ПРИДНЕСТРОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМ. Т.Г.ШЕВЧЕНКО
**Рыбницкий филиал**
Кафедра информатики и программной инженерии

**ПРАКТИЧЕСКИЙ ПРОЕКТ**

**на тему:**

# «РЕАЛИЗАЦИЯ ТЕКСТОВОГО РЕДАКТОРА»

Студентов I курса направления
«Программная инженерия»
профиля «Разработка программно-информационных систем»
Черного Александра Александровича,
Джосана Никиты Денисовича

Научные руководители:

Тягульская Л.А. Гарбузняк Е.С.

Рыбница, 2023

# АКТУАЛЬНОСТЬ, ОБЪЕКТ, ПРЕДМЕТ ИССЛЕДОВАНИЯ

**Актуальность** исследования является то, что в современном образе жизни нас окружает большое количество информации, которую необходимо организовывать и запоминать. Блокнот является решением этой проблемы, предоставляя удобный и эффективный способ для хранения и управления этой информацией.

**Объект исследования** - текстовые редакторы и их возможности.

**Предмет исследования** – программный продукт, реализованный при помощи языка *C++* и Windows Forms в Microsoft Visual Studio.

# ЦЕЛЬ И ЗАДАЧИ

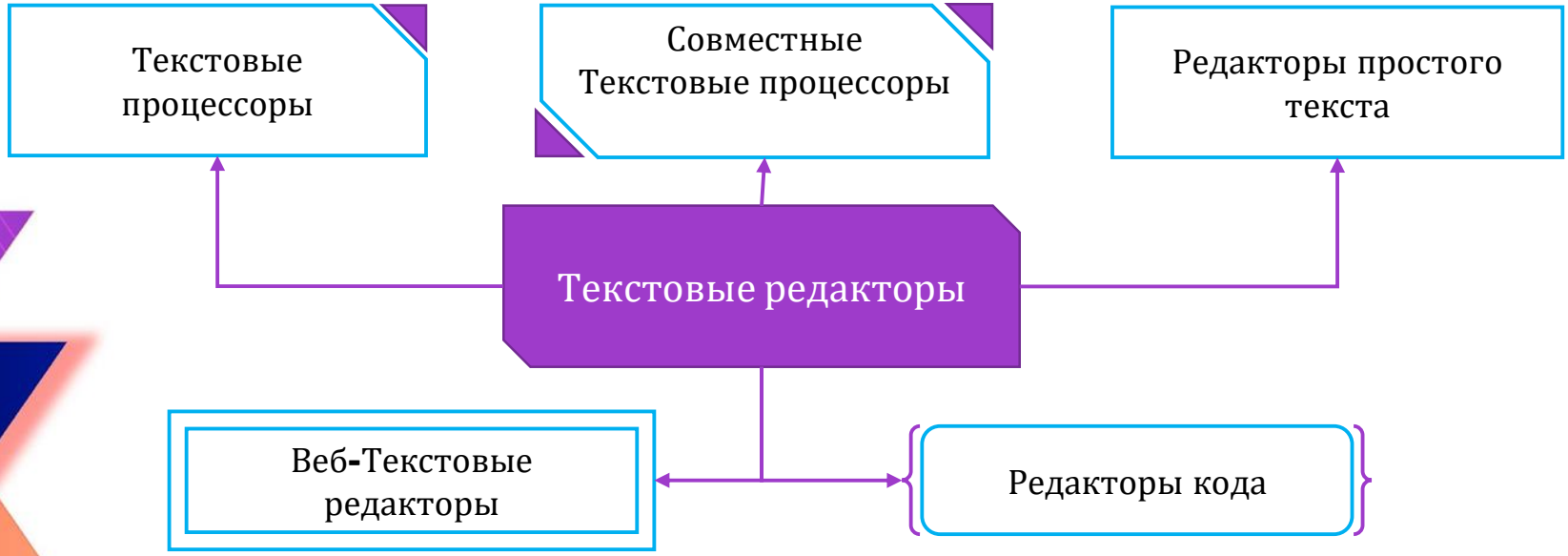
**Целью исследования** является создание программы текстового редактора с модуля визуального структурирования файлов с целью упростить запись и хранение информации.

