МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
**«Национальный исследовательский
Нижегородский государственный университет им. Н.И. Лобачевского»
(ННГУ)**

Институт информационных технологий, математики и механики

Направление подготовки: «Программная инженерия»

ОТЧЕТ

по лабораторной работе №3

Вычисление арифметических выражений (стеки)

Выполнил: студент группы
3822Б1ПР2

_____В. Е. Филатьев
Подпись

Нижний Новгород
2023

Содержание

1.Введение:

Арифметическое выражение - выражение, в котором операндами являются объекты, над которыми выполняются арифметические операции. Например, (1+2)/(3+4*6.7)-5.3*4.4

При такой форме записи (называемой инфиксной, где знаки операций стоят между операндами) порядок действий определяется расстановкой скобок и приоритетом операций. Постфиксная (или обратная польская) форма записи не содержит скобок, а знаки операций следуют после соответствующих операндов. Тогда для приведённого примера постфиксная форма будет иметь вид: 1 2+ 3 4 6.7*+/ 5.3 4.4* -

Обратная польская нотация была разработана австралийским ученым Чарльзом Хэмблином в середине 50-х годов прошлого столетия на основе польской нотации, которая была предложена в 1920 году польским математиком Яном Лукасевичем. Эта нотация лежит в основе организации вычислений для арифметических выражений. Известный ученый Эдсгер Дейкстра предложил алгоритм для перевода выражений из инфиксной в постфиксную форму. Данный алгоритм основан на использовании стека.

Стек (англ. stack), магазин – схема запоминания информации, при которой каждый вновь поступающий ее элемент как бы «проталкивает» вглубь отведенного участка памяти находящиеся там элементы (подобно патрону, помещаемому в магазин винтовки) и занимает крайнее положение (так называемую вершину стека). При выдаче информации из стека выдается элемент, расположенный в вершине стека, а оставшиеся элементы продвигаются к вершине; следовательно, элемент, поступивший последним, выдается первым

2. Постановка задачи:

В рамках лабораторной работы ставится задача создания программных средств, поддерживающих обработку аритмических выражений в инфиксной форме в постфиксную и последующего вычисления ответа. Так же создания визуального интерфейса для визуализации работы программы:

Класс Арифметическое Выражение должен предоставлять, как минимум, следующие операции

- Получение, хранение и возврат исходной (инфиксной) формы
- Преобразование в постфиксную форму
- Возврат постфиксной формы
- Вычисление арифметического выражения при заданных значениях операндов

3. Руководство пользователя

3.1. Консольное приложение

При начале работы приложения, пользовать заходит в главное меню (*Рис 1.*) и может задать арифметическое выражение во вкладке Set formula (*Рис 2.*) или получить постфиксную запись данного выражения (*Рис 3.*).

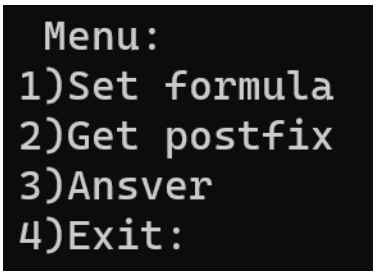

Рисунок 1

Рисунок 2

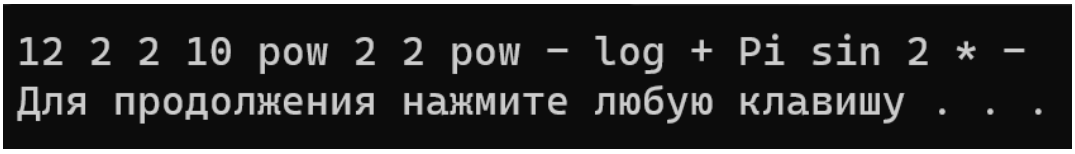

Рисунок 3

Так же из главного меню пользователь может получить ответ, если в выражение были переменные, то пользователя программа попросит вести их значение (*Рис 4.*), после чего пользователь получит ответ (*Рис 5.*).

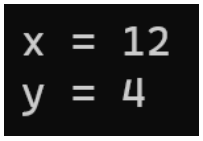

Рисунок 5

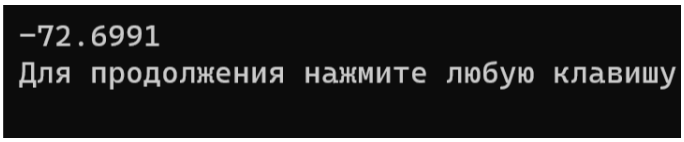

Рисунок 4

3.2. Визуальное приложение:

В визуальном приложении для задания формулы используются кнопки на форме (Рис 6.), а для получения ответа нужно нажать на клавишу равно, если в формуле были допущены ошибки, то калькулятор выведет ошибку (Рис 7.) если в формуле не было ошибок, то он выведет ответ на выражение (Рис.8).

