МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ
ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ

«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

Кафедра САПР

Отчет по Курсовой работе

по дисциплине

«Алгоритмы и структуры данных»

Тема: «Преобразование алгебраических формул из инфиксной в постфиксную форму записи и вычисление значения выражения»

Вариант 1

Студент 0301 _____ Сморжок В. Е.

Преподаватель _____ Тутуева А.В.

Санкт-Петербург,

2021

Постановка задачи

Необходимо реализовать простейшую версию калькулятора. Пользователю должен быть доступен ввод математического выражения, состоящего из чисел и арифметических знаков. Программа должна выполнить проверку корректности введенного выражения. В случае некорректного ввода необходимо вывести сообщение об ошибке с указанием позиции некорректного ввода. В противном выводится обратная польская нотация введенного выражения, а также отображается результат вычисления.

Входные данные:

● арифметическое выражение

● поддерживаемый тип данных: вещественные числа (double)

● поддерживаемые знаки: +, -, *, /, ^, унарный “-”, функции с одним аргументом (cos, sin, tg, ctg, ln, log, sqrt и др. (хотя бы одну не из списка)), константы pi, e открывающая и закрывающая скобки

Выходные данные:

● постфиксная ФЗ

● результат вычисления

Входные данные по желанию можно читать из файла.

Обоснование выбора используемых структур данных

1. Стек использовался для преобразования исходной строки в польскую нотацию. В стек подавались операции из строки, с помощью специальной функции определялся приоритет операции и в зависимости от этого из стека выводились операторы в постфиксную строку
2. Одномерные массивы использовались в программе для подсчета арифметического выражения. Один массив отвечал за числовые данные, второй за операторы. Таким образом, оперируя индексами можно было добиться корректного подсчета

Описание алгоритма решения

На вход подается строка, она проверяется с помощью функции correct на правильность ввода, если строка введена с ошибкой, выводится сообщение о типе ошибки и позиция некорректного ввода. Далее заполняется постфиксная строка с помощью функции filling, которая в свою очередь использует стек, в который записываются операции и с помощью функции priority определяется приоритет данной операции, в зависимости от чего в постфиксную строку выводятся операции в нужном порядке. Далее вводятся два динамических массива, которые впоследствии будут хранить в себе введенные числа и операции. Вызывается функция processing, в которой с помощью заполненных массивов подсчитывается результат операций, используя циклы и вызов функции clear, которая очищает пустые парные клетки двух массивов, и возвращается ответ. Функция notation преобразовывает постфиксную строку в «красивый» вид с добавлением пробелов и возвращением к понятным пользователю символов и выводит на экран обратную польскую нотацию с пробелами.