Для достижения цели были поставлены следующие **задачи**:

- Изучить понятие текстового редактора;
- Изучить функции текстового редактора;
- Изучить возможности работы с графикой в Microsoft Visual Studio;
- Изучить аналогичные программные продукты;
- Создать программный продукт, реализующий текстовый редактор;
- Провести тестирование и отладку программы.

# ПОНЯТИЕ И ВИДЫ ТЕКСТОВОГО РЕДАКТОРА

**Текстовый редактор** — это программа, которая используется для создания и редактирования текстовых файлов. это инструмент, который позволяет пользователям различными способами редактировать текст, например изменять шрифты, форматирование и цвет.



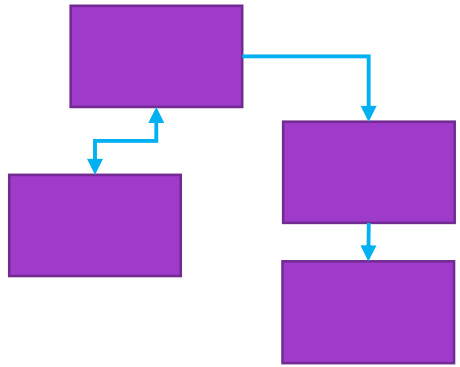
# ВИДЫ ВИЗУАЛЬНОГО ПРЕДСТАВЛЕНИЯ ИНФОРМАЦИИ

**Визуализация информации** — это научная и инженерная дисциплина, направленная на поиск и реализацию форм и способов визуального (в том числе интерактивного) представления абстрактных данных для облегчения их восприятия человеком.

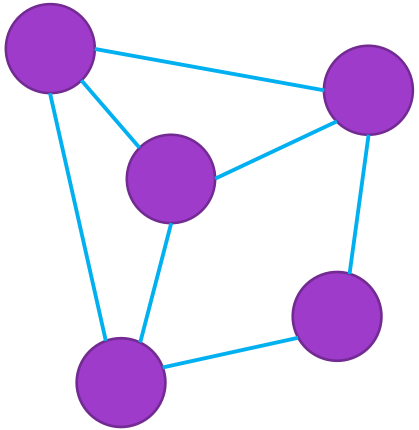
**Виды визуализации информации:**

1. Картограмма
2. Кладограмма
3. Графические концепции (схемы)
4. Древовидная схема.
5. Эталонная модель
6. Графы
7. Тепловая карта
8. Гиперболическое дерево
9. Многомерное шкалирование
10. Параллельные координаты
11. Treemapping (древовидная карта)

Графические концепции



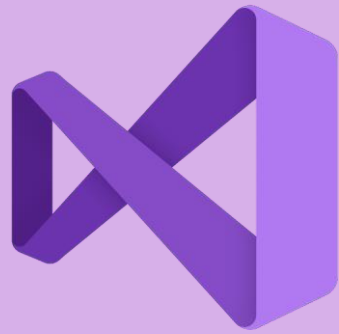
Графы



# ИНСТРУМЕНТЫ РАЗРАБОТКИ

**C++** обеспечивает низкоуровневый доступ к аппаратным ресурсам, что позволит реализовать более высокую производительность и эффективность по сравнению с языками высокого уровня, такими как ***Java*** и ***Python***. Он также поддерживает объектно-ориентированное программирование, что упрощает разработку и поддержку сложных приложений.

В Microsoft Visual Studio есть мощный редактор кода, предоставляющий такие функции, как подсветка синтаксиса, завершение кода и инструменты рефакторинга. Он также имеет отладчик, который позволяет разработчикам выполнять код, выявлять и исправлять ошибки. Кроме того в данной среде есть поддержка ***Git*** для совместной работы над проектом.