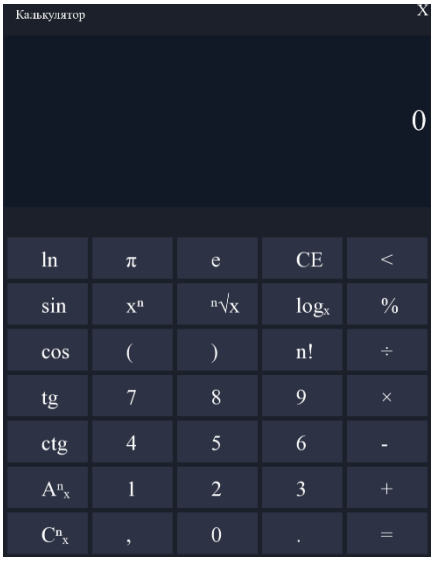

Рисунок 6

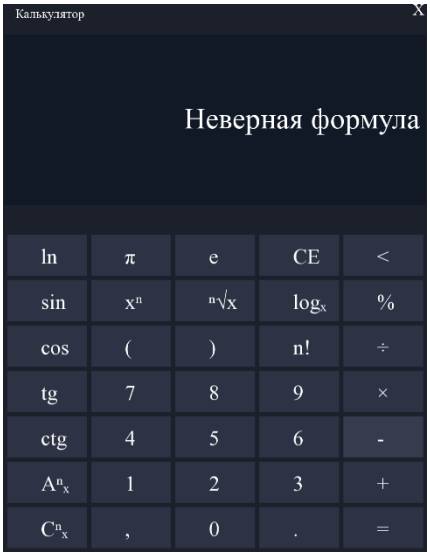

Рисунок 7

Рисунок 8

3. Руководство программиста:

3.1 Описание алгоритмов:

Алгоритма преобразования инфиксной строки в обратную польскую запись:

Рассматриваем поочередно каждый символ:

1. Если этот символ - число (или переменная), то просто помещаем его в выходную строку.
2. Если символ - знак операции (+, -, *, /), то проверяем приоритет данной операции:
    1. Умножение и деление имеют наивысший приоритет (например, он равен 3)
    2. Сложение и вычитание имеют меньший приоритет (например, 2)
    3. Наименьший приоритет у открывающейся скобки (1)

Получив один из этих символов мы должны проверить стек:

a. Если стек все еще пуст или находящиеся в нем символы (а находиться в нем могут только знаки операций и открывающая скобка) имеют меньший приоритет, чем приоритет текущего символа, то символ помещаем в стек
b. Если символ, находящийся на вершине стека имеет приоритет больший или равный приоритету текущего символа, то извлекаем символы из стека в выходную строку до тех пор, пока выполняется это условие, затем переходим к пункту А

3. Если текущий символ - открывающая скобка, то помещаем ее в стек
4. Если текущий символ - закрывающая скобка, то извлекаем символы из стека в выходную строку, пока не встретим открывающую скобку (т.е. символ с приоритетом, равным 1), которую следует просто уничтожить. Закрывающая скобка также уничтожается

Если вся входная строка разобрана, а в стеке еще остаются знаки операций, то извлекаем их из стека в выходную строку.

Алгоритм вычисления по постфиксной форме:
Для каждой лексемы в постфиксной форме
– Если лексема –операнд, поместить ее значение в стек
– Если лексема –операция
• Извлечь из стека значения двух операндов
• Выполнить операцию (верхний элемент из стека является правым операндом, следующий за ним – левым
• Положить результат операции в стек
По исчерпании лексем из постфиксной формы на вершине стека будет
результат вычисления выражения

3.2.Описание программы:

Программа состоит из нескольких основных классов:

Класс стек:

Класс стек – шаблонный класс с изменяемым типом хранения содержащий:

Внутренний класс:

- class Note; - Данный класс хранит значение ячейки (T value;) и указатель на следующую ячейку (Note* next;). В классе перегружен конструктор инициализации.

Поля:

- Note<T>* head; - указатель на первый элемент стека
- size_t sz; - размер стека

Методы:

- MyStack() noexcept; - конструктор по умолчанию
- bool empty() noexcept; - метод проверки на пустоту стак
- void push(T value); - добавление элемента в начала стека
- void pop(); - удаление элемента из начала стека
- void clear(); - очистка стека
- size_t size() noexcept – метод возвращает размер стека
- T& top(); - метод возвращает верхний элемент стека
- ~MyStack(); - деструктор

Класс матрица:

Поля:

- string formula; - строка содержащая изначальную формулу
- string postfix; - строка содержащая постфиксную запись формулы
- map<string, double> var; - словарь содержащий переменную и ее значение

Методы:

- int priority(char c); - метод возвращает порядок данного оператора
- calculator(); - конструктор по умолчанию
- calculator(string formula); - конструктор инициализации
- string GetFormula(); - возвращает инфиксную запись формулы
- string GetPostfix(); - возвращает постфиксную запись
- void SetFormula(string formula); - метод задает инфиксную запись
- void FormulaConverrt(); - преобразует инфиксную формулу в постфиксную
- double Ansver(); - возвращает ответ на данное арифметическое выражение
- map<string, double> GetVar() { return var; } – возвращает список переменных
- void SetVar(map<string,double> var); - метод задает список переменных

