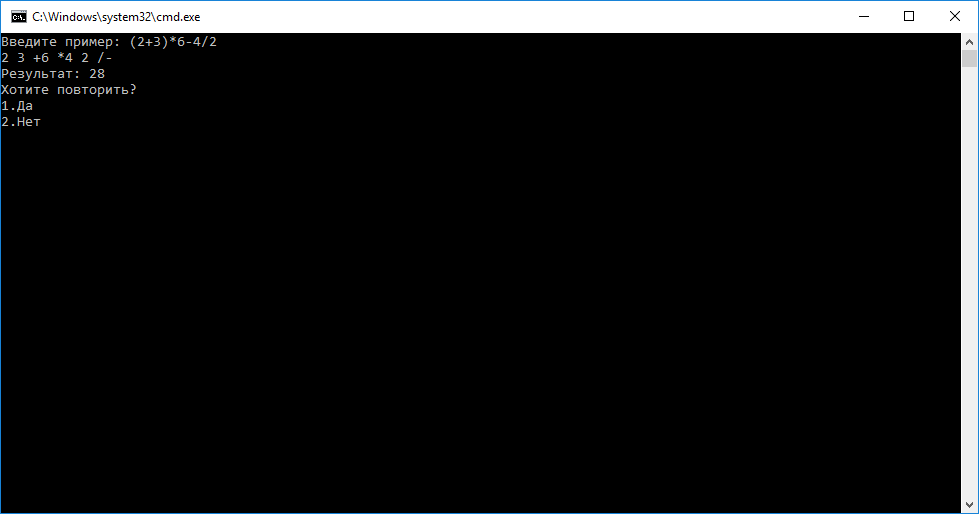
**Test\_calculator.cpp -** файл, в котором находится реализация фреймворка Google Test.

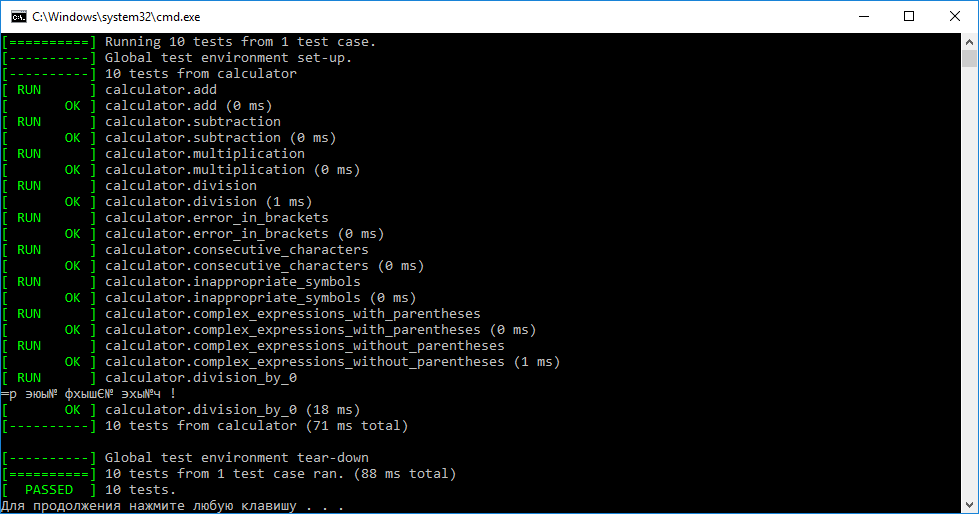
**Заключение**

В результате выполнения данной лабораторной работы было создано программное приложение, позволяющее переводить арифметические выражения из инфиксной формы записи в ОПН и производить различные вычислительные операции над ними.

Программа была протестирована на примере, некоторые операции были продемонстрированы выше. Ниже представлены результаты программы.







**Список используемой литературы:**

* Гергель В.П. «Рабочие материалы к учебному курсу «Методы программирования» ННГУ, 2002
* Б. Страуструп Язык программирования С++. Специальное издание. Пер. с англ. – М.: ООО «Бином-Пресс», 2008 г. – 1104 с.: ил.
* Сайт <https://habrahabr.ru>.
* Шилдт, Герберт Полный справочник по C++, 4-е издание. : Пер. с англ. : -M. : Издательский дом "Вильямс" , 2004.-800 с.