# СОЗДАНИЕ ПРОГРАММЫ

Функция импорта текстового файла с помощью события “*Click*” привязанного к кнопке “Open a file”:

```
private: System::Void Open_file_button_Click(System::Object^ sender, System::EventArgs^ e)
{
    OpenFileDialog^ openFileDialog1 = gcnew OpenFileDialog();

    openFileDialog1->Filter = "Text Files|*.txt|All Files|*.*";
    openFileDialog1->Title = "Open a TXT File";

    if (openFileDialog1->ShowDialog() == System::Windows::Forms::DialogResult::OK)
    {
        System::IO::StreamReader^ sr = gcnew System::IO::StreamReader(openFileDialog1->FileName);
        textBox1->Text = sr->ReadToEnd();
        sr->Close();
    }
```

Функция сохранения текстового файла с помощью события “*Click*” привязанного к кнопке “Save as”:

```
SaveFileDialog^ saveFileDialog1 = gcnew SaveFileDialog;
saveFileDialog1->Filter = "Text Files|*.txt|All Files|*.*";
saveFileDialog1->FilterIndex = 1;
saveFileDialog1->RestoreDirectory = true;
if (saveFileDialog1->ShowDialog() == System::Windows::Forms::DialogResult::OK
    && saveFileDialog1->FileName->Length > 0)
{
    IO::File::WriteAllText(saveFileDialog1->FileName, textBox1->Text);
}
```

# ВНУТРЕННИЙ ФУНКЦИОНАЛ ПРОГРАММЫ (MyForm.h : основная форма)

В программе используются пространства имен для упрощения работы с кодом

```
using namespace System;
using namespace System::ComponentModel;
using namespace System::Collections;
using namespace System::Windows::Forms;
using namespace System::Data;
using namespace System::Drawing;
using namespace System::IO;
using namespace System::Security::Cryptography;
using namespace System::Text;
using namespace System::Windows::Forms;
using namespace System::Drawing::Drawing2D;
```

# ВНУТРЕННИЙ ФУНКЦИОНАЛ ПРОГРАММЫ (MyForm.h : форма с визуальным модулем)

Создание формы на основе класса *Form. Эта форма позволяет создать основное окно программы.*

```
public ref class MyForm : public System::Windows::Forms::Form
{
public:
    MyForm()
    {
        InitializeComponent();
        this->Resize += gcnew System::EventHandler(this, &MyForm::MyForm_Resize);
        this->Text = "NoteBK";
        initialText = textBox1->Text;
        //
        //TODO: добавьте код конструктора
        //
    }


protected:
    /// <summary>
    /// Освободить все используемые ресурсы.
    /// </summary>
    ~MyForm()
    {
        if (components)
        {
            delete components;
        }
    }
```

Далее по коду идет объявление элементов формы пример которого представлен на скриншоте.

```
    private: System::Windows::Forms::TextBox^ textBox1;
    private: System::Windows::Forms::OpenFileDialog^ openFileDialog1;
    private: System::Windows::Forms::SaveFileDialog^ saveFileDialog1;
    private: System::Windows::Forms::Button^ Open_file_button;
    private: System::Windows::Forms::Button^ Save_file_button;
```