Текст программы

```
#include <iostream>
#include <string>
#include <cmath>
#include <math.h>
#define _USE_MATH_DEFINES
#define M_PI 3.14159265358979323846

using namespace std;



int size_stack;
string poststring; // postfix string
char* arrayoper;
double* arraynumber;

class Stack
{
private:
	class Node
	{
	public:
		char element;
		Node* previous;
	};
	Node* current;
	Node* head;

public:

	Stack()
	{
		Node* buffer = new Node;
		buffer->element = NULL;
		buffer->previous = NULL;
```

```
        current = head = buffer;
        size_stack = 0;
}
~Stack()
{
        Node* buffer;
        while (current)
        {
                buffer = current->previous;
                delete(current);
                current = buffer;
        }
}

void add(int element)
{
        if (size_stack == 0)
        {
                Node* buffer = new Node;
                buffer->element = element;
                buffer->previous = nullptr;
                head = current = buffer;
        }
        else
        {
                Node* buffer = current;
                current = new Node;
                current->element = element;
                current->previous = buffer;
        }
        size_stack++;
}
void parenthesis(char element, string poststroka)
{
        if (element == ')')
        {
```

```
                Node* buffer = current;
                while (buffer->element != '(')
                {
                        vitalkivanie(buffer, poststroka);
                        buffer = current;
                }
                remove();
        }
}
void vitalkivanie(Node* buffer, string poststring)
{
        poststring += buffer->element;
        poststring += ' ';
        buffer = buffer->previous;
        remove();
}

void remove()
{
        Node* buffer = current;
        current = current->previous;
        delete(buffer);
        size_stack--;
}
char getlast()
{
        if (size_stack != 0)
        return current->element;
}
int size()
{
        return size_stack;
}
bool IsEmpty()
{
        if (size() == 0)
```

```
		{
			return true;
		}
		else return false;
	}
};

int priority(int element) // prioritization of operations
{
	int temp = 0;
	switch (element)
	{
	case 40: // (
		temp = 0;
		break;
	case 43:// +
		temp = 1;
		break;
	case 45: // -
		temp = 1;
		break;
	case (42): // *
		temp = 2;
		break;
	case 47: // /
		temp = 2;
		break;
	case 94: // ^
		temp = 3;
		break;
	case 126: // ~
		temp = 4;
		break;
	case 33: // sin
		temp = 5;
		break;
```

```
	case 39: // cos
		temp = 5;
		break;
	case 36: // sqrt
		temp = 5;
		break;
	case 34: // ctg
		temp = 5;
		break;
	case 37: // log2
		temp = 5;
		break;
	case 60: // lg
		temp = 5;
		break;
	case 38: // ln
		temp = 5;
		break;
	case 58: // tg
		temp = 5; break;
	}
	return temp;
}

int clear(int maxindex) // clearing empty array elements
{
	bool answer = 1;
	int i = 0;
	int maxindex2 = maxindex;
	int maxindex1 = 0;
	while (answer == 1) // cleaning empty items
	{
		for (i = 0; i <= maxindex; i++)
		{
			if (arraynumber[i] == 0 && arrayoper[i] == ' ')
			{
```

```
                if (maxindex = maxindex2)
                {
                        arrayoper[maxindex + 1] = ' ';
                        arraynumber[maxindex + 1] = 0;
                }
                for (int j = i; j <= maxindex; j++)
                {
                        arraynumber[j] = arraynumber[j + 1];
                        arrayoper[j] = arrayoper[j + 1];
                }
                arrayoper[maxindex + 1] = ' ';
                arraynumber[maxindex + 1] = 0;
        }
}
answer = 0;
for (i = 0; i <= maxindex; i++)
{
        if (arraynumber[i] == 0 && arrayoper[i] == ' ')
        {
                answer = 1; break;
        }
}
maxindex1 = maxindex;
if (answer != 0)
{
        for (i = 0; i <= maxindex; i++)
        {
                if (arraynumber[i] != 0)
                {
                        maxindex1 = i;
                }
                else
                {
                        if (arrayoper[i] != ' ')
                        {
                                maxindex1 = i;
```

```
                                }
                        }
                }
                for (i = maxindex1 + 1; i < maxindex; i++)
                {
                        arraynumber[i] = 0;
                        arrayoper[i] = ' ';
                }
                maxindex = maxindex1;
        }
    }
    i = 0;
    return maxindex;
}
int forbinaroption(int i, int maxindex) // clearing empty array windows for binary operations
{
    for (int j = i - 1; j < maxindex; j++)
    {
        arraynumber[j] = arraynumber[j + 1];
    }
    for (int j = i; j <= maxindex + 1; j++)
    {
        arrayoper[j] = arrayoper[j + 1];
    }
    arrayoper[maxindex] = ' ';
    arraynumber[maxindex] = 0;
    maxindex--;
    return maxindex;

}

double processing(string poststring) // calculating the whole example
{
    int i;
    int base = 0;
    double result;
```

```
	double first;
	double second;
	string number;
	int n = poststring.size();

	bool answer = 0;
	for (i = 0; i < n; i++) // filling arrays with empty characters
	{
		arraynumber[i] = 0;
		arrayoper[i] = ' ';
	}
	for (i = 0; i < poststring.size(); i++) //filling arrays of operators and numbers
	{
