**MyForm.h:**  
  
#pragma once

#include "Parser.h"

#include "dop\_func.h"

namespace WinForms {

using namespace System;

using namespace System::ComponentModel;

using namespace System::Collections;

using namespace System::Windows::Forms;

using namespace System::Data;

using namespace System::Drawing;

/// <summary>

/// Сводка для MyForm

/// </summary>

public ref class MyForm : public System::Windows::Forms::Form

{

public:

MyForm(void)

{

InitializeComponent();

//

//TODO: добавьте код конструктораs

//

}

protected:

/// <summary>

/// Освободить все используемые ресурсы.

/// </summary>

~MyForm()

{

if (components)

{

delete components;

}

}

private: System::Windows::Forms::MenuStrip^ menuStrip1;

private: System::Windows::Forms::ToolStripMenuItem^ управлениеToolStripMenuItem;

private: System::Windows::Forms::ToolStripMenuItem^ помощьToolStripMenuItem;

private: System::Windows::Forms::ToolStripMenuItem^ очToolStripMenuItem;

private: System::Windows::Forms::ToolStripMenuItem^ выполнитьToolStripMenuItem;

private: System::Windows::Forms::ToolStripMenuItem^ завершитьToolStripMenuItem;

private: System::Windows::Forms::TextBox^ textBox8;

private: System::Windows::Forms::TextBox^ textBox7;

private: System::Windows::Forms::TextBox^ textBox6;

private: System::Windows::Forms::TextBox^ textBox5;

private: System::Windows::Forms::Label^ label9;

private: System::Windows::Forms::Label^ label10;

private: System::Windows::Forms::Label^ label11;

private: System::Windows::Forms::Label^ label12;

private: System::Windows::Forms::Label^ label8;

private: System::Windows::Forms::ListBox^ listBox3;

private: System::Windows::Forms::ListBox^ listBox4;

private: System::Windows::Forms::ListBox^ listBox2;

private: System::Windows::Forms::ListBox^ listBox1;

private: System::Windows::Forms::DataVisualization::Charting::Chart^ chart1;

private: System::Windows::Forms::Label^ label3;

private: System::Windows::Forms::Label^ label2;

private: System::Windows::Forms::Label^ label1;

private: System::Windows::Forms::TextBox^ textBox4;

private: System::Windows::Forms::TextBox^ textBox3;

private: System::Windows::Forms::TextBox^ textBox2;

private: System::Windows::Forms::TextBox^ textBox1;

private: System::Windows::Forms::ComboBox^ comboBox1;

private: System::Windows::Forms::TextBox^ textBox12;

private: System::Windows::Forms::TextBox^ textBox11;

private: System::Windows::Forms::TextBox^ textBox10;

private: System::Windows::Forms::TextBox^ textBox9;

private: System::Windows::Forms::Label^ label4;

private: System::Windows::Forms::Label^ label5;

private: System::Windows::Forms::Label^ label6;

private: System::Windows::Forms::Label^ label7;

private: System::Windows::Forms::Label^ label13;

private: System::Windows::Forms::ListBox^ listBox5;

private: System::Windows::Forms::ListBox^ listBox6;

private: System::Windows::Forms::Label^ label14;

private: System::Windows::Forms::Label^ label15;

private: System::Windows::Forms::Label^ label16;

private: System::Windows::Forms::Label^ label17;

private: System::Windows::Forms::Label^ label18;

private: System::Windows::Forms::Label^ label19;

private: System::Windows::Forms::ComboBox^ comboBox2;

private: System::Windows::Forms::Label^ label20;

private: System::Windows::Forms::CheckBox^ checkBox1;

/// <summary>

/// Обязательная переменная конструктора.

/// </summary>

System::ComponentModel::Container^ components;

#pragma region Windows Form Designer generated code

/// <summary>

/// Требуемый метод для поддержки конструктора — не изменяйте

/// содержимое этого метода с помощью редактора кода.

/// </summary>

void InitializeComponent(void)

{

System::Windows::Forms::DataVisualization::Charting::ChartArea^ chartArea1 = (gcnew System::Windows::Forms::DataVisualization::Charting::ChartArea());

System::Windows::Forms::DataVisualization::Charting::Legend^ legend1 = (gcnew System::Windows::Forms::DataVisualization::Charting::Legend());

System::Windows::Forms::DataVisualization::Charting::Series^ series1 = (gcnew System::Windows::Forms::DataVisualization::Charting::Series());

System::Windows::Forms::DataVisualization::Charting::Series^ series2 = (gcnew System::Windows::Forms::DataVisualization::Charting::Series());

System::Windows::Forms::DataVisualization::Charting::Series^ series3 = (gcnew System::Windows::Forms::DataVisualization::Charting::Series());

this->menuStrip1 = (gcnew System::Windows::Forms::MenuStrip());

this->управлениеToolStripMenuItem = (gcnew System::Windows::Forms::ToolStripMenuItem());

this->помощьToolStripMenuItem = (gcnew System::Windows::Forms::ToolStripMenuItem());

this->очToolStripMenuItem = (gcnew System::Windows::Forms::ToolStripMenuItem());

this->выполнитьToolStripMenuItem = (gcnew System::Windows::Forms::ToolStripMenuItem());

this->завершитьToolStripMenuItem = (gcnew System::Windows::Forms::ToolStripMenuItem());

this->textBox8 = (gcnew System::Windows::Forms::TextBox());

this->textBox7 = (gcnew System::Windows::Forms::TextBox());

this->textBox6 = (gcnew System::Windows::Forms::TextBox());

this->textBox5 = (gcnew System::Windows::Forms::TextBox());

this->label9 = (gcnew System::Windows::Forms::Label());

this->label10 = (gcnew System::Windows::Forms::Label());

this->label11 = (gcnew System::Windows::Forms::Label());

this->label12 = (gcnew System::Windows::Forms::Label());

this->label8 = (gcnew System::Windows::Forms::Label());

this->listBox3 = (gcnew System::Windows::Forms::ListBox());

this->listBox4 = (gcnew System::Windows::Forms::ListBox());

this->listBox2 = (gcnew System::Windows::Forms::ListBox());

this->listBox1 = (gcnew System::Windows::Forms::ListBox());

this->chart1 = (gcnew System::Windows::Forms::DataVisualization::Charting::Chart());

this->label3 = (gcnew System::Windows::Forms::Label());

this->label2 = (gcnew System::Windows::Forms::Label());

this->label1 = (gcnew System::Windows::Forms::Label());

this->textBox4 = (gcnew System::Windows::Forms::TextBox());

this->textBox3 = (gcnew System::Windows::Forms::TextBox());

this->textBox2 = (gcnew System::Windows::Forms::TextBox());

this->textBox1 = (gcnew System::Windows::Forms::TextBox());

this->comboBox1 = (gcnew System::Windows::Forms::ComboBox());

this->textBox12 = (gcnew System::Windows::Forms::TextBox());

this->textBox11 = (gcnew System::Windows::Forms::TextBox());

this->textBox10 = (gcnew System::Windows::Forms::TextBox());

this->textBox9 = (gcnew System::Windows::Forms::TextBox());

this->label4 = (gcnew System::Windows::Forms::Label());

this->label5 = (gcnew System::Windows::Forms::Label());

this->label6 = (gcnew System::Windows::Forms::Label());

this->label7 = (gcnew System::Windows::Forms::Label());

this->label13 = (gcnew System::Windows::Forms::Label());

this->listBox5 = (gcnew System::Windows::Forms::ListBox());

this->listBox6 = (gcnew System::Windows::Forms::ListBox());

this->label14 = (gcnew System::Windows::Forms::Label());

this->label15 = (gcnew System::Windows::Forms::Label());

this->label16 = (gcnew System::Windows::Forms::Label());

this->label17 = (gcnew System::Windows::Forms::Label());

this->label18 = (gcnew System::Windows::Forms::Label());

this->label19 = (gcnew System::Windows::Forms::Label());

this->comboBox2 = (gcnew System::Windows::Forms::ComboBox());

this->label20 = (gcnew System::Windows::Forms::Label());

this->checkBox1 = (gcnew System::Windows::Forms::CheckBox());

this->menuStrip1->SuspendLayout();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->chart1))->BeginInit();

this->SuspendLayout();

