File DTC.cpp:

// DTC.cpp : main project file.

#include "stdafx.h"

#include "Form1.h"

using namespace DTC;

[STAThreadAttribute]

int main(array<System::String ^> ^args)

{

// Enabling Windows XP visual effects before any controls are created

Application::EnableVisualStyles();

Application::SetCompatibleTextRenderingDefault(false);

// Create the main window and run it

Application::Run(gcnew Form1());

return 0;

}

File Form1.h:

#pragma once

namespace DTC {

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>

/// Summary for Form1

///

/// WARNING: If you change the name of this class, you will need to change the

/// 'Resource File Name' property for the managed resource compiler tool

/// associated with all .resx files this class depends on. Otherwise,

/// the designers will not be able to interact properly with localized

/// resources associated with this form.

/// </summary>

public ref class Form1 : public System::Windows::Forms::Form

{

public:

Form1(void)

{

InitializeComponent();

//

//TODO: Add the constructor code here

//

}

protected:

/// <summary>

/// Clean up any resources being used.

/// </summary>

~Form1()

{

if (components)

{

delete components;

}

}

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

public: System::Windows::Forms::Button^ button1;

protected:

protected:

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

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

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

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

private: System::Windows::Forms::ContextMenuStrip^ contextMenuStrip1;

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

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

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

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

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

private: System::Windows::Forms::StatusStrip^ statusStrip1;

private: System::Windows::Forms::ToolStripStatusLabel^ toolStripStatusLabel1;

public: System::Windows::Forms::ToolStripProgressBar^ toolStripProgressBar1;

private:

private: System::Windows::Forms::ToolStrip^ toolStrip1;

private: System::Windows::Forms::ToolStripButton^ toolStripButton1;

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

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

private: System::Windows::Forms::RadioButton^ radioButton1;

private: System::Windows::Forms::RadioButton^ radioButton2;

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

private: System::Windows::Forms::Button^ button2;

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

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

private: System::Windows::Forms::Button^ button3;

private: System::Windows::Forms::ContextMenuStrip^ contextMenuStrip2;

private: System::Windows::Forms::ToolStripMenuItem^ kopiujToolStripMenuItem1;

private: System::Windows::Forms::ToolStripMenuItem^ zaznaczWszystkoToolStripMenuItem1;

private: System::Windows::Forms::ToolStripMenuItem^ wyczyśćToolStripMenuItem1;

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

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

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

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

private: System::Windows::Forms::SaveFileDialog^ saveFileDialog1;

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

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

private: System::Windows::Forms::ToolStripSeparator^ toolStripSeparator1;

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

private: System::Windows::Forms::ToolStripMenuItem^ kopiujToolStripMenuItem2;

private: System::Windows::Forms::ToolStripMenuItem^ wklejToolStripMenuItem1;

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

private: System::ComponentModel::IContainer^ components;

public:

private:

private:

/// <summary>

/// Required designer variable.

/// </summary>

#pragma region Windows Form Designer generated code

/// <summary>

/// Required method for Designer support - do not modify

/// the contents of this method with the code editor.

/// </summary>

void InitializeComponent(void)

{

this->components = (gcnew System::ComponentModel::Container());

System::ComponentModel::ComponentResourceManager^ resources = (gcnew System::ComponentModel::ComponentResourceManager(Form1::typeid));

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

this->contextMenuStrip1 = (gcnew System::Windows::Forms::ContextMenuStrip(this->components));

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

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

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

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

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

this->button1 = (gcnew System::Windows::Forms::Button());

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

this->contextMenuStrip2 = (gcnew System::Windows::Forms::ContextMenuStrip(this->components));

this->kopiujToolStripMenuItem1 = (gcnew System::Windows::Forms::ToolStripMenuItem());

this->zaznaczWszystkoToolStripMenuItem1 = (gcnew System::Windows::Forms::ToolStripMenuItem());

this->wyczyśćToolStripMenuItem1 = (gcnew System::Windows::Forms::ToolStripMenuItem());

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

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

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

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

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

this->toolStripSeparator1 = (gcnew System::Windows::Forms::ToolStripSeparator());

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

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

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

this->kopiujToolStripMenuItem2 = (gcnew System::Windows::Forms::ToolStripMenuItem());

this->wklejToolStripMenuItem1 = (gcnew System::Windows::Forms::ToolStripMenuItem());

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

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

this->statusStrip1 = (gcnew System::Windows::Forms::StatusStrip());

this->toolStripStatusLabel1 = (gcnew System::Windows::Forms::ToolStripStatusLabel());

this->toolStripProgressBar1 = (gcnew System::Windows::Forms::ToolStripProgressBar());

this->toolStrip1 = (gcnew System::Windows::Forms::ToolStrip());

this->toolStripButton1 = (gcnew System::Windows::Forms::ToolStripButton());

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

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

this->radioButton1 = (gcnew System::Windows::Forms::RadioButton());

this->radioButton2 = (gcnew System::Windows::Forms::RadioButton());

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

this->button2 = (gcnew System::Windows::Forms::Button());

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

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

this->button3 = (gcnew System::Windows::Forms::Button());

this->saveFileDialog1 = (gcnew System::Windows::Forms::SaveFileDialog());

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

this->contextMenuStrip1->SuspendLayout();

this->contextMenuStrip2->SuspendLayout();

this->menuStrip1->SuspendLayout();

this->statusStrip1->SuspendLayout();

this->toolStrip1->SuspendLayout();

this->SuspendLayout();

//

// textBox1

//

this->textBox1->AccessibleRole = System::Windows::Forms::AccessibleRole::Text;

this->textBox1->Anchor = static\_cast<System::Windows::Forms::AnchorStyles>(((System::Windows::Forms::AnchorStyles::Top | System::Windows::Forms::AnchorStyles::Left)

| System::Windows::Forms::AnchorStyles::Right));

this->textBox1->ContextMenuStrip = this->contextMenuStrip1;

this->textBox1->ImeMode = System::Windows::Forms::ImeMode::On;

this->textBox1->Location = System::Drawing::Point(15, 82);

this->textBox1->MinimumSize = System::Drawing::Size(130, 30);

this->textBox1->Multiline = true;

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

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

this->textBox1->Size = System::Drawing::Size(665, 114);

this->textBox1->TabIndex = 0;

this->textBox1->TextChanged += gcnew System::EventHandler(this, &Form1::textBox1\_TextChanged);

//

// contextMenuStrip1

//

this->contextMenuStrip1->Items->AddRange(gcnew cli::array< System::Windows::Forms::ToolStripItem^ >(5) {this->kopiujToolStripMenuItem,

this->wklejToolStripMenuItem, this->zaznaczWszystkoToolStripMenuItem, this->cofnijToolStripMenuItem, this->wyczyśćToolStripMenuItem});

this->contextMenuStrip1->Name = L"contextMenuStrip1";

this->contextMenuStrip1->Size = System::Drawing::Size(211, 114);

this->contextMenuStrip1->Opened += gcnew System::EventHandler(this, &Form1::skup);

//

// kopiujToolStripMenuItem

//

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

this->kopiujToolStripMenuItem->ShortcutKeyDisplayString = L"";

this->kopiujToolStripMenuItem->ShortcutKeys = static\_cast<System::Windows::Forms::Keys>((System::Windows::Forms::Keys::Control | System::Windows::Forms::Keys::C));

this->kopiujToolStripMenuItem->Size = System::Drawing::Size(210, 22);

this->kopiujToolStripMenuItem->Text = L"&Kopiuj";

this->kopiujToolStripMenuItem->Click += gcnew System::EventHandler(this, &Form1::kopiujToolStripMenuItem\_Click);

//

// wklejToolStripMenuItem

//

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

this->wklejToolStripMenuItem->ShortcutKeys = static\_cast<System::Windows::Forms::Keys>((System::Windows::Forms::Keys::Control | System::Windows::Forms::Keys::V));

this->wklejToolStripMenuItem->Size = System::Drawing::Size(210, 22);

this->wklejToolStripMenuItem->Text = L"&Wklej";

this->wklejToolStripMenuItem->Click += gcnew System::EventHandler(this, &Form1::wklejToolStripMenuItem\_Click);

//

// zaznaczWszystkoToolStripMenuItem

//

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

this->zaznaczWszystkoToolStripMenuItem->ShortcutKeys = static\_cast<System::Windows::Forms::Keys>((System::Windows::Forms::Keys::Control | System::Windows::Forms::Keys::A));

this->zaznaczWszystkoToolStripMenuItem->Size = System::Drawing::Size(210, 22);

this->zaznaczWszystkoToolStripMenuItem->Text = L"&Zaznacz Wszystko";

this->zaznaczWszystkoToolStripMenuItem->Click += gcnew System::EventHandler(this, &Form1::zaznaczWszystkoToolStripMenuItem\_Click);

//

// cofnijToolStripMenuItem

//

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

this->cofnijToolStripMenuItem->ShortcutKeys = static\_cast<System::Windows::Forms::Keys>((System::Windows::Forms::Keys::Control | System::Windows::Forms::Keys::Z));

this->cofnijToolStripMenuItem->Size = System::Drawing::Size(210, 22);

this->cofnijToolStripMenuItem->Text = L"&Cofnij";

this->cofnijToolStripMenuItem->Click += gcnew System::EventHandler(this, &Form1::cofnijToolStripMenuItem\_Click);

//

// wyczyśćToolStripMenuItem

//

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

this->wyczyśćToolStripMenuItem->Size = System::Drawing::Size(210, 22);

this->wyczyśćToolStripMenuItem->Text = L"W&yczyść";

this->wyczyśćToolStripMenuItem->Click += gcnew System::EventHandler(this, &Form1::wyczyśćToolStripMenuItem\_Click);

//

// button1

//

this->button1->Enabled = false;

this->button1->Location = System::Drawing::Point(336, 224);

this->button1->Name = L"button1";

this->button1->Size = System::Drawing::Size(75, 23);

this->button1->TabIndex = 1;

this->button1->Text = L"S&zyfruj";

this->button1->UseCompatibleTextRendering = true;

this->button1->UseVisualStyleBackColor = true;

this->button1->Click += gcnew System::EventHandler(this, &Form1::button1\_Click);

//

// textBox2

//

this->textBox2->Anchor = static\_cast<System::Windows::Forms::AnchorStyles>(((System::Windows::Forms::AnchorStyles::Top | System::Windows::Forms::AnchorStyles::Left)

| System::Windows::Forms::AnchorStyles::Right));

this->textBox2->ContextMenuStrip = this->contextMenuStrip2;

this->textBox2->ImeMode = System::Windows::Forms::ImeMode::Off;

this->textBox2->Location = System::Drawing::Point(13, 270);

this->textBox2->MaxLength = 278527;

this->textBox2->MinimumSize = System::Drawing::Size(130, 30);

this->textBox2->Multiline = true;

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

this->textBox2->ReadOnly = true;

this->textBox2->ScrollBars = System::Windows::Forms::ScrollBars::Vertical;

this->textBox2->Size = System::Drawing::Size(665, 156);

this->textBox2->TabIndex = 2;

this->textBox2->TabStop = false;

//

// contextMenuStrip2

//

this->contextMenuStrip2->Items->AddRange(gcnew cli::array< System::Windows::Forms::ToolStripItem^ >(3) {this->kopiujToolStripMenuItem1,

this->zaznaczWszystkoToolStripMenuItem1, this->wyczyśćToolStripMenuItem1});

this->contextMenuStrip2->Name = L"contextMenuStrip2";

this->contextMenuStrip2->Size = System::Drawing::Size(211, 70);

this->contextMenuStrip2->Opened += gcnew System::EventHandler(this, &Form1::skup2);

//

// kopiujToolStripMenuItem1

//

this->kopiujToolStripMenuItem1->Name = L"kopiujToolStripMenuItem1";

this->kopiujToolStripMenuItem1->ShortcutKeyDisplayString = L"";

this->kopiujToolStripMenuItem1->ShortcutKeys = static\_cast<System::Windows::Forms::Keys>((System::Windows::Forms::Keys::Control | System::Windows::Forms::Keys::C));

this->kopiujToolStripMenuItem1->Size = System::Drawing::Size(210, 22);

this->kopiujToolStripMenuItem1->Text = L"&Kopiuj";

this->kopiujToolStripMenuItem1->Click += gcnew System::EventHandler(this, &Form1::kopiujToolStripMenuItem1\_Click);