		if (!(poststring[i] >= 48 && poststring[i] <= 57 || poststring[i] == 46))
// if the operator
		{
			arrayoper[i] = poststring[i];
		}
		if (poststring.substr(i, 4) == "2.71" || poststring.substr(i, 4) == "3.14")
		{
			if (poststring[i] == '2')
			{
				arraynumber[i] = exp(1.0);
				i += 4;
			}
			else
			{
				arraynumber[i] = M_PI;
				i += 4;
			}
		}
		else
		{
			if (poststring[i] >= 48 && poststring[i] <= 57 || poststring[i] ==
46) // if number or dot
			{
				while (poststring[i] >= 48 && poststring[i] <= 57 ||
poststring[i] == 46)
```

```
                        {
                                number += poststring[i]; i++;
                        }
                        result = atof(number.c_str());
                        arraynumber[i] = result;
                        number.clear();
                }
        }
}
for (i = 0; i < n; i++)
{
        if (arraynumber[i] == 0 && arrayoper[i] == ' ')
        {
                answer = 1; break;
        }
}
i = 0;
int maxindex = n-1;
maxindex = clear(maxindex);

answer = 1;
while (answer == 1)
{
        for (i = 0; i <= maxindex; i++)
        {
                if (arrayoper[i] != ' ')
                {
                        break;
                }
        }
        first = arraynumber[i - 2];
        switch (arrayoper[i])
        {
        case 43: // +
                second = arraynumber[i - 1];
                result = first + second;
```

```
        arraynumber[i - 2] = result;
        maxindex = forbinaroption(i, maxindex);
        break;
case 45: // -
        first = arraynumber[i - 2];
        second = arraynumber[i - 1];
        result = first - second;
        arraynumber[i - 2] = result;
        maxindex = forbinaroption(i, maxindex);
        break;
case 42: // *
        first = arraynumber[i - 2];
        second = arraynumber[i - 1];
        result = first * second;
        arraynumber[i - 2] = result;
        maxindex = forbinaroption(i, maxindex);
        break;
case 47: // /
        first = arraynumber[i - 2];
        second = arraynumber[i - 1];
        result = first / second;
        arraynumber[i - 2] = result;
        maxindex = forbinaroption(i, maxindex);
        break;
case 94: // ^
        first = arraynumber[i - 2];
        second = arraynumber[i - 1];
        result = pow(first, second);
        arraynumber[i - 2] = result;
        maxindex = forbinaroption(i, maxindex);
        break;
case 126: //unary minus
        first = arraynumber[i - 1];
        result = -first;
        arraynumber[i - 1] = result;
        arrayoper[i] = ' ';
```

```
        maxindex = clear(maxindex);
        break;
case 36:// sqrt
        first = arraynumber[i - 1];
        result = sqrt(first);
        arraynumber[i - 1] = result;
        arrayoper[i] = ' ';
        maxindex = clear(maxindex);
        break;
case 38: // ln
        first = arraynumber[i - 1];
        result = log(first);
        arraynumber[i - 1] = result;
        arrayoper[i] = ' ';
        maxindex = clear(maxindex);
        break;
case 60: // lg
        first = arraynumber[i - 1];
        result = log10(first);
        arraynumber[i - 1] = result;
        arrayoper[i] = ' ';
        maxindex = clear(maxindex);
        break;
case 37:// log2
        first = arraynumber[i - 1];
        base = 2;
        result = log(first) / log(base);
        arraynumber[i - 1] = result;
        arrayoper[i] = ' ';
        maxindex = clear(maxindex);
        break;
case 33: // sin
        first = arraynumber[i - 1];
        result = sin(first);
        arraynumber[i - 1] = result;
        arrayoper[i] = ' ';
```

```
                maxindex = clear(maxindex);
                break;
        case 39: // cos
                first = arraynumber[i - 1];
                result = cos(first);
                arraynumber[i - 1] = result;
                arrayoper[i] = ' ';
                maxindex = clear(maxindex);
                break;
        case 58: // tg
                first = arraynumber[i - 1];
                result = tan(first);
                arraynumber[i - 1] = result;
                arrayoper[i] = ' ';
                maxindex = clear(maxindex);
                break;
        case 34: // ctg
                first = arraynumber[i - 1];
                result = 1/tan(first);
                arraynumber[i - 1] = result;
                arrayoper[i] = ' ';
                maxindex = clear(maxindex);
                break;
        }

        clear(maxindex);
        answer = 0;
        for (i = 0; i <= maxindex; i++)
        {
                if (arrayoper[i] != ' ')
                {
                        answer = 1;
                        break;
                }
        }
}
```

```
	return arraynumber[0];


}
string notation(string poststring)
{
	string correctpoststring;
	for (int i = 0; i < poststring.size(); i++)
	{
		if (poststring.substr(i, 4) == "2.71" || poststring.substr(i, 4) == "3.14")
		{
			if (poststring[i] == '2')
			{
				correctpoststring += 'e';
				poststring.erase(i, 4);
				correctpoststring += ' ';
			}
			else
			{
				correctpoststring += "pi";
				poststring.erase(i, 4);
				correctpoststring += ' ';
			}
		}
		else {
			if (poststring[i] >= 48 && poststring[i] <= 57 || poststring[i] ==
46) //if the number
			{