Функции:

- string RemoveSpace(string s); - функция удаляет все пробелы из строки
- string Remove0(string s); - класс удаляет все незначимые числа при переводи числа из double в string
- bool CheckFormula(string s); функция проверяет корректность веденной формулы
- double ToDouble(string s, int& i); - приводит число из строки к double
- string ToStringVar(string s, int& i); - приводит строку к имени переменной
- bool CheckNumber(char c); - проверяет символ на число
- bool CheckChar(char c); - проверяет символ на букву
- bool CheckOperator(char c); - проверяет символ на оператор
- double StandartOperator(double a, double b, char c); - функция вызова стандартых операторов
- int CheckConst(string s); - проверка на константу π и e
- int Func(string s); - проверка на созданные функции
- bool CheckFunc(string s, string fun, int &i); - метод проверяет верность введенной функции
- double DuFunc(string fun, double a); - вычисляет значения функции с параметрами
- double DuFunc(string fun, double a, double b); - вычисляет значения функции с параметрами

- bool Eror(char c, MyStack<string>& stack); - функция проверяет на нахождение ошибки в формуле

Класс MainForm:

Класс основного окна, выполняет бесконечный цикл проверяет события и отображает объекты на окне

Поля:

- RenderWindow* window; - окно приложения
- button* exit; - кнопка выхода из приложения
- Text name; - текстовое поле для названия приложения
- Font font; - поля для хранения шрифта
- PanelFormul* panel; - панель отображения текста формулы на окне
- button*** knopki; - матрица кнопок

Методы:

- void Update(); - метод обрабатывает события
- void Draw(); - метод отрисовывает объекты на форме
- void Init(); - метод инициализирует объекты
- MainForm(int Windht = 1000, int Height = 1300); - конструктор инициализации
- ~MainForm(); - деструктор
- void Start(); - запускает окно и выполняет бесконечный цикл работы приложения
- void zap_knopki(); - заполняет имена кнопок
- void MousKlic(wstring s); - метод обрабатывающий нажатие на кнопки

Класс PanelFormul:

Класс отображает формулу на окне приложения

Поля:

- Font font; - поле содержавшие шрифт текста
- Text text; - поле текста
- RectangleShape rectagle; - прямоугольники выделяющий область формулы

Методы:

- PanelFormul(int Windht, int Height); - конструктор
- void draw(RenderWindow* window); - отрисовка поля на форме
- void SetText(wstring s); - метод задает строку формулы
- Text& Text() { return text; } – метод возвращает текст данного поля

Класс button:

Поля:

- RectangleShape rectagle; - прямоугольник ограничивавший область кнопки
- Font font; - поле содержавшие шрифт текста
- Text text; - поле текста кнопки
- Color Hover; - цвет кнопки при наведения на него мышки
- Color Fill; - цвет кнопки

Методы:

- button(int Windht = 100, int Height = 100); - конструктор
- Color GetColor() – выводит цвет кнопки
- Vector2f GetSize() - размеры кнопки
- Vector2f GetPosition() – положение кнопки
- void SetColor(Color color); - метод задает цвет кнопке
- void SetHoverColor(Color color) – метод задает цвет кнопки при наведение
- Text& Text() – получить текст кнопки
- void SetPosition(Vector2f vec); - задает положение кнопки
- void draw(RenderWindow* window); - отрисовка кнопки на форме
- bool MousHover(RenderWindow* window); - проверяет наведена ли курсор на кнопку

4.Эксперементы:

Введем в приложение формулы выведем инфиксные записи и ответ и проверим с их.

Формула	Постфиксная запись	Ответ
12 + 4	12 4 +	16
((12 + 8) * 4 + 7) / 3 + 9 * 0 - 1	12 8 + 4 * 7 + 3 / 9 0 * + 1 -	28
pow(2,10)	2 10 pow	1024
25^3-7*88	25 3 ^ 7 88 * -	15009
100 + pow(ln(e),3) - sin(pi)! + pow(2,pow(2,3))	100 e ln 3 pow + pi sin ! - 2 2 3 pow pow +	356
(100 + 5!) * 2 - 4 / 2 + 13	100 5 ! + 2 * 4 2 / - 13 +	451
-10 – 12	10 12 - -	-22

Получили верные постфиксные записи и ответы, значит алгоритмы верно работают.

5.Заключение

В ходе выполнения этой лабораторной работы, я успешно справился со всеми поставленными задачами. После изучения темы постфиксной записи был реализован класс для обработки арифметических выражений, после в него были добавлены различные улучшения такие как определения функций, факториала и отрицания переменной. Реализовали тесты для проверки основных функций класса. Для данного класса был реализован дополнительно класс стек для обработки арифметического выражения, который был построен на упрошенном одностороннем списке. Так же мы реализовали main для проверки работоспособности программы, после мы реализовали визуальный интерфейс для приложения который имел вид стандартного калькулятора, визуальный интерфейс создавался с помощью библиотеки SFML, которая отрисовывает примитивы на экране. Для работы с приложением были созданы кнопки управления приложением, а так же окно отображения формулы.