//

// zaznaczWszystkoToolStripMenuItem1

//

this->zaznaczWszystkoToolStripMenuItem1->Name = L"zaznaczWszystkoToolStripMenuItem1";

this->zaznaczWszystkoToolStripMenuItem1->ShortcutKeys = static\_cast<System::Windows::Forms::Keys>((System::Windows::Forms::Keys::Control | System::Windows::Forms::Keys::A));

this->zaznaczWszystkoToolStripMenuItem1->Size = System::Drawing::Size(210, 22);

this->zaznaczWszystkoToolStripMenuItem1->Text = L"&Zaznacz Wszystko";

this->zaznaczWszystkoToolStripMenuItem1->Click += gcnew System::EventHandler(this, &Form1::zaznaczWszystkoToolStripMenuItem1\_Click);

//

// wyczyśćToolStripMenuItem1

//

this->wyczyśćToolStripMenuItem1->Name = L"wyczyśćToolStripMenuItem1";

this->wyczyśćToolStripMenuItem1->Size = System::Drawing::Size(210, 22);

this->wyczyśćToolStripMenuItem1->Text = L"Wyczyść";

this->wyczyśćToolStripMenuItem1->Click += gcnew System::EventHandler(this, &Form1::wyczyśćToolStripMenuItem1\_Click);

//

// menuStrip1

//

this->menuStrip1->Items->AddRange(gcnew cli::array< System::Windows::Forms::ToolStripItem^ >(4) {this->menuToolStripMenuItem,

this->edycjaToolStripMenuItem, this->opcjeToolStripMenuItem, this->pomocToolStripMenuItem});

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

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

this->menuStrip1->Size = System::Drawing::Size(690, 24);

this->menuStrip1->TabIndex = 3;

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

//

// menuToolStripMenuItem

//

this->menuToolStripMenuItem->DropDownItems->AddRange(gcnew cli::array< System::Windows::Forms::ToolStripItem^ >(5) {this->nowyToolStripMenuItem,

this->otwórzToolStripMenuItem, this->zapiszToolStripMenuItem, this->toolStripSeparator1, this->zamknijToolStripMenuItem});

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

this->menuToolStripMenuItem->Size = System::Drawing::Size(38, 20);

this->menuToolStripMenuItem->Text = L"Plik";

//

// nowyToolStripMenuItem

//

this->nowyToolStripMenuItem->Enabled = false;

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

this->nowyToolStripMenuItem->Size = System::Drawing::Size(147, 22);

this->nowyToolStripMenuItem->Text = L"Nowy";

//

// otwórzToolStripMenuItem

//

this->otwórzToolStripMenuItem->Enabled = false;

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

this->otwórzToolStripMenuItem->Size = System::Drawing::Size(147, 22);

this->otwórzToolStripMenuItem->Text = L"Otwórz";

//

// zapiszToolStripMenuItem

//

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

this->zapiszToolStripMenuItem->ShortcutKeys = static\_cast<System::Windows::Forms::Keys>((System::Windows::Forms::Keys::Control | System::Windows::Forms::Keys::S));

this->zapiszToolStripMenuItem->Size = System::Drawing::Size(147, 22);

this->zapiszToolStripMenuItem->Text = L"Zapisz";

this->zapiszToolStripMenuItem->Click += gcnew System::EventHandler(this, &Form1::zapiszToolStripMenuItem\_Click);

//

// toolStripSeparator1

//

this->toolStripSeparator1->Name = L"toolStripSeparator1";

this->toolStripSeparator1->Size = System::Drawing::Size(144, 6);

//

// zamknijToolStripMenuItem

//

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

this->zamknijToolStripMenuItem->Size = System::Drawing::Size(147, 22);

this->zamknijToolStripMenuItem->Text = L"Zamknij";

this->zamknijToolStripMenuItem->Click += gcnew System::EventHandler(this, &Form1::zamknijToolStripMenuItem\_Click);

//

// edycjaToolStripMenuItem

//

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

this->kopiujToolStripMenuItem2, this->wklejToolStripMenuItem1});

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

this->edycjaToolStripMenuItem->Size = System::Drawing::Size(53, 20);

this->edycjaToolStripMenuItem->Text = L"Edycja";

//

// wytnijToolStripMenuItem

//

this->wytnijToolStripMenuItem->Enabled = false;

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

this->wytnijToolStripMenuItem->Size = System::Drawing::Size(150, 22);

this->wytnijToolStripMenuItem->Text = L"Wytnij";

//

// kopiujToolStripMenuItem2

//

this->kopiujToolStripMenuItem2->Name = L"kopiujToolStripMenuItem2";

this->kopiujToolStripMenuItem2->ShortcutKeys = static\_cast<System::Windows::Forms::Keys>((System::Windows::Forms::Keys::Control | System::Windows::Forms::Keys::C));

this->kopiujToolStripMenuItem2->Size = System::Drawing::Size(150, 22);

this->kopiujToolStripMenuItem2->Text = L"Kopiuj";

this->kopiujToolStripMenuItem2->Click += gcnew System::EventHandler(this, &Form1::kopiujToolStripMenuItem2\_Click);

//

// wklejToolStripMenuItem1

//

this->wklejToolStripMenuItem1->Name = L"wklejToolStripMenuItem1";

this->wklejToolStripMenuItem1->ShortcutKeys = static\_cast<System::Windows::Forms::Keys>((System::Windows::Forms::Keys::Control | System::Windows::Forms::Keys::V));

this->wklejToolStripMenuItem1->Size = System::Drawing::Size(150, 22);

this->wklejToolStripMenuItem1->Text = L"Wklej";

this->wklejToolStripMenuItem1->Click += gcnew System::EventHandler(this, &Form1::wklejToolStripMenuItem1\_Click);

//

// opcjeToolStripMenuItem

//

this->opcjeToolStripMenuItem->Enabled = false;

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

this->opcjeToolStripMenuItem->Size = System::Drawing::Size(50, 20);

this->opcjeToolStripMenuItem->Text = L"Opcje";

//

// pomocToolStripMenuItem

//

this->pomocToolStripMenuItem->Enabled = false;

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

this->pomocToolStripMenuItem->Size = System::Drawing::Size(57, 20);

this->pomocToolStripMenuItem->Text = L"Pomoc";

//

// statusStrip1

//

this->statusStrip1->Items->AddRange(gcnew cli::array< System::Windows::Forms::ToolStripItem^ >(2) {this->toolStripStatusLabel1,

this->toolStripProgressBar1});

this->statusStrip1->Location = System::Drawing::Point(0, 448);

this->statusStrip1->Name = L"statusStrip1";

this->statusStrip1->Size = System::Drawing::Size(690, 22);

this->statusStrip1->SizingGrip = false;

this->statusStrip1->TabIndex = 5;

this->statusStrip1->Text = L"statusStrip1";

//

// toolStripStatusLabel1

//

this->toolStripStatusLabel1->Name = L"toolStripStatusLabel1";

this->toolStripStatusLabel1->Size = System::Drawing::Size(48, 17);

this->toolStripStatusLabel1->Text = L"Gotowe";

//

// toolStripProgressBar1

//

this->toolStripProgressBar1->AccessibleRole = System::Windows::Forms::AccessibleRole::ProgressBar;

this->toolStripProgressBar1->Name = L"toolStripProgressBar1";

this->toolStripProgressBar1->Size = System::Drawing::Size(200, 16);

this->toolStripProgressBar1->Visible = false;

//

// toolStrip1

//

this->toolStrip1->Dock = System::Windows::Forms::DockStyle::None;

this->toolStrip1->Items->AddRange(gcnew cli::array< System::Windows::Forms::ToolStripItem^ >(1) {this->toolStripButton1});

this->toolStrip1->Location = System::Drawing::Point(15, 27);

this->toolStrip1->Name = L"toolStrip1";

this->toolStrip1->Size = System::Drawing::Size(35, 25);

this->toolStrip1->TabIndex = 6;

this->toolStrip1->Text = L"toolStrip1";

//

// toolStripButton1

//

this->toolStripButton1->DisplayStyle = System::Windows::Forms::ToolStripItemDisplayStyle::Image;

this->toolStripButton1->Image = (cli::safe\_cast<System::Drawing::Image^ >(resources->GetObject(L"toolStripButton1.Image")));

this->toolStripButton1->ImageTransparentColor = System::Drawing::Color::Magenta;

this->toolStripButton1->Name = L"toolStripButton1";

this->toolStripButton1->Size = System::Drawing::Size(23, 22);

this->toolStripButton1->Text = L"Zapisz";

this->toolStripButton1->Click += gcnew System::EventHandler(this, &Form1::toolStripButton1\_Click);

//

// label1

//

this->label1->AutoSize = true;

this->label1->Location = System::Drawing::Point(12, 208);

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

this->label1->Size = System::Drawing::Size(40, 13);

this->label1->TabIndex = 7;

this->label1->Text = L"Status:";

//

// label2

//

this->label2->AutoSize = true;

this->label2->ForeColor = System::Drawing::SystemColors::ControlText;

this->label2->Location = System::Drawing::Point(12, 234);

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

this->label2->Size = System::Drawing::Size(109, 13);

this->label2->TabIndex = 8;

this->label2->Text = L"Sprawdź poprawność";

//

// radioButton1

//

this->radioButton1->AutoSize = true;

this->radioButton1->Checked = true;

this->radioButton1->Location = System::Drawing::Point(468, 222);

this->radioButton1->Name = L"radioButton1";

this->radioButton1->Size = System::Drawing::Size(87, 17);

this->radioButton1->TabIndex = 9;

this->radioButton1->TabStop = true;

this->radioButton1->Text = L"Standardowy";

this->radioButton1->UseVisualStyleBackColor = true;

this->radioButton1->CheckedChanged += gcnew System::EventHandler(this, &Form1::radioButton1\_CheckedChanged);

//

// radioButton2

//

this->radioButton2->AutoSize = true;

this->radioButton2->Location = System::Drawing::Point(468, 248);

this->radioButton2->Name = L"radioButton2";

this->radioButton2->Size = System::Drawing::Size(90, 17);

this->radioButton2->TabIndex = 10;

this->radioButton2->TabStop = true;

this->radioButton2->Text = L"Kompleksowy";

this->radioButton2->UseVisualStyleBackColor = true;

//

// label3

//

this->label3->AutoSize = true;

this->label3->Location = System::Drawing::Point(465, 199);

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

this->label3->Size = System::Drawing::Size(102, 13);

this->label3->TabIndex = 11;

this->label3->Text = L"Poziom szyfrowania:";

//

// button2

//

this->button2->Location = System::Drawing::Point(227, 224);

this->button2->Name = L"button2";

this->button2->Size = System::Drawing::Size(75, 23);

this->button2->TabIndex = 12;

this->button2->Text = L"&Sprawdź";

this->button2->UseVisualStyleBackColor = true;

this->button2->Click += gcnew System::EventHandler(this, &Form1::button2\_Click);

//

// comboBox1

//

this->comboBox1->DropDownStyle = System::Windows::Forms::ComboBoxStyle::DropDownList;

this->comboBox1->FormattingEnabled = true;

this->comboBox1->Items->AddRange(gcnew cli::array< System::Object^ >(2) {L"Szyfrowanie", L"Deszyfrowanie"});

this->comboBox1->Location = System::Drawing::Point(49, 55);

this->comboBox1->MaxDropDownItems = 2;

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

this->comboBox1->Size = System::Drawing::Size(100, 21);

this->comboBox1->TabIndex = 13;

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

this->comboBox1->SelectionChangeCommitted += gcnew System::EventHandler(this, &Form1::comboBox1\_SelectionChangeCommitted);

//

// label4

//

this->label4->AutoSize = true;

this->label4->Location = System::Drawing::Point(12, 58);

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

this->label4->Size = System::Drawing::Size(31, 13);

this->label4->TabIndex = 14;

this->label4->Text = L"Tryb:";

//

// button3

//

this->button3->Location = System::Drawing::Point(289, 198);

this->button3->Name = L"button3";

this->button3->Size = System::Drawing::Size(57, 23);

this->button3->TabIndex = 15;

this->button3->Text = L"&Czyść";

this->button3->UseVisualStyleBackColor = true;

this->button3->Click += gcnew System::EventHandler(this, &Form1::button3\_Click);

//

// saveFileDialog1

//

this->saveFileDialog1->DefaultExt = L"txt";

this->saveFileDialog1->Filter = L"\*.txt|Wszystkie pliki";

this->saveFileDialog1->Title = L"Zapisz jako";

this->saveFileDialog1->FileOk += gcnew System::ComponentModel::CancelEventHandler(this, &Form1::saveFileDialog1\_FileOk);