				correctpoststring += poststring[i];
				if (poststring[i + 1] != 46 && poststring[i + 1] < 48 ||
poststring[i + 1]>57) // if the next is not a number
					correctpoststring += ' ';
			}
		}
		if (poststring[i] >= 42 && poststring[i] <= 47 && poststring[i] != 46 ||
poststring[i] == 94 || poststring[i] == 126) // if the operator
		{
			correctpoststring += poststring[i];
			correctpoststring += ' ';
```

```
        }

        // converting notation to readable form
        switch (poststring[i])
        {
        case '!':
               correctpoststring += "sin";
               correctpoststring += ' '; break;
        case '$':
               correctpoststring += "sqrt";
               correctpoststring += ' '; break;
        case 39:
               correctpoststring += "cos";
               correctpoststring += ' '; break;
        case'"':
               correctpoststring += "ctg";
               correctpoststring += ' '; break;
        case'%':
               correctpoststring += "log";
               correctpoststring += ' '; break;
        case '<':
               correctpoststring += "lg";
               correctpoststring += ' '; break;
        case'&':
               correctpoststring += "ln";
               correctpoststring += ' '; break;
        case':':
               correctpoststring += "tg";
               correctpoststring += ' '; break;
        default:
               break;
        }
}
poststring.clear();
poststring = correctpoststring;
return poststring;
```

```
}
int correct(string String,int answer)
{
	int open = 0;
	int close = 0;
	int position = 0;
	for (int i = 0; i < String.size(); i++)
	{
		if (String[i] == '(')
		{
			open++;
			position = i;
		}
		if (String[i] == ')')
		{
			if (i > 0 && (String[i - 1] = '(' || String[i] >= 42 && String[i] <=
47 && String[i] != 46 || String[i] == 94 || String[i] == 126)) // if previous is operator
or (
			{
				if (answer != 0)
				{
					position = i;
					return position;
				}
				throw invalid_argument("Incorrect contents of parentheses");

			}
			if (i == 0 && String[i] == ')')
			{
				if (answer != 0)
				{
					position = i;
					return position;
				}
				throw invalid_argument("Incorrect contents of parentheses");
			}
```

```
        close++;
        position = i;
    }
    switch (String[i])
    {
    case 'c':
        if (String[i + 3] == '0')
        {
            if (answer != 0)
            {
                position = i + 3;
                return position;
            }
            throw invalid_argument("Incorrect argument for ctg");
        }
        break;
    case 's':
        if (String[i + 4] == '~')
        {
            if (answer != 0)
            {
                position = i + 4;
                return position;
            }
            throw invalid_argument("Incorrect argument for sqrt");
        }break;
    case 'l':
        if (String[i + 1] == 'o')
        {
            if (String[i + 3] == '~')
            {
                if (answer != 0)
                {
                    position = i + 1;
                    return position;
                }
```

```
                        throw invalid_argument("Incorrect argument for log");
                    }
                }
                else
                {
                    if (String[i + 1] == 'n')
                    {
                        if (String[i + 2] == '~')
                        {
                            if (answer != 0)
                            {
                                position = i + 2;
                                return position;
                            }
                            throw invalid_argument("Incorrect argument for
ln");
                        }
                    }
                    if (String[i + 1] == 'g')
                    {
                        if (String[i + 2] == '~')
                        {
                            if (answer != 0)
                            {
                                position = i + 2;
                                return position;
                            }
                            throw invalid_argument("Incorrect argument for
lg");
                        }
                    }
                }
                break;
            case 't':
                if (String[i + 2] == 'p' && String[i + 5] == '2' && String[i + 4] ==
'/')
                {
                    if (answer != 0)
```

```
                        {

                        position = i + 2;
                        return position;
                        }
                        throw invalid_argument("Incorrect argument for tg");

                }
                break;

        }
    }
    if (open != close)
    {
        if (answer != 0)
        {
            return position;
        }
        throw invalid_argument("Invalid number of parenthesis");
    }

}

string filling(string String,string poststring)
{
    char symb;
    Stack stack;
    for (int i = 0; i < String.size(); i++)
    {
        bool answer = false;
        answer = false;
        if (String[i] >= 48 && String[i] <= 57 || String[i] == 46) // if the number
        {
            poststring += String[i];
            if (String[i + 1] != 46 && String[i + 1] < 48 || String[i + 1]>57)