6.Литература:

1. Википедия/Стек - https://ru.wikipedia.org/wiki/Стек
2. Википедия/Список - https://ru.wikipedia.org/wiki/Связный_список
3. Учебно-методическое пособие - https://drive.google.com/drive
4. Библиотека SFML - https://www.sfml-dev.org/

7.Приложение:

Класс MyStack:

```
template<class T>
class MyStack {

public:
	MyStack() noexcept;
	bool empty() noexcept;
	void push(T value);
	void pop();
	void clear();
	size_t size() noexcept { return sz; }
	T& top();
	~MyStack();
private:
	template<class T>
	class Note {
	public:
		T value;
		Note* next;
		Note(T value = T(),Note* next = nullptr) {
			this->value = value;
			this->next = next;
		}
	};

	Note<T>* head;
	size_t sz;
};

template<class T>
inline MyStack<T>::MyStack() noexcept
{
	head = nullptr;
	sz = 0;
}

template<class T>
bool MyStack<T>::empty() noexcept
{
	if (sz == 0)
		return true;
	return false;
}

template<class T>
void MyStack<T>::push(T value)
{
	if (head == nullptr)
		head = new Note<T>(value);
	else {
		Note<T>* temp =  new Note<T>(value,head);
		this->head = temp;
	}
	sz++;
}

template<class T>
void MyStack<T>::pop()
{
	if (this->empty())
```

```
                throw exception("Отсутствуют элементы в стеке");
        Note<T>* temp = head;
        head = head->next;
        delete temp;
        sz--;
}

template<class T>
inline void MyStack<T>::clear()
{
        while (!this->empty())
                this->pop();
}

template<class T>
T& MyStack<T>::top()
{
        if (this->empty())
                throw exception("Отсутствуют элементы в стеке");
        return head->value;
}

template<class T>
inline MyStack<T>::~MyStack()
{
        this->clear();
}
```