//

// checkBox1

//

this->checkBox1->AutoSize = true;

this->checkBox1->Location = System::Drawing::Point(289, 58);

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

this->checkBox1->Size = System::Drawing::Size(93, 18);

this->checkBox1->TabIndex = 16;

this->checkBox1->Text = L"&Krok po kroku";

this->checkBox1->UseCompatibleTextRendering = true;

this->checkBox1->UseVisualStyleBackColor = true;

//

// Form1

//

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

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

this->ClientSize = System::Drawing::Size(690, 470);

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

this->Controls->Add(this->button3);

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

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

this->Controls->Add(this->radioButton2);

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

this->Controls->Add(this->button2);

this->Controls->Add(this->radioButton1);

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

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

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

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

this->Controls->Add(this->button1);

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

this->Controls->Add(this->statusStrip1);

this->Controls->Add(this->toolStrip1);

this->FormBorderStyle = System::Windows::Forms::FormBorderStyle::Fixed3D;

this->MainMenuStrip = this->menuStrip1;

this->MaximizeBox = false;

this->MinimumSize = System::Drawing::Size(700, 500);

this->Name = L"Form1";

this->SizeGripStyle = System::Windows::Forms::SizeGripStyle::Hide;

this->StartPosition = System::Windows::Forms::FormStartPosition::CenterScreen;

this->Text = L"DTC 1.0 - Dominic\'s Text Coder";

this->Load += gcnew System::EventHandler(this, &Form1::Form1\_Load);

this->contextMenuStrip1->ResumeLayout(false);

this->contextMenuStrip2->ResumeLayout(false);

this->menuStrip1->ResumeLayout(false);

this->menuStrip1->PerformLayout();

this->statusStrip1->ResumeLayout(false);

this->statusStrip1->PerformLayout();

this->toolStrip1->ResumeLayout(false);

this->toolStrip1->PerformLayout();

this->ResumeLayout(false);

this->PerformLayout();

}

#pragma endregion

// zmienna globalna używana przy szyfrowaniu krok po kroku;

// pozwala zapamiętać tekst na etapie "mixed" i powrócić do niego po utworzeniu segmentów;

public: System::String^ zapamietanyText;

// funkcja opisująca wciśnięcie przycisku do szyfrowania lub odszyfrowania;

private: System::Void button1\_Click(System::Object^ sender, System::EventArgs^ e) {

button1->Enabled::set(false);

// deszyfrowanie krok po kroku/ standardowe/ ostatni krok/

// kontroluje aby po wciśnięciu przycisku Dalej została wykonana odpowiednia funkcja;

if ( (comboBox1->SelectedIndex == 1) && (button1->Text == "&Dalej") && (label2->Text == "...Krok2...") && (radioButton1->Checked) )

{

KrokPoKroku5K(textBox2->Text);

button1->Enabled::set(true);

}

// deszyfrowanie krok po kroku/ kompleksowe/ ostatni krok;

if ( (comboBox1->SelectedIndex == 1) && (button1->Text == "&Dalej") && (label2->Text == "...Krok4...") && (radioButton2->Checked) )

{

KrokPoKroku5K(textBox2->Text);

button1->Enabled::set(true);

}

// deszyfrowanie krok po kroku/ standardowe/ krok 2 - z "mixed" do "inverted";

if ( (comboBox1->SelectedIndex == 1) && (button1->Text == "&Dalej") && (label2->Text == "Krok1...") && (radioButton1->Checked) )

{

KrokPoKroku4K(textBox2->Text);

button1->Enabled::set(true);

}

// deszyfrowanie krok po kroku/ kompleksowe/ krok 4 - z "mixed" do "inverted";

if ( (comboBox1->SelectedIndex == 1) && (button1->Text == "&Dalej") && (label2->Text == "...Krok3...") && (radioButton2->Checked) )

{

KrokPoKroku4K(textBox2->Text);

button1->Enabled::set(true);

}

// deszyfrowanie krok po kroku/ kompleksowe/ krok 3 - ze "Standard" do "mixed";

if ( (comboBox1->SelectedIndex == 1) && (button1->Text == "&Dalej") && (label2->Text == "...Krok2...") && (radioButton2->Checked) )

{

KrokPoKroku3K(textBox2->Text);

button1->Enabled::set(true);

}

// deszyfrowanie krok po kroku/ kompleksowe/ krok 2 - z heksadecymalnych do "Standard";

if ( (comboBox1->SelectedIndex == 1) && (label2->Text == "Krok1...") && (radioButton2->Checked) )

{

KrokPoKroku2K(textBox2->Text);

button1->Enabled::set(true);

}

// szyfrowanie krok po kroku/ kompleksowe/ ostatni krok - z "heksa" do binarnych;

if ( (comboBox1->SelectedIndex == 0) && (label2->Text == "...Krok5...") && (radioButton2->Checked) )

{

KrokPoKroku6(textBox2->Text);

button1->Enabled::set(true);

}

// szyfrowanie krok po kroku/ kompleksowe/ krok 5 - ze "Standard" do "heksa";

if ( (comboBox1->SelectedIndex == 0) && (label2->Text == "...Krok4...") && (radioButton2->Checked) )

{

KrokPoKroku5(textBox2->Text);

button1->Enabled::set(true);

}

// szyfrowanie krok po kroku/ nie ważne czy standardowe czy kompleksowe/

// krok 4 z "mixed" do "Standard"/ pomija się tu "segmented" i korzysta z

// tekstu zapisanego w globalnej zmiennej w kroku 2;

if ( (comboBox1->SelectedIndex == 0) && (button1->Text == "&Dalej") && (label2->Text == "...Krok3...") )

{

KrokPoKroku4();

button1->Enabled::set(true);

}

// szyfrowanie krok po kroku/ krok 3 - z "mixed" do "segmented";

if ( (comboBox1->SelectedIndex == 0) && (button1->Text == "&Dalej") && (label2->Text == "...Krok2...") )

{

KrokPoKroku3(textBox2->Text);

button1->Enabled::set(true);

}

// szyfrowanie krok po kroku/ krok 2 - z "inverted" do "mixed";

if ( (comboBox1->SelectedIndex == 0) && (button1->Text == "&Dalej") && (label2->Text == "Krok1...") )

{

KrokPoKroku2(textBox2->Text);

button1->Enabled::set(true);

}

// dla trybu szyfrowania;

if (button1->Text == "S&zyfruj")

{

Form1::toolStripStatusLabel1->Text::set("Szyfrowanie tekstu");

// dla zaznaczonej opcji Krok po kroku;

if (Form1::checkBox1->Checked::get())

{

// są dezaktywowane wszystkie elementy niepotrzebne w Krok po kroku;

nowyToolStripMenuItem->Enabled::set(false);

otwórzToolStripMenuItem->Enabled::set(false);

zapiszToolStripMenuItem->Enabled::set(false);

edycjaToolStripMenuItem->Enabled::set(false);

opcjeToolStripMenuItem->Enabled::set(false);

pomocToolStripMenuItem->Enabled::set(false);

toolStrip1->Enabled::set(false);

comboBox1->Enabled::set(false);

checkBox1->Enabled::set(false);

textBox1->Enabled::set(false);

button3->Enabled::set(false);

button2->Enabled::set(false);

// jeśli szyfrujemy standardowo to najpierw wyłączamy drugi przycisk;

if (radioButton1->Checked)

{

radioButton2->Enabled::set(false);

radioButton1->Enabled::set(false);

}

// a jeśli kompleksowo to najpierw pierwszy przycisk;

if (radioButton2->Checked)

{

radioButton1->Enabled::set(false);

radioButton2->Enabled::set(false);

}

// szyfrowanie krok po kroku/ krok 1 - z "Normal" do "inverted";

KrokPoKroku(textBox1->Text);

button1->Enabled::set(true);

button1->Text::set("&Dalej");

}

else // dla szyfrowania bez opcji krok po kroku;

{

// szyfrowanie standardowe;

if (Form1::radioButton1->Checked::get())

textBox2->Text::set(DisplayStandard(textBox1->Text::get()));

// szyfrowanie kompleksowe;

if (Form1::radioButton2->Checked::get())

textBox2->Text::set( DisplayComplex( DisplayStandard( textBox1->Text::get() ) ) );

}

}

// dla trybu deszyfrowania;

if (button1->Text == "&Odszyfruj")

{

Form1::toolStripStatusLabel1->Text::set("Odszyfrowywanie tekstu");

// dla zaznaczonej opcji krok po kroku;

if (checkBox1->Checked)

{

nowyToolStripMenuItem->Enabled::set(false);

otwórzToolStripMenuItem->Enabled::set(false);

zapiszToolStripMenuItem->Enabled::set(false);

edycjaToolStripMenuItem->Enabled::set(false);

opcjeToolStripMenuItem->Enabled::set(false);

pomocToolStripMenuItem->Enabled::set(false);

toolStrip1->Enabled::set(false);

comboBox1->Enabled::set(false);

checkBox1->Enabled::set(false);

textBox1->Enabled::set(false);

button3->Enabled::set(false);

button2->Enabled::set(false);

if (radioButton2->Checked)

{

radioButton1->Enabled::set(false);

radioButton2->Enabled::set(false);

}

if (radioButton1->Checked)

{

radioButton2->Enabled::set(false);

radioButton1->Enabled::set(false);

}

// deszyfrowanie kompleksowe krok po kroku/ krok 1 - z binarnych do "heksa";

if (radioButton2->Checked)

KrokPoKroku1K(textBox1->Text);

// deszyfrowanie standardowe krok po kroku/ krok 1 - ze "Standard" do "mixed";

if (radioButton1->Checked)

{

label2->ForeColor::set(System::Drawing::SystemColors::ControlText);

KrokPoKroku3K(textBox1->Text);

}

button1->Text::set("&Dalej");

button1->Enabled::set(true);

}

else // deszyfrowanie bez opcji Krok po kroku;

{

// deszyfrowanie standardowe;

if (Form1::radioButton1->Checked::get())

textBox2->Text::set(OdszyfrujStandard(textBox1->Text));

// deszyfrowanie kompleksowe;

if (Form1::radioButton2->Checked::get())

textBox2->Text::set(OdszyfrujKompleks(textBox1->Text));

}

}

// po zakończeniu standardowego szyfrowania krok po kroku;

if ( (label2->Text == "...Krok4...Gotowe") && (radioButton1->Checked) )

{

// są przywracane wszystkie wyłączone wcześniej opcje;

button1->Text::set("S&zyfruj");

button1->Enabled::set(false);

Form1::toolStripStatusLabel1->Text::set("Gotowe");

zapiszToolStripMenuItem->Enabled::set(true);

edycjaToolStripMenuItem->Enabled::set(true);

toolStrip1->Enabled::set(true);

comboBox1->Enabled::set(true);

checkBox1->Enabled::set(true);

textBox1->Enabled::set(true);

button3->Enabled::set(true);

button2->Enabled::set(true);

radioButton1->Enabled::set(true);

radioButton2->Enabled::set(true);

}

// po zakończeniu kompleksowego szyfrowania krok po kroku;

if ( (label2->Text == "...Krok6...Gotowe") && (radioButton2->Checked) )

{

button1->Text::set("S&zyfruj");

button1->Enabled::set(false);

Form1::toolStripStatusLabel1->Text::set("Gotowe");

zapiszToolStripMenuItem->Enabled::set(true);

edycjaToolStripMenuItem->Enabled::set(true);

toolStrip1->Enabled::set(true);

comboBox1->Enabled::set(true);

checkBox1->Enabled::set(true);

textBox1->Enabled::set(true);

button3->Enabled::set(true);

button2->Enabled::set(true);

radioButton2->Enabled::set(true);

radioButton1->Enabled::set(true);

}

// po zakończeniu standardowego deszyfrowania krok po kroku;

if ( (label2->Text == "...Krok3...Gotowe") && (radioButton1->Checked) && (comboBox1->SelectedIndex == 1) )

{

button1->Text::set("&Odszyfruj");

button1->Enabled::set(false);

Form1::toolStripStatusLabel1->Text::set("Gotowe");

zapiszToolStripMenuItem->Enabled::set(true);

edycjaToolStripMenuItem->Enabled::set(true);

toolStrip1->Enabled::set(true);

comboBox1->Enabled::set(true);

checkBox1->Enabled::set(true);

textBox1->Enabled::set(true);

button3->Enabled::set(true);

button2->Enabled::set(true);

radioButton1->Enabled::set(true);

radioButton2->Enabled::set(true);