                poststring += ' ';
```

```
            }
            if (String[i] >= 42 && String[i] <= 47 && String[i] != 46 || String[i] ==
94 || String[i] == 126) //if the operator
            {
                  answer = true;
                  symb = String[i];
            }
            switch (String[i])
            {
            case 'p': // if the number pi
                  poststring += "3.14";
                  poststring += ' ';
                  i += 1;
                  break;
            case 'e': // if the number e
                  poststring += "2.71";
                  poststring += ' '; break;
            case '(': // if the opening parenthesis
                  stack.add(String[i]); break;
            case ')': // if the closing parenthesis
                  stack.parenthesis(String[i], poststring); break;
            case 's':
                  if (String[i + 1] == 'i') // if sine
                  {
                        String[i] = 33;
                        symb = String[i];
                        answer = true;
                        i += 2;
                  }
                  else
                  { // if sqrt
                        String[i] = 36;
                        symb = String[i];
                        answer = true;
                        i += 3;
                  }
                  break;
```

```
case 'c':
	if (String[i + 1] == 'o') // if cosine
	{
		String[i] = 39;
		symb = String[i];
		answer = true;
		i += 2;
	}
	else
	{ // если котангенс
		String[i] = 34;
		symb = String[i];
		answer = true;
		i += 2;
	}
	break;
case 'l':
	if (String[i + 1] == 'o') // if logarithm base 2
	{
		String[i] = 37;
		symb = String[i];
		answer = true;
		i += 2;
	}
	else
	{
		if (String[i + 1] == 'n') // if natural logarithm
		{
			String[i] = 38;
			symb = String[i];
			answer = true;
			i += 1;
		}
		else
		{ // if logarithm to base 10
			String[i] = 60;
```

```
                                symb = String[i];
                                answer = true;
                                i += 1;
                        }
                }
                break;
        case 't': // if tangent
                String[i] = 58;
                symb = String[i];
                answer = true;
                i += 1;
                break;
        default:
                break;
        }

        if (answer == true)
        {
                if (stack.IsEmpty() == true)
                {
                        stack.add(symb);
                }
                else {
                        if ((priority(stack.getlast()) >= priority(symb))) // if the
priority is lower or the same than the highest on the stack
                        {
                                while (priority(stack.getlast()) >= priority(symb) &&
stack.IsEmpty() == false)
                                {
                                        poststring += ' ';
                                        poststring += stack.getlast();
                                        stack.remove();
                                }
                                stack.add(symb);
                        }
                        else
                        {
```

```
                                stack.add(symb);
                        }
                }
        }
        while (stack.IsEmpty() == false)// until empty, return everything from the stack
        {
                poststring += stack.getlast();
                stack.remove();
        }
        return poststring;
}

int main()
{
        cout << "available operations : \n" << "+, -, / , *, sin, cos, tg, ctg,
sqrt."<<endl;
        cout << "ln - natural logarithm\n" << "lg - logarithm to base 10\n" << "log -
logarithm to base 2\n" << "~- unary minus\n";
        cout << "Available constants : pi, e\n" << "(, ) - admissible\n" << "enter
expression without spaces\n"<< endl;
        cout << "Enter task" << endl;
        string String;

        bool correctanswer = false;
        int position = 0;
        cin >> String;
        while (correctanswer == false)
        {
                correctanswer = true;
                position = 0;
                try
                {
                        correct(String, position);
                }
                catch (const invalid_argument error)
                {
                        cout << error.what() << endl;
```

```
                        position++;
                        cout << "Mistake in position " << correct(String, position) << endl;
                        correctanswer = false;
                        cout << "Enter new task" << endl;
                        String.erase();
                        cin >> String;
                }
        }
        if (correctanswer == true)
        {
                poststring = filling(String, poststring);
                arrayoper = (char*)malloc(poststring.size() * sizeof(char));
                arraynumber = (double*)malloc(poststring.size() * sizeof(double));
                double result = processing(poststring);
                cout << "Solution = " << result << endl;
                poststring = notation(poststring);
                cout << "Postfics notation: " << poststring;
        }


        return 0;
}
```