# ВНУТРЕННИЙ ФУНКЦИОНАЛ ПРОГРАММЫ (MyForm.h : форма с визуальным модулем)

Пример настройки элемента формы представлен на скриншоте

```
//
// MyForm
//
this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Inherit;
this->BackColor = System::Drawing::Color::White;
this->ClientSize = System::Drawing::Size(1264, 681);
this->Controls->Add(this->as_background_panel);
this->Controls->Add(this->as_panel);
this->Controls->Add(this->idle_background_panel);
this->Controls->Add(this->idle_panel);
this->Controls->Add(this->Settings_panel_left_background);
this->Controls->Add(this->panel_back_password_enter_field);
this->Controls->Add(this->Password_panel_background);
this->Controls->Add(this->settings_button);
this->Controls->Add(this->panel_settings);
this->Controls->Add(this->textBox1);
this->DoubleBuffered = true;
this->Icon = (cli::safe_cast<System::Drawing::Icon^>(resources->GetObject(L"$this.Icon")));
this->MinimumSize = System::Drawing::Size(430, 510);
this->Name = L"MyForm";
this->Text = L"MyForm";
this->Load += gcnew System::EventHandler(this, &MyForm::MyForm_Load);
this->panel_settings->ResumeLayout(false);
this->Password_panel_background->ResumeLayout(false);
this->Password_panel_background->PerformLayout();
```

# ВНУТРЕННИЙ ФУНКЦИОНАЛ ПРОГРАММЫ (MyForm.h : форма с визуальным модулем)

Обработчик событий settings_button_click отвечающий за появление скрывающегося меню с настройками программы

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

    idle_timer->Stop();
    idle_timer->Start();


    panel_settings->Visible = true;
    if (isPanelOpen) {
        return;
    }


    panel_settings->Location = System::Drawing::Point(-PANEL_WIDTH, 0);

    // Set the current size of the panel to zero
    currentSize = 0;



    openPanelTimer->Interval = 1;
    openPanelTimer->Tick += gcnew EventHandler(this, &MyForm::openPanelTimer_Tick);
    openPanelTimer->Start();


    //Settings_panel_left_background->BackColor = System::Drawing::Color::YellowGreen;
    Settings_panel_left_background->Visible = true;
    Settings_panel_left_background->Dock = DockStyle::Fill;

    isPanelOpen = true;

}
```

# ВНУТРЕННИЙ ФУНКЦИОНАЛ ПРОГРАММЫ (MyForm.h : форма с визуальным модулем)

Обработчик событий block_button_click отвечает за блокировку основной формы с помощью вспомогательной PasswordForm.h

```
private: System::Void Block_button_Click_1(System::Object^ sender, System::EventArgs^ e)
{


    idle_timer->Stop();

    Settings_panel_left_background->Visible = false;
    currentSize = panel_settings->Size.Width;

    // Скрыть панель по таймеру
    closePanelTimer->Interval = 10; // задать интервал таймера в миллисекундах
    closePanelTimer->Tick += gcnew EventHandler(this, &MyForm::closePanelTimer_Tick);
    closePanelTimer->Start();

    isPanelOpen = false;


    PasswordForm^ bkmn = gcnew PasswordForm();
    bkmn->StartPosition = FormStartPosition::Manual; // Устанавливаем ручное позиционирование формы
    bkmn->Location = this->Location; // Устанавливаем положение формы password как у myform формы
    bkmn->Size = this->Size; // Устанавливаем размеры формы password как у myform формы
    bkmn->Owner = this;
    this->Hide();
    bkmn->Show();
}
```

# ВНУТРЕННИЙ ФУНКЦИОНАЛ ПРОГРАММЫ (MyForm.h : форма с визуальным модулем)

Обработчик событий passcode_button_click отвечает за появление панели с вводом пароля

```
private: System::Void Passcode_button__Click(System::Object^ sender, System::EventArgs^ e)
{
    idle_timer->Stop();
    idle_timer->Start();

    Settings_panel_left_background->Visible = false;
    currentSize = panel_settings->Size.Width;

    // Скрыть панель по таймеру
    closePanelTimer->Interval = 10; // задать интервал таймера в миллисекундах
    closePanelTimer->Tick += gcnew EventHandler(this, &MyForm::closePanelTimer_Tick);
    closePanelTimer->Start();

    isPanelOpen = false;


    Password_panel_background->Location = System::Drawing::Point((this->Width / 2) - (Password_panel_background->Width / 2), (this->Height / 2) -



    Password_panel_background->Visible = true;
    password_pictureBox->Visible = true;

    panel_back_password_enter_field->Visible = true;
    panel_back_password_enter_field->Dock = DockStyle::Fill;