//

// menuStrip1

//

this->menuStrip1->Items->AddRange(gcnew cli::array< System::Windows::Forms::ToolStripItem^ >(2) {

this->управлениеToolStripMenuItem,

this->завершитьToolStripMenuItem

});

this->menuStrip1->Location = System::Drawing::Point(0, 0);

this->menuStrip1->Name = L"menuStrip1";

this->menuStrip1->Size = System::Drawing::Size(931, 24);

this->menuStrip1->TabIndex = 16;

this->menuStrip1->Text = L"menuStrip1";

//

// управлениеToolStripMenuItem

//

this->управлениеToolStripMenuItem->DropDownItems->AddRange(gcnew cli::array< System::Windows::Forms::ToolStripItem^ >(3) {

this->помощьToolStripMenuItem,

this->очToolStripMenuItem, this->выполнитьToolStripMenuItem

});

this->управлениеToolStripMenuItem->Name = L"управлениеToolStripMenuItem";

this->управлениеToolStripMenuItem->Size = System::Drawing::Size(85, 20);

this->управлениеToolStripMenuItem->Text = L"Управление";

//

// помощьToolStripMenuItem

//

this->помощьToolStripMenuItem->Name = L"помощьToolStripMenuItem";

this->помощьToolStripMenuItem->Size = System::Drawing::Size(136, 22);

this->помощьToolStripMenuItem->Text = L"Помощь";

this->помощьToolStripMenuItem->Click += gcnew System::EventHandler(this, &MyForm::помощьToolStripMenuItem\_Click);

//

// очToolStripMenuItem

//

this->очToolStripMenuItem->Name = L"очToolStripMenuItem";

this->очToolStripMenuItem->Size = System::Drawing::Size(136, 22);

this->очToolStripMenuItem->Text = L"Очистить";

this->очToolStripMenuItem->Click += gcnew System::EventHandler(this, &MyForm::очиститьToolStripMenuItem\_Click);

//

// выполнитьToolStripMenuItem

//

this->выполнитьToolStripMenuItem->Name = L"выполнитьToolStripMenuItem";

this->выполнитьToolStripMenuItem->Size = System::Drawing::Size(136, 22);

this->выполнитьToolStripMenuItem->Text = L"Выполнить";

this->выполнитьToolStripMenuItem->Click += gcnew System::EventHandler(this, &MyForm::выполнитьToolStripMenuItem\_Click);

//

// завершитьToolStripMenuItem

//

this->завершитьToolStripMenuItem->Name = L"завершитьToolStripMenuItem";

this->завершитьToolStripMenuItem->Size = System::Drawing::Size(80, 20);

this->завершитьToolStripMenuItem->Text = L"Завершить";

this->завершитьToolStripMenuItem->Click += gcnew System::EventHandler(this, &MyForm::завершитьToolStripMenuItem\_Click);

//

// textBox8

//

this->textBox8->Location = System::Drawing::Point(431, 497);

this->textBox8->Name = L"textBox8";

this->textBox8->Size = System::Drawing::Size(112, 20);

this->textBox8->TabIndex = 86;

//

// textBox7

//

this->textBox7->Location = System::Drawing::Point(431, 471);

this->textBox7->Name = L"textBox7";

this->textBox7->Size = System::Drawing::Size(112, 20);

this->textBox7->TabIndex = 85;

//

// textBox6

//

this->textBox6->Location = System::Drawing::Point(431, 445);

this->textBox6->Name = L"textBox6";

this->textBox6->Size = System::Drawing::Size(112, 20);

this->textBox6->TabIndex = 84;

//

// textBox5

//

this->textBox5->Location = System::Drawing::Point(431, 419);

this->textBox5->Name = L"textBox5";

this->textBox5->ScrollBars = System::Windows::Forms::ScrollBars::Horizontal;

this->textBox5->Size = System::Drawing::Size(182, 20);

this->textBox5->TabIndex = 83;

//

// label9

//

this->label9->AutoSize = true;

this->label9->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 9.75F, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(204)));

this->label9->Location = System::Drawing::Point(318, 497);

this->label9->Name = L"label9";

this->label9->Size = System::Drawing::Size(33, 16);

this->label9->TabIndex = 82;

this->label9->Text = L"Шаг";

//

// label10

//

this->label10->AutoSize = true;

this->label10->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 9.75F, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(204)));

this->label10->Location = System::Drawing::Point(318, 472);

this->label10->Name = L"label10";

this->label10->Size = System::Drawing::Size(105, 16);

this->label10->TabIndex = 81;

this->label10->Text = L"Конец отрезка";

//

// label11

//

this->label11->AutoSize = true;

this->label11->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 9.75F, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(204)));

this->label11->Location = System::Drawing::Point(318, 449);

this->label11->Name = L"label11";

this->label11->Size = System::Drawing::Size(115, 16);

this->label11->TabIndex = 80;

this->label11->Text = L"Начало отрезка";

//

// label12

//

this->label12->AutoSize = true;

this->label12->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 9.75F, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(204)));

this->label12->Location = System::Drawing::Point(318, 423);

this->label12->Name = L"label12";

this->label12->Size = System::Drawing::Size(83, 16);

this->label12->TabIndex = 79;

this->label12->Text = L"Функция (2)";

//

// label8

//

this->label8->AutoSize = true;

this->label8->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 9.75F, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(204)));

this->label8->Location = System::Drawing::Point(0, 497);

this->label8->Name = L"label8";

this->label8->Size = System::Drawing::Size(33, 16);

this->label8->TabIndex = 78;

this->label8->Text = L"Шаг";

//

// listBox3

//

this->listBox3->FormattingEnabled = true;

this->listBox3->Location = System::Drawing::Point(621, 49);

this->listBox3->Name = L"listBox3";

this->listBox3->Size = System::Drawing::Size(64, 329);

this->listBox3->TabIndex = 77;

//

// listBox4

//

this->listBox4->FormattingEnabled = true;

this->listBox4->Location = System::Drawing::Point(691, 49);

this->listBox4->Name = L"listBox4";

this->listBox4->Size = System::Drawing::Size(64, 329);

this->listBox4->TabIndex = 76;

//

// listBox2

//

this->listBox2->FormattingEnabled = true;

this->listBox2->Location = System::Drawing::Point(549, 49);

this->listBox2->Name = L"listBox2";

this->listBox2->Size = System::Drawing::Size(64, 329);

this->listBox2->TabIndex = 75;

//

// listBox1

//

this->listBox1->FormattingEnabled = true;

this->listBox1->Location = System::Drawing::Point(479, 49);

this->listBox1->Name = L"listBox1";

this->listBox1->Size = System::Drawing::Size(64, 329);

this->listBox1->TabIndex = 74;