Текст Unit тестов

```
#include "pch.h"
#include "CppUnitTest.h"
#include "..\kursovaya.h"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace UnitTestKursovaya
{
        TEST_CLASS(UnitTestKursovaya)
        {
        public:

                TEST_METHOD(TestCorrect)
                {
                        string String = "()";
                        int position = 0;
                        try
                        {
                                correct(String, position);
                        }
                        catch (const invalid_argument error)
                        {
                                cout << error.what() << endl;
                                position++;
                        }
```

```
		Assert::IsTrue(position != 0);
		String.erase();
		String = "(2+6*)";
		position = 0;
		try
		{
			correct(String, position);
		}
		catch (const invalid_argument error)
		{
			cout << error.what() << endl;
			position++;
		}
		String.erase();
		Assert::IsTrue(position != 0);
		String = "ln~e";
		position = 0;
		try
		{
			correct(String, position);
		}
		catch (const invalid_argument error)
		{
			cout << error.what() << endl;
			position++;
		}
		Assert::IsTrue(position != 0);
}

TEST_METHOD(TestFilling)
{
		string stroka = "1+2*3-4^2";
		arrayoper = (char*)malloc(poststring.size() * sizeof(char));
		arraynumber = (double*)malloc(poststring.size() * sizeof(double));
		poststring = filling(stroka, poststring);
		Assert::IsTrue(poststring == "1 2 3  * +4 2 ^-");
}
TEST_METHOD(TestProsessing)
{
		string stroka = "1+2*3-4^2";
		arrayoper = (char*)malloc(poststring.size() * sizeof(char));
		arraynumber = (double*)malloc(poststring.size() * sizeof(double));
		poststring = filling(stroka, poststring);
		double result = processing(poststring);
		Assert::IsTrue(result == -9);
		poststring.clear();
		stroka.clear();
		stroka = "cos1-pi+lne-ctg3+20^lg10";
		delete(arrayoper);
		delete(arraynumber);
		arrayoper = (char*)malloc(poststring.size() * sizeof(char));
		arraynumber = (double*)malloc(poststring.size() * sizeof(double));
		poststring = filling(stroka, poststring);
		result = processing(poststring);
		Assert::IsTrue(result == 25.413962203712881);
		poststring.clear();
		stroka.clear();
		stroka = "sin1+~log8-tgpi+sqrt81";
		delete(arrayoper);
		delete(arraynumber);
		arrayoper = (char*)malloc(poststring.size() * sizeof(char));
		arraynumber = (double*)malloc(poststring.size() * sizeof(double));
		poststring = filling(stroka, poststring);
		result = processing(poststring);
		Assert::IsTrue(result == 6.8414709848078967);
```

```
		}
		TEST_METHOD(TestNotation)
		{
			string stroka = "sin1+~log8-tgpi+sqrt81";
			arrayoper = (char*)malloc(poststring.size() * sizeof(char));
			arraynumber = (double*)malloc(poststring.size() * sizeof(double));
			poststring = filling(stroka, poststring);
			poststring = notation(poststring);
			Assert::IsTrue(poststring == "1 sin 8 log ~ + pi tg - 81 sqrt + ");
			poststring.clear();
			stroka.clear();
			delete(arrayoper);
			delete(arraynumber);
			arrayoper = (char*)malloc(poststring.size() * sizeof(char));
			arraynumber = (double*)malloc(poststring.size() * sizeof(double));
			stroka = "cos1-pi+lne-ctg3+20^lg10";
			poststring = filling(stroka, poststring);
			poststring = notation(poststring);
			Assert::IsTrue(poststring == "1 cos pi - e ln + 3 ctg - 20 10 lg ^ +
");
			delete(arrayoper);
			delete(arraynumber);
			arrayoper = (char*)malloc(poststring.size() * sizeof(char));
			arraynumber = (double*)malloc(poststring.size() * sizeof(double));
			poststring.clear();
			stroka.clear();
			stroka = "1+2*3-4^2";
			poststring = filling(stroka, poststring);
			poststring = notation(poststring);
			Assert::IsTrue(poststring == "1 2 3 * + 4 2 ^ - ");
		}
	};
}
```