}

// po zakończeniu kompleksowego deszyfrowania krok po kroku;

if ( (label2->Text == "...Krok5...Gotowe") && (radioButton2->Checked) && (comboBox1->SelectedIndex == 1) )

{

button1->Text::set("&Odszyfruj");

button1->Enabled::set(false);

Form1::toolStripStatusLabel1->Text::set("Gotowe");

zapiszToolStripMenuItem->Enabled::set(true);

edycjaToolStripMenuItem->Enabled::set(true);

toolStrip1->Enabled::set(true);

comboBox1->Enabled::set(true);

checkBox1->Enabled::set(true);

textBox1->Enabled::set(true);

button3->Enabled::set(true);

button2->Enabled::set(true);

radioButton2->Enabled::set(true);

radioButton1->Enabled::set(true);

}

// przywrócenie tekstu Gotowe na pasku stanu gdy

// nie używamy opcji Krok po kroku;

if ( !(checkBox1->Checked) )

Form1::toolStripStatusLabel1->Text::set("Gotowe");

}

// funkcja sprawdzająca czy nie ma błędów we wprowadzonym tekście

// przeznaczonym do zaszyfrowania;

// zwraca kod błędu error;

public: int SprawdzPoprawnosc(System::String ^ sourceText)

{

int error = 0;

int dlugosc = sourceText->Length;

int i = 0;

int j = 0;

cli::array<System::String ^, 1> ^ wyrazy;

int ileWyrazow = 0;

int dlugoscWyrazu = 0;

// tablica znaków typu String/ zawiera wszystkie dozwolone znaki

// jakie można wprowadzić do pola tekstowego w celu zaszyfrowania;

cli::array<System::String ^, 1> ^ dozwoloneZnaki = {"a","ą","b","c","ć","d","e","ę","f","g","h","i","j","k","l","ł","m","n","ń","o","ó","p","q","r","s","ś","t","u","v","w","x","y","z","ź","ż","A","Ą","B","C","Ć","D","E","Ę","F","G","H","I","J","K","L","Ł","M","N","Ń","O","Ó","P","Q","R","S","Ś","T","U","V","W","X","Y","Z","Ź","Ż","0","1","2","3","4","5","6","7","8","9",".",",","#","!","@","$","%","^","&","\*","(",")","-","`","\_","+","=","[","]","{","}","\\","|",":",";","\"","\'","<",">","/","?"," "};

System::String^ znak = "";

// pętla, w której są zawarte instrukcje sprawdzania błędów;

while(1)

{

// występuje gdy długość tekstu nie przekracza jednego znaku;

if (dlugosc < 2)

error = 1;

// po każdym sprawdzeniu błędu pojawia się ta instruckja, która sprawia,

// że jeśli wykryto błąd to przerywa dalsze sprawdzanie błędów;

if (error != 0)

break;

// występuje gdy tekst zaczyna się lub kończy spacją;

if ( (sourceText->StartsWith(" ")) || (sourceText->EndsWith(" ")) )

error = 2;

if (error != 0)

break;

// występuje gdy obok jednej spacji wykryje od razu drugą;

for (i = 0; i < dlugosc; i++)

{

if ( ( sourceText->Substring(i) )->Contains(" ") )

{

i = sourceText->IndexOf(' ', i);

if ( ( sourceText->Substring(i+1, 1) ) == " " )

{

error = 3;

break;

}

}

else

break;

}

if (error != 0)

break;

// dzieli tekst na wyrazy;

wyrazy = sourceText->Split(' ');

ileWyrazow = wyrazy->Length;

// szuka przynajmniej jednego wyrazu dłuższego niż 1 znak;

for (i = 0; i < ileWyrazow; i++)

{

dlugoscWyrazu = wyrazy[i]->Length;

if (dlugoscWyrazu > 1)

break;

}

// jeśli nie znalazło, czyli pętla doszła do końca, zgłasza błąd;

if (i == ileWyrazow)

error = 4;

if (error != 0)

break;

// występuje jeśli tekst zawiera któryś z tych niedozwolonych znaków;

if ( (sourceText->Contains("~")) || (sourceText->Contains("#")) )

error = 5;

if (error != 0)

break;

// występuje jeśli wprowadzimy za dużo wyrazów;

if (ileWyrazow > 999)

error = 6;

if (error != 0)

break;

// występuje gdy znajdzie w tekście użycie klawisza Enter;

if ( sourceText->Contains("\n") )

error = 7;

if (error != 0)

break;

// występuje jeśli przynajmniej jeden wyraz przekracza dozwoloną długość;

for (i = 0; i < ileWyrazow; i++)

{

dlugoscWyrazu = wyrazy[i]->Length;

if (dlugoscWyrazu > 99)

{

error = 8;

break;

}

}

if (error != 0)

break;

// sprawdza każdy znak w tekście i porównuje z tablicą dozwolonych

// znaków; jeśli nie znajdzie go w tablicy to znaczy, że jest niedozwolony

// i zgłasza błąd;

for (i = 0; i < dlugosc; i++)

{

znak = sourceText->Substring(i, 1);

for (j = 0; j < dozwoloneZnaki->Length; j++)

{

if ( znak == dozwoloneZnaki[j] )

break;

}

if (j == dozwoloneZnaki->Length)

{

error = 10;

break;

}

}

if (error != 0)

break;

//następne instrukcje sprawdzania błędów wprowadzać tutaj;\

// jeśli nie wykryto żadnego błędu przerywa pętlę;

break;

}

// w zależności od numeru błędu zgłasza odpowiedni komunikat;

// jeśli nie ma błędu czyli error ma wartość 0, przerywa instrukcję;

switch(error)

{

case 0:

break;

case 1:

System::Windows::Forms::MessageBox::Show("Tekst musi zawierać przynajmniej dwa znaki!", "Błąd 01", System::Windows::Forms::MessageBoxButtons::OK, System::Windows::Forms::MessageBoxIcon::Error);

break;

case 2:

System::Windows::Forms::MessageBox::Show("Tekst nie może zaczynać się ani kończyć spacją!", "Błąd 02", System::Windows::Forms::MessageBoxButtons::OK, System::Windows::Forms::MessageBoxIcon::Error);

break;

case 3:

System::Windows::Forms::MessageBox::Show("Wyrazy mogą być oddzielane tylko jedną spacją!\nNie używaj kilku spacji obok siebie.", "Błąd 03", System::Windows::Forms::MessageBoxButtons::OK, System::Windows::Forms::MessageBoxIcon::Error);

break;

case 4:

System::Windows::Forms::MessageBox::Show("Przynajmniej jeden wyraz musi zawierać więcej niż jeden znak!", "Błąd 04", System::Windows::Forms::MessageBoxButtons::OK, System::Windows::Forms::MessageBoxIcon::Error);

break;

case 5:

System::Windows::Forms::MessageBox::Show("Tekst zawiera niedozwolony znak!\nNie używaj znaków:\t\'~\', \'#\'.", "Błąd 05", System::Windows::Forms::MessageBoxButtons::OK, System::Windows::Forms::MessageBoxIcon::Error);

break;

case 6:

System::Windows::Forms::MessageBox::Show("Tekst zawiera zbyt dużo wyrazów!\nMaksymalna ilość to 999.", "Błąd 06", System::Windows::Forms::MessageBoxButtons::OK, System::Windows::Forms::MessageBoxIcon::Error);

break;

case 7:

System::Windows::Forms::MessageBox::Show("Nie wolno używać klawisza Enter podczas wpisywania tekstu!", "Błąd 07", System::Windows::Forms::MessageBoxButtons::OK, System::Windows::Forms::MessageBoxIcon::Error);

break;

case 8:

System::Windows::Forms::MessageBox::Show("Przekroczono dozwoloną długość wyrazu!\nMaksymalna długość to 99 znaków.", "Błąd 08", System::Windows::Forms::MessageBoxButtons::OK, System::Windows::Forms::MessageBoxIcon::Error);

break;

case 10:

System::Windows::Forms::MessageBox::Show("Tekst zawiera nieobsługiwany znak!", "Błąd 10", System::Windows::Forms::MessageBoxButtons::OK, System::Windows::Forms::MessageBoxIcon::Error);

break;

default:

System::Windows::Forms::MessageBox::Show("Błąd nieznanego pochodzenia.", "Błąd 00", System::Windows::Forms::MessageBoxButtons::OK, System::Windows::Forms::MessageBoxIcon::Error);

break;

}

return error;

}

// funkcja dzieląca tekst na wyrazy przez wykrywanie spacji;

public: cli::array<System::String ^, 1> ^ Wyraz(System::String ^ zdanie)

{

cli::array<wchar\_t, 1> ^ spacja = {' '};

cli::array<System::String ^, 1> ^ wyrazy = zdanie->Split(spacja, System::StringSplitOptions::RemoveEmptyEntries);

return wyrazy;

}

// funkcja, która odwraca wyrazy zapisując je od końca;

public: cli::array<System::String ^, 1> ^ Invert(cli::array<System::String ^, 1> ^ wyrazy)

{

int ileWyrazow = wyrazy->GetLength(0);

int i;

cli::array<wchar\_t, 1> ^ znaki;

int ileZnakow;

wchar\_t bufor;

int j;

int licznik;

int k;

for (i = 0; (i < ileWyrazow); i++)

{

znaki = wyrazy[i]->ToCharArray(); // zamienia każdy wyraz na tablicę znaków;

ileZnakow = znaki->GetLength(0); // mierzy jego długość;

// zapamiętuje jeden znak w zmiennej bufor, zmienia go na inny

// a następnie znak z bufora zastępuje kolejnym;

for ( (j = (ileZnakow-1), licznik = 0); (licznik <= j); (j--, licznik++) )

{

bufor = znaki[j];

znaki[j] = znaki[licznik];

znaki[licznik] = bufor;

}

wyrazy[i] = "";

// każdy znak jest zamieniany na typ String i dopisywany

// do tablicy wyrazy;

for (k=0; k<ileZnakow; k++)

{

wyrazy[i] += wchar\_t::ToString(znaki[k]);

}

// do każdego wyrazu, który nie jest ostatnim, dopisujemy spację;

if (i < (ileWyrazow - 1) )

wyrazy[i] += " ";

}

return wyrazy;

}

// funkcja łączy wyrazy z tablicy wyrazów w jeden cały tekst typu String;

public: System::String ^ ZwrocZdanie(cli::array<System::String ^, 1> ^ wyrazy)

{

System::String ^ odwroconeZdanie;

odwroconeZdanie = String::Concat(wyrazy);

return odwroconeZdanie;

}

// Ta funkcja przeprowadza standardowe szyfrowanie tekstu;

public: System::String ^ DisplayStandard(System::String ^ sourceText)

{

// wyrazy są odwracane;

cli::array<System::String ^, 1> ^ rozbiteZdanie = Wyraz(sourceText);

cli::array<System::String ^, 1> ^ odwroconeWyrazy = Invert(rozbiteZdanie);

System::String ^ reversedText = ZwrocZdanie(odwroconeWyrazy);

// znaki w wyrazach są przestawiane;

cli::array<System::String ^, 1> ^ rozbityText = Wyraz(reversedText);

cli::array<System::String ^, 1> ^ przestawioneWyrazy = Reorder(rozbityText);

System::String ^ mixedText = ZwrocZdanie(przestawioneWyrazy);

// tworzona jest postać segmentowana/ jest ona jednak wykorzystywana

// tylko w szyfrowaniu Krok po kroku;

cli::array<System::String ^, 1> ^ segmentowaneZdanie = Wyraz(mixedText);

System::String ^ posegmentowanyText = Segment(segmentowaneZdanie);

// tworzona jest gotowa forma zaszyfrowana wykorzystując tekst

// z poprzestawianymi znakami a nie posegmentowany;

System::String ^ gotowyStandard = MakeStandard(segmentowaneZdanie);

return gotowyStandard;

}

// funkcja odpowiedzialna za przestawianie znaków w odwróconych wyrazach;

public: cli::array<System::String ^, 1> ^ Reorder(cli::array<System::String ^, 1> ^ wyrazy)

{

int ileWyrazow = wyrazy->GetLength(0);

int i;

cli::array<wchar\_t, 1> ^ znaki;

cli::array<wchar\_t, 1> ^ wynik;

int ileZnakow;

int j;

int licznik;

int czyParzysta;

int k;

for (i = 0; (i < ileWyrazow); i++)