Calculator.lib:

```
#include "calculator_lib.h"
bool CheckNumber(char c)
{
	return (c >= '0' && c <= '9' || c == '.');
}
bool CheckChar(char c) {
	return (c >= 'A' && c <= 'Z' || c >= 'a' && c <= 'z' || c == '_');
}

bool CheckOperator(char c)
{
	return (c == '+' || c == '-' || c == '*' || c == '/' || c == '%' || c == '^');
}
bool CheckOperator(string s)
{
	return CheckOperator(s[0]);
}

int CheckConst(string s) {
	if (s == "E" || s == "e")
		return 1;
	else if (s == "PI" || s == "pi" || s == "Pi")
	{
		return 2;
	}
	return 0;
}

int Func(string s)
{
	if (s == "sin" || s == "cos" || s == "tg" || s == "ctg" || s == "ln")
		return 1;
	if (s == "pow" || s == "sqrt" || s == "log" || s == "A" || s == "C")
		return 2;
	return 0;
}

bool CheckFunc(string s, string func, int& i)
{
	int kol = 1;
	int zap = 0, ti = i + 1;
	i += 2;
	string temp = "(", temp2 = "(";
	while (i < s.size() && kol != 0) {
		switch (s[i])
		{
		case '(':
			kol += 1;
			break;
		case ')':
			kol -= 1;
			break;
		case ',':
			if (zap == 0 && kol == 1) {
				zap = 1;
				temp += ')';
			}
		default:
			break;
		}
```

```
                        if (zap == 0)
                                temp += s[i];
                        else if (zap == 2) {
                                temp2 += s[i];
                        }
                        else
                                zap = 2;
                        i++;
                }
                i--;
                if (Func(func) == 1) {
                        if (zap != 0)
                                return false;
                        return CheckFormula(temp);
                }
                if (Func(func) == 2) {
                        if (zap == 0)
                                return false;
                        return CheckFormula(temp) && CheckFormula(temp2);
                }
                return true;
}
double DuFunc(string fun, double a)
{
                if (fun == "sin")
                        return sin(a);
                if (fun == "cos")
                        return cos(a);
                if (fun == "tg")
                        return sin(a) / cos(a);
                if (fun == "ctg")
                        return cos(a) / sin(a);
                if (fun == "ln")
                        return log(a);
                return 0.0;
}
double DuFunc(string fun, double a, double b)
{
                if (fun == "pow")
                        return pow(a, b);
                if (fun == "sqrt")
                        return pow(a, (1 / b));
                if (fun == "log")
                        return Log(b, a);
                if (fun == "A")
                        return A(a, b);
                if (fun == "C")
                        return C(a, b);
                return 0.0;
}
double StandartOperator(double a, double b, char c)
{
                switch (c)
                {
                case '+':
                        return a + b;
                case '-':
                        return a - b;
                case '*':
                        return a * b;
                case '/':
                        return a / b;
```

```
		case '%':
			return (int)a % (int)b;
		case '^':
			return pow(a, b);
		default:
			return 0;
		}
}
string RemoveSpace(string s)
{
		int i = 0;
		while (i < s.size()) {
			if (s[i] == ' ')
				s.erase(i, 1);
			else
				i++;
		}
		return s;
}
string Remove0(string s) {
		int i = s.size() - 1;
		while (i >= 0 && s[i] == '0') {
			s.pop_back();
			i--;
		}
		if (s.back() == '.')
			s.pop_back();
		return s;

}
double ToDouble(string s, int &i) {
		double k = 0;
		int kol = 0;
		while (s[i] != '\0' && CheckNumber(s[i])) {
			if (s[i] != '.')
				k = k * 10 + (s[i] - 48);
			if (s[i] == '.' || kol != 0)
				kol++;
			i++;
		}
		i--;
		return k / pow(10, ((kol == 0)?0:kol - 1));
}
string ToStringVar(string s, int& i) {
		string temp;
		char c = s[i];
		while (c != '\0' && CheckChar(c)) {
			temp += s[i];
			i++;
			c = s[i];
		}
		i--;
		return temp;
}
bool Eror(char c, MyStack<string>& stack)
{
		bool flag = true;
		switch (c)
		{
		case '(':
			if (!stack.empty() && stack.top() == "!")
				flag = false;
```

```
                break;
        case '~':
                if (stack.empty() || stack.top() != "(" || stack.top() == "!")
                        flag = false;
                break;
        case ')':
                if (stack.size() < 2 || !(stack.top() == "P" || stack.top() == "SK" || CheckNumber(stack.top()[0]) || Check-
Char(stack.top()[0])))
                        flag = false;
                break;
        case '+':
                if (stack.empty() || stack.top() == "(" || CheckOperator(stack.top()) || stack.top() == "~")
                        flag = false;
                break;
        case '!':
                if (stack.empty() || stack.top() == "+" || stack.top() == "(" || stack.top() == "~")
                        flag = false;
                break;
        default:
                if (!stack.empty() && stack.top() == "!")
                        flag = false;
                break;
        }
        return flag;
}
bool CheckFormula(string s)
{
        s = RemoveSpace(s);
        MyStack<string>stack;
        int i = 0;
        while (i < s.size()) {
                char c = (CheckOperator(s[i])?'+':s[i]);

                if (!Eror(c, stack))
                        return false;

                switch (c){
                case '!':
                        stack.pop();
                        stack.push("P");
                        break;
                case '~':
                        stack.push("~");
                        break;
                case '(':
                        stack.push("(");
                        break;
                case ')':
                        stack.pop(); stack.pop();
                        if (stack.empty() || stack.top() == "(")
                                stack.push("SK");
                        else if (CheckOperator(stack.top())) {
                                stack.pop();
                                stack.push("P");
                        }
                        else
                                return false;
                        break;

                case '+':

                        stack.pop();
```

```
                                stack.push(string(1,s[i]));
                                break;

                        default:
                                string var;
                                if (CheckNumber(c)) {
                                        double num = ToDouble(s, i);
                                        var = to_string(num);
                                }
                                if (c != '\0' && CheckChar(c)) {
                                        var = ToStringVar(s, i);
                                }

                                if (Func(var)) {
                                        if (!CheckFunc(s,var, i))
                                                return false;
                                        var = "SK";
                                }


                                if (stack.top() == "(")
                                        stack.push(var);
                                else if (CheckOperator(stack.top())) {
                                        stack.pop();
                                        stack.push("P");