***Приложение:***

*TStack.cpp*

#include "TStack.h"

TStack::TStack(const TStack & v)

{

Size = v.Size;

pMem = new int[Size];

top = v.top;

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

{

pMem[i] = v.pMem[i];

}

}

TStack::TStack()

{

top = -1;

Size = MemSize;

pMem = new int[Size];

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

{

pMem[i] = 0;

}

}

TStack::~TStack(void)

{

delete[]pMem;

}

TStack::TStack(int Si)

{

top = -1;

Size = Si;

pMem = new int[Size];

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

{

pMem[i] = 0;

}

}

int TStack::IsEmpty(void) const

{

return top == -1;

}

int TStack::IsFull(void) const

{

return top == Size - 1;

}

void TStack::Put(int val)

{

pMem[++top] = val;

}

int TStack::Get(void)

{

return pMem[top--];

}

int TStack::Peek(void)

{

return pMem[top];

}

*Calculator.h*

#include"TStack.h"

#include<iostream>

#include<string.h>

#include<cstring>

#include<ctype.h>

#include<locale.h>

using namespace std;

const int Max\_Size\_Str = 50;

int priority(char str\_s);

void polish\_notation(char \*str, char \*result);

int foolproof(char \*str);

int priority(char str\_s)

{

if (str\_s == '(')

{

return 0;

}

if (str\_s == ')')

{

return 1;

}

if (str\_s == '+')

{

return 2;

}

if (str\_s == '-')

{

return 2;

}

if (str\_s == '\*')

{

return 3;

}

if (str\_s == '/')

{

return 3;

}

else

{

return -1;

}

}

void polish\_notation(char \* str, char \* result)

{

TStack Stack;

int str\_i = 0, i = 0;

while (str[i] != NULL)

{

while ((str[i] <= '9') && (str[i] >= '0'))

{

result[str\_i++] = str[i++];

if ((str[i] > '9') || (str[i] < '0'))

{

result[str\_i++] = ' ';

}

}

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

{

if ((Stack.IsEmpty()) || (str[i] == '(') || (priority(str[i]) > priority(Stack.Peek())))

{

Stack.Put(str[i]);

}

else

{

while ((!Stack.IsEmpty()) && (priority(str[i]) <= priority(Stack.Peek())))

{

result[str\_i++] = Stack.Get();

}

Stack.Put(str[i]);

}

i++;

}

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

{

while (Stack.Peek() != '(')

{

result[str\_i++] = Stack.Get();

}

Stack.Get();

i++;

}

}

while (!Stack.IsEmpty())

{

result[str\_i++] = Stack.Get();

}

}

int calculator(char \* str)

{

TStack Stack;

int result = 0, elem;

int a, i = 0;

while (str[i] != NULL)

{

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

{

elem = 0;

while (str[i] != ' ')

{

elem = elem \* 10 + (str[i++] - 48);

}

Stack.Put(elem);

i++;

}

if (str[i] == '\*')

{

a = Stack.Get();

result = Stack.Get() \* a;

Stack.Put(result);

i++;

}

if (str[i] == '/')

{

a = Stack.Get();

if (a != 0)

{

result = Stack.Get() / a;

Stack.Put(result);

i++;

}

else

{

cout << "На ноль делить нельзя!" << endl;

result = NULL;

return result;

}

}

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

{

a = Stack.Get();

result = Stack.Get() + a;

Stack.Put(result);

i++;

}

if (str[i] == '-')

{

a = Stack.Get();

if (Stack.IsEmpty())

{

Stack.Put(0);

}

result = Stack.Get() - a;

Stack.Put(result);

i++;

}

}

return result;

}

int foolproof(char \*str)

{

int ch = 0;

int i = 0;

int q = 0, p = 0;

TStack skob;

while (str[i] != NULL) //Неподходящие символы(ASCII)

{

if ((str[i]<40) || (str[i]>57) || (str[i] == 44) || (str[i] == 46) || (str[i] == 32) )