Описание Unit тестов

1.TestZapolnenie – в данном тесте проверялась корректность работы стека и определения приоритета функциями

2. TestProsessing – в данном тесте проверялась корректность подсчета заданного выражения

3. TestNotation – в данном тесте проверялось корректность вывода обратной польской нотации

4. TestCorrect — в данном тесте проверялась корректность вводимой строки. Иначе выводилась позиция ошибки

Пример работы

```
available operations :
+, -, / , *, sin, cos, tg, ctg, sqrt.
ln - natural logarithm
lg - logarithm to base 10
log - logarithm to base 2
~- unary minus
Available constants : pi, e
(, ) - admissible
enter expression without spaces

Enter task
2+sinpi+tg4-ctg8+sqrt81/9+12^2+~5
Solution = 143.305
Postfics notation: 2 pi sin + 4 tg + 8 ctg - 81 sqrt 9 / + 12 2 ^ + 5 ~ +
```

```
available operations :
+, -, / , *, sin, cos, tg, ctg, sqrt.
ln - natural logarithm
lg - logarithm to base 10
log - logarithm to base 2
~- unary minus
Available constants : pi, e
(, ) - admissible
enter expression without spaces

Enter task
128-lne+log1024+pi-12*5
Solution = 80.1416
Postfics notation: 128 e ln - 1024 log + pi + 12 5 * -
```

```
available operations :
+, -, / , *, sin, cos, tg, ctg, sqrt.
ln - natural logarithm
lg - logarithm to base 10
log - logarithm to base 2
~- unary minus
Available constants : pi, e
(, ) - admissible
enter expression without spaces

Enter task
1+2*(5/2
Invalid number of parenthesis
Mistake in position 4
```

```
available operations :
+, -, / , *, sin, cos, tg, ctg, sqrt.
ln - natural logarithm
lg - logarithm to base 10
log - logarithm to base 2
~- unary minus
Available constants : pi, e
(, ) - admissible
enter expression without spaces

Enter task
cospi+32^4-48/2+tgpi/2
Incorrect argument for tg
Mistake in position 18
```

```
available operations :
+, -, / , *, sin, cos, tg, ctg, sqrt.
ln - natural logarithm
lg - logarithm to base 10
log - logarithm to base 2
~- unary minus
Available constants : pi, e
(, ) - admissible
enter expression without spaces

Enter task
sin45+cos45-tg60*ctg60+lne-sqrt9/lg32+e^5
Solution = 147.796
Postfics notation: 45 sin 45 cos + 60 tg 60 ctg * - e ln + 9 sqrt 32 lg / - e 5 ^ +
```

```
available operations :
+, -, / , *, sin, cos, tg, ctg, sqrt.
ln - natural logarithm
lg - logarithm to base 10
log - logarithm to base 2
~- unary minus
Available constants : pi, e
(, ) - admissible
enter expression without spaces

Enter task
()
Incorrect contents of parentheses
Mistake in position 1
```

```
TEST_METHOD(TestCorrect)
{
    string String = "()";
    int position = 0;
    try
    {
        correct(String, position);
    }
    catch (const invalid_argument error)
    {
        cout << error.what() << endl;
        position++;
    }
    Assert::IsTrue(position != 0);
    String.erase();
    String = "(2+6*)";
    position = 0;
    try
    {
        correct(String, position);
    }
    catch (const invalid_argument error)
    {
        cout << error.what() << endl;