                                }
                                else if (stack.top() == "~") {
                                        stack.pop();
                                        stack.push("P");
                                }
                                else
                                        return false;
                        }
                        i++;
                }
                if (stack.top() != "SK" || stack.size() != 1)
                        return false;
                return true;
}
int calculator::priority(char c)
{
                switch (c)
                {
                case '(':
                        return 0;
                case '+': case '-':
                        return 1;
                case '*': case '/': case '%':
                        return 2;
                case '^':
                        return 3;
                case '~':
                        return 4;
                default:
                        return -1;
                }
}
calculator::calculator()
{
                this->formula = "";
                this->postfix = "";
}
```

```
calculator::calculator(string formula)
{
	SetFormula(formula);
}
void calculator::SetFormula(string formula)
{
	formula = RemoveSpace("(" + formula  + ")");
	//Замена унарного минуса на ~
	size_t start{ formula.find("(-")};
	while (start != std::string::npos)
	{
		formula.replace(start, 2, "(~");
		start = formula.find("(-", start + 2);
	}

	postfix.clear();
	if (!CheckFormula(formula))
		throw exception("Введено неверная формула");
	this->formula = formula;
	this->FormulaConverrt();
}
string calculator::GetFormula()
{
	string temp;
	for (auto a : formula) {
		if (CheckOperator(a))
			temp = temp + " " + a + " ";
		else if (a == '~')
			temp += '-';
		else
			temp += a;
	}
	return temp;
}
string calculator::GetPostfix()
{
	string temp = postfix;
	for (char &a : temp) {
		if (a == '~')
			a = '-';
	}
	return temp;
}
void calculator::FormulaConverrt()
{
	var.clear();
	if (formula.empty())
		throw exception("Формула не заданна");
	MyStack<string> stack;
	int i = 0;
	while (i < formula.size()) {
		char c = (CheckOperator(formula[i]) ? '+' : formula[i]);
		switch (c)
		{
		case '(':
			stack.push("(");
			break;
		case ')':
			while (stack.top() != "(") {
				if (!postfix.empty() && postfix.back() != ' ')
					postfix += " ";
				postfix +=  stack.top();
```

```
                                        stack.pop();
                                }
                                stack.pop();
                                if (!stack.empty() && Func(stack.top())) {
                                        if (!postfix.empty() && postfix.back() != ' ')
                                                postfix += " ";
                                        postfix +=  stack.top();
                                        stack.pop();
                                }
                                if (!postfix.empty() && postfix.back() != ' ')
                                        postfix += " ";
                                break;
                        case ',':
                                while (stack.top() != "(") {
                                        if (!postfix.empty() && postfix.back() != ' ')
                                                postfix += " ";
                                        postfix += stack.top();
                                        stack.pop();
                                }
                                if (!postfix.empty() && postfix.back() != ' ')
                                        postfix += " ";
                                break;
                        case '+': case '~':
                                while (CheckOperator(stack.top()) && priority(formula[i]) <= priority(stack.top()[0])) {
                                        if (!postfix.empty() && postfix.back() != ' ')
                                                postfix += " ";
                                        postfix += stack.top();
                                        stack.pop();
                                }
                                stack.push(string(1,formula[i]));
                                if (!postfix.empty() && postfix.back() != ' ')
                                        postfix += " ";
                                break;

                        default:
                                if (CheckChar(c)) {
                                        string temp = ToStringVar(formula, i);
                                        if (Func(temp))
                                                stack.push(temp);
                                        else {
                                                if (!CheckConst(temp))
                                                        var.insert({ temp,0 });
                                                postfix += temp;
                                        }
                                }
                                else if (CheckNumber(c)) {
                                        double num = ToDouble(formula, i);
                                        postfix += Remove0(to_string(num));
                                }
                                else
                                        postfix += c;
                                if (!postfix.empty() && postfix.back() != ' ')
                                        postfix += " ";
                        }
                        i++;

        }
}
double calculator::Ansver()
{
        MyStack<double> stack;
        double a, b;
```

```
	int i = 0;
	while(i < postfix.size()){
		char c = (CheckOperator(postfix[i]) ? '+': postfix[i]);
		switch (c)
		{
		case '~':
			a = stack.top(); stack.pop();
			stack.push(-a);
			break;
		case '+':
			a = stack.top(); stack.pop();
			b = stack.top(); stack.pop();
			stack.push(StandartOperator(b, a, postfix[i]));
			break;
		case ' ':
			break;
		default:
			if (c == '!') {
				int n = (int)stack.top(); stack.pop();
				stack.push(fac(n));
				break;
			}
			if (CheckNumber(postfix[i]))
				stack.push(ToDouble(postfix, i));
			else {
				string temp = ToStringVar(postfix, i);
				if (!Func(temp)) {
					if (CheckConst(temp) == 0)
						stack.push(var.at(temp));
					else if (CheckConst(temp) == 1)
						stack.push(E);
					else
						stack.push(PI);
				}
				else {
					if (Func(temp) == 1) {
						double a = stack.top(); stack.pop();
						stack.push(DuFunc(temp,a));
					}
					if (Func(temp) == 2) {
						double a = stack.top(); stack.pop();
						double b = stack.top(); stack.pop();
						stack.push(DuFunc(temp, b, a));
					}
				}
			}
		}
		i++;
	}
	return stack.top();
}
void calculator::SetVar(map<string, double> var)
{
	for (auto a : var) {
		this->var[a.first] = a.second;
	}
}

double st(double x, int p)
{
	double ans = 1;
	while (p > 0) {
```

```
                if (p % 2)
                        ans *= x;
                p /= 2;
                x *= x;
        }
        return ans;
}

double fac(int n)
{
        double ans = 1;
        for (int i = 2; i <= n; i++) {
                ans *= i;
        }
        return ans;
}

double Log(double a, double b)
{
        return log(a) / log(b);
}

double A(int n, int k)
{
        return fac(n) / fac(n - k);
}

double C(int n, int k)
{
        return fac(n) / ((long long)fac(n - k) * fac(k));
}
```