{

znaki = wyrazy[i]->ToCharArray();

wynik = wyrazy[i]->ToCharArray();

ileZnakow = znaki->GetLength(0);

// począwszy od drugiego znaku pobierany jest co drugi znak

// i dopisywany do wyniku (aż do końca wyrazu);

for ( licznik = 1, j = 0; (licznik < ileZnakow); j++ )

{

wynik[j] = znaki[licznik];

licznik += 2;

}

// w zależności czy liczba znaków w wyrazie jest parzysta czy nie

// w zmiennej czyParzysta zapisujemy index przedostatniego lub

// ostatniego znaku;

if ( (ileZnakow%2) == 0 )

czyParzysta = ileZnakow-2;

else

czyParzysta = ileZnakow-1;

// do wyniku dopisujemy pozostałe znaki zaczynając od końca wyrazu

// albo od ostatniego albo od przedostatniego znaku;

for ( licznik = czyParzysta; licznik >= 0; j++ )

{

wynik[j] = znaki[licznik];

licznik -= 2;

}

// czyścimy dany element tablicy wyrazy, aby poniżej zapisać w nim

// cały wyraz z tablicy wynik, który przed chwilą stworzyliśmy;

wyrazy[i] = "";

for (k=0; k<ileZnakow; k++)

{

wyrazy[i] += wchar\_t::ToString(wynik[k]);

}

// jeśli nie jest to ostatni wyraz, dopisujemy jeszcze spację;

if (i < (ileWyrazow - 1) )

wyrazy[i] += " ";

}

return wyrazy;

}

// funkcja z tekstu "mixed" tworzy tekst z segmentami;

public: System::String ^ Segment(cli::array<System::String ^, 1> ^ wyrazy)

{

int i;

int ileWyrazow = wyrazy->GetLength(0);

int ileZnakow;

int j;

int s;

int theLongest = 1;

System::String ^ wynik = "";

for (j = 0, s = j+1; j < theLongest; j++, s++)

{

// najpierw do wyniku dopisujemy numer segmentu

// zamieniając liczbę integer na String;

wynik += System::String::Concat(s);

if (j == 0)

{

// dla pierwszego znaku z każdego wyrazu (j == 0):

for (i = 0; i < ileWyrazow; i++)

{

ileZnakow = wyrazy[i]->Length;

// gdy skończy nam się dany wyraz dopisujemy tyldę;

if (j >= ileZnakow)

wynik += "~";

else

// zwykle dopisujemy znak z każdego wyrazu o tym samym indeksie;

wynik += wyrazy[i]->Substring(j, 1);

// mierzymy długość każdego wyrazu i rekordową zapisujemy

// w zmiennej theLongest;

if (ileZnakow > theLongest)

theLongest = ileZnakow;

}

}

else

{

// to samo co wyżej lecz dla wszystkich pozostałych znaków

// oprócz pierwszego (j > 0)

// bez sprawdzania ile znaków ma najdłuższy wyraz;

for (i = 0; i < ileWyrazow; i++)

{

ileZnakow = wyrazy[i]->Length;

if (j >= ileZnakow)

wynik += "~";

else

wynik += wyrazy[i]->Substring(j, 1);

}

} // zakończenie else od głównej funkcji if

} // zakończenie głównej pętli for

if (wynik == "1")

wynik = "";

return wynik;

} // zakończenie definicji funkcji

// funkcja tworzy gotową postać zaszyfrowaną z postaci "mixed";

public: System::String ^ MakeStandard(cli::array<System::String ^, 1> ^ wyrazy)

{

int i = 0;

int ileWyrazow = wyrazy->GetLength(0);

int j = 0;

int l = 0;

cli::array<wchar\_t, 1> ^ tymczasowa;

int czyTylda = 0;

System::String ^ text = "";

int rozmiar = 0;

int dlugosc = 0;

// sprawdzamy jaka jest długość najdłuższego wyrazu;

while(i < ileWyrazow)

{

dlugosc = wyrazy[i]->Length;

if (dlugosc > rozmiar)

rozmiar = dlugosc;

i++;

}

// ustalamy rozmiar tablicy String;

cli::array<System::String ^, 1> ^ wynik;

System::Array::Resize(wynik, rozmiar);

// do wyniku dopisujemy kolejne znaki kolejnych wyrazów bez

// numeracji segmentów

while(j < rozmiar)

{

for (i = 0; i < ileWyrazow; i++)

{

dlugosc = wyrazy[i]->Length;

if (j >= dlugosc)

wynik[j] += "~";

else

wynik[j] += wyrazy[i]->Substring(j, 1);

}

j++;

}

// usuwamy tyldy z poszczególnych wyrazów;

for (i = 0; i < rozmiar; i++)

{

l = 0;

czyTylda = wynik[i]->Contains("~");

while(czyTylda == 1) // jeśli wyraz zawiera tyldę;

{

dlugosc = wynik[i]->Length;

j = wynik[i]->IndexOf("~"); // znajdujemy ją;

tymczasowa = wynik[i]->ToCharArray(); // zamieniamy wyraz na tablicę znaków;

tymczasowa[j] = '#'; // znalezioną tyldę zamieniamy na hash;

wynik[i] = ""; // wynik czyścimy;

for (int k = 0; k < dlugosc; k++)

{

// dopisujemy do wyniku całą zawartość tymczasowej łącznie z hashem;

wynik[i] += wchar\_t::ToString(tymczasowa[k]);

}

// z pozostałych wyrazów usuwamy tyldy znajdujące się na tej

// samej pozycji co nasz hash;

for (int s = i + 1; s < rozmiar; s++)

{

wynik[s] = wynik[s]->Remove( (j-l), 1);

}

// powtarzamy pętlę dopóki są tyldy w wyrazach;

l++;

czyTylda = wynik[i]->Contains("~");

}

}

// na początku wynikowego tekstu zapisujemy symbolikę

// jeśli długość wyrazów nie przekracza 9 dopisujemy pojedyncze

// zero oraz rekordową długość wyrazu; w przeciwnym razie zapisujemy

// poprostu długość wyrazu;

if (rozmiar < 10)

text += System::String::Concat("0", rozmiar);

else

text += System::String::Concat(rozmiar);

// następnie drugi symbol; w zależności od ilości wyrazów

// dopisujemy (lub nie) przed nimi jeszcze albo jedno albo dwa zera;

if (ileWyrazow < 10)

text += System::String::Concat("00", ileWyrazow);

if ( (ileWyrazow > 9) && (ileWyrazow < 100) )

text += System::String::Concat("0", ileWyrazow);

if (ileWyrazow > 99)

text += System::String::Concat(ileWyrazow);

// ostatnim krokiem jest dopisanie za symboliką wszystkich

// przetworzonych wyrazów z tablicy wynik;

text += System::String::Concat(wynik);

return text;

}

// zdarzenia, które wywołuje wciśnięcie przycisku Sprawdź;

private: System::Void button2\_Click(System::Object^ sender, System::EventArgs^ e) {

Form1::toolStripStatusLabel1->Text::set("Sprawdzanie poprawności tekstu");

int error = 1;

int stopien = 0;

// jeśli tryb jest ustawiony na szyfrowanie to wywoływana jest funkcja

// SprawdzPoprawnosc; jeśli nie wystąpi błąd to error przyjmie wartość 0

// w przeciwnym wypadku przyjmuje wartość kodu błędu;

if ( (comboBox1->SelectedIndex::get()) == 0 )

error = SprawdzPoprawnosc(textBox1->Text);

else

{

// przy deszyfrowaniu najpierw uruchamiana jest funkcja CzyDobrzeZaszyfrowane

// i jeśli nie wykaże błędu startuje funkcja StandardCzyKompleks;

error = CzyDobrzeZaszyfrowane(textBox1->Text);

if (error == 0)

stopien = StandardCzyKompleks(textBox1->Text);

}

// jeśli nie ma błędu i jest zaszyfrowane standardowo;

if ( (error == 0) && (stopien == 1) )

{

label2->Text::set("Tekst zaszyfrowany standardowo");

label2->ForeColor::set(System::Drawing::Color::Green);

button1->Enabled::set(true);

button2->Enabled::set(false);

Form1::toolStripStatusLabel1->Text::set("Gotowe");

radioButton1->Checked::set(true);

radioButton2->Checked::set(false);

}

// jeśli nie ma błędu i jest zaszyfrowane kompleksowo;

if ( (error == 0) && (stopien == 2) )

{

label2->Text::set("Tekst zaszyfrowany kompleksowo");

label2->ForeColor::set(System::Drawing::Color::Green);

button1->Enabled::set(true);

button2->Enabled::set(false);

Form1::toolStripStatusLabel1->Text::set("Gotowe");

radioButton1->Checked::set(false);

radioButton2->Checked::set(true);

}

// jeśli nie ma błędu i jest gotowe do zaszyfrowania;

if ( (error == 0) && (stopien == 0) )

{

label2->Text::set("OK");

label2->ForeColor::set(System::Drawing::Color::Green);

button1->Enabled::set(true);

button2->Enabled::set(false);

Form1::toolStripStatusLabel1->Text::set("Gotowe");

}

// jeśli któraś z funkcji sprawdzających zgłosiła błąd;

if ( (error != 0) || (stopien == 3) )

{

textBox2->Text::set(System::String::Empty);

button1->Enabled::set(false);

button2->Enabled::set(false);

label2->Text::set("Błąd w tekście!");

label2->ForeColor::set(System::Drawing::Color::Red);

Form1::toolStripStatusLabel1->Text::set("Gotowe");

}

}

private: System::Void textBox1\_TextChanged(System::Object^ sender, System::EventArgs^ e) {

label2->Text::set("Sprawdź poprawność");

label2->ForeColor::set(System::Drawing::SystemColors::ControlText);

if (button1->Enabled)

button1->Enabled::set(false);

if ( !(button2->Enabled) )

button2->Enabled::set(true);

}

// kliknięcie na opcję Zamknij w menu Plik powoduje zamknięcie aplikcaji;

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

Form1::toolStripStatusLabel1->Text::set("Zamykanie programu");

System::Windows::Forms::Application::Exit();

}

private: System::Void button3\_Click(System::Object^ sender, System::EventArgs^ e) {

textBox1->Clear();

textBox2->Clear();

}

// funkcja, która powoduje dalsze szyfrowanie tekstu zaszyfrowanego

// już standardowo;

public: System::String ^ DisplayComplex(System::String ^ sourceText)

{

System::String ^ szesnastkowy = MakeHex(sourceText);

System::String ^ binarny = MakeBinary(szesnastkowy);

return binarny;

}

public: System::String ^ MakeHex(System::String ^ Text)

{

int i = 0;

int ileZnakow = Text->Length;

System::String ^ znak = "";

// tablice "pierwotny" oraz "koncowy" stanowią tzw. klucz do zaszyfrowania

// tekstu na poziomie kompleksowym;

cli::array<System::String ^, 1> ^ pierwotny = {"a","ą","b","c","ć","d","e","ę","f","g","h","i","j","k","l","ł","m","n","ń","o","ó","p","q","r","s","ś","t","u","v","w","x","y","z","ź","ż","A","Ą","B","C","Ć","D","E","Ę","F","G","H","I","J","K","L","Ł","M","N","Ń","O","Ó","P","Q","R","S","Ś","T","U","V","W","X","Y","Z","Ź","Ż","0","1","2","3","4","5","6","7","8","9",".",",","#","!","@","$","%","^","&","\*","(",")","-","`","\_","+","=","[","]","{","}","\\","|",":",";","\"","\'","<",">","/","?"};

cli::array<System::String ^, 1> ^ koncowy = {"20","31","0F","43","54","01","44","30","62","10","6D","55","42","02","0E","72","1F","32","21","53","45","11","2F","61","56","41","33","0D","1E","46","22","6C","63","52","40","34","2E","03","12","1D","0C","57","23","47","60","6B","64","51","35","3F","04","2D","24","13","1C","58","0B","48","5F","6A","6E","50","65","36","3E","25","14","2C","49","1B","05","59","5E","4F","37","2B","26","0A","15","3D","4A","66","5A","6F","5D","69","38","06","16","27","4B","4E","5B","3C","2A","67","09","70","19","4C","39","28","17","5C","07","71","68","4D","3A","29","08"};

int indexWtablicy = 0;

System::String ^ hexText = "";

// bierzemy pojedynczy znak z tekstu, wyszukujemy jego indeks w tablicy

// pierwotny, zastępujemy dwoma znakami z tablicy koncowy o tym samym

// indeksie aż wreszcie dopisujemy je do wynikowego "hexTextu";

for (i = 0; i < ileZnakow; i++)