}
```

# ВНУТРЕННИЙ ФУНКЦИОНАЛ ПРОГРАММЫ (MyForm.h : форма с визуальным модулем)

Обработчик событий enter_password_button_click принимающий пароль с поля password_field и сохраняющего его в определенную папку

```
private: System::Void enter_password_button_Click(System::Object^ sender, System::EventArgs^ e)
{
    idle_timer->Stop();
    idle_timer->Start();


    String^ password = Password_field->Text;
    String^ documentsPath = Environment::GetFolderPath(Environment::SpecialFolder::MyDocuments);
    String^ notebookPath = Path::Combine(documentsPath, "NoteBk");
    String^ passwordFile = Path::Combine(notebookPath, "password.txt");

    if (!Directory::Exists(notebookPath))
    {
        Directory::CreateDirectory(notebookPath);
    }
    // Шифруем пароль
    String^ encryptedPassword = EncryptPassword(password);
    // Записываем зашифрованный пароль в файл
    File::WriteAllText(passwordFile, encryptedPassword);

    MessageBox::Show("Пароль сохранен");
    panel_back_password_enter_field->Visible = false;
    Password_panel_background->Visible = false;
}
    // Функция для шифрования пароля
    String^ EncryptPassword(String^ password)
    {
        // Произвольный секретный ключ для шифрования
        String^ secretKey = "A5KP7Y4YZyRA9Qz";

        // Создаем экземпляр класса RijndaelManaged для шифрования
        RijndaelManaged^ aes = gcnew RijndaelManaged();
        aes->BlockSize = 128;
        aes->KeySize = 256;
        aes->Padding = PaddingMode::PKCS7;
        aes->Mode = CipherMode::CBC;

        // Генерируем ключ и вектор инициализации на основе секретного ключа
        Rfc2898DeriveBytes^ key = gcnew Rfc2898DeriveBytes(secretKey, Encoding::UTF8->GetBytes(secretKey));
        aes->Key = key->GetBytes(aes->KeySize / 8);
        aes->IV = key->GetBytes(aes->BlockSize / 8);

        // Шифруем пароль
        ICryptoTransform^ encryptor = aes->CreateEncryptor();
        array<Byte>^ encryptedData = encryptor->TransformFinalBlock(Encoding::UTF8->GetBytes(password), 0, password->Length);
        // Возвращаем зашифрованный пароль в виде строки в формате Base64
        return Convert::ToBase64String(encryptedData);
}
```

# ВНУТРЕННИЙ ФУНКЦИОНАЛ ПРОГРАММЫ (MyForm.h : форма с визуальным модулем)

settings_panel_left_background_click пример одного из обработчиков событий реализующего скрывающего основной элемент управления

```
private: System::Void Settings_panel_left_background_Click(System::Object^ sender, System::EventArgs^ e)
{
    Settings_panel_left_background->Visible = false;


    currentSize = panel_settings->Size.Width;

    // Скрыть панель по таймеру
    closePanelTimer->Interval = 10; // задать интервал таймера в миллисекундах
    closePanelTimer->Tick += gcnew EventHandler(this, &MyForm::closePanelTimer_Tick);
    closePanelTimer->Start();

    isPanelOpen = false;

}
```

# ВНУТРЕННИЙ ФУНКЦИОНАЛ ПРОГРАММЫ (Passwordform.h : блокирующая форма)

PasswordForm_FormClosed обработчик срабатывающий при закрытии формы passwordform.h, предназначенный для передачи размера и позиции для основной формы

```
private: System::Void PasswordForm_FormClosed(System::Object^ sender, System::Windows::Forms::FormClosedEventArgs^ e) {
    // Передача положения и размера формы в первую форму
    this->Owner->Location = this->Location;
    this->Owner->Size = this->Size;
}
```

# ВНУТРЕННИЙ ФУНКЦИОНАЛ ПРОГРАММЫ (SchemeSpace.h : форма с визуальным модулем)

Упрощение работы с пространствами имён *«System».* Поскольку в разных пространствах имён могут быть одинаковые имена и часть разработки выполнена с помощью встроенного конструктора *Microsoft Visual Studio,* данные «упрощения» применяются не вегда.

