МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное

образовательное учреждение высшего образования

«ЧЕРЕПОВЕЦКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Институт информационных технологий\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

наименование института (факультета)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_МПО ЭВМ\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

наименование кафедры \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Теория автоматов и формальных языков\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ наименование дисциплины в соответствии с учебным планом

ЛАБОРАТОРНАЯ РАБОТА № 3-4

Исполнитель

студент 1ПИб-02-2оп-22

группа

\_\_Зернов В. З.

Фамилия, имя, отчество

Руководитель \_Ганичева О.Г.\_

Ф.И.О. преподавателя

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Подпись \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_2024\_ год

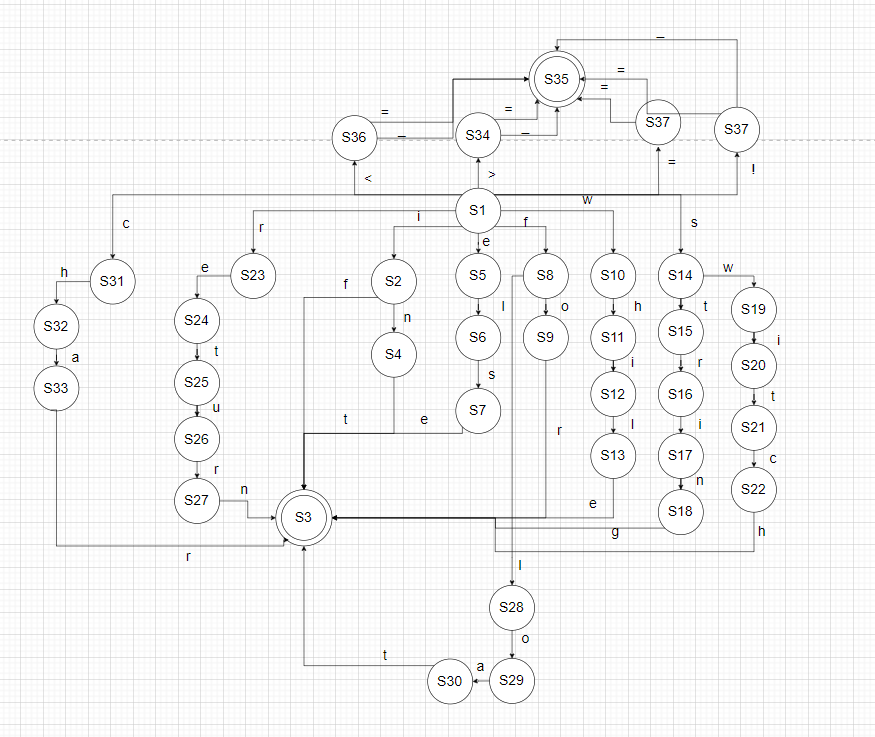
Задание

1. Написать программу для распознавания   слов языка С++  по своему варианту задания (ключевые слова, знаки отношений).

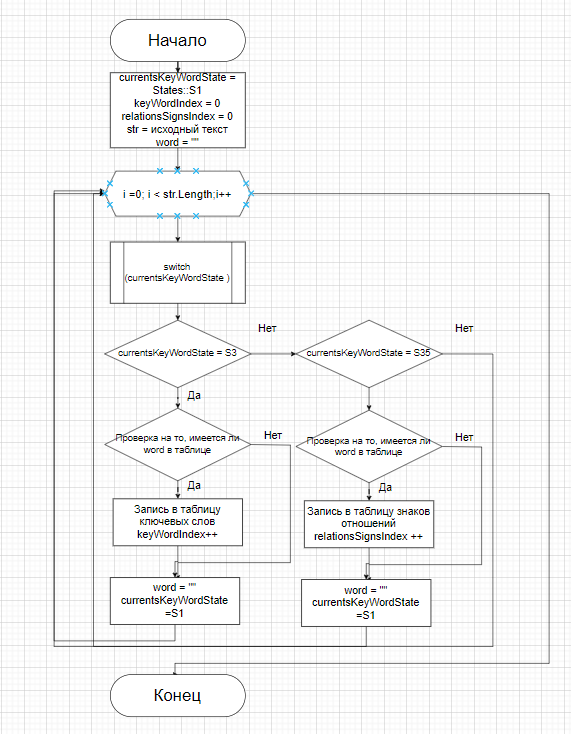
2. Составить блок-схему алгоритма решения задачи.

3. Оформить отчет (задание, блок-схема алгоритма, код программы с комментариями, результаты тестирования программы)