{

znak = Text->Substring(i, 1);

indexWtablicy = pierwotny->IndexOf(pierwotny, znak);

znak = koncowy[indexWtablicy];

hexText += znak;

}

return hexText;

}

// gdy zmienimy poziom już po zaszyfrowaniu tekstu, to zdarzenie

// czyści pole z wynikiem oraz uaktywnia przycisk Szyfruj w celu

// ponownego zaszyfrowania lecz na innym poziomie;

private: System::Void radioButton1\_CheckedChanged(System::Object^ sender, System::EventArgs^ e) {

textBox2->Clear();

if ( !(Form1::button2->Enabled::get()) )

Form1::button1->Enabled::set(true);

}

// funkcja zamienia tekst heksadecymalny na postać binarną

// co jest ostatnim krokiem w szyfrowaniu kompleksowym;

public: System::String^ MakeBinary(System::String^ hexText)

{

int i = 0;

int ileZnakow = hexText->Length;

System::String^ znak = "";

System::String^ binarnyText = "";

cli::array<System::String^, 1>^ tablica = {"0","1","2","3","4","5","6","7","8","9","A","B","C","D","E","F"};

int id = 0;

// każdy pojedynczy znak z tekstu występuje w tablicy o nazwie "tablica"

// na podstawie jego indeksu w tej tablicy przyporządkowywany jest ciąg

// binarny odpowiadający wartości danego znaku;

for (i = 0; i < ileZnakow; i++)

{

znak = hexText->Substring(i, 1);

id = System::Array::IndexOf(tablica, znak);

switch(id)

{

case(0):

binarnyText += "0000";

break;

case(1):

binarnyText += "0001";

break;

case(2):

binarnyText += "0010";

break;

case(3):

binarnyText += "0011";

break;

case(4):

binarnyText += "0100";

break;

case(5):

binarnyText += "0101";

break;

case(6):

binarnyText += "0110";

break;

case(7):

binarnyText += "0111";

break;

case(8):

binarnyText += "1000";

break;

case(9):

binarnyText += "1001";

break;

case(10):

binarnyText += "1010";

break;

case(11):

binarnyText += "1011";

break;

case(12):

binarnyText += "1100";

break;

case(13):

binarnyText += "1101";

break;

case(14):

binarnyText += "1110";

break;

case(15):

binarnyText += "1111";

break;

}

// znalezione wartości binarne dopisywane są do wynikowej

// zmiennej binarnyText;

}

return binarnyText;

}

// jeśli klikniemy pozycję kopiuj w menu kontekstowym górnego pola tekstowego

// funkcja ta skopiuje zaznaczony w tym polu tekst do schowka Windows;

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

if ( (textBox1->SelectionLength) > 0 )

System::Windows::Forms::Clipboard::SetText(textBox1->SelectedText);

}

// to samo jak wyżej lecz dotyczy dolnego pola tekstowego i jego menu;

private: System::Void kopiujToolStripMenuItem1\_Click(System::Object^ sender, System::EventArgs^ e) {

if ( (textBox2->SelectionLength) > 0 )

System::Windows::Forms::Clipboard::SetText(textBox2->SelectedText);

}

// ten kod wkleja zawartość schowka do górnego pola tekstowego;

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

textBox1->Paste(System::Windows::Forms::Clipboard::GetText());

}

// tutaj działa opcja Zaznacz wszystko górnego menu kontekstowego;

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

if (textBox1->Focused)

textBox1->SelectAll();

else

textBox1->SelectAll();

}

// tutaj czyścimy zawartość górnego pola za pomocą przycisku

// Wyczyść w menu kontekstowym;

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

textBox1->Clear();

}

// cofa wpisywanie tekstu do pierwszego pola;

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

textBox1->Undo();

}

// zaznacza zawartość dolnego pola tekstowego;

private: System::Void zaznaczWszystkoToolStripMenuItem1\_Click(System::Object^ sender, System::EventArgs^ e) {

textBox2->SelectAll();

}

private: System::Void skup(System::Object^ sender, System::EventArgs^ e) {

textBox1->Focus();

}

private: System::Void skup2(System::Object^ sender, System::EventArgs^ e) {

textBox2->Focus();

}

// czyści zawartość dolnego pola tekstowego;

private: System::Void wyczyśćToolStripMenuItem1\_Click(System::Object^ sender, System::EventArgs^ e) {

textBox2->Clear();

}

// kliknięcie przycisku Zapisz w menu Plik powoduje otwarcie okna

// zapisywania pliku;

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

Form1::saveFileDialog1->ShowDialog();

}

// po wciśnięciu OK tworzony jest plik \*.txt, do którego zapisywana

// jest zawartość dolnego pola tekstowego;

private: System::Void saveFileDialog1\_FileOk(System::Object^ sender, System::ComponentModel::CancelEventArgs^ e) {

Form1::saveFileDialog1->DefaultExt::set("txt");

System::String^ nazwa = Form1::saveFileDialog1->FileName;

System::IO::StreamWriter^ strumien = System::IO::File::CreateText(nazwa);

strumien->Write(textBox2->Text);

strumien->Flush();

}

// bezpośrednio po uruchomieniu aplikacji tworzona jest tablica obiektów

// do której zapisywane są elementy rozsuwanego pola do wyboru trybu pracy

// oraz domyślnie jest ustawiany pierwszy element jako tryb startowy;

private: System::Void Form1\_Load(System::Object^ sender, System::EventArgs^ e) {

cli::array<System::Object^, 1>^ tablicaObiektow;

System::Array::Resize(tablicaObiektow, 2);

comboBox1->Items->CopyTo(tablicaObiektow, 0);

comboBox1->SelectedItem::set(tablicaObiektow[0]);

}

// zmiana trybu pracy - szyfrowanie lub deszyfrowanie;

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

label2->Text::set("Sprawdź poprawność");

label2->ForeColor::set(System::Drawing::SystemColors::ControlText);

if ( !(button2->Enabled::get()) )

button2->Enabled::set(true);

if ( button1->Enabled::get() )

button1->Enabled::set(false);

// po zmianie opcji tekst przycisku Szyfruj/Odszyfruj również ulega

// zmianie/ przyciski wyboru poziomu szyfrowania są ukrywane w trybie

// deszyfrowania i aktywowane w trybie przeciwnym;

// ilość możliwych do wprowadzenia znaków w górnym polu tekstowym

// także ulega zmianie;

if ( (comboBox1->SelectedIndex::get()) == 1 )

{

button1->Text::set("&Odszyfruj");

radioButton1->Enabled::set(false);

radioButton2->Enabled::set(false);

textBox1->MaxLength::set(262200);

}

else

{

button1->Text::set("S&zyfruj");

radioButton1->Enabled::set(true);

radioButton2->Enabled::set(true);

textBox1->MaxLength::set(32767);

}

textBox2->Clear();

}

// funkcja sprawdzania błędów w zaszyfrowanym tekście zanim

// zostanie on odszyfrowany;

// zawarte tu mechanizmy są wspólne zarówno dla tekstu zaszyfrowanego

// standardowo jak i kompleksowo;

public: int CzyDobrzeZaszyfrowane(System::String ^ sourceText)

{

int error = 0;

int dlugosc = sourceText->Length;

int i = 0;

System::String^ cyfry = "";

int ileWyrazow = 0;

int dlugoscWyrazu = 0;

while(1)

{

// najkrótszy możliwy szyfr ma 7 znaków;

if (dlugosc < 7)

error = 9;

if (error != 0)

break;

// szyfr nie może zawierać tyld ani spacji;

if ( (sourceText->Contains("~")) || (sourceText->Contains(" ")) )

error = 9;

if (error != 0)

break;

// nie można używać klawisza Enter;

if ( sourceText->Contains("\n") )

error = 7;

if (error != 0)

break;

// sprawdzamy czy pierwsze pięć elementów stanowią cyfry;

cyfry = "0123456789";

for (i = 0; i < 5; i++)

{

if ( !(cyfry->Contains(sourceText->Substring(i, 1))) )

{

error = 9;

break;

}

}

if (error != 0)

break;

//następne instrukcje sprawdzania błędów

break;

}

// wypisywane komunikaty błędów;

switch(error)

{

case 0:

break;

case 9:

System::Windows::Forms::MessageBox::Show("Tekst niezaszyfrowany lub zaszyfrowany nieprawidłowo!", "Błąd 09", System::Windows::Forms::MessageBoxButtons::OK, System::Windows::Forms::MessageBoxIcon::Error);

break;

case 7:

System::Windows::Forms::MessageBox::Show("Nie wolno używać klawisza Enter podczas wpisywania tekstu!", "Błąd 07", System::Windows::Forms::MessageBoxButtons::OK, System::Windows::Forms::MessageBoxIcon::Error);

break;

default:

System::Windows::Forms::MessageBox::Show("Błąd nieznanego pochodzenia.", "Błąd 00", System::Windows::Forms::MessageBoxButtons::OK, System::Windows::Forms::MessageBoxIcon::Error);

break;

}

return error;

}

// funkcja zamienia dwa umowne symbole zawarte w tekście zaszyfrowanym

// kompleksowo na liczby całkowite, aby umożliwić ich sprawdzenie

// w funkcji StandardCzyKompleks;

public: int MakeSymbol(System::String^ sourceText)

{

int i = 0;

int j = 0;

int dlugosc = sourceText->Length;

System::String^ znaki = "";

System::String^ symbol = "";

int id = 0;

cli::array<System::String^, 1>^ binarna = {"0000","0001","0010","0011","0100","0101","0110","0111","1000","1001","1010","1011","1100","1101","1110","1111"};

cli::array<System::String^, 1>^ cyfry = {"0","1","2","3","4","5","6","7","8","9","A","B","C","D","E","F"};

cli::array<System::String^, 1>^ hex = {"05","59","5E","4F","37","2B","26","0A","15","3D"};

// zamieniamy tekst binarny na heksadecymalny z tablicy cyfry;

for (i = 0; i < dlugosc; )

{

znaki = sourceText->Substring(i, 4);

id = System::Array::IndexOf(binarna, znaki);

symbol += cyfry[id];

i += 4;

}

dlugosc = symbol->Length;

id = 0;

for (i = 0; i < dlugosc; )

{

// sprawdzamy czy wszystkie znaki z tekstu należą do tablicy hex

// w tablicy tej znajdują się wartości oznaczające cyfry od 0 do 9

// jeśli jest jakiś znak, który nie występuje w tablicy zgłaszamy

// kod błędu id = 1;

znaki = symbol->Substring(i, 2);

for (j = 0; j < hex->Length; j++)

{

if (znaki == hex[j])

break;

}

if (j == hex->Length)

{

id = 1;

break;

}

i += 2;

}

// jeśli nie było błędu to tekst zamieniamy na postać heksadecymalną

// z tablicy cyfry a następnie ...

if (id == 0)

{

sourceText = symbol;

symbol = "";

dlugosc = sourceText->Length;

for (i = 0; i < dlugosc; )

{

znaki = sourceText->Substring(i, 2);

id = System::Array::IndexOf(hex, znaki);

symbol += cyfry[id];

i += 2;

}

// ... zamieniamy go na liczby całkowite

// uzyskamy w ten sposób wartość symbolu;

dlugosc = System::Convert::ToInt32(symbol);

}

else

dlugosc = 9999;

// jeśli wystąpił błąd jest on zgłaszany przez oznaczenie zmiennej

// dlugosc wartością 9999;

return dlugosc;

}

// funkcja ta rozróżnia czy tekst został zaszyfrowany standardowo czy kompleksowo

// przeprowadzając przy tym ostatnie sprawdzenia błędów;

public: int StandardCzyKompleks(System::String^ sourceText)

{

int stopien = 1;

int i;

int dlugosc = sourceText->Length;

bool pierwszy = false;

bool drugi = false;

int symbol1 = 0;

int symbol2 = 0;

int startIndex = 5;

int ileZostalo = 0;

System::String^ wynik = "";

System::String^ wycinek = "";

int ileHashy = 0;

int hashIndex = 0;

int j = 0;

cli::array<System::String ^, 1> ^ dozwoloneZnaki = {"a","ą","b","c","ć","d","e","ę","f","g","h","i","j","k","l","ł","m","n","ń","o","ó","p","q","r","s","ś","t","u","v","w","x","y","z","ź","ż","A","Ą","B","C","Ć","D","E","Ę","F","G","H","I","J","K","L","Ł","M","N","Ń","O","Ó","P","Q","R","S","Ś","T","U","V","W","X","Y","Z","Ź","Ż","0","1","2","3","4","5","6","7","8","9",".",",","#","!","@","$","%","^","&","\*","(",")","-","`","\_","+","=","[","]","{","}","\\","|",":",";","\"","\'","<",">","/","?"};