        position++;
    }
    String.erase();
    Assert::IsTrue(position != 0);
    String = "ln~e";
    position = 0;
    try
    {
        correct(String, position);
    }
    catch (const invalid_argument error)
    {
        cout << error.what() << endl;
        position++;
    }
    Assert::IsTrue(position != 0);
}
```

```
TEST_METHOD(TestFilling)
{
    string stroka = "1+2*3-4^2";
    arrayoper = (char*)malloc(poststroka.size() * sizeof(char));
    arraynumber = (double*)malloc(poststroka.size() * sizeof(double));
    poststroka = filling(stroka, poststroka);
    Assert::IsTrue(poststroka == "1 2 3  * +4 2 ^-");
}
TEST_METHOD(TestProsessing)
{
    string stroka = "1+2*3-4^2";
    arrayoper = (char*)malloc(poststroka.size() * sizeof(char));
    arraynumber = (double*)malloc(poststroka.size() * sizeof(double));
    poststroka = filling(stroka, poststroka);
    double result = processing(poststroka);
    Assert::IsTrue(result == -9);
    poststroka.clear();
    stroka.clear();
    stroka = "cos1-pi+lne-ctg3+20^lg10";
    delete(arrayoper);
    delete(arraynumber);
    arrayoper = (char*)malloc(poststroka.size() * sizeof(char));
    arraynumber = (double*)malloc(poststroka.size() * sizeof(double));
    poststroka = filling(stroka, poststroka);
    result = processing(poststroka);
    Assert::IsTrue(result == 25.413962203712881);
    poststroka.clear();
    stroka.clear();
    stroka = "sin1+~log8-tgpi+sqrt81";
    delete(arrayoper);
    delete(arraynumber);
    arrayoper = (char*)malloc(poststroka.size() * sizeof(char));
    arraynumber = (double*)malloc(poststroka.size() * sizeof(double));
    poststroka = filling(stroka, poststroka);
    result = processing(poststroka);
    Assert::IsTrue(result == 6.8414709848078967);

TEST_METHOD(TestNotation)
{
    string stroka = "sin1+~log8-tgpi+sqrt81";
    arrayoper = (char*)malloc(poststroka.size() * sizeof(char));
    arraynumber = (double*)malloc(poststroka.size() * sizeof(double));
    poststroka = filling(stroka, poststroka);
    poststroka = notation(poststroka);
    Assert::IsTrue(poststroka == "1 sin 8 log ~ + pi tg - 81 sqrt + ");
    poststroka.clear();
    stroka.clear();
    delete(arrayoper);
    delete(arraynumber);
    arrayoper = (char*)malloc(poststroka.size() * sizeof(char));
    arraynumber = (double*)malloc(poststroka.size() * sizeof(double));
    stroka = "cos1-pi+lne-ctg3+20^lg10";
    poststroka = filling(stroka, poststroka);
    poststroka = notation(poststroka);
    Assert::IsTrue(poststroka == "1 cos pi - e ln + 3 ctg - 20 10 lg ^ + ");
    delete(arrayoper);
    delete(arraynumber);
    arrayoper = (char*)malloc(poststroka.size() * sizeof(char));
    arraynumber = (double*)malloc(poststroka.size() * sizeof(double));
    poststroka.clear();
    stroka.clear();
    stroka = "1+2*3-4^2";
    poststroka = filling(stroka, poststroka);
    poststroka = notation(poststroka);
    Assert::IsTrue(poststroka == "1 2 3 * + 4 2 ^ - ");
}
```