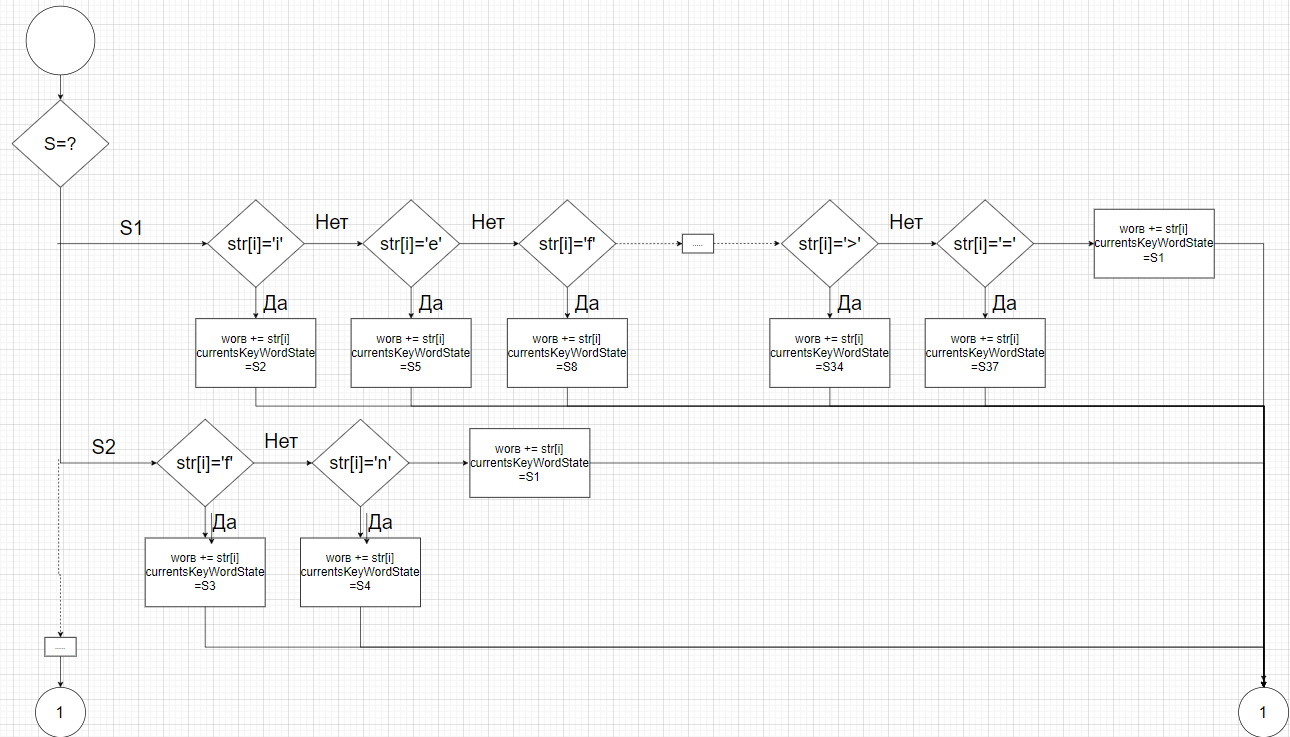
Граф конечного автомата



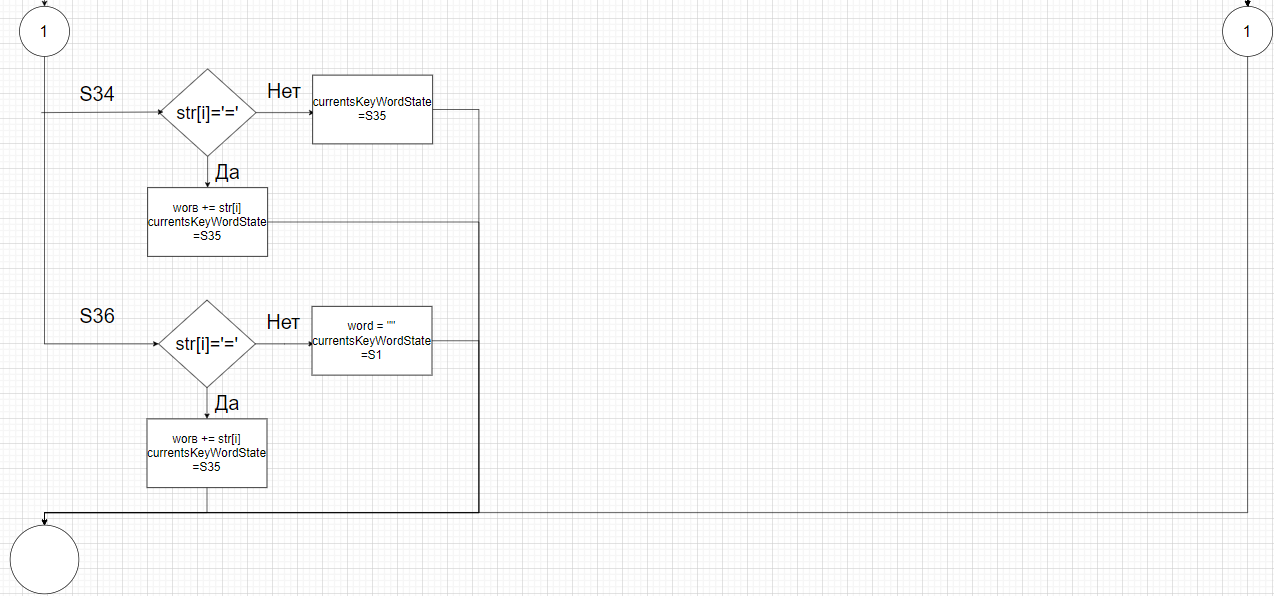
Блок-схема алгоритма



Общая схема блока switch

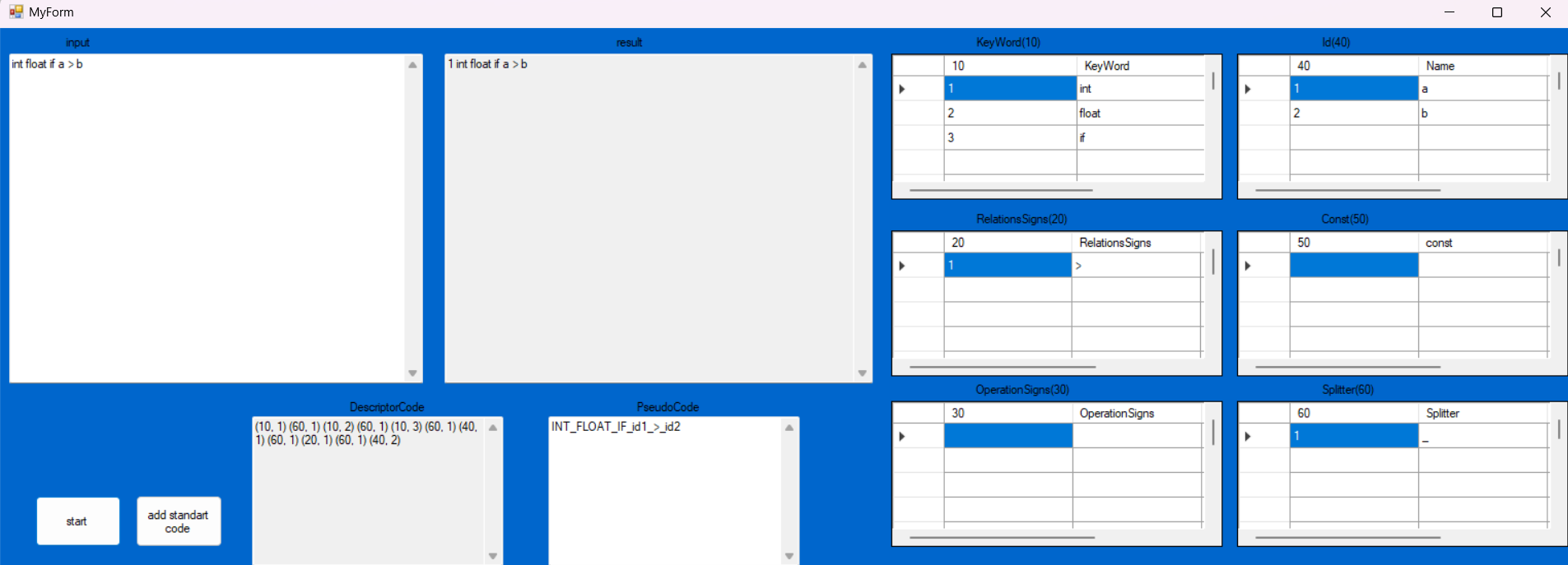


Продолжение схемы блока switch

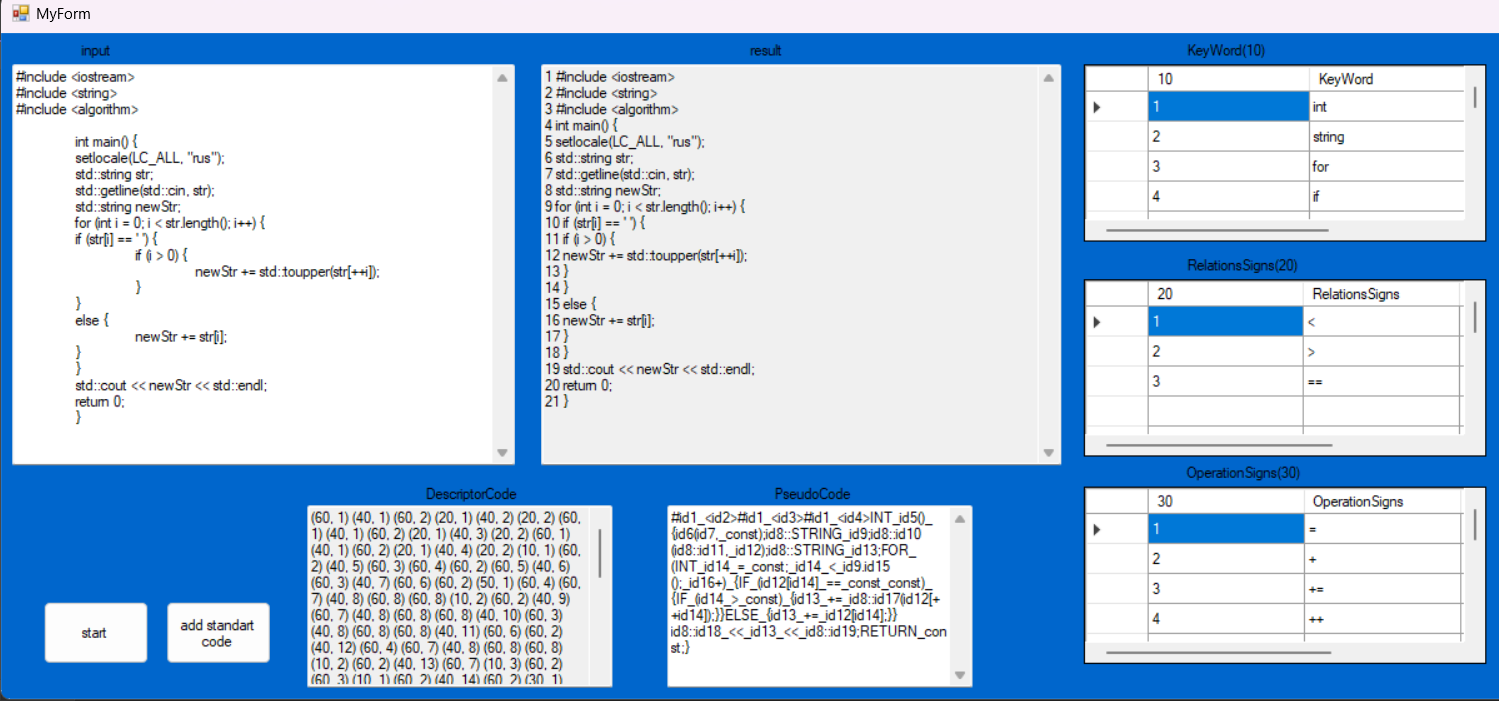


Тестирование

Случайная комбинация



Тестовый код



Текст программы:

Примечание:

Код имеет уже реализацию определения знаков операций, идентификаторов, констант и разделителей.

MyForm.cpp

#include "MyForm.h"

#include <iostream>

using namespace System;

using namespace System::Windows::Forms;

[STAThreadAttribute]

int main(array<String^>^ args)

{

Application::SetCompatibleTextRenderingDefault(false);

Application::EnableVisualStyles();

Lexer::MyForm MyForm;

Application::Run(% MyForm);

}

MyForm.h

#pragma once

#include "LexerStates.cpp"

#include "States.cpp"

#include <set>

namespace Lexer {

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>

/// ������ ��� MyForm

/// </summary>

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

{

public:

MyForm(void)

{

InitializeComponent();

//

//TODO: �������� ��� ������������

//

}

protected:

/// <summary>

/// ���������� ��� ������������ �������.

/// </summary>

~MyForm()

{

if (components)

{

delete components;

}

}

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

protected:

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

private: System::Windows::Forms::Button^ startProcessing;

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

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

private: System::Windows::Forms::DataGridView^ keyWordsTable;

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

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

private: System::Windows::Forms::DataGridView^ relationsSignsTable;

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

private: System::Windows::Forms::DataGridView^ operationSignsTable;

private: System::Windows::Forms::Button^ standartCode;

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

private: System::Windows::Forms::DataGridView^ splitterTable;

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

private: System::Windows::Forms::DataGridView^ constTable;

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

private: System::Windows::Forms::DataGridView^ idTable;

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

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

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

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

private: System::Windows::Forms::DataGridViewTextBoxColumn^ keyWord10;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ KeyWord;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ Pseudocode;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ relSigns20;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ RelationsSigns;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ dataGridViewTextBoxColumn3;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ oper30;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ OperationSigns;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ dataGridViewTextBoxColumn5;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ split60;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ Splitter;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ dataGridViewTextBoxColumn7;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ const50;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ Const;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ dataGridViewTextBoxColumn10;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ id40;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ Name;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ dataGridViewTextBoxColumn13;

private:

/// <summary>

/// ������������ ���������� ������������.

/// </summary>

System::ComponentModel::Container^ components;

#pragma region Windows Form Designer generated code

/// <summary>

/// ��������� ����� ��� ��������� ������������ � �� ���������

/// ���������� ����� ������ � ������� ��������� ����.

/// </summary>

void InitializeComponent(void)

{

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

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

this->startProcessing = (gcnew System::Windows::Forms::Button());

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

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