System::String^ znak = "";

// jeśli warunek jest prawdziwy to może to być tekst zaszyfrowany

// kompleksowo (pierwszy ustawiamy na prawda);

if (dlugosc > 55)

pierwszy = true;

// jeśli pierwszy był prawdziwy to sprawdzamy drugi warunek

// polega on na tym, że jeśli przynajmniej jeden znak nie jest

// zerem albo jedynką to tekst nie może być kompleksowy;

if (pierwszy)

{

for (i = 0; i < dlugosc; i++)

{

if ( !(sourceText->Substring(i, 1) == "0") && !(sourceText->Substring(i, 1) == "1") )

break;

}

if (i == dlugosc)

drugi = true;

}

// jeśli drugi był prawdziwy to sprawdzamy czy łączna ilość znaków

// jest podzielna przez 8; jeśli tak to mamy pewność, że jest to tekst

// zaszyfrowany kompleksowo; trzeba jeszcze sprawdzić błędy;

if (drugi)

{

if (dlugosc % 8 == 0)

stopien = 2;

}

// dla szyfru standardowego;

while (stopien == 1)

{

// pętla poniżej sprawdza czy każdy znak tekstu należy do tablicy

// dozwoloneZnaki, jeśli nie to ustawia kod błędu startIndex == 0;

for (i = 0; i < dlugosc; i++)

{

znak = sourceText->Substring(i, 1);

for (j = 0; j < dozwoloneZnaki->Length; j++)

{

if ( znak == dozwoloneZnaki[j] )

break;

}

if (j == dozwoloneZnaki->Length)

{

startIndex = 0;

break;

}

}

// jeśli powyżej wykryto błąd to stopien jest ustawiany na 3 co

// oznacza wartość błędu i dalsze warunki nie są już sprawdzane;

if (startIndex == 0)

{

stopien = 3;

break;

}

// pierwsze 5 znaków jest traktowane jako symbole;

symbol1 = System::Convert::ToInt32(sourceText->Substring(0, 2));

symbol2 = System::Convert::ToInt32(sourceText->Substring(2, 3));

// w tekście musi znajdować się przynajmniej jeden wyraz o długości

// co najmniej 2 znaków i symbol2 musi być większe od zera co oznacza

// że w tekście jest co najmniej jeden wyraz;

if ( (symbol1 < 2) || (symbol2 < 1) )

{

stopien = 3;

break;

}

// jeśli tekst składa się tylko z jednego wyrazu to nie może

// zawierać żadnego znaku hash;

if (symbol2 == 1)

{

if (sourceText->Contains("#"))

{

stopien = 3;

break;

}

}

// pętla zlicza ilość hashy w tekście;

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

{

if ( (sourceText->Substring(i, 1)) == "#" )

ileHashy += 1;

}

// ilość hashy musi być mniejsza niż ilość wyrazów; jeśli nie - zgłoś błąd;

if ( !(ileHashy < symbol2) )

{

stopien = 3;

break;

}

ileHashy = 0;

// pętla służy do próbnego odkodowania tekstu, w celu sprawdzenia

// czy symbolika jest zgodna z faktyczną ilością znaków;

while (1)

{

// ile znaków zostało jeszcze do wypisania;

ileZostalo = dlugosc - startIndex;

if ( (ileZostalo >= (symbol2 - ileHashy)) || (ileZostalo > 0) )

{

// do każdego segmentu dopisujemy tyle tyld ile mamy łącznie hashy;

for (i = ileHashy; i > 0; i--)

{

wynik += "~";

}

// wypisujemy kolejne segmenty z uwzględnieniem ilości hashy;

if ( ileZostalo < (symbol2 - ileHashy) )

wynik += sourceText->Substring(startIndex, ileZostalo);

else

wynik += sourceText->Substring(startIndex, (symbol2 - ileHashy));

}

else

break;

// jeśli jest to pierwszy segment i znajdziemy w nim hash

// to zgłaszamy błąd;

if ( (wycinek->Length) == 0 )

{

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

{

if ( (wynik->Substring(i, 1)) == "#" )

{

startIndex = 0;

break;

}

}

}

if (startIndex == 0)

break;

// to samo co wypisaliśmy wyżej wypisujemy tu w celu

// sprawdzenia ile wypisaliśmy hashy w danym etapie;

if ( ileZostalo < (symbol2 - ileHashy) )

wycinek = sourceText->Substring(startIndex, ileZostalo);

else

wycinek = sourceText->Substring(startIndex, (symbol2 - ileHashy));

if ( (wycinek->Length) < 1 )

break;

// przesuwamy się o 1 segment dalej;

startIndex += (symbol2 - ileHashy);

// jeśli wypisaliśmy jakieś hashe zwiększ odpowiednio ich liczbę;

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

{

if ( (wycinek->Substring(i, 1)) == "#" )

ileHashy += 1;

}

}

// sprawdzamy czy długość naszego wyniku z segmentami jest zgodna

// z symboliką; jeśli tak to znaczy, że tekst jest poprawnie

// zaszyfrowany na poziomie standardowym;

if ( ((wynik->Length) / symbol2) == symbol1 )

{

if ( ((wynik->Length) % symbol2) == 0 )

break;

else

stopien = 3;

}

else

stopien = 3;

}

while (stopien == 2)

{

//instrukcje sprawdzania błędów dla kompleksowego

symbol1 = MakeSymbol(sourceText->Substring(0, 16));

symbol2 = MakeSymbol(sourceText->Substring(16, 24));

// jeśli w funkcji MakeSymbol wystąpił błąd zostanie on zgłoszony tutaj;

if ( (symbol1 == 9999) || (symbol2 == 9999) )

{

stopien = 3;

break;

}

// jeśli symbol1 jest równe 0 lub 1 zgłoś błąd;

if (sourceText->Substring(0, 8) == "00000101")

{

if ( (sourceText->Substring(8, 8) == "00000101") || (sourceText->Substring(8, 8) == "01011001") )

{

stopien = 3;

break;

}

}

// jeśli symbol2 jest równy 0 zgłoś błąd;

if (sourceText->Substring(16, 24) == "000001010000010100000101")

{

stopien = 3;

break;

}

// jeśli symbol2 jest równy 1 to tekst nie może zawierać hashy;

if (sourceText->Substring(16, 24) == "000001010000010101011001")

{

for (i = 0; i < dlugosc; )

{

if (sourceText->Substring(i, 8) == "01011010")

break;

i += 8;

}

if (i < dlugosc)

{

stopien = 3;

break;

}

}

// zliczamy ilość hashy;

for (i = 0; i < dlugosc; )

{

if (sourceText->Substring(i, 8) == "01011010")

ileHashy += 1;

i += 8;

}

if ( !(ileHashy < symbol2) )

{

stopien = 3;

break;

}

ileHashy = 0;

startIndex = 40;

dlugosc = sourceText->Length;

// pętla while jest analogiczna do tej użytej w sprawdzeniu standardowym

// jedyne różnice dotyczą ilości znaków, gdyż jest ich 8 razy więcej;

while (1)

{

ileZostalo = (dlugosc - startIndex) / 8;

if ( (ileZostalo >= (symbol2 - ileHashy)) || (ileZostalo > 0) )

{

for (i = ileHashy; i > 0; i--)

{

wynik += "11111111";

}

if ( ileZostalo < (symbol2 - ileHashy) )

wynik += sourceText->Substring(startIndex, (ileZostalo \* 8));

else

wynik += sourceText->Substring(startIndex, ((symbol2 - ileHashy) \* 8));

}

else

break;

if ( (wycinek->Length) == 0 )

{

for (i = 0; i < (wynik->Length); )

{

if ( (wynik->Substring(i, 8)) == "01011010" )

{

startIndex = 0;

break;

}

i += 8;

}

}

if (startIndex == 0)

break;

if ( ileZostalo < (symbol2 - ileHashy) )

wycinek = sourceText->Substring(startIndex, (ileZostalo \* 8));

else

wycinek = sourceText->Substring(startIndex, ((symbol2 - ileHashy) \* 8));

if ( (wycinek->Length) < 8 )

break;

startIndex += (symbol2 - ileHashy) \* 8;

for (i = 0; i < (wycinek->Length); )

{

if ( (wycinek->Substring(i, 8)) == "01011010" )

ileHashy += 1;

i += 8;

}

}

// sprawdzenie symboliki i wyniku;

if ( (((wynik->Length) / 8) / symbol2) == symbol1 )

{

if ( (((wynik->Length) / 8) % symbol2) == 0 )

break;

else

stopien = 3;

}

else

stopien = 3;

}

// sprawdzamy jeszcze za pomocą funkcji CheckText czy wszystkie znaki

// są dozwolone, aby można je było poprawnie odszyfrować;

if (stopien == 2)

{

startIndex = CheckText(sourceText);

if (startIndex == 1)

stopien = 3;

}

// jeśli napotkaliśmy błąd zostanie on tutaj zgłoszony komunikatem;

if (stopien == 3)

System::Windows::Forms::MessageBox::Show("Tekst niezaszyfrowany lub zaszyfrowany nieprawidłowo!", "Błąd 09", System::Windows::Forms::MessageBoxButtons::OK, System::Windows::Forms::MessageBoxIcon::Error);

return stopien;

}

public: int CheckText(System::String^ sourceText)

{

int i = 0;

int j = 0;

int dlugosc = sourceText->Length;

System::String^ znaki = "";

System::String^ symbol = "";

int id = 0;

cli::array<System::String^, 1>^ binarna = {"0000","0001","0010","0011","0100","0101","0110","0111","1000","1001","1010","1011","1100","1101","1110","1111"};

cli::array<System::String^, 1>^ cyfry = {"0","1","2","3","4","5","6","7","8","9","A","B","C","D","E","F"};

cli::array<System::String^, 1>^ dozwoloneZnaki = {"20","31","0F","43","54","01","44","30","62","10","6D","55","42","02","0E","72","1F","32","21","53","45","11","2F","61","56","41","33","0D","1E","46","22","6C","63","52","40","34","2E","03","12","1D","0C","57","23","47","60","6B","64","51","35","3F","04","2D","24","13","1C","58","0B","48","5F","6A","6E","50","65","36","3E","25","14","2C","49","1B","05","59","5E","4F","37","2B","26","0A","15","3D","4A","66","5A","6F","5D","69","38","06","16","27","4B","4E","5B","3C","2A","67","09","70","19","4C","39","28","17","5C","07","71","68","4D","3A","29","08"};

// funkcja jest bliźniacza do funkcji MakeSymbol, lecz tutaj sprawdzamy tylko

// czy każdy znak tekstu występuje w tablicy dozwoloneZnaki;

for (i = 0; i < dlugosc; )

{

znaki = sourceText->Substring(i, 4);

id = System::Array::IndexOf(binarna, znaki);

symbol += cyfry[id];

i += 4;

}

dlugosc = symbol->Length;

id = 0;

for (i = 0; i < dlugosc; )

{

znaki = symbol->Substring(i, 2);

for (j = 0; j < dozwoloneZnaki->Length; j++)

{

if (znaki == dozwoloneZnaki[j])

break;

}

if (j == dozwoloneZnaki->Length)

{

id = 1;

break;

}

i += 2;

}

return id;

}

// funkcja zamienia tekst binarny na heksadecymalny;

public: System::String^ BinaryToHex(System::String^ sourceText)

{

int i = 0;

int dlugosc = sourceText->Length;

System::String^ czworka = "";

System::String^ text = "";

int index = 0;

cli::array<System::String^, 1>^ binarna = {"0000","0001","0010","0011","0100","0101","0110","0111","1000","1001","1010","1011","1100","1101","1110","1111"};

cli::array<System::String^, 1>^ hex = {"0","1","2","3","4","5","6","7","8","9","A","B","C","D","E","F"};

for (i = 0; i < dlugosc; )

{

czworka = sourceText->Substring(i, 4);

index = System::Array::IndexOf(binarna, czworka);

text += hex[index];

i += 4;

}

return text;

}

// funkcja zamienia tekst heksadecymalny na postać standardową;

public: System::String^ HexToStandard(System::String^ sourceText)

{

int i = 0;

int dlugosc = sourceText->Length;