//

// chart1

//

this->chart1->BackColor = System::Drawing::SystemColors::ScrollBar;

chartArea1->Name = L"Ch1";

this->chart1->ChartAreas->Add(chartArea1);

legend1->Name = L"Legend1";

this->chart1->Legends->Add(legend1);

this->chart1->Location = System::Drawing::Point(12, 27);

this->chart1->Name = L"chart1";

series1->BorderWidth = 2;

series1->ChartArea = L"Ch1";

series1->ChartType = System::Windows::Forms::DataVisualization::Charting::SeriesChartType::Line;

series1->Color = System::Drawing::Color::Blue;

series1->Legend = L"Legend1";

series1->Name = L"Function(1)";

series2->BorderWidth = 2;

series2->ChartArea = L"Ch1";

series2->ChartType = System::Windows::Forms::DataVisualization::Charting::SeriesChartType::Line;

series2->Color = System::Drawing::Color::FromArgb(static\_cast<System::Int32>(static\_cast<System::Byte>(0)), static\_cast<System::Int32>(static\_cast<System::Byte>(192)),

static\_cast<System::Int32>(static\_cast<System::Byte>(0)));

series2->Legend = L"Legend1";

series2->Name = L"Function(2)";

series3->BorderWidth = 2;

series3->ChartArea = L"Ch1";

series3->ChartType = System::Windows::Forms::DataVisualization::Charting::SeriesChartType::Line;

series3->Color = System::Drawing::Color::Red;

series3->Legend = L"Legend1";

series3->Name = L"Function(3)";

this->chart1->Series->Add(series1);

this->chart1->Series->Add(series2);

this->chart1->Series->Add(series3);

this->chart1->Size = System::Drawing::Size(458, 367);

this->chart1->TabIndex = 73;

this->chart1->Text = L"chart1";

//

// label3

//

this->label3->AutoSize = true;

this->label3->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 9.75F, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(204)));

this->label3->Location = System::Drawing::Point(0, 472);

this->label3->Name = L"label3";

this->label3->Size = System::Drawing::Size(105, 16);

this->label3->TabIndex = 72;

this->label3->Text = L"Конец отрезка";

//

// label2

//

this->label2->AutoSize = true;

this->label2->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 9.75F, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(204)));

this->label2->Location = System::Drawing::Point(0, 449);

this->label2->Name = L"label2";

this->label2->Size = System::Drawing::Size(115, 16);

this->label2->TabIndex = 71;

this->label2->Text = L"Начало отрезка";

//

// label1

//

this->label1->AutoSize = true;

this->label1->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 9.75F, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(204)));

this->label1->Location = System::Drawing::Point(0, 423);

this->label1->Name = L"label1";

this->label1->Size = System::Drawing::Size(83, 16);

this->label1->TabIndex = 70;

this->label1->Text = L"Функция (1)";

//

// textBox4

//

this->textBox4->Location = System::Drawing::Point(121, 497);

this->textBox4->Name = L"textBox4";

this->textBox4->Size = System::Drawing::Size(112, 20);

this->textBox4->TabIndex = 69;

//

// textBox3

//

this->textBox3->Location = System::Drawing::Point(121, 471);

this->textBox3->Name = L"textBox3";

this->textBox3->Size = System::Drawing::Size(112, 20);

this->textBox3->TabIndex = 68;

//

// textBox2

//

this->textBox2->Location = System::Drawing::Point(121, 445);

this->textBox2->Name = L"textBox2";

this->textBox2->Size = System::Drawing::Size(112, 20);

this->textBox2->TabIndex = 67;

//

// textBox1

//

this->textBox1->Location = System::Drawing::Point(121, 419);

this->textBox1->Name = L"textBox1";

this->textBox1->ScrollBars = System::Windows::Forms::ScrollBars::Horizontal;

this->textBox1->Size = System::Drawing::Size(182, 20);

this->textBox1->TabIndex = 66;

//

// comboBox1

//

this->comboBox1->FormattingEnabled = true;

this->comboBox1->Items->AddRange(gcnew cli::array< System::Object^ >(3) { L"1 функция", L"2 функции", L"3 функции " });

this->comboBox1->Location = System::Drawing::Point(598, 390);

this->comboBox1->Name = L"comboBox1";

this->comboBox1->Size = System::Drawing::Size(84, 21);

this->comboBox1->TabIndex = 90;

this->comboBox1->Text = L"Не указано";

this->comboBox1->SelectedIndexChanged += gcnew System::EventHandler(this, &MyForm::comboBox1\_SelectedIndexChanged);

//

// textBox12

//

this->textBox12->Location = System::Drawing::Point(737, 497);

this->textBox12->Name = L"textBox12";

this->textBox12->Size = System::Drawing::Size(112, 20);

this->textBox12->TabIndex = 98;

//

// textBox11

//

this->textBox11->Location = System::Drawing::Point(737, 471);

this->textBox11->Name = L"textBox11";

this->textBox11->Size = System::Drawing::Size(112, 20);

this->textBox11->TabIndex = 97;

//

// textBox10

//

this->textBox10->Location = System::Drawing::Point(737, 445);

this->textBox10->Name = L"textBox10";

this->textBox10->Size = System::Drawing::Size(112, 20);

this->textBox10->TabIndex = 96;

//

// textBox9

//

this->textBox9->Location = System::Drawing::Point(737, 419);

this->textBox9->Name = L"textBox9";

this->textBox9->ScrollBars = System::Windows::Forms::ScrollBars::Horizontal;

this->textBox9->Size = System::Drawing::Size(182, 20);

this->textBox9->TabIndex = 95;

//

// label4

//

this->label4->AutoSize = true;

this->label4->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 9.75F, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(204)));

this->label4->Location = System::Drawing::Point(624, 497);

this->label4->Name = L"label4";

this->label4->Size = System::Drawing::Size(33, 16);

this->label4->TabIndex = 94;

this->label4->Text = L"Шаг";

//

// label5

//

this->label5->AutoSize = true;

this->label5->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 9.75F, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(204)));

this->label5->Location = System::Drawing::Point(624, 472);

this->label5->Name = L"label5";

this->label5->Size = System::Drawing::Size(105, 16);

this->label5->TabIndex = 93;

this->label5->Text = L"Конец отрезка";

//

// label6

//

this->label6->AutoSize = true;

this->label6->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 9.75F, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(204)));

this->label6->Location = System::Drawing::Point(624, 449);

this->label6->Name = L"label6";

this->label6->Size = System::Drawing::Size(115, 16);

this->label6->TabIndex = 92;

this->label6->Text = L"Начало отрезка";

//

// label7

//

this->label7->AutoSize = true;

this->label7->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 9.75F, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(204)));

this->label7->Location = System::Drawing::Point(624, 423);

this->label7->Name = L"label7";

this->label7->Size = System::Drawing::Size(83, 16);

this->label7->TabIndex = 91;

this->label7->Text = L"Функция (3)";

//

// label13

//

this->label13->AutoSize = true;

this->label13->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 9.75F, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(204)));

this->label13->Location = System::Drawing::Point(476, 391);

this->label13->Name = L"label13";

this->label13->Size = System::Drawing::Size(116, 16);

this->label13->TabIndex = 99;

this->label13->Text = L"Кол-во функций:";

//

// listBox5

//

this->listBox5->FormattingEnabled = true;

this->listBox5->Location = System::Drawing::Point(761, 49);

this->listBox5->Name = L"listBox5";

this->listBox5->Size = System::Drawing::Size(64, 329);

this->listBox5->TabIndex = 100;

//

// listBox6

//

this->listBox6->FormattingEnabled = true;

this->listBox6->Location = System::Drawing::Point(831, 49);

this->listBox6->Name = L"listBox6";

this->listBox6->Size = System::Drawing::Size(64, 329);

this->listBox6->TabIndex = 101;

//

// label14

//

this->label14->AutoSize = true;

this->label14->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 9.75F, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(204)));

this->label14->Location = System::Drawing::Point(492, 30);

this->label14->Name = L"label14";

this->label14->Size = System::Drawing::Size(34, 16);

this->label14->TabIndex = 102;

this->label14->Text = L"X(1):";

//

// label15

//

this->label15->AutoSize = true;

this->label15->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 9.75F, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(204)));

this->label15->Location = System::Drawing::Point(562, 30);

this->label15->Name = L"label15";

this->label15->Size = System::Drawing::Size(35, 16);

this->label15->TabIndex = 103;

this->label15->Text = L"Y(1):";