this->keyWordsTable = (gcnew System::Windows::Forms::DataGridView());

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

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

this->relationsSignsTable = (gcnew System::Windows::Forms::DataGridView());

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

this->operationSignsTable = (gcnew System::Windows::Forms::DataGridView());

this->standartCode = (gcnew System::Windows::Forms::Button());

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

this->splitterTable = (gcnew System::Windows::Forms::DataGridView());

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

this->constTable = (gcnew System::Windows::Forms::DataGridView());

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

this->idTable = (gcnew System::Windows::Forms::DataGridView());

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

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

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

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

this->relSigns20 = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->RelationsSigns = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->dataGridViewTextBoxColumn3 = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->keyWord10 = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->KeyWord = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->Pseudocode = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->id40 = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->Name = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->dataGridViewTextBoxColumn13 = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->const50 = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->Const = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->dataGridViewTextBoxColumn10 = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->oper30 = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->OperationSigns = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->dataGridViewTextBoxColumn5 = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->split60 = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->Splitter = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->dataGridViewTextBoxColumn7 = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

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

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

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

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

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

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

this->SuspendLayout();

//

// input

//

this->input->Location = System::Drawing::Point(12, 31);

this->input->Margin = System::Windows::Forms::Padding(3, 2, 3, 2);

this->input->Multiline = true;

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

this->input->ScrollBars = System::Windows::Forms::ScrollBars::Both;

this->input->Size = System::Drawing::Size(535, 394);

this->input->TabIndex = 0;

//

// result

//

this->result->Location = System::Drawing::Point(576, 31);

this->result->Margin = System::Windows::Forms::Padding(3, 2, 3, 2);

this->result->Multiline = true;

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

this->result->ReadOnly = true;

this->result->ScrollBars = System::Windows::Forms::ScrollBars::Both;

this->result->Size = System::Drawing::Size(553, 394);

this->result->TabIndex = 1;

//

// startProcessing

//

this->startProcessing->Location = System::Drawing::Point(45, 560);

this->startProcessing->Margin = System::Windows::Forms::Padding(3, 2, 3, 2);

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

this->startProcessing->Size = System::Drawing::Size(112, 62);

this->startProcessing->TabIndex = 2;

this->startProcessing->Text = L"start";

this->startProcessing->UseVisualStyleBackColor = true;

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

//

// label1

//

this->label1->AutoSize = true;

this->label1->Location = System::Drawing::Point(83, 9);

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

this->label1->Size = System::Drawing::Size(35, 16);

this->label1->TabIndex = 3;

this->label1->Text = L"input";

//

// label2

//

this->label2->AutoSize = true;

this->label2->Location = System::Drawing::Point(796, 9);

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

this->label2->Size = System::Drawing::Size(39, 16);

this->label2->TabIndex = 4;

this->label2->Text = L"result";