Класс MainForm:

```
#include "MainForm.h"

void MainForm::Update()
{
	sf::Event event;
	while (window->pollEvent(event))
	{
		if (event.type == sf::Event::Closed)
			window->close();
		if (event.type == Event::MouseButtonReleased) {
			if (event.mouseButton.button == Mouse::Left) {
				if (exit->MousHover(window)) {
					window->close();
				}
				for (int i = 0; i < 7; i++) {
					for (int j = 0; j < 5; j++) {
						if (knopki[i][j]->MousHover(window)) {
							MousKlic(knopki[i][j]->Text().getString());
						}
					}
				}
			}
		}

		exit->MousHover(window);

		for (int i = 0; i < 7; i++) {
			for (int j = 0; j < 5; j++) {
				knopki[i][j]->MousHover(window);
			}
		}

}
void MainForm::Draw()
{
	window->clear(Color(28,32,42));

	exit->draw(window);
	panel->draw(window);

	window->draw(name);

	for (int i = 0; i < 7; i++) {
		for (int j = 0; j < 5; j++) {
			knopki[i][j]->draw(window);
		}
	}

	window->display();
}

void MainForm::Init()
{
	exit = new button(40, 50);
	exit->SetColor(Color(28, 32, 42));
	exit->Text().setString("X");
	exit->SetPosition(Vector2f(Vector2f(window->getSize().x - exit->GetSize().x, 0)));
	exit->SetHoverColor(Color::Red);
```

```
	if (!font.loadFromFile("Times.ttf"))
		throw exception("NON font");

	name.setFont(font);
	name.setCharacterSize(30);
	name.setPosition(Vector2f(30, 10));
	name.setFillColor(Color::White);
	name.setString(L"Калькулятор");

	panel = new PanelFormul(window->getSize().x, 400);

	knopki = new button ** [7];
	for (int i = 0; i < 7; i++) {
		knopki[i] = new button*[5];
		for (int j = 0; j < 5; j++) {
			Vector2u r = window->getSize();
			int rx = r.x / 5 - 12;
			int ry = (r.y - 550) / 7 - 10;
			knopki[i][j] = new  button(rx, ry);
			knopki[i][j]->SetPosition(Vector2f(j * (rx + 10) + 10, i * (ry + 10) + 550 ));
			knopki[i][j]->Text().setString("X");
			knopki[i][j]->Text().setCharacterSize(50);
			knopki[i][j]->SetColor(Color(45, 50, 68));
			knopki[i][j]->SetHoverColor(Color(54, 59, 77));
		}
	}
	zap_knopki();

}

MainForm::MainForm(int Windht, int Height)
{
	window = new RenderWindow(VideoMode(Windht, Height), L"Калькулятор", Style::None);
	window->setPosition(Vector2i(100, 100));
	this->Init();
}

MainForm::~MainForm()
{
	delete window;
	delete exit;

	for (int i = 0; i < 7; i++) {
		for (int j = 0; j < 5; j++) {
			delete knopki[i][j];
		}
		delete[] knopki[i];
	}
	delete[]knopki;
}

void MainForm::Start()
{
	window->setFramerateLimit(60);
	while (window->isOpen()) {
		Update();


		Draw();
	}

}
```

```
void MainForm::zap_knopki()
{
	int k = 1;
	for (int i = 5; i >= 3; i--) {
		for (int j = 1; j <= 3; j++,k++) {
			knopki[i][j]->Text().setString(to_string(k));
		}
	}
	knopki[6][2]->Text().setString(L"0");
	knopki[6][3]->Text().setString(L".");
	knopki[6][1]->Text().setString(L",");

	knopki[0][1]->Text().setString(L"π");
	knopki[0][2]->Text().setString(L"e");
	knopki[0][3]->Text().setString(L"CE");
	knopki[0][4]->Text().setString(L"<");
	knopki[1][0]->Text().setString(L"sin");
	knopki[1][1]->Text().setString(L"xⁿ");
	knopki[1][2]->Text().setString(L"ⁿ√x");
	knopki[1][3]->Text().setString(L"logₓ");


	knopki[1][4]->Text().setString(L"%");
	knopki[2][1]->Text().setString(L"(");
	knopki[2][2]->Text().setString(L")");
	knopki[2][3]->Text().setString(L"n!");
	knopki[2][4]->Text().setString(L"÷");
	knopki[3][4]->Text().setString(L"×");
	knopki[4][4]->Text().setString(L"-");
	knopki[5][4]->Text().setString(L"+");
	knopki[6][4]->Text().setString(L"=");

	knopki[2][0]->Text().setString(L"cos");
	knopki[3][0]->Text().setString(L"tg");
	knopki[4][0]->Text().setString(L"ctg");

	knopki[0][0]->Text().setString(L"ln");

	knopki[5][0]->Text().setString(L"Aⁿₓ");
	knopki[6][0]->Text().setString(L"Cⁿₓ");
}
void MainForm::MousKlic(wstring s)
{
	wstring temp = panel->Text().getString();
	if (s == "CE")
		panel->SetText(L"0");
	if (s == "<") {
		if (temp.size() == 1)
			temp = L"0";
		else
			temp.pop_back();
		panel->SetText(temp);
	}

	if (s >= "0" && s <= "9" || s == L"π" || s == "e") {
		if (temp == "0")
			temp = s;
		else
			temp += s;
		panel->SetText(temp);
	}
```

```
if (s == "+" || s == "-" || s == L"×" || s == L"÷" || s == "%" || s == "." || s == ",")
		panel->SetText(temp + s);
if (s >= "(" && s <= ")") {
		if (temp == "0")
			temp = s;
		else
			temp += s;
		panel->SetText(temp);
}
if (s == "n!")
		panel->SetText(temp + "!");
if (s == "ln" || s == "sin" || s == "cos" || s == "tg" || s == "ctg" ) {
		if (temp == "0")
			temp = s + "(";
		else
			temp += s + "(";
		panel->SetText(temp);
}
if (s == L"log ₓ") {
		s.pop_back();
		if (temp == "0")
			temp = s + "(";
		else
			temp += s + "(";
		panel->SetText(temp);
}
if (s == L"Aⁿ ₓ" || s == L"Cⁿ ₓ") {
		s.pop_back();
		s.pop_back();
		if (temp == "0")
			temp = s + "(";
		else
			temp += s + "(";
		panel->SetText(temp);
}
if (s == L"xⁿ") {
		if (temp == "0")
			temp = L"pow(";
		else
			temp += L"pow(";
		panel->SetText(temp);
}
if (s == L"ⁿ√x") {
		if (temp == "0")
			temp = L"sqrt(";
		else
			temp += L"sqrt(";
		panel->SetText(temp);
}
if (s == "=") {
		try {
			string str;
			for (wchar_t a : temp) {
				if (a == *L"π")
					str += "pi";
				else if (a == *L"×")
					str += "*";
				else if (a == *L"÷")
					str += "/";
				else
					str += a;
			}
```

```
                    calculator c(str);
                    str = Remove0(to_string(c.Ansver()));
                    temp = L"";
                    for (auto a: str){
                        temp += a;
                    }
                    panel->SetText(temp);
                }
                catch (exception& error) {
                    temp = L"Неверная формула";
                    panel->SetText(temp);
                }
            }
}
```