//

// label16

//

this->label16->AutoSize = true;

this->label16->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 9.75F, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(204)));

this->label16->Location = System::Drawing::Point(705, 30);

this->label16->Name = L"label16";

this->label16->Size = System::Drawing::Size(35, 16);

this->label16->TabIndex = 105;

this->label16->Text = L"Y(2):";

//

// label17

//

this->label17->AutoSize = true;

this->label17->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 9.75F, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(204)));

this->label17->Location = System::Drawing::Point(635, 30);

this->label17->Name = L"label17";

this->label17->Size = System::Drawing::Size(34, 16);

this->label17->TabIndex = 104;

this->label17->Text = L"X(2):";

//

// label18

//

this->label18->AutoSize = true;

this->label18->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 9.75F, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(204)));

this->label18->Location = System::Drawing::Point(848, 30);

this->label18->Name = L"label18";

this->label18->Size = System::Drawing::Size(35, 16);

this->label18->TabIndex = 107;

this->label18->Text = L"Y(3):";

//

// label19

//

this->label19->AutoSize = true;

this->label19->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 9.75F, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(204)));

this->label19->Location = System::Drawing::Point(778, 30);

this->label19->Name = L"label19";

this->label19->Size = System::Drawing::Size(34, 16);

this->label19->TabIndex = 106;

this->label19->Text = L"X(3):";

//

// comboBox2

//

this->comboBox2->DisplayMember = L"asd";

this->comboBox2->FormattingEnabled = true;

this->comboBox2->Items->AddRange(gcnew cli::array< System::Object^ >(4) { L"Выключить", L"По оси Х", L"По оси У", L"По обоим осям" });

this->comboBox2->Location = System::Drawing::Point(749, 390);

this->comboBox2->Name = L"comboBox2";

this->comboBox2->Size = System::Drawing::Size(100, 21);

this->comboBox2->TabIndex = 108;

this->comboBox2->Text = L"Не указано";

this->comboBox2->SelectedIndexChanged += gcnew System::EventHandler(this, &MyForm::comboBox2\_SelectedIndexChanged);

//

// label20

//

this->label20->AutoSize = true;

this->label20->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 9.75F, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(204)));

this->label20->Location = System::Drawing::Point(688, 391);

this->label20->Name = L"label20";

this->label20->Size = System::Drawing::Size(55, 16);

this->label20->TabIndex = 109;

this->label20->Text = L"Курсор";

//

// checkBox1

//

this->checkBox1->AutoSize = true;

this->checkBox1->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 9.75F, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(204)));

this->checkBox1->Location = System::Drawing::Point(857, 391);

this->checkBox1->Name = L"checkBox1";

this->checkBox1->Size = System::Drawing::Size(62, 20);

this->checkBox1->TabIndex = 110;

this->checkBox1->Text = L"Zoom";

this->checkBox1->UseVisualStyleBackColor = true;

this->checkBox1->CheckedChanged += gcnew System::EventHandler(this, &MyForm::checkBox1\_CheckedChanged);

//

// MyForm

//

this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);

this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;

this->BackColor = System::Drawing::SystemColors::InactiveCaption;

this->ClientSize = System::Drawing::Size(931, 532);

this->ControlBox = false;

this->Controls->Add(this->checkBox1);

this->Controls->Add(this->label20);

this->Controls->Add(this->comboBox2);

this->Controls->Add(this->label18);

this->Controls->Add(this->label19);

this->Controls->Add(this->label16);

this->Controls->Add(this->label17);

this->Controls->Add(this->label15);

this->Controls->Add(this->label14);

this->Controls->Add(this->listBox6);

this->Controls->Add(this->listBox5);

this->Controls->Add(this->label13);

this->Controls->Add(this->textBox12);

this->Controls->Add(this->textBox11);

this->Controls->Add(this->textBox10);

this->Controls->Add(this->textBox9);

this->Controls->Add(this->label4);

this->Controls->Add(this->label5);

this->Controls->Add(this->label6);

this->Controls->Add(this->label7);

this->Controls->Add(this->comboBox1);

this->Controls->Add(this->textBox8);

this->Controls->Add(this->textBox7);

this->Controls->Add(this->textBox6);

this->Controls->Add(this->textBox5);

this->Controls->Add(this->label9);

this->Controls->Add(this->label10);

this->Controls->Add(this->label11);

this->Controls->Add(this->label12);

this->Controls->Add(this->label8);

this->Controls->Add(this->listBox3);

this->Controls->Add(this->listBox4);

this->Controls->Add(this->listBox2);

this->Controls->Add(this->listBox1);

this->Controls->Add(this->chart1);

this->Controls->Add(this->label3);

this->Controls->Add(this->label2);

this->Controls->Add(this->label1);

this->Controls->Add(this->textBox4);

this->Controls->Add(this->textBox3);

this->Controls->Add(this->textBox2);

this->Controls->Add(this->textBox1);

this->Controls->Add(this->menuStrip1);

this->MainMenuStrip = this->menuStrip1;

this->MaximizeBox = false;

this->MinimizeBox = false;

this->Name = L"MyForm";

this->ShowIcon = false;

this->Text = L"Esfunc";

this->menuStrip1->ResumeLayout(false);

this->menuStrip1->PerformLayout();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->chart1))->EndInit();

this->ResumeLayout(false);

this->PerformLayout();

}

void activefunction() // включает использующиеся элементы ввода-вывода

{

textBox5->Enabled = true; textBox6->Enabled = true; textBox7->Enabled = true; textBox8->Enabled = true;

textBox9->Enabled = true; textBox10->Enabled = true; textBox11->Enabled = true; textBox12->Enabled = true;

listBox3->Enabled = true; listBox4->Enabled = true;

listBox3->BackColor = Color::White; listBox4->BackColor = Color::White;

listBox5->Enabled = true; listBox6->Enabled = true;

listBox5->BackColor = Color::White; listBox6->BackColor = Color::White;

}

void deactivefunction() // выключает неиспользующиеся элементы ввода-вывода

{

if (comboBox1->SelectedIndex == 0)

{

textBox5->Enabled = false; textBox6->Enabled = false; textBox7->Enabled = false; textBox8->Enabled = false;

textBox9->Enabled = false; textBox10->Enabled = false; textBox11->Enabled = false; textBox12->Enabled = false;

listBox3->Enabled = false; listBox4->Enabled = false;

listBox5->Enabled = false; listBox6->Enabled = false;

listBox3->BackColor = Color::LightGray; listBox4->BackColor = Color::LightGray;

listBox5->BackColor = Color::LightGray; listBox6->BackColor = Color::LightGray;

}

else if (comboBox1->SelectedIndex == 1)

{

textBox9->Enabled = false; textBox10->Enabled = false; textBox11->Enabled = false; textBox12->Enabled = false;

listBox5->Enabled = false; listBox6->Enabled = false;

listBox5->BackColor = Color::LightGray;

listBox6->BackColor = Color::LightGray;

}

}

void use\_cursor() {

chart1->ChartAreas["Ch1"]->CursorX->Interval = 0.1; // интервал курсора по х

chart1->ChartAreas["Ch1"]->CursorY->Interval = 0.1; // интервал курсора по у

chart1->ChartAreas["Ch1"]->CursorX->LineColor = Color::DeepPink; // цвет курсора по Х

chart1->ChartAreas["Ch1"]->CursorY->LineColor = Color::DeepPink; // цвет курсора по У

if (comboBox2->SelectedIndex == 0) { // выключение курсора

chart1->ChartAreas["Ch1"]->CursorX->IsUserEnabled = false;

chart1->ChartAreas["Ch1"]->CursorY->IsUserEnabled = false;

chart1->ChartAreas["Ch1"]->CursorX->LineColor = Color::Transparent; // если курсор уже был использован, то отключить

chart1->ChartAreas["Ch1"]->CursorY->LineColor = Color::Transparent; // если курсор уже был использован, то отключить