//

// keyWordsTable

//

this->keyWordsTable->ColumnHeadersHeightSizeMode = System::Windows::Forms::DataGridViewColumnHeadersHeightSizeMode::AutoSize;

this->keyWordsTable->Columns->AddRange(gcnew cli::array< System::Windows::Forms::DataGridViewColumn^ >(3) {

this->keyWord10,

this->KeyWord, this->Pseudocode

});

this->keyWordsTable->Location = System::Drawing::Point(1155, 31);

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

this->keyWordsTable->RowHeadersWidth = 51;

this->keyWordsTable->RowTemplate->Height = 24;

this->keyWordsTable->Size = System::Drawing::Size(430, 175);

this->keyWordsTable->TabIndex = 5;

//

// label3

//

this->label3->AutoSize = true;

this->label3->Location = System::Drawing::Point(1262, 9);

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

this->label3->Size = System::Drawing::Size(85, 16);

this->label3->TabIndex = 6;

this->label3->Text = L"KeyWord(10)";

//

// label4

//

this->label4->AutoSize = true;

this->label4->Location = System::Drawing::Point(1262, 220);

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

this->label4->Size = System::Drawing::Size(120, 16);

this->label4->TabIndex = 8;

this->label4->Text = L"RelationsSigns(20)";

//

// relationsSignsTable

//

this->relationsSignsTable->ColumnHeadersHeightSizeMode = System::Windows::Forms::DataGridViewColumnHeadersHeightSizeMode::AutoSize;

this->relationsSignsTable->Columns->AddRange(gcnew cli::array< System::Windows::Forms::DataGridViewColumn^ >(3) {

this->relSigns20,

this->RelationsSigns, this->dataGridViewTextBoxColumn3

});

this->relationsSignsTable->Location = System::Drawing::Point(1155, 242);

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

this->relationsSignsTable->RowHeadersWidth = 51;

this->relationsSignsTable->RowTemplate->Height = 24;

this->relationsSignsTable->Size = System::Drawing::Size(430, 175);

this->relationsSignsTable->TabIndex = 7;

//

// label5

//

this->label5->AutoSize = true;

this->label5->Location = System::Drawing::Point(1262, 425);

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

this->label5->Size = System::Drawing::Size(122, 16);

this->label5->TabIndex = 10;

this->label5->Text = L"OperationSigns(30)";

//

// operationSignsTable

//

this->operationSignsTable->ColumnHeadersHeightSizeMode = System::Windows::Forms::DataGridViewColumnHeadersHeightSizeMode::AutoSize;

this->operationSignsTable->Columns->AddRange(gcnew cli::array< System::Windows::Forms::DataGridViewColumn^ >(3) {

this->oper30,

this->OperationSigns, this->dataGridViewTextBoxColumn5

});

this->operationSignsTable->Location = System::Drawing::Point(1155, 447);

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

this->operationSignsTable->RowHeadersWidth = 51;

this->operationSignsTable->RowTemplate->Height = 24;

this->operationSignsTable->Size = System::Drawing::Size(430, 175);

this->operationSignsTable->TabIndex = 9;

//

// standartCode

//

this->standartCode->Location = System::Drawing::Point(176, 560);

this->standartCode->Margin = System::Windows::Forms::Padding(3, 2, 3, 2);

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

this->standartCode->Size = System::Drawing::Size(112, 62);

this->standartCode->TabIndex = 11;

this->standartCode->Text = L"add standart code";

this->standartCode->UseVisualStyleBackColor = true;

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

//

// label6

//

this->label6->AutoSize = true;

this->label6->Location = System::Drawing::Point(1710, 425);

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

this->label6->Size = System::Drawing::Size(70, 16);

this->label6->TabIndex = 17;

this->label6->Text = L"Splitter(60)";

//

// splitterTable

//

this->splitterTable->ColumnHeadersHeightSizeMode = System::Windows::Forms::DataGridViewColumnHeadersHeightSizeMode::AutoSize;

this->splitterTable->Columns->AddRange(gcnew cli::array< System::Windows::Forms::DataGridViewColumn^ >(3) {

this->split60,

this->Splitter, this->dataGridViewTextBoxColumn7

});

this->splitterTable->Location = System::Drawing::Point(1603, 447);

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

this->splitterTable->RowHeadersWidth = 51;

this->splitterTable->RowTemplate->Height = 24;

this->splitterTable->Size = System::Drawing::Size(430, 175);

this->splitterTable->TabIndex = 16;

//

// label7

//

this->label7->AutoSize = true;

this->label7->Location = System::Drawing::Point(1710, 220);

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

this->label7->Size = System::Drawing::Size(63, 16);

this->label7->TabIndex = 15;

this->label7->Text = L"Const(50)";

//

// constTable

//

this->constTable->ColumnHeadersHeightSizeMode = System::Windows::Forms::DataGridViewColumnHeadersHeightSizeMode::AutoSize;

this->constTable->Columns->AddRange(gcnew cli::array< System::Windows::Forms::DataGridViewColumn^ >(3) {

this->const50, this->Const,

this->dataGridViewTextBoxColumn10

});

this->constTable->Location = System::Drawing::Point(1603, 242);

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

this->constTable->RowHeadersWidth = 51;

this->constTable->RowTemplate->Height = 24;

this->constTable->Size = System::Drawing::Size(430, 175);

this->constTable->TabIndex = 14;

//

// label8

//

this->label8->AutoSize = true;

this->label8->Location = System::Drawing::Point(1710, 9);

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

this->label8->Size = System::Drawing::Size(40, 16);

this->label8->TabIndex = 13;

this->label8->Text = L"Id(40)";

//

// idTable

//

this->idTable->ColumnHeadersHeightSizeMode = System::Windows::Forms::DataGridViewColumnHeadersHeightSizeMode::AutoSize;

this->idTable->Columns->AddRange(gcnew cli::array< System::Windows::Forms::DataGridViewColumn^ >(3) {

this->id40, this->Name,

this->dataGridViewTextBoxColumn13

});

this->idTable->Location = System::Drawing::Point(1603, 31);

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

this->idTable->RowHeadersWidth = 51;

this->idTable->RowTemplate->Height = 24;

this->idTable->Size = System::Drawing::Size(430, 175);

this->idTable->TabIndex = 12;

//

// descriptorCode

//

this->descriptorCode->Location = System::Drawing::Point(327, 465);

this->descriptorCode->Multiline = true;

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

this->descriptorCode->ReadOnly = true;

this->descriptorCode->ScrollBars = System::Windows::Forms::ScrollBars::Both;

this->descriptorCode->Size = System::Drawing::Size(324, 179);

this->descriptorCode->TabIndex = 18;

//

// pseudoCodes

//

this->pseudoCodes->Location = System::Drawing::Point(711, 465);

this->pseudoCodes->Multiline = true;

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

this->pseudoCodes->ScrollBars = System::Windows::Forms::ScrollBars::Both;

this->pseudoCodes->Size = System::Drawing::Size(324, 179);

this->pseudoCodes->TabIndex = 19;

//

// label9

//

this->label9->AutoSize = true;

this->label9->Location = System::Drawing::Point(449, 446);

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

this->label9->Size = System::Drawing::Size(102, 16);

this->label9->TabIndex = 20;

this->label9->Text = L"DescriptorCode";

//

// label10

//

this->label10->AutoSize = true;

this->label10->Location = System::Drawing::Point(822, 446);