System::String^ fragment = "";

System::String^ text = "";

int index = 0;

cli::array<System::String^, 1>^ hex = {"20","31","0F","43","54","01","44","30","62","10","6D","55","42","02","0E","72","1F","32","21","53","45","11","2F","61","56","41","33","0D","1E","46","22","6C","63","52","40","34","2E","03","12","1D","0C","57","23","47","60","6B","64","51","35","3F","04","2D","24","13","1C","58","0B","48","5F","6A","6E","50","65","36","3E","25","14","2C","49","1B","05","59","5E","4F","37","2B","26","0A","15","3D","4A","66","5A","6F","5D","69","38","06","16","27","4B","4E","5B","3C","2A","67","09","70","19","4C","39","28","17","5C","07","71","68","4D","3A","29","08"};

cli::array<System::String ^, 1> ^ normal = {"a","ą","b","c","ć","d","e","ę","f","g","h","i","j","k","l","ł","m","n","ń","o","ó","p","q","r","s","ś","t","u","v","w","x","y","z","ź","ż","A","Ą","B","C","Ć","D","E","Ę","F","G","H","I","J","K","L","Ł","M","N","Ń","O","Ó","P","Q","R","S","Ś","T","U","V","W","X","Y","Z","Ź","Ż","0","1","2","3","4","5","6","7","8","9",".",",","#","!","@","$","%","^","&","\*","(",")","-","`","\_","+","=","[","]","{","}","\\","|",":",";","\"","\'","<",">","/","?"};

for (i = 0; i < dlugosc; )

{

fragment = sourceText->Substring(i, 2);

index = System::Array::IndexOf(hex, fragment);

text += normal[index];

i += 2;

}

return text;

}

// funkcja zamienia tekst zaszyfrowany standardowo na tzw. "mixed",

// czyli z poprzestawianymi literami;

public: System::String^ StandardToMixed(System::String^ sourceText)

{

int i = 0;

int dlugosc = sourceText->Length;

int startIndex = 5;

int symbol1;

int symbol2;

cli::array<System::String^, 1>^ wyrazy;

System::String^ wynik = "";

symbol1 = System::Convert::ToInt32(sourceText->Substring(0, 2));

symbol2 = System::Convert::ToInt32(sourceText->Substring(2, 3));

System::Array::Resize(wyrazy, symbol2);

// przypisujemy puste wartości każdemu elementowi tablicy wyrazy;

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

{

wyrazy[i] = "";

}

while (startIndex < dlugosc)

{

// pętla for kończy się po każdym wypisaniu segmentu;

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

{

// jeśli dany wyraz już się skończył to zawiera tyldę

// a jeśli zawiera tyldę to dopisuj tyldy do końca segmentu;

if ( wyrazy[i]->Contains("~") )

{

wyrazy[i] += "~";

continue;

}

if ( sourceText->Substring(startIndex, 1) == "#" )

wyrazy[i] += "~";

else

wyrazy[i] += sourceText->Substring(startIndex, 1);

startIndex += 1;

}

}

// usuwamy tyldy z wyrazów;

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

{

if ( wyrazy[i]->Contains("~") )

{

startIndex = wyrazy[i]->IndexOf("~");

wyrazy[i] = wyrazy[i]->Substring(0, startIndex);

}

}

// na końcu każdego wyrazu dopisuj spację oprócz ostatniego;

for (i = 0; i < (wyrazy->Length - 1); i++)

{

wyrazy[i] += " ";

}

// połącz tablicę wyrazów w gotowy tekst;

wynik = System::String::Concat(wyrazy);

return wynik;

}

// funkcja odpowiada za odszyfrowanie tekstu na poziomie kompleksowym;

public: System::String^ OdszyfrujKompleks(System::String^ sourceText)

{

System::String^ hexText = BinaryToHex(sourceText);

System::String^ standardText = HexToStandard(hexText);

System::String^ mixedText = StandardToMixed(standardText);

System::String^ invertedText = MixedToInverted(mixedText);

System::String^ normalText = InvertedToNormal(invertedText);

return normalText;

}

// natomiast ta funkcja na poziomie standardowym;

public: System::String^ OdszyfrujStandard(System::String^ sourceText)

{

System::String^ mixedText = StandardToMixed(sourceText);

System::String^ invertedText = MixedToInverted(mixedText);

System::String^ normalText = InvertedToNormal(invertedText);

return normalText;

}

// funkcja zamienia tekst z pomieszanymi znakami na tekst z wyrazami

// zapisanymi od końca;

public: System::String^ MixedToInverted(System::String^ sourceText)

{

int i = 0;

int dlWyrazu = 0;

cli::array<System::String^, 1>^ spacja = {" "};

cli::array<System::String^, 1>^ wyrazy = sourceText->Split(spacja, System::StringSplitOptions::RemoveEmptyEntries);

int wyrazyLength = wyrazy->Length;

System::String^ wynik = "";

for (i = 0; i < wyrazyLength; i++)

{

dlWyrazu = wyrazy[i]->Length;

while (dlWyrazu > 0)

{

// ostatni znak danego wyrazu dopisujemy do wyniku, a następnie

// usuwamy go z tablicy źródłowej;

wynik += wyrazy[i]->Substring(dlWyrazu - 1, 1);

wyrazy[i] = wyrazy[i]->Substring(0, dlWyrazu - 1);

// jeśli wyraz się skończył przerywamy, jeśli nie to do wyniku

// dopisujemy pierwszy znak wyrazu i również likwidujemy w źródle;

if ( wyrazy[i]->Length == 0 )

break;

else

{

dlWyrazu = wyrazy[i]->Length;

wynik += wyrazy[i]->Substring(0, 1);

wyrazy[i] = wyrazy[i]->Substring(1, dlWyrazu - 1);

}

dlWyrazu = wyrazy[i]->Length;

}

// jeśli dany wyraz nie jest ostatni dopisz do niego jeszcze spację;

if ( i < (wyrazyLength - 1) )

wynik += " ";

}

return wynik;

}

// funkcja zamienia tekst zapisany wspak na normalny, całkowicie

// odszyfrowany tekst;

public: System::String^ InvertedToNormal(System::String^ sourceText)

{

int i = 0;

int dlWyrazu = 0;

cli::array<System::String^, 1>^ spacja = {" "};

cli::array<System::String^, 1>^ wyrazy = sourceText->Split(spacja, System::StringSplitOptions::RemoveEmptyEntries);

int wyrazyLength = wyrazy->Length;

System::String^ wynik = "";

for (i = 0; i < wyrazyLength; i++)

{

dlWyrazu = wyrazy[i]->Length;

while (dlWyrazu > 0)

{

// ostatni znak dopisujemy do wyniku i usuwamy ze źródła

// aż do usunięcia całego wyrazu; następnie robimy to samo

// ze wszystkimi kolejnymi wyrazami;

wynik += wyrazy[i]->Substring(dlWyrazu - 1, 1);

wyrazy[i] = wyrazy[i]->Substring(0, dlWyrazu - 1);

dlWyrazu = wyrazy[i]->Length;

}

if ( i < (wyrazyLength - 1) )

wynik += " ";

}

return wynik;

}

// funkcja odpowiada za odwrócenie wszystkich wyrazów w szyfrowanym tekście

// z użyciem opcji Krok po kroku; zmienia też tekst etykiety Status;

public: int KrokPoKroku(System::String^ sourceText)

{

cli::array<System::String ^, 1> ^ rozbiteZdanie = Wyraz(sourceText);

cli::array<System::String ^, 1> ^ odwroconeWyrazy = Invert(rozbiteZdanie);

textBox2->Text = System::String::Concat(odwroconeWyrazy);

label2->ForeColor::set(System::Drawing::SystemColors::ControlText);

label2->Text::set("Krok1...");

return 0;

}

// funkcja ta przestawia znaki w tekście z opcją krok po kroku;

public: int KrokPoKroku2(System::String^ sourceText)

{

cli::array<System::String ^, 1> ^ rozbityText = Wyraz(sourceText);

cli::array<System::String ^, 1> ^ przestawioneWyrazy = Reorder(rozbityText);

textBox2->Text = ZwrocZdanie(przestawioneWyrazy);

label2->Text::set("...Krok2...");

zapamietanyText = textBox2->Text;

return 0;

}

// funkcja ta pokazuje tekst posegmentowany podczas szyfrowania krok

// po kroku; jest to jedyny moment kiedy można zobaczyć tę postać;

public: int KrokPoKroku3(System::String^ sourceText)

{

cli::array<System::String ^, 1> ^ segmentowaneZdanie = Wyraz(sourceText);

textBox2->Text = Segment(segmentowaneZdanie);

label2->Text::set("...Krok3...");

return 0;

}

// funkcja zamienia tekst na postać standardową z postaci "mixed";

public: int KrokPoKroku4()

{

cli::array<System::String ^, 1> ^ podzieloneZdanie = Wyraz(zapamietanyText);

textBox2->Text = MakeStandard(podzieloneZdanie);

if (radioButton1->Checked)

label2->Text::set("...Krok4...Gotowe");

if (radioButton2->Checked)

label2->Text::set("...Krok4...");

return 0;

}

// funkcja zamienia tekst na heksadecymalny w szyfrowaniu

// kompleksowym krok po kroku;

public: int KrokPoKroku5(System::String^ sourceText)

{

textBox2->Text = MakeHex(sourceText);

label2->Text::set("...Krok5...");

return 0;

}

// ostatni etap szyfrowania kompleksowego krok po kroku;

public: int KrokPoKroku6(System::String^ sourceText)

{

textBox2->Text = MakeBinary(sourceText);

label2->Text::set("...Krok6...Gotowe");

return 0;

}

// kopiuje zaznaczony tekst w jednym z pól tekstowych do schowka po naciśnięciu

// Kopiuj w menu Edycja;

private: System::Void kopiujToolStripMenuItem2\_Click(System::Object^ sender, System::EventArgs^ e) {

if (textBox1->SelectionLength > 0)

System::Windows::Forms::Clipboard::SetText(textBox1->SelectedText);

if (textBox2->SelectionLength > 0)

System::Windows::Forms::Clipboard::SetText(textBox2->SelectedText);

}

// wkleja tekst ze schowka do górnego pola tekstowego po naciśnięciu

// Wklej w menu Edycja;

private: System::Void wklejToolStripMenuItem1\_Click(System::Object^ sender, System::EventArgs^ e) {

textBox1->Paste(System::Windows::Forms::Clipboard::GetText());

}

// po naciśnięciu przycisku Zapisz na pasku narzędzi otwiera okno zapisu;

private: System::Void toolStripButton1\_Click(System::Object^ sender, System::EventArgs^ e) {

Form1::saveFileDialog1->ShowDialog();

}

// zamienia tekst zaszyfrowany kompleksowo z opcją krok po kroku

// na tekst heksadecymalny;

public: int KrokPoKroku1K(System::String^ sourceText)

{

textBox2->Text = BinaryToHex(sourceText);

label2->ForeColor::set(System::Drawing::SystemColors::ControlText);

label2->Text::set("Krok1...");

return 0;

}

// jak wyżej lecz na tekst standardowy;

public: int KrokPoKroku2K(System::String^ sourceText)

{

textBox2->Text = HexToStandard(sourceText);

label2->Text::set("...Krok2...");

return 0;

}

// zamienia tekst z postaci zaszyfrowanej standardowo na postać

// pomieszaną (mixed); zmienia tekst etykiety odpowiednio do poziomu;

public: int KrokPoKroku3K(System::String^ sourceText)

{

textBox2->Text = StandardToMixed(sourceText);

if (radioButton2->Checked)

label2->Text::set("...Krok3...");

if (radioButton1->Checked)

label2->Text::set("Krok1...");

return 0;

}

// zamienia "mixed" na odwrócony;

public: int KrokPoKroku4K(System::String^ sourceText)

{

textBox2->Text = MixedToInverted(sourceText);

if (radioButton2->Checked)

label2->Text::set("...Krok4...");

if (radioButton1->Checked)

label2->Text::set("...Krok2...");

return 0;

}

// ostatni krok w deszyfrowaniu Krok po kroku

// zamienia tekst na całkowicie czytelny;

public: int KrokPoKroku5K(System::String^ sourceText)

{

textBox2->Text = InvertedToNormal(sourceText);

if (radioButton2->Checked)

label2->Text::set("...Krok5...Gotowe");

if (radioButton1->Checked)

label2->Text::set("...Krok3...Gotowe");

return 0;

}

};

}