}

else if (comboBox2->SelectedIndex == 1) { // включение курсора по оси x

chart1->ChartAreas["Ch1"]->CursorY->IsUserEnabled = false;

chart1->ChartAreas["Ch1"]->CursorY->LineColor = Color::Transparent; // если курсор уже был использован, то отключить

chart1->ChartAreas["Ch1"]->CursorX->IsUserEnabled = true;

}

else if (comboBox2->SelectedIndex == 2) { // включение курсора по оси y

chart1->ChartAreas["Ch1"]->CursorX->IsUserEnabled = false;

chart1->ChartAreas["Ch1"]->CursorX->LineColor = Color::Transparent; // если курсор уже был использован, то отключить

chart1->ChartAreas["Ch1"]->CursorY->IsUserEnabled = true;

}

else if (comboBox2->SelectedIndex == 3) { // включение курсора по осям х и y

chart1->ChartAreas["Ch1"]->CursorX->IsUserEnabled = false;

chart1->ChartAreas["Ch1"]->CursorY->IsUserEnabled = false;

chart1->ChartAreas["Ch1"]->CursorX->IsUserEnabled = true;

chart1->ChartAreas["Ch1"]->CursorY->IsUserEnabled = true;

}

}

#pragma endregion

private: System::Void comboBox1\_SelectedIndexChanged(System::Object^ sender, System::EventArgs^ e) {

activefunction(); // включает использующиеся элементы ввода-вывода

deactivefunction(); // выключает неиспользующиеся элементы ввода-вывода

}

private: System::Void comboBox2\_SelectedIndexChanged(System::Object^ sender, System::EventArgs^ e) {

use\_cursor();

}

private: System::Void checkBox1\_CheckedChanged(System::Object^ sender, System::EventArgs^ e) {

if (checkBox1->Checked)

chart1->ChartAreas["Ch1"]->CursorX->IsUserSelectionEnabled = true; // включить увеличение

else

chart1->ChartAreas["Ch1"]->CursorX->IsUserSelectionEnabled = false; // выключить увеличение

}

private: System::Void выполнитьToolStripMenuItem\_Click(System::Object^ sender, System::EventArgs^ e) {

//section 1:

if (comboBox1->SelectedIndex == 0 || comboBox1->SelectedIndex == 1 || comboBox1->SelectedIndex == 2)

{

tokens texpr1, pexpr1; // Токен это отдельная самостоятельная единица в выражении, например, константа, имя переменной или операция.

Variables expvars1; // Массив переменны

Massives expmasvars1; // Массив массивов под доп нужны (как я понимаю функции и т.д.)

String^ function1 = textBox1->Text; // Функция

if (correct\_function(function1, 1)) // проверка, корректна ли введена функция

goto link; // в случае, если введена некорректно

std::string expr1 = ""; // Формула, в формате СТД строки

for (int i = 0; i < function1->Length; i++)

expr1 += (char)function1[i];

double start\_digit1, end\_digit1, step1; // интервал, шаг

String^ some\_start\_digit1 = textBox2->Text; // начало интервала

String^ some\_end\_digit1 = textBox3->Text; // конец интервала

String^ some\_step1 = textBox4->Text; // указанный шаг (если таковой имеется)

if (user\_data(start\_digit1, end\_digit1, step1, some\_start\_digit1, some\_end\_digit1, some\_step1)) //проверка, корректно ли введены доп. условия

goto link; // в случае, если введена некорректно

std::string var\_dig1 = "x="; // переменное значение x

String^ p1 = ""; // вспомогательная переменная для присваивания значения x

double result1 = 0;

/////

chart1->Series["Function(1)"]->Points->Clear(); // Очистка графика 1

for (double i = start\_digit1; i < end\_digit1 + step1; i += step1)

{

p1 = Convert::ToString(i); // передача х значения

for (int i = 0; i < p1->Length; i++)

{

var\_dig1 += (char)p1[i];

}

std::string::iterator it1 = find(begin(var\_dig1), end(var\_dig1), ','); // замена "," на "." для корректной работы

if (it1 != end(var\_dig1))

{

\*it1 = '.';

}

ReadExpressionFromStream(expr1, var\_dig1, expvars1, expmasvars1); // считываем функцию

CreateTokensFromExpression(expr1, texpr1); // разбиваем на токены

CreatePostfixFromTokens(texpr1, pexpr1); // преобразовываем в постфиксную запись

result1 = ResultExpr(pexpr1, expvars1, expmasvars1); // получем значением в заданной точке

listBox1->Items->Add(i); // выводим точку

listBox2->Items->Add(result1); // выводим значение функции

chart1->Series["Function(1)"]->Points->AddXY(i, result1); // Строим график

clear\_el(expvars1, expmasvars1, texpr1, pexpr1); // очистка элементов парсера

clear\_str(var\_dig1, p1); // очистка вспомогательных строк

}

}

else

{

MessageBox::Show("Пожалуйста, уточните количество используемых функций", "Внимание!");

link: // В разных ситуациях приходится выходить из события, чтобы сто раз не писать сделал через ссылку на место выхода

return; // выход из события

}

//section 2:

if (comboBox1->SelectedIndex == 1 || comboBox1->SelectedIndex == 2)

{

tokens texpr2, pexpr2; // Токен это отдельная самостоятельная единица в выражении, например, константа, имя переменной или операция.

Variables expvars2; // Массив переменны

Massives expmasvars2; // Массив массивов под доп нужны (как я понимаю функции и т.д.)

String^ function2 = textBox5->Text; // Функция

if (correct\_function(function2, 2)) // проверка, корректна ли введена функция

goto link; // в случае, если введена некорректно

std::string expr2 = ""; // Формула, в формате СТД строки

for (int i = 0; i < function2->Length; i++)

expr2 += (char)function2[i];

double start\_digit2, end\_digit2, step2; // интервал шаг

String^ some\_start\_digit2 = textBox6->Text; // начало интервала

String^ some\_end\_digit2 = textBox7->Text; // конец интервала

String^ some\_step2 = textBox8->Text; // указанный шаг (если таковой имеется)

if (user\_data(start\_digit2, end\_digit2, step2, some\_start\_digit2, some\_end\_digit2, some\_step2)) // проверка, корректно ли введены доп. условия

goto link; // в случае, если введена некорректно

std::string var\_dig2 = "y="; // переменное значение y

String^ p2 = ""; // вспомогательная переменная для присваивания значения y

double result2 = 0;

////////

chart1->Series["Function(2)"]->Points->Clear(); // Очистка графика 2

for (double j = start\_digit2; j < end\_digit2 + step2; j += step2)

{

p2 = Convert::ToString(j); // передача y значения

for (int i = 0; i < p2->Length; i++)

{

var\_dig2 += (char)p2[i];

}

std::string::iterator it2 = find(begin(var\_dig2), end(var\_dig2), ','); // замена "," на "." для корректной работы

if (it2 != end(var\_dig2))

{

\*it2 = '.';

}

ReadExpressionFromStream(expr2, var\_dig2, expvars2, expmasvars2); // считываем функцию

CreateTokensFromExpression(expr2, texpr2); // разбиваем на токены

CreatePostfixFromTokens(texpr2, pexpr2); // преобразовываем в постфиксную запись

result2 = ResultExpr(pexpr2, expvars2, expmasvars2); // получем значением в заданной точке

listBox3->Items->Add(j); // выводим точку

listBox4->Items->Add(result2); // выводим значение функции

chart1->Series["Function(2)"]->Points->AddXY(j, result2); // Строим график

clear\_el(expvars2, expmasvars2, texpr2, pexpr2); // очистка элементов парсера

clear\_str(var\_dig2, p2); // очистка вспомогательных строк

}

}

//section 3

if (comboBox1->SelectedIndex == 2)

{

tokens texpr3, pexpr3; // Токен это отдельная самостоятельная единица в выражении, например, константа, имя переменной или операция.