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

this->label10->Size = System::Drawing::Size(87, 16);

this->label10->TabIndex = 21;

this->label10->Text = L"PseudoCode";

//

// relSigns20

//

this->relSigns20->HeaderText = L"20";

this->relSigns20->MinimumWidth = 6;

this->relSigns20->Name = L"relSigns20";

this->relSigns20->Width = 125;

//

// RelationsSigns

//

this->RelationsSigns->HeaderText = L"RelationsSigns";

this->RelationsSigns->MinimumWidth = 6;

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

this->RelationsSigns->Width = 125;

//

// dataGridViewTextBoxColumn3

//

this->dataGridViewTextBoxColumn3->HeaderText = L"Pseudocode";

this->dataGridViewTextBoxColumn3->MinimumWidth = 6;

this->dataGridViewTextBoxColumn3->Name = L"dataGridViewTextBoxColumn3";

this->dataGridViewTextBoxColumn3->Width = 125;

//

// keyWord10

//

this->keyWord10->HeaderText = L"10";

this->keyWord10->MinimumWidth = 6;

this->keyWord10->Name = L"keyWord10";

this->keyWord10->Width = 125;

//

// KeyWord

//

this->KeyWord->HeaderText = L"KeyWord";

this->KeyWord->MinimumWidth = 6;

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

this->KeyWord->Width = 125;

//

// Pseudocode

//

this->Pseudocode->HeaderText = L"Pseudocode";

this->Pseudocode->MinimumWidth = 6;

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

this->Pseudocode->Width = 125;

//

// id40

//

this->id40->HeaderText = L"40";

this->id40->MinimumWidth = 6;

this->id40->Name = L"id40";

this->id40->Width = 125;

//

// Name

//

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

this->Name->MinimumWidth = 6;

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

this->Name->Width = 125;

//

// dataGridViewTextBoxColumn13

//

this->dataGridViewTextBoxColumn13->HeaderText = L"Pseudocode";

this->dataGridViewTextBoxColumn13->MinimumWidth = 6;

this->dataGridViewTextBoxColumn13->Name = L"dataGridViewTextBoxColumn13";

this->dataGridViewTextBoxColumn13->Width = 125;

//

// const50

//

this->const50->HeaderText = L"50";

this->const50->MinimumWidth = 6;

this->const50->Name = L"const50";

this->const50->Width = 125;

//

// Const

//

this->Const->HeaderText = L"const";

this->Const->MinimumWidth = 6;

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

this->Const->Width = 125;

//

// dataGridViewTextBoxColumn10

//

this->dataGridViewTextBoxColumn10->HeaderText = L"Pseudocode";

this->dataGridViewTextBoxColumn10->MinimumWidth = 6;

this->dataGridViewTextBoxColumn10->Name = L"dataGridViewTextBoxColumn10";

this->dataGridViewTextBoxColumn10->Width = 125;

//

// oper30

//

this->oper30->HeaderText = L"30";

this->oper30->MinimumWidth = 6;

this->oper30->Name = L"oper30";

this->oper30->Width = 125;

//

// OperationSigns

//

this->OperationSigns->HeaderText = L"OperationSigns";

this->OperationSigns->MinimumWidth = 6;

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

this->OperationSigns->Width = 125;

//

// dataGridViewTextBoxColumn5

//

this->dataGridViewTextBoxColumn5->HeaderText = L"Pseudocode";

this->dataGridViewTextBoxColumn5->MinimumWidth = 6;

this->dataGridViewTextBoxColumn5->Name = L"dataGridViewTextBoxColumn5";

this->dataGridViewTextBoxColumn5->Width = 125;

//

// split60

//

this->split60->HeaderText = L"60";

this->split60->MinimumWidth = 6;

this->split60->Name = L"split60";

this->split60->Width = 125;

//

// Splitter

//

this->Splitter->HeaderText = L"Splitter";

this->Splitter->MinimumWidth = 6;

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

this->Splitter->Width = 125;

//

// dataGridViewTextBoxColumn7

//

this->dataGridViewTextBoxColumn7->HeaderText = L"Pseudocode";

this->dataGridViewTextBoxColumn7->MinimumWidth = 6;

this->dataGridViewTextBoxColumn7->Name = L"dataGridViewTextBoxColumn7";

this->dataGridViewTextBoxColumn7->Width = 125;

//

// MyForm

//

this->AutoScaleDimensions = System::Drawing::SizeF(8, 16);

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

this->AutoSize = true;

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

this->ClientSize = System::Drawing::Size(1924, 656);

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

this->Margin = System::Windows::Forms::Padding(3, 2, 3, 2);

this->Text = L"MyForm";

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

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

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

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

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

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

this->ResumeLayout(false);

this->PerformLayout();

}

#pragma endregion

private: LexerStates currentLexerState = LexerStates::S0;

private: String^ resultWithoutNumbers = "";

private: States currentsKeyWordState = States::S1;

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

LexerAnalyze();

ReadingKeyWords();

}

// if\_ int\_ else\_ for\_ while\_ string\_ switch\_ return\_ float\_ char\_

// double const do class enum struct using namespace static public

// private protected auto try catch throw case bool break continue

// default delete true false friend goto long short template operator

// typedef typename sizeof new this virtual void explicit inline nullptr

// dynamic\_cast extern mutable noexcept static\_cast thread\_local typeid wchar\_t static\_assert const\_cast

// + - \* / % ++ -- = += -= \*= /= %= >> << >>= <<=

// > < >= <= == !=

private: void ReadingKeyWords() {

keyWordsTable->Rows->Clear();

relationsSignsTable->Rows->Clear();

operationSignsTable->Rows->Clear();

splitterTable->Rows->Clear();

idTable->Rows->Clear();

constTable->Rows->Clear();

keyWordsTable->RowCount = 60;

idTable->RowCount = 1000;

splitterTable->RowCount = 20;

constTable->RowCount = 100;

relationsSignsTable->RowCount = 15;

operationSignsTable->RowCount = 15;

this->pseudoCodes->Text = "";

this->descriptorCode->Text = "";

currentsKeyWordState = States::S1;

String^ str = resultWithoutNumbers;

String^ word = "";

int keyWordIndex = 0;

int relationsSignsIndex = 0;

int operationsSignsIndex = 0;

int idTableIndex = 0;

int splitterTableIndex = 0;

int constTableIndex = 0;

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

switch (currentsKeyWordState)

{

case States::S1:

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

currentsKeyWordState = States::S2;

}

else if (str[i] == 'e') {

currentsKeyWordState = States::S5;

}

else if (str[i] == 'f') {

currentsKeyWordState = States::S8;

}

else if (str[i] == 'w') {

currentsKeyWordState = States::S10;

}

else if (str[i] == 's') {

currentsKeyWordState = States::S14;

}

else if (str[i] == 'r') {

currentsKeyWordState = States::S23;

}

else if (str[i] == 'c') {

currentsKeyWordState = States::S31;

}

else if (str[i] == '>') {

currentsKeyWordState = States::S34;

}

else if (str[i] == '<') {

currentsKeyWordState = States::S36;

}

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

currentsKeyWordState = States::S37;

}

else if (str[i] == '!') {

currentsKeyWordState = States::S38;

}

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

currentsKeyWordState = States::S40;

}

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

currentsKeyWordState = States::S41;

}

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

currentsKeyWordState = States::S42;

}

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

currentsKeyWordState = States::S43;