```
using namespace System;
using namespace System::ComponentModel;
using namespace System::Collections;
using namespace System::Windows::Forms;
using namespace System::Data;
using namespace System::Drawing;
```

# ВНУТРЕННИЙ ФУНКЦИОНАЛ ПРОГРАММЫ (SchemeSpace.h : форма с визуальным модулем)

Обработка события при нажатии кнопки *«open_fldr»* для выбора рабочей папки. Служит для вызова диалогового окна браузера файлов и выбора папки, путь к которой записывается в переменную *targetDirectory.*

```
private: String^ targetDirectory = " "; // путь к папке, откуда берём файлы

          // Загружаем файлы из выбранной дерикетории в список файлов
private: System::Void open_fldr_Click(System::Object^ sender, System::EventArgs^ e) {

        FolderBrowserDialog^ dlg = gcnew FolderBrowserDialog();

        if (dlg->ShowDialog() == System::Windows::Forms::DialogResult::OK) {
            targetDirectory = dlg->SelectedPath;
            this->fldr_lable->Text = targetDirectory;
        }

        // Заполняем масив файлов, если выбрана папка
        if (targetDirectory != " ") {
            filesView->Clear();
            array<String^>^ Result = System::IO::Directory::GetFiles(targetDirectory, "*.txt", IO::SearchOption::AllDirectories);
            for (int i = 0; i < Result->Length; i++) {
                    String^ name = Result[i]->Substring(targetDirectory->Length+1); //отсечение пути из выводимой строки (вывод только имени файла)
                filesView->Items->Add(name);
                }
        }
        targetDirectory = " ";
    }
```

# ВНУТРЕННИЙ ФУНКЦИОНАЛ ПРОГРАММЫ (SchemeSpace.h : форма с визуальным модулем)

Обработка события нажатия кнопки *«addNodeButton».* Данный блок кода служит для добавления на форму (в рабочую область) нового элемента пользовательского класса *«RectangleContainer»,* служащего узлами, подразумевающими файлы, лежащие в выбранной папке. При создании задаются положение, размеры, заголовок и контент узла при помощи соответствующих свойств объекта. В будущем контент и заголовок будут определяться выбранным в списке файлом.

```
private: System::Void addNodeButton_Click(System::Object^ sender, System::EventArgs^ e) {
    // Создать новый объект RectangleContainer
    RectangleContainer^ newContainer = gcnew RectangleContainer();

    // Установить размер и положение контейнера на форме
    newContainer->Location = System::Drawing::Point(500, 100);
    newContainer->Size = System::Drawing::Size(200, 200);
    newContainer->Title = "Ваш Новый заголовок";
    newContainer->Content = "Здесь находится некоторый текст, который будет показывать небольшой отрывок из текстового файла";
    // Добавить контейнер на форму
    this->Controls->Add(newContainer);

}
```

# ВНУТРЕННИЙ ФУНКЦИОНАЛ ПРОГРАММЫ (SchemeSpace.h : форма с визуальным модулем)

Объявление пользовательского класса *RectangleContainer* на основе класса *Panel.* Добавление вспомогательных переменных для перемещения узлов этого класса и элементов для заголовка и контента узлов.