Variables expvars3; // Массив переменны

Massives expmasvars3; // Массив массивов под доп нужны (как я понимаю функции и т.д.)

String^ function3 = textBox9->Text; // Функция

if (correct\_function(function3, 3)) // проверка, корректна ли введена функция

goto link; // в случае, если введена некорректно

std::string expr3 = ""; // Формула, в формате СТД строки

for (int i = 0; i < function3->Length; i++)

expr3 += (char)function3[i];

double start\_digit3, end\_digit3, step3; // интервал шаг

String^ some\_start\_digit3 = textBox10->Text; // начало интервала

String^ some\_end\_digit3 = textBox11->Text; // конец интервала

String^ some\_step3 = textBox12->Text; // указанный шаг (если таковой имеется)

if (user\_data(start\_digit3, end\_digit3, step3, some\_start\_digit3, some\_end\_digit3, some\_step3)) // проверка, корректно ли введены доп. условия

goto link; // в случае, если введена некорректно

std::string var\_dig3 = "z="; // переменное значение z

String^ p3 = ""; // вспомогательная переменная для присваивания значения z

double result3 = 0;

////////

chart1->Series["Function(3)"]->Points->Clear(); // Очистка графика 3

for (double l = start\_digit3; l < end\_digit3 + step3; l += step3)

{

p3 = Convert::ToString(l); // передача z значения

for (int i = 0; i < p3->Length; i++)

{

var\_dig3 += (char)p3[i];

}

std::string::iterator it3 = find(begin(var\_dig3), end(var\_dig3), ','); // замена "," на "." для корректной работы

if (it3 != end(var\_dig3))

{

\*it3 = '.';

}

ReadExpressionFromStream(expr3, var\_dig3, expvars3, expmasvars3); // считываем функцию

CreateTokensFromExpression(expr3, texpr3); // разбиваем на токены

CreatePostfixFromTokens(texpr3, pexpr3); // преобразовываем в постфиксную запись

result3 = ResultExpr(pexpr3, expvars3, expmasvars3); // получем значением в заданной точке

listBox5->Items->Add(l); // выводим точку

listBox6->Items->Add(result3); // выводим значение функции

chart1->Series["Function(3)"]->Points->AddXY(l, result3); // Строим график

clear\_el(expvars3, expmasvars3, texpr3, pexpr3); // очистка элементов парсера

clear\_str(var\_dig3, p3); // очистка вспомогательных строк

}

}

}

private: System::Void помощьToolStripMenuItem\_Click(System::Object^ sender, System::EventArgs^ e) { // Информация

MessageBox::Show

("1. Записывать функции (1),(2),(3) соответственно через \"x\",\"y\",\"z\".\n"+

"2. Перед работой предварительно выбрать кол-во функций для использования.\n"+

"3. Пример ввода: x^2-3\*sin(x+15)+9.\n" +

"4. Доступные тригометрические функции sin, cos, tan, asin, acos, atan.\n"+

"5. Для включение курсора выбрать нужные оси. \n"+

"6. При надобности есть возможность увеличения графика, после активации \"Zoom\". Для максимального увеличения подключить курсор.\n", "Информация по использованию");

}

private: System::Void очиститьToolStripMenuItem\_Click(System::Object^ sender, System::EventArgs^ e) { // Очистка всего

textBox1->Text = ""; textBox2->Text = ""; textBox3->Text = ""; textBox4->Text = ""; // очистка текстбоксов

textBox5->Text = ""; textBox6->Text = ""; textBox7->Text = ""; textBox8->Text = ""; // очистка текстбоксов

textBox9->Text = ""; textBox10->Text = ""; textBox11->Text = ""; textBox12->Text = ""; // очистка текстбоксов

listBox1->Items->Clear(); listBox2->Items->Clear(); // очистка листбоксов

listBox3->Items->Clear(); listBox4->Items->Clear(); // очистка листбоксов

listBox5->Items->Clear(); listBox6->Items->Clear(); // очистка листбоксов

chart1->Series["Function(1)"]->Points->Clear(); // Очистка графика

chart1->Series["Function(2)"]->Points->Clear(); // Очистка графика

chart1->Series["Function(3)"]->Points->Clear(); // Очистка графика

comboBox1->SelectedIndex = -1; // Очистка комбобокса

}

private: System::Void завершитьToolStripMenuItem\_Click(System::Object^ sender, System::EventArgs^ e) { // Выход

this->Close();

}

};

}

**Parser.h:**  
#pragma once

#include <iostream>

#include <string>

#include <vector>

#include <map>

#include <stack>

#include <set>

#include <iterator>

//Массив переменных

typedef std::map<std::string, double> Variables;

//Массив "массивов"

typedef std::map<std::string, std::vector<double> > Massives;

//Функция считывает выражение в строку "expr" и ищем переменные или массивы

void ReadExpressionFromStream(std::string expr, std::string utoch, Variables& var, Massives& mas) {

std::string temp;

int pos;

//Если встретили '=', то это переменная, заносим ее имя и значение в массив

temp = utoch;

pos = temp.find('=');

if (pos > 0) {

std::string name = temp.substr(0, pos);

double value = atof(temp.substr(pos + 1).c\_str());

var[name] = value;

}

//Если нашли '{', это массив, заносим имя массива и значения в соответствующие массивы

pos = temp.find('{');

if (pos > 0) {

std::string name = temp.substr(0, pos);

//Ищем значения массива и запоминаем их

int pos1 = pos, pos2;

do {

pos2 = temp.find(',');

double value = atof(temp.substr(pos1 + 1, pos2).c\_str());

mas[name].push\_back(value);

if (pos2 == -1)

break;

temp[pos2] = ' ';

pos1 = pos2;

} while (pos2 > 0);

}

}

// Типы токенов

enum tokentype {

//Переменная, константа, (, ), функция, операция, массив, {, }

var, num, op\_br, cl\_br, func, op, mas, op\_sbr, cl\_sbr

};

// Структура токена

struct token {

std::string name;

tokentype type;

//Конструкторы

token(std::string str, tokentype typ) {

name = str;

type = typ;

}

token() {}

};

//Массив токенов

typedef std::vector<token> tokens;

//Множество разделителей

std::set<char> DelimSet;

//Разделители

const std::string delimiters = " ()+/\*-^&|!%[]";

//Инициализирует множество разделителей

void CreateSetOfDelimiters() {

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

DelimSet.insert(delimiters[i]);

return;

}

//Проверка, является ли символ разделителем

bool IsDelimiter(char sym) {

return DelimSet.count(sym) > 0;

}

//Разбиваем выражение на токены

void CreateTokensFromExpression(std::string& expr, tokens& texpr) {

CreateSetOfDelimiters();

std::string ex = expr + " ";

std::string name;

//Получаем имя токена

int i = 0;

while (i < ex.size() - 1) {

name = "";

//Если текущий символ разделитель

if (IsDelimiter(ex[i])) {

if (ex[i] == ' ') { //Пробел просто перепрыгиваем

i++;

continue;

}

name = ex[i]; //Любой другой добавляем в имя токена

i++;

}

else {

while (!IsDelimiter(ex[i])) {

/\*Если не разделитель например, переменная или имя массива,

Считываем его полностью \*/

name += ex[i];

i++;

}

}

//Заносим получившийся токен в список токенов

texpr.push\_back(token(name, var));

}

//Раздаем получившимся токенам типы

for (int j = 0; j < texpr.size(); j++) {

if (texpr[j].name[0] == '[') {

texpr[j].type = op\_sbr;

continue;

}

if (texpr[j].name[0] == ']') {

texpr[j].type = cl\_sbr;

continue;

}

if (texpr[j].name[0] == '(') {

texpr[j].type = op\_br;

continue;

}

if (texpr[j].name[0] == ')') {

texpr[j].type = cl\_br;

continue;