}

else if (str[i] == '%') {

currentsKeyWordState = States::S44;

}

else if (str[i] == '\r' || str[i] == '\n') continue;

else if (isSplitterForTable(str[i])) {

currentsKeyWordState = States::S46;

}

else if (isConstant(str[i])) {

currentsKeyWordState = States::S47;

}

else {

currentsKeyWordState = States::S45;

}

word += str[i];

break;

case States::S2: // if int

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

if(i + 1 == str->Length)

currentsKeyWordState = States::S3;

else if (isWord(str[i+1])) {

currentsKeyWordState = States::S45;

}

else

currentsKeyWordState = States::S3;

}

else if (str[i] == 'n') {

currentsKeyWordState = States::S4;

}

else {

currentsKeyWordState = States::S45;

}

if (!isSplitterForTable(str[i]))

word += str[i];

else i--;

break;

case States::S4:

if (str[i] == 't') {

if (i + 1 == str->Length)

currentsKeyWordState = States::S3;

else if (isWord(str[i + 1])) {

currentsKeyWordState = States::S45;

}

else

currentsKeyWordState = States::S3;

}

else {

currentsKeyWordState = States::S45;

}

if (!isSplitterForTable(str[i]))

word += str[i];

else i--;

break;

case States::S5: // else

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

currentsKeyWordState = States::S6;

}

else {

currentsKeyWordState = States::S45;

}

if (!isSplitterForTable(str[i]))

word += str[i];

else i--;

break;

case States::S6:

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

currentsKeyWordState = States::S7;

}

else {

currentsKeyWordState = States::S45;

}

if (!isSplitterForTable(str[i]))

word += str[i];

else i--;

break;

case States::S7:

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

if (i + 1 == str->Length)

currentsKeyWordState = States::S3;

else if (isWord(str[i + 1])) {

currentsKeyWordState = States::S45;

}

else

currentsKeyWordState = States::S3;

}

else {

currentsKeyWordState = States::S45;

}

if (!isSplitterForTable(str[i]))

word += str[i];

else i--;

break;

case States::S8: // for float

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

currentsKeyWordState = States::S9;

}

else if (str[i] == 'l') {

currentsKeyWordState = States::S28;

}

else {

currentsKeyWordState = States::S45;

}

if (!isSplitterForTable(str[i]))

word += str[i];

else i--;

break;

case States::S9:

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

if (i + 1 == str->Length)

currentsKeyWordState = States::S3;

else if (isWord(str[i + 1])) {

currentsKeyWordState = States::S45;

}

else

currentsKeyWordState = States::S3;

}

else {

currentsKeyWordState = States::S45;

}

if (!isSplitterForTable(str[i]))

word += str[i];

else i--;

break;

case States::S10: // while

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

currentsKeyWordState = States::S11;

}

else {

currentsKeyWordState = States::S45;

}

if (!isSplitterForTable(str[i]))

word += str[i];

else i--;

break;

case States::S11:

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

currentsKeyWordState = States::S12;

}

else {

currentsKeyWordState = States::S45;

}

if (!isSplitterForTable(str[i]))

word += str[i];

else i--;

break;

case States::S12:

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

currentsKeyWordState = States::S13;

}

else {

currentsKeyWordState = States::S45;

}

if (!isSplitterForTable(str[i]))

word += str[i];

else i--;

break;

case States::S13:

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

if (i + 1 == str->Length)

currentsKeyWordState = States::S3;

else if (isWord(str[i + 1])) {

currentsKeyWordState = States::S45;

}

else

currentsKeyWordState = States::S3;

}

else {

currentsKeyWordState = States::S45;

}

if (!isSplitterForTable(str[i]))

word += str[i];

else i--;

break;

case States::S14: // string switch

if (str[i] == 't') {

currentsKeyWordState = States::S15;

}

else if (str[i] == 'w') {

currentsKeyWordState = States::S19;

}

else {

currentsKeyWordState = States::S45;

}

if (!isSplitterForTable(str[i]))

word += str[i];

else i--;

break;

case States::S15:

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

currentsKeyWordState = States::S16;

}

else {

currentsKeyWordState = States::S45;

}

if (!isSplitterForTable(str[i]))

word += str[i];

else i--;

break;

case States::S16:

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

currentsKeyWordState = States::S17;

}

else {

currentsKeyWordState = States::S45;

}

if (!isSplitterForTable(str[i]))

word += str[i];

else i--;

break;

case States::S17:

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

currentsKeyWordState = States::S18;

}

else {

currentsKeyWordState = States::S45;

}

if (!isSplitterForTable(str[i]))

word += str[i];

else i--;

break;

case States::S18:

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

if (i + 1 == str->Length)

currentsKeyWordState = States::S3;

else if (isWord(str[i + 1])) {

currentsKeyWordState = States::S45;

}

else

currentsKeyWordState = States::S3;

}

else {

currentsKeyWordState = States::S45;

}

if (!isSplitterForTable(str[i]))

word += str[i];

else i--;

break;

case States::S19:

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

currentsKeyWordState = States::S20;

}

else {

currentsKeyWordState = States::S45;

}

if (!isSplitterForTable(str[i]))

word += str[i];

else i--;

break;

case States::S20:

if (str[i] == 't') {

currentsKeyWordState = States::S21;

}

else {

currentsKeyWordState = States::S45;

}

if (!isSplitterForTable(str[i]))

word += str[i];

else i--;

break;

case States::S21:

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

currentsKeyWordState = States::S22;

}

else {

currentsKeyWordState = States::S45;

}

if (!isSplitterForTable(str[i]))

word += str[i];

else i--;

break;

case States::S22:

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

if (i + 1 == str->Length)

currentsKeyWordState = States::S3;

else if (isWord(str[i + 1])) {

currentsKeyWordState = States::S45;

}

else

currentsKeyWordState = States::S3;

}

else {

currentsKeyWordState = States::S45;

}

if (!isSplitterForTable(str[i]))

word += str[i];

else i--;

break;

case States::S23: // return

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

currentsKeyWordState = States::S24;

}

else {

currentsKeyWordState = States::S45;

}

if (!isSplitterForTable(str[i]))

word += str[i];

else i--;

break;

case States::S24:

if (str[i] == 't') {

currentsKeyWordState = States::S25;

}

else {

currentsKeyWordState = States::S45;

}

if (!isSplitterForTable(str[i]))

word += str[i];

else i--;

break;

case States::S25:

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

currentsKeyWordState = States::S26;

}

else {

currentsKeyWordState = States::S45;

}

if (!isSplitterForTable(str[i]))

word += str[i];

else i--;

break;

case States::S26:

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

currentsKeyWordState = States::S27;

}

else {

currentsKeyWordState = States::S45;

}

if (!isSplitterForTable(str[i]))

word += str[i];

else i--;

break;

case States::S27:

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

if (i + 1 == str->Length)

currentsKeyWordState = States::S3;

else if (isWord(str[i + 1])) {

currentsKeyWordState = States::S45;

}

else

currentsKeyWordState = States::S3;

}

else {

currentsKeyWordState = States::S45;

}

if (!isSplitterForTable(str[i]))

word += str[i];

else i--;

break;

case States::S28:

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

currentsKeyWordState = States::S29;

}

else {

currentsKeyWordState = States::S45;

}

if (!isSplitterForTable(str[i]))

word += str[i];

else i--;

break;

case States::S29:

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

currentsKeyWordState = States::S30;

}

else {

currentsKeyWordState = States::S45;

}

if (!isSplitterForTable(str[i]))

word += str[i];

else i--;

break;

case States::S30:

if (str[i] == 't') {

if (i + 1 == str->Length)

currentsKeyWordState = States::S3;

else if (isWord(str[i + 1])) {

currentsKeyWordState = States::S45;

}

else

currentsKeyWordState = States::S3;

}

else {

currentsKeyWordState = States::S45;

}

if (!isSplitterForTable(str[i]))

word += str[i];

else i--;

break;

case States::S31: // char

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

currentsKeyWordState = States::S32;

}

else {

currentsKeyWordState = States::S45;

}

if (!isSplitterForTable(str[i]))

word += str[i];

else i--;

break;

case States::S32:

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

currentsKeyWordState = States::S33;

}

else {

currentsKeyWordState = States::S45;

}

if (!isSplitterForTable(str[i]))

word += str[i];

else i--;

break;

case States::S33:

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

if (i + 1 == str->Length)

currentsKeyWordState = States::S3;

else if (isWord(str[i + 1])) {

currentsKeyWordState = States::S45;

}

else

currentsKeyWordState = States::S3;

}

else {

currentsKeyWordState = States::S45;

}

if (!isSplitterForTable(str[i]))

word += str[i];

else i--;

break;

case States::S34: // > >= >>

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

word += str[i];

currentsKeyWordState = States::S35;

}

else if (str[i] == '>') {

word += str[i];

currentsKeyWordState = States::S39;

}

else {

i--;

currentsKeyWordState = States::S35;

}

break;

case States::S36: // < <= <<

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

word += str[i];

currentsKeyWordState = States::S35;

}

else if (str[i] == '<') {

word += str[i];

currentsKeyWordState = States::S39;

}

else {

i--;

currentsKeyWordState = States::S35;

}

break;

case States::S37: // = ==

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

word += str[i];

currentsKeyWordState = States::S35;

}

else {

i--;

currentsKeyWordState = States::S39;

}

break;

case States::S38: // ! !=

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

word += str[i];

currentsKeyWordState = States::S35;

}

else {

i--;

currentsKeyWordState = States::S30;

}

break;

case States::S40: // + += ++

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

word += str[i];

currentsKeyWordState = States::S39;

}

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

word += str[i];

currentsKeyWordState = States::S39;

}

else {

i--;

currentsKeyWordState = States::S39;

}

break;

case States::S41: // - -= --

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

word += str[i];

currentsKeyWordState = States::S39;

}

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

word += str[i];

currentsKeyWordState = States::S39;

}

else {

i--;

currentsKeyWordState = States::S39;

}

break;

case States::S42: // \* \*=

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

word += str[i];

currentsKeyWordState = States::S39;

}

else {

i--;

currentsKeyWordState = States::S39;

}

break;

case States::S43: // / /=

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

word += str[i];

currentsKeyWordState = States::S39;

}

else {

i--;

currentsKeyWordState = States::S39;

}

break;

case States::S44: // % %=

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

word += str[i];

currentsKeyWordState = States::S39;

}

else {

i--;

currentsKeyWordState = States::S39;

}

break;

case States::S45: // id

if (i + 1 != str->Length) {

if (isSplitterForTable(str[i + 1]) ||

isSplitterForTable(str[i]) ||

isSplitter(str[i + 1]) ||

isSplitter(str[i]) ||

isRelationsSigns(str[i]) ||

isOperationSigns(str[i])) {

if (!isSplitterForTable(str[i]) && !isRelationsSigns(str[i]) && !isOperationSigns(str[i]))

word += str[i];

else i--;

if (!isContainInTable(word, idTable)) {

WriteInIdTable(idTable, word, idTableIndex);

idTableIndex++;

}

this->descriptorCode->Text += getDescriptor(idTable, word);

this->pseudoCodes->Text += getPseudo(idTable, word);

word = "";

currentsKeyWordState = States::S1;

}

else {

word += str[i];

currentsKeyWordState = States::S45;

}

}

else {

if (isSplitterForTable(str[i])) {

i--;

}

else word += str[i];

if (!isContainInTable(word, idTable)) {

WriteInIdTable(idTable, word, idTableIndex);

idTableIndex++;

}

this->descriptorCode->Text += getDescriptor(idTable, word);

this->pseudoCodes->Text += getPseudo(idTable, word);

word = "";

currentsKeyWordState = States::S1;

}

break;

case States::S47: // constants

if (Char::IsDigit(str[i])) {

currentsKeyWordState = States::S47;

word += str[i];

}

else if (isSplitterForTable(str[i]) && str[i] != '.') {

if (!isContainInTable(word, constTable)) {

WriteInConstTable(constTable, word, constTableIndex);

constTableIndex++;

}

this->descriptorCode->Text += getDescriptor(constTable, word);

this->pseudoCodes->Text += getPseudo(constTable, word);

i--;

word = "";

currentsKeyWordState = States::S1;

}

else word += str[i];

break;

default:

break;

}

if (currentsKeyWordState == States::S3) {

if (!isContainInTable(word, keyWordsTable)) {

WriteInTable(keyWordsTable, word, keyWordIndex);

keyWordIndex++;

}

this->descriptorCode->Text += getDescriptor(keyWordsTable, word);

this->pseudoCodes->Text += getPseudo(keyWordsTable, word);

word = "";

currentsKeyWordState = States::S1;

}

if (currentsKeyWordState == States::S35) {

if (!isContainInTable(word, relationsSignsTable)) {

WriteInTable(relationsSignsTable, word, relationsSignsIndex);

relationsSignsIndex++;

}

this->descriptorCode->Text += getDescriptor(relationsSignsTable, word);

this->pseudoCodes->Text += getPseudo(relationsSignsTable, word);

word = "";

currentsKeyWordState = States::S1;

}

if (currentsKeyWordState == States::S46) {

if (!isContainInTable(word, splitterTable)) {

WriteInTable(splitterTable, word, splitterTableIndex);

splitterTableIndex++;

}

this->descriptorCode->Text += getDescriptor(splitterTable, word);

this->pseudoCodes->Text += getPseudo(splitterTable, word);

word = "";

currentsKeyWordState = States::S1;

}

if (currentsKeyWordState == States::S39) {

if (!isContainInTable(word, operationSignsTable)) {

WriteInTable(operationSignsTable, word, operationsSignsIndex);

operationsSignsIndex++;

}

this->descriptorCode->Text += getDescriptor(operationSignsTable, word);

this->pseudoCodes->Text += getPseudo(operationSignsTable, word);

word = "";

currentsKeyWordState = States::S1;

}

if (currentsKeyWordState == States::S45) {

if (i != str->Length - 1) {

if (isWord(str[i + 1])) continue;

}

if (isSplitterForTable(str[i])) {

i--;

}

else if(i != 0) word = str[i].ToString();

if (!isContainInTable(word, idTable)) {

if(word->Length > 1)

WriteInIdTable(idTable, word->Substring(0, word->Length-1), idTableIndex);

else

WriteInIdTable(idTable, word, idTableIndex);

idTableIndex++;

}

this->descriptorCode->Text += getDescriptor(idTable, word);

this->pseudoCodes->Text += getPseudo(idTable, word);

word = "";

currentsKeyWordState = States::S1;