{

ch = 1;

}

i++;

}

i = 0;

if ((str[0] > 57) || (str[0] < 48) && (str[0] != 45) && (str[0] != 40))//Корректность 1-го символа

{

ch = 1;

}

if ((str[strlen(str)-1] > 57) || (str[strlen(str) - 1] < 48) && (str[strlen(str) - 1] != 41))//Корректность последнего символа

{

ch = 1;

}

while (str[i] != NULL) //Парность/непарность скобок

{

if (str[i] == 40)

{

q++;

}

if (str[i] == 41)

{

p++;

}

i++;

}

if (q != p)

{

ch = 1;

}

i = 0; q = 0; p = 0;

while (str[i] != NULL) //Порядок скобок

{

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

{

skob.Put(str[i]);

i++;

continue;

}

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

{

if (skob.IsEmpty())

{

ch = 1;

}

}

i++;

}

i = 0;

while (str[i] != NULL) //Повторение символов

{

if ((str[i] >= 42) && (str[i] <= 47) && (str[i + 1] >= 42) && (str[i + 1] <= 47))

{

ch = 1;

}

i++;

}

i = 0;

return ch;

}

*Main.cpp*

#include <calculator.h>

int main()

{

int a = 0;

do {

int result = 0;

char str\_1[Max\_Size\_Str];

char str[Max\_Size\_Str] = " ";

int ch;

int i = 0;

setlocale(LC\_ALL, "RUS");

system("cls");

cout << "Введите пример: ";

cin >> str\_1;

ch = foolproof(str\_1);

if (ch == 1)

{

cout << "Некоректное выражение" << endl;

goto C2;

}

polish\_notation(str\_1, str);

cout << str << endl;

result=calculator(str);

cout << "Результат: " << result << endl;

C2:

cout << "Хотите повторить?" << endl;

cout << "1.Да" << endl;

cout << "2.Нет" << endl;

cin >> a;

cout << endl;

} while (a != 2);

return 0;

}

*Test\_calculator.cpp*

#include <gtest.h>

#include "calculator.h"

TEST(calculator, add)

{

char s[] = "25+5";

char s2[10] = { 0 };

polish\_notation(s, s2);

int result = calculator(s2);

EXPECT\_EQ(30, result);

}

TEST(calculator, subtraction)

{

char s[] = "25-5";

char s2[10] = {0};

polish\_notation(s, s2);

int result=calculator(s2);

EXPECT\_EQ(20, result);

}

TEST(calculator, multiplication)

{

char s[] = "25\*5";

char s2[10] = { 0 };

polish\_notation(s, s2);

int result = calculator(s2);

EXPECT\_EQ(125, result);

}

TEST(calculator,division)

{

char s[] = "25/5";

char s2[10] = { 0 };

polish\_notation(s, s2);

int result = calculator(s2);

EXPECT\_EQ(5, result);

}

TEST(calculator, error\_in\_brackets)

{

char s[] = "2-)3+4(";

int result = foolproof(s);

EXPECT\_EQ(1, result);

}

TEST(calculator,consecutive\_characters)

{

char s[] = "2-+4";

int result = foolproof(s);

EXPECT\_EQ(1, result);

}

TEST(calculator, inappropriate\_symbols)

{

char s[] = "3+5%r\*6";

int result = foolproof(s);

EXPECT\_EQ(1, result);

}

TEST(calculator, complex\_expressions\_with\_parentheses)

{

char s[] = "10-5\*(4\*(4/1)/2-3)\*2";

char s2[100] = { 0 };

polish\_notation(s, s2);

int result = calculator(s2);

EXPECT\_EQ(-40, result);

}

TEST(calculator, complex\_expressions\_without\_parentheses)

{

char s[] = "10-5\*4\*4/1/2-3\*2";

char s2[100] = { 0 };

polish\_notation(s, s2);

int result = calculator(s2);

EXPECT\_EQ(-36, result);

}

TEST(calculator, division\_by\_0)

{

char s[] = "10/(5-5)";

char s2[100] = { 0 };

polish\_notation(s, s2);

int result = calculator(s2);

EXPECT\_EQ(NULL, result);

}