}

if (isdigit(texpr[j].name[0])) {

texpr[j].type = num;

continue;

}

//mas

if (isalpha(texpr[j].name[0])) {

if (j < texpr.size() - 1 && texpr[j + 1].name[0] == '[')

texpr[j].type = mas;

//continue;

}

if (isalpha(texpr[j].name[0])) {

if (j < texpr.size() - 1 && texpr[j + 1].name[0] == '(')

texpr[j].type = func;

continue;

}

texpr[j].type = op;

}

//Проверяем минус и !, что это префиксные операции

for (int j = 0; j < texpr.size(); j++) {

if (texpr[j].name == "-" && (j == 0 || texpr[j - 1].type == op\_br))

texpr[j].name = "opposite";

if (texpr[j].name == "!" && (j == texpr.size() - 1 || texpr[j + 1].type == cl\_br || texpr[j + 1].type == op))

texpr[j].name = "factorial";

}

return;

}

//Приоритеты операций

std::map <std::string, int> prior;

//Функция выставляет приоритеты операций

void CreatePrior() {

prior["+"] = 10;

prior["-"] = 10;

prior["\*"] = 20;

prior["/"] = 20;

prior["^"] = 30;

prior["opposite"] = 10;

prior["factorial"] = 30;

prior["%"] = 20;

prior["&"] = 5;

prior["|"] = 5;

prior["!"] = 40;

}

//Переводим выражение в постфиксную запись

void CreatePostfixFromTokens(tokens& texpr, tokens& pexpr) {

//Задаем приоритеты операций

CreatePrior();

std::stack <token> TStack;

//Ловим токены и работаем по алгоритму

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

switch (texpr[i].type) {

case var:

case num:

pexpr.push\_back(texpr[i]);

break;

case op\_br:

TStack.push(texpr[i]);

break;

case cl\_br:

while (TStack.top().type != op\_br) {

pexpr.push\_back(TStack.top());

TStack.pop();

}

TStack.pop();

break;

case op\_sbr:

TStack.push(texpr[i]);

break;

case cl\_sbr:

while (TStack.top().type != op\_sbr) {

pexpr.push\_back(TStack.top());

TStack.pop();

}

TStack.pop();

break;

case op:

if (TStack.size()) {

while (TStack.size() && ((TStack.top().type == op && prior[texpr[i].name] <= prior[TStack.top().name]) ||

TStack.top().type == func ||

TStack.top().type == mas)) {

pexpr.push\_back(TStack.top());

TStack.pop();

}

}

TStack.push(texpr[i]);

break;

case mas:

while (TStack.size() && TStack.top().type == mas) {

pexpr.push\_back(TStack.top());

TStack.pop();

}

TStack.push(texpr[i]);

break;

case func:

while (TStack.size() && TStack.top().type == func) {

pexpr.push\_back(TStack.top());

TStack.pop();

}

TStack.push(texpr[i]);

break;

}

}

while (TStack.size()) {

pexpr.push\_back(TStack.top());

TStack.pop();

}

return;

}

//Реализация доступных операций

double fact(double n) {

if (n == 0)

return 1;

return n \* fact(n - 1);

}

double op\_plus(std::stack <double>& s) {

double a, b;

a = s.top();

s.pop();

b = s.top();

s.pop();

return a + b;

}

double op\_minus(std::stack <double>& s) {

double a, b;

a = s.top();

s.pop();

b = s.top();

s.pop();

return b - a;

}

double op\_mul(std::stack <double>& s) {

double a, b;

a = s.top();

s.pop();

b = s.top();

s.pop();

return a \* b;

}

double op\_div(std::stack <double>& s) {

double a, b;

a = s.top();

s.pop();

b = s.top();

s.pop();

return b / a;

}

double op\_deg(std::stack <double>& s) {

double a, b;

a = s.top();

s.pop();

b = s.top();

s.pop();

//b^a!!

return pow(b, a);

}

double op\_opposite(std::stack <double>& s) {

double a;

a = s.top();

s.pop();

return -a;

}

double op\_factorial(std::stack <double>& s) {

double a;

a = s.top();

s.pop();

return fact(a);

}

double op\_odiv(std::stack <double>& s) {

long long a, b;

a = s.top();

s.pop();

b = s.top();

s.pop();

return b % a;

}

double op\_and(std::stack <double>& s) {

double a, b;

a = s.top();

s.pop();

b = s.top();

s.pop();

return a && b;

}

double op\_or(std::stack <double>& s) {

double a, b;

a = s.top();

s.pop();

b = s.top();

s.pop();

return a || b;

}

double op\_not(std::stack <double>& s) {

double a;

a = s.top();

s.pop();

return !a;

}

double op\_sin(std::stack <double>& s) {

double a;

a = s.top();

s.pop();

return sin(a);

}

double op\_cos(std::stack <double>& s) {

double a;

a = s.top();

s.pop();

return cos(a);

}

double op\_tan(std::stack <double>& s) {

double a;

a = s.top();

s.pop();

return tan(a);

}

double op\_asin(std::stack <double>& s) {

double a;

a = s.top();

s.pop();

return asin(a);

}

double op\_acos(std::stack <double>& s) {

double a;

a = s.top();

s.pop();

return acos(a);

}

double op\_atan(std::stack <double>& s) {

double a;

a = s.top();

s.pop();

return atan(a);

}

//Указатель на функцию(для операций)

typedef double(\*func\_type)(std::stack<double>&);

//Массив операций

typedef std::map<std::string, func\_type> Ops;

Ops ops;

//Инициализация массива операций

void CreateOps() {

ops["+"] = op\_plus;

ops["-"] = op\_minus;

ops["\*"] = op\_mul;

ops["/"] = op\_div;

ops["^"] = op\_deg;

ops["opposite"] = op\_opposite;

ops["factorial"] = op\_factorial;

ops["%"] = op\_odiv;

ops["&"] = op\_and;

ops["|"] = op\_or;

ops["!"] = op\_not;

ops["sin"] = op\_sin;

ops["cos"] = op\_cos;

ops["tan"] = op\_tan;

ops["asin"] = op\_asin;

ops["acos"] = op\_acos;

ops["atan"] = op\_atan;

return;

}

//Считаем результат выражения

double ResultExpr(tokens& pexpr, Variables& expvars, Massives& varmas) {

CreateOps();

std::stack <double> s;

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

switch (pexpr[i].type) {

case num: {

s.push(atoi(pexpr[i].name.c\_str()));

}

break;

case var: {

Variables::iterator Vit;

for (Vit = expvars.begin(); Vit != expvars.end(); Vit++) {

if (Vit->first == pexpr[i].name) {

s.push(Vit->second);

break;

}

}

}

break;

case func:

case op: {

Ops::iterator Oit;

for (Oit = ops.begin(); Oit != ops.end(); Oit++) {

if (Oit->first == pexpr[i].name) {

s.push(Oit->second(s));

break;

}

}

}

break;

case mas: {

int index = s.top();

s.pop();

Massives::iterator it;

for (it = varmas.begin(); it != varmas.end(); it++) {

if (it->first == pexpr[i].name)

s.push(it->second[index]);

}

}

}

}

return s.top();

}

using namespace System;

void clear\_el(Variables& expvars, Massives& expmasvars, tokens& texpr, tokens& pexpr) // очищаем все данные, для при новом рассчёте значения

{

expvars.clear();

expmasvars.clear();

texpr.clear();

pexpr.clear();

};

**dop\_func.h:**

#pragma once

using namespace System;

using namespace System::ComponentModel;

using namespace System::Collections;

using namespace System::Windows::Forms;

using namespace System::Data;

using namespace System::Drawing;

// обработка пользовательских данных(Начало-конец интервала, шаг)

bool user\_data(double& start\_digit, double& end\_digit, double& step, String^& some\_start\_digit, String^& some\_end\_digit, String^& some\_step)