Класс PanelFormul:

```
#include "PanelFormul.h"

PanelFormul::PanelFormul(int Windht, int Height)
{
	rectagle.setPosition(0, 0);
	rectagle.setSize(Vector2f(Windht - 8, Height));
	rectagle.setPosition(Vector2f(4, 80));
	rectagle.setFillColor(Color(17, 24, 38));
	/*rectagle.setOutlineThickness(1);
	rectagle.setOutlineColor(Color::White);*/


	if (!font.loadFromFile("Times.ttf"))
		throw exception("NON font");

	text.setFont(font);
	text.setCharacterSize(70);
	text.setOrigin(text.getLocalBounds().width, 0);
	text.setString("0");
	text.setPosition(rectagle.getSize().x - text.getLocalBounds().width - 10,rectagle.getPosition().y + rectagle.getSize().y
/ 2 - 50);
	text.setFillColor(Color::White);
}

void PanelFormul::draw(RenderWindow* window)
{
	window->draw(rectagle);
	window->draw(text);
}

void PanelFormul::SetText(wstring s)
{
	text.setString(s);
	text.setPosition(rectagle.getSize().x - text.getLocalBounds().width - 10, rectagle.getPosition().y + rectagle.getSize().y
/ 2 - 50);
}
```

Класс button:

```
#include "button.h"

button::button(int Windht, int Height)
{
	rectagle.setPosition(0, 0);
	rectagle.setSize(Vector2f(Windht, Height));
	Fill = Hover = Color::Green;
	rectagle.setFillColor(Fill);

	if (!font.loadFromFile("Times.ttf"))
		throw exception("NON font");
	text.setFont(font);
	text.setCharacterSize(Height - 12);
	text.setFillColor(Color::White);



void button::SetColor(Color color)
{
	rectagle.setFillColor(color);
	Fill = color;
	if (Hover != Color::Green)
		Hover = color;

}

void button::SetPosition(Vector2f vec)
{
	rectagle.setPosition(vec);
	text.setPosition(Vector2f(vec.x + rectagle.getSize().x / 2 - 15, vec.y + rectagle.getSize().y / 2 - 30));
}

void button::draw(RenderWindow* window)
{
	window->draw(rectagle);
	window->draw(text);
}

bool button::MousHover(RenderWindow* window)
{
	int x = Mouse::getPosition(*window).x;
	int y = Mouse::getPosition(*window).y;
	bool flag = true;
	if (x < rectagle.getPosition().x || x > rectagle.getPosition().x + rectagle.getSize().x)
		flag = false;
	if (y < rectagle.getPosition().y || y > rectagle.getPosition().y + rectagle.getSize().y)
		flag = false;
	flag ? rectagle.setFillColor(Hover) : rectagle.setFillColor(Fill);
	return flag;
}
```