```
public ref class RectangleContainer : public Panel
{
    // Переменные ----------------------------------------
private:
    // Флаг перемещения контейнера
    bool isMoving = false;
    // Последние координаты мыши
    Point lastMousePosition;

    //  Элементы управления для заголовка и содержания --
    Label^ titleLabel;
    Label^ contentLabel;
```

```
public:
    RectangleContainer()
    {
        // Установить размер и цвет фона контейнера
        this->Size = System::Drawing::Size(200, 200);
        this->BackColor = Color::Black;

        // Создать текстовое поле для заголовка с шрифтом 15pt
        titleLabel = gcnew Label();
        titleLabel->Font = gcnew Drawing::Font("Arial", 15);
        titleLabel->Size = System::Drawing::Size(180, 30);
        titleLabel->Location = Point(10, 10);
        titleLabel->Text = "Заголовок контейнера";
        titleLabel->ForeColor = Color::Wheat;
        titleLabel->AutoEllipsis = true;
        this->Controls->Add(titleLabel);

        // Создать текстовое поле для содержания с шрифтом 10pt
        contentLabel = gcnew Label();
        contentLabel->Font = gcnew Drawing::Font("Arial", 10);
        contentLabel->Size = System::Drawing::Size(180, 150);
        contentLabel->Location = Point(10, 40);
        contentLabel->Text = "Содержание контейнера";
        contentLabel->ForeColor = Color::White;
        contentLabel->AutoEllipsis = true;
        this->Controls->Add(contentLabel);
```

# ВНУТРЕННИЙ ФУНКЦИОНАЛ ПРОГРАММЫ (SchemeSpace.h : форма с визуальным модулем)

```
//  Свойства контейнера ------------------------------------
property String^ Title {
    String^ get() { return titleLabel->Text; }
    void set(String^ value) { titleLabel->Text = value; }
}

property String^ Content {
    String^ get() { return contentLabel->Text; }
    void set(String^ value) { contentLabel->Text = value; }
}
```

Объявление новых свойств для класса *RectangleContainer* для задания заголовка и контента узлов. Все остальные свойства, применяемые в коде наследуются от класса *Panel*

```
//рисуем прямоугольник с закруглёнными углами
System::Drawing::Drawing2D::GraphicsPath^ path = gcnew System::Drawing::Drawing2D::GraphicsPath();
int cornerRadius = 20;
path->AddLine(cornerRadius, 0, this->Width - cornerRadius, 0);
path->AddArc(this->Width - cornerRadius, 0, cornerRadius, cornerRadius, 270, 90);
path->AddLine(this->Width, cornerRadius, this->Width, this->Height - cornerRadius);
path->AddArc(this->Width - cornerRadius, this->Height - cornerRadius, cornerRadius, cornerRadius, 0, 90);
path->AddLine(this->Width - cornerRadius, this->Height, cornerRadius, this->Height);
path->AddArc(0, this->Height - cornerRadius, cornerRadius, cornerRadius, 90, 90);
path->AddLine(0, this->Height - cornerRadius, 0, cornerRadius);
path->AddArc(0, 0, cornerRadius, cornerRadius, 180, 90);
path->CloseFigure();
this->Region = gcnew System::Drawing::Region(path);
```

Задание формы контейнера узла при помощи арок и линий. Нужно для отображения узла в виде прямоугольника с закруглёнными углами. Для этого используются свойства класса *Width* (ширина) и *Height* (высота) и переменная *cornerRadius,* отвечающая за радиус скругления углов

# ВНУТРЕННИЙ ФУНКЦИОНАЛ ПРОГРАММЫ (SchemeSpace.h : форма с визуальным модулем)

В методе *RectangleContainer()* (показанном на слайде 12) также описаны обработчики событий нажатия, перемещения и отпускания мыши для реализации перетаскивания узлов внутри формы (рабочей области)

```
//обработчики события перемещения
this->MouseDown += gcnew MouseEventHandler(this, &RectangleContainer::OnMouseDown);
this->MouseMove += gcnew MouseEventHandler(this, &RectangleContainer::OnMouseMove);
this->MouseUp += gcnew MouseEventHandler(this, &RectangleContainer::OnMouseUp);
```

А в самом классе описана обработка этих событий. При нажатии/отпускании на кнопку мыши поднимается/опускается флаг перемещения, при перемещении курсора просчитывается изменение его положения и прибавляется к текущему положению узла