{

start\_digit = -5;

if (some\_start\_digit != "")

{

for (int i = 0; i < some\_start\_digit->Length; i++)

{

if (isalpha(some\_start\_digit[i]))

{

MessageBox::Show("Некорректно указано начало интервала", "Ошибка");

return 1;

}

}

start\_digit = Convert::ToDouble(some\_start\_digit);

}

end\_digit = 5;

if (some\_end\_digit != "")

{

for (int i = 0; i < some\_end\_digit->Length; i++)

{

if (isalpha(some\_end\_digit[i]))

{

MessageBox::Show("Некорректно указан конец интервала", "Ошибка");

return 1;

}

}

end\_digit = Convert::ToDouble(some\_end\_digit);

}

step = 1;

if (some\_step != "")

{

for (int i = 0; i < some\_step->Length; i++)

{

if (isalpha(some\_step[i]))

{

MessageBox::Show("Некорректно указан шаг", "Ошибка");

return 1;

}

}

step = Convert::ToDouble(some\_step);

}

}

bool correct\_function(String^ function, int f) // Корректно ли введена функция

{

int l1 = 0, l2=0, r1 = 0, r2=0;

bool arg = 0;

if (function == "") // если функция пустая

{

MessageBox::Show("Функция не задана. Задайте функцию", "Ошибка в функции");

return 1;

}

for (int i = 0; i < function->Length; i++) { // подсчёт открывающих и закрывающих скобок

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

++l1;

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

++l2;

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

++r1;

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

++r2;

}

if (l1 != r1 || l2 != r2) { // если количество открывающих скобок не равно количеству закрывающих

MessageBox::Show("Количество открывающихся скобок не равно количеству закрывающихся", "Ошибка в функции");

return 1;

}

for (int i = 0; i < function->Length; i++) // проверка все символов функции

{

switch (function[i])

{

case 's': // проверка ввода sin

if (function[i + 1] != 'i' || function[i + 2] != 'n' || function[i+3] != '(')

{

MessageBox::Show("Использован неизвестный символ или не указан аргумент в функции №" + char(f), "Ошибка в функции");

return 1;

}

for (int i = 3; i < function->Length; i++) { // проверка аргумента для функции

if (function[i] == 'x' || function[i] == 'y' || function[i] == 'z' || isdigit(function[i]))

arg = 1;

}

if (!arg){

MessageBox::Show("Использован неизвестный символ или не указан аргумент в функции №" + char(f), "Ошибка в функции");

return 1;

}

arg = 0;

i += 3;

continue;

break;

case 'c': // проверка ввода cos

if ((function[i + 1] != 'o' || function[i + 2] != 's' || function[i + 3] != '('))

{

MessageBox::Show("Использован неизвестный символ в функции №" + char(f), "Ошибка в функции");

return 1;

}

for (int i = 3; i < function->Length; i++) { // проверка аргумента для функции

if (function[i] == 'x' || function[i] == 'y' || function[i] == 'z' || isdigit(function[i]))

arg = 1;

}

if (!arg) {

MessageBox::Show("Использован неизвестный символ или не указан аргумент в функции №" + char(f), "Ошибка в функции");

return 1;

}

arg = 0;

i += 3;

continue;

break;

case 't': // проверка ввода tan

if ((function[i + 1] != 'a' && function[i + 2] != 'n' && function[i + 3] != '('))

{

MessageBox::Show("Использован неизвестный символ в функции №" + char(f), "Ошибка в функции");

return 1;

}

for (int i = 3; i < function->Length; i++) { // проверка аргумента для функции

if (function[i] == 'x' || function[i] == 'y' || function[i] == 'z' || isdigit(function[i]))

arg = 1;

}

if (!arg) {

MessageBox::Show("Использован неизвестный символ или не указан аргумент в функции №" + char(f), "Ошибка в функции");

return 1;

}

arg = 0;

i += 3;

continue;

break;

case 'a':

switch (function[i+1])

{

case 's': // проверка ввода asin

if ((function[i + 2] != 'i' && function[i + 3] != 'n' && function[i + 4] != '('))

{

MessageBox::Show("Использован неизвестный символ в функции №" + char(f), "Ошибка в функции");

return 1;

}

for (int i = 4; i < function->Length; i++) { // проверка аргумента для функции

if (function[i] == 'x' || function[i] == 'y' || function[i] == 'z' || isdigit(function[i]))

arg = 1;

}

if (!arg) {

MessageBox::Show("Использован неизвестный символ или не указан аргумент в функции №" + char(f), "Ошибка в функции");

return 1;

}

arg = 0;

i += 4;

continue;

break;

case 'c': // проверка ввода acos

if ((function[i + 2] != 'o' && function[i + 3] != 's' && function[i + 4] != '('))

{

MessageBox::Show("Использован неизвестный символ в функции №" + char(f), "Ошибка в функции");

return 1;

}

for (int i = 4; i < function->Length; i++) { // проверка аргумента для функции

if (function[i] == 'x' || function[i] == 'y' || function[i] == 'z' || isdigit(function[i]))

arg = 1;

}

if (!arg) {

MessageBox::Show("Использован неизвестный символ или не указан аргумент в функции №" + char(f), "Ошибка в функции");

return 1;

}

arg = 0;

i += 4;

continue;

break;

case 't': // проверка ввода atan

if ((function[i + 2] != 'a' && function[i + 3] != 'n' && function[i + 4] != '('))

{

MessageBox::Show("Использован неизвестный символ в функции №" + char(f), "Ошибка в функции");

return 1;

}

for (int i = 4; i < function->Length; i++) { // проверка аргумента для функции

if (function[i] == 'x' || function[i] == 'y' || function[i] == 'z' || isdigit(function[i]))

arg = 1;

}

if (!arg) {

MessageBox::Show("Использован неизвестный символ или не указан аргумент в функции №" + char(f), "Ошибка в функции");

return 1;

}

arg = 0;

i += 4;

continue;

break;

}

}

switch (f)

{

case 1:

if (!isdigit(function[i]) &&

function[i] != 'x' &&

function[i] != '+' && function[i] != '-' && function[i] != '\*' && function[i] != '/' && function[i] != '^' &&

function[i] != '(' && function[i] != ')' && function[i] != '[' && function[i] != ']' && function[i] != ',' && function[i] != '.' && function[i] != ' ') {

MessageBox::Show("Использован неизвестный символ в функции №" + char(f), "Ошибка в функции");

return 1;

}

break;

case 2:

if (!isdigit(function[i]) &&

function[i] != 'y' &&

function[i] != '+' && function[i] != '-' && function[i] != '\*' && function[i] != '/' && function[i] != '^' &&

function[i] != '(' && function[i] != ')' && function[i] != '[' && function[i] != ']' && function[i] != ',' && function[i] != '.' && function[i] != ' ') {

MessageBox::Show("Использован неизвестный символ в функции №" + char(f), "Ошибка в функции");

return 1;

}

break;

case 3:

if (!isdigit(function[i]) &&

function[i] != 'z' &&

function[i] != '+' && function[i] != '-' && function[i] != '\*' && function[i] != '/' && function[i] != '^' &&

function[i] != '(' && function[i] != ')' && function[i] != '[' && function[i] != ']' && function[i] != ',' && function[i] != '.' && function[i] != ' ') {

MessageBox::Show("Использован неизвестный символ в функции №" + char(f), "Ошибка в функции");

return 1;

}

}

}

return 0;

}

void clear\_str(std::string& var\_dig, String^& p) // очищаем строки при новом расчёте значения (для 1-ой функции)

{

if (var\_dig[0] == 'x')

var\_dig = "x=";

else if (var\_dig[0] == 'y')

var\_dig = "y=";

else if (var\_dig[0] == 'z')

var\_dig = "z=";

p = "";

}