}

}

}

private: String^ getDescriptor(DataGridView^ wordsTable, String^ word) {

if (word == " ") word = "\_";

String^ id = "";

for (int i = 0; i < wordsTable->Rows->Count; i++) {

if (wordsTable->Rows[i]->Cells[1]->Value->ToString() == word) {

id = wordsTable->Rows[i]->Cells[0]->Value->ToString();

break;

}

}

return "(" + wordsTable->Columns[0]->HeaderText + ", " + id + ") ";

}

private: String^ getPseudo(DataGridView^ wordsTable, String^ word) {

if (word == " ") word = "\_";

for (int i = 0; i < wordsTable->Rows->Count; i++) {

if (wordsTable->Rows[i]->Cells[1]->Value->ToString() == word) {

return wordsTable->Rows[i]->Cells[2]->Value->ToString();

}

}

return "";

}

private: void WriteInTable(DataGridView^ wordsTable, String^ word, int wordIndex) {

wordsTable->Rows[wordIndex]->Cells[0]->Value = wordIndex + 1;

if (word != " ") {

wordsTable->Rows[wordIndex]->Cells[1]->Value = word;

wordsTable->Rows[wordIndex]->Cells[2]->Value = word->ToUpper();

}

else {

wordsTable->Rows[wordIndex]->Cells[1]->Value = "\_";

wordsTable->Rows[wordIndex]->Cells[2]->Value = "\_";

}

}

private: void WriteInIdTable(DataGridView^ wordsTable, String^ word, int wordIndex) {

wordsTable->Rows[wordIndex]->Cells[0]->Value = wordIndex + 1;

wordsTable->Rows[wordIndex]->Cells[1]->Value = word;

wordsTable->Rows[wordIndex]->Cells[2]->Value = "id" + (wordIndex+1).ToString();

}

private: void WriteInConstTable(DataGridView^ wordsTable, String^ word, int wordIndex) {

wordsTable->Rows[wordIndex]->Cells[0]->Value = wordIndex + 1;

wordsTable->Rows[wordIndex]->Cells[1]->Value = word;

wordsTable->Rows[wordIndex]->Cells[2]->Value = "const";

}

private: bool isWord(char ch) {

std::set<char> st = { ' ', '(', ')', '[', ']', '{', '}', ':', '\r', '\n', '\t' };

bool bl = st.count(ch);

return !bl;

}

private: bool isSplitter(char ch) {

std::set<char> st = { ' ', '(', ')', '[', ']', '{', '}', ':', '\r', '\n', '\t'};

bool bl = st.count(ch);

return bl;

}

private: bool isRelationsSigns(char ch) {

std::set<char> st = { '>', '<', '=', '!' };

bool bl = st.count(ch);

return bl;

}

private: bool isOperationSigns(char ch) {

std::set<char> st = { '+', '-', '=', '\*', '/', '%' };

bool bl = st.count(ch);

return bl;

}

private: bool isSplitterForTable(char ch) {

std::set<char> st = { ' ', '(', ')', '[', ']', '{', '}', ':', '.', ';', '#', ','};

bool bl = st.count(ch);

return bl;

}

private: bool isNewWord(String^ str, int i) {

if (i == 0) return true;

return isSplitter(str[i - 1]);

}

private: bool isConstant(char ch) {

std::set<char> st = { '1', '2', '3','4', '5', '6', '7', '8', '9', '0', '\'', '.', '\"' };

return st.count(ch);

}

private: bool isContainInTable(String^ word, DataGridView^ table) {

if (word == " ") word = "\_";

for (int i = 0; i < table->Rows->Count; i++) {

if (table->Rows[i]->Cells[1]->Value == nullptr) return false;

if (table->Rows[i]->Cells[1]->Value->ToString() == word) return true;

}

return false;

}

private: void LexerAnalyze() {

this->result->Text = "";

String^ str = "";

currentLexerState = LexerStates::S0;

for (int i = 0; i < this->input->Text->Length; i++) {

switch (currentLexerState) {

case LexerStates::S0:

if (this->input->Text[i] == '/') {

currentLexerState = LexerStates::S1;

}

else if (this->input->Text[i] == ' ') {

currentLexerState = LexerStates::S5;

}

else if (this->input->Text[i] == '\t') {

currentLexerState = LexerStates::S6;

}

else if (this->input->Text[i] == '\n') {

str += this->input->Text[i];

currentLexerState = LexerStates::S7;

}

else {

str += this->input->Text[i];

}

break;

case LexerStates::S1:

if (this->input->Text[i] == '/') {

currentLexerState = LexerStates::S2;

}

else if (this->input->Text[i] == '\*') {

currentLexerState = LexerStates::S3;

}

else {

str += "/" + this->input->Text[i];

currentLexerState = LexerStates::S0;

}

break;

case LexerStates::S2:

if (this->input->Text[i] == '\n') {

str += "\r\n";

currentLexerState = LexerStates::S0;

}

break;

case LexerStates::S3:

if (this->input->Text[i] == '\*') {

currentLexerState = LexerStates::S4;

}

break;

case LexerStates::S4:

if (this->input->Text[i] == '/') {

currentLexerState = LexerStates::S0;

}

else {

currentLexerState = LexerStates::S3;

}

break;

case LexerStates::S5:

if (this->input->Text[i] == '/') {

currentLexerState = LexerStates::S1;

}

else if (this->input->Text[i] != ' ') {

currentLexerState = LexerStates::S0;

str += " ";

str += this->input->Text[i];

}

break;

case LexerStates::S6:

if (this->input->Text[i] == '/') {

currentLexerState = LexerStates::S1;

}

else if (this->input->Text[i] != '\t') {

currentLexerState = LexerStates::S0;

str += this->input->Text[i];

}

break;

case LexerStates::S7:

if (this->input->Text[i] == '\n' || this->input->Text[i] == '\r') {

currentLexerState = LexerStates::S7;

}

else if (this->input->Text[i] == '/')

currentLexerState = LexerStates::S1;

else if (this->input->Text[i] == '\t') {

currentLexerState = LexerStates::S6;

}

else if (this->input->Text[i] == ' ') {

currentLexerState = LexerStates::S5;

}

else {

str += this->input->Text[i];

currentLexerState = LexerStates::S0;

}

break;

}

}

int numberString = 1;

resultWithoutNumbers = str;

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

if (i == 0) {

this->result->Text += numberString.ToString() + " " + str[i];

numberString++;

}

else if (str[i] == '\n') {

this->result->Text += str[i] + numberString.ToString() + " ";

numberString++;

}

else {

this->result->Text += str[i];

}

}

}

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

this->input->Text = "#include <iostream>\r\n#include <string>\r\n#include <algorithm>\r\n\r\n\tint main() {\r\n\tsetlocale(LC\_ALL, \"rus\");\r\n\tstd::string str;\r\n\tstd::getline(std::cin, str);\r\n\tstd::string newStr;\r\n\tfor (int i = 0; i < str.length(); i++) {\r\n\tif (str[i] == ' ') {\r\n\t\tif (i > 0) {\r\n\t\t\tnewStr += std::toupper(str[++i]);\r\n\t\t}\r\n\t}\r\n\telse {\r\n\t\tnewStr += str[i];\r\n\t}\r\n\t}\r\n\tstd::cout << newStr << std::endl;\r\n\treturn 0;\r\n\t}";

}

};

};