```
void OnMouseMove(Object^ sender, MouseEventArgs^ e)
{
    // Обработка события перемещения мыши по контейнеру
    if (this->isMoving)
    {
        Point currentLocation = e->Location; // текущая позиция мыши
        int deltaX = currentLocation.X - lastMousePosition.X; // вычислить разницу по оси X
        int deltaY = currentLocation.Y - lastMousePosition.Y; // вычислить разницу по оси Y
        this->Location = Point(this->Left + deltaX, this->Top + deltaY); // изменить положение объекта
    }
}
```

```
void OnMouseDown(Object^ sender, MouseEventArgs^ e)
{
    // Обработка события нажатия кнопки мыши на контейнере
    if (e->Button == System::Windows::Forms::MouseButtons::Left)
    {
        // Запомнить текущие координаты мыши
        this->lastMousePosition = e->Location;
        // Установить флаг "перемещение"
        if (!isMoving) this->isMoving = true;
    }
}
```

```
void OnMouseUp(Object^ sender, MouseEventArgs^ e)
{
    // Обработка события отпускания кнопки мыши на контейнере
    if (e->Button == System::Windows::Forms::MouseButtons::Left)
    {
        // Сбросить флаг "перемещение"
        this->isMoving = false;
    }
}
```

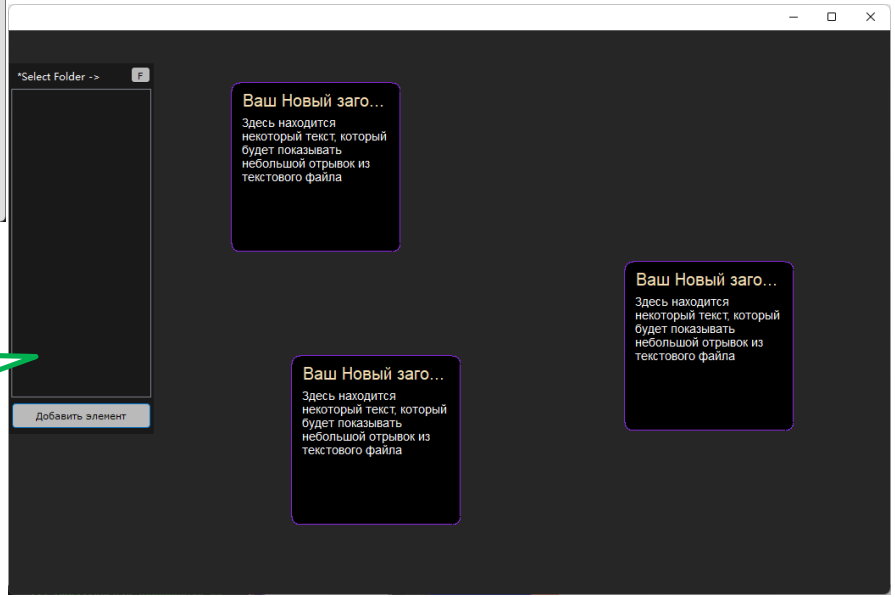
# ВНУТРЕННИЙ ФУНКЦИОНАЛ ПРОГРАММЫ (SchemeSpace.h : форма с визуальным модулем)

Блок кода из класса *RectangleContainer*, служащий для отрисовки контура (обводки) узлов. Цвет обводки планируется использовать для обозначения выбранного узла.

```
    //  Прорисовка обводки объекта ----------------------------------------------------------------------
protected:
    // Метод для рисования обводки контейнера
    virtual void OnPaint(PaintEventArgs^ e) override {
        Panel::OnPaint(e);
        Graphics^ g = e->Graphics;
        Pen^ pen = gcnew Pen(Color::BlueViolet, 2);

        // Создаем GraphicsPath и добавляем прямоугольник с закругленными углами
        int radius = 20;
        System::Drawing::Drawing2D::GraphicsPath^ path = gcnew System::Drawing::Drawing2D::GraphicsPath();
        System::Drawing::Rectangle rect = this->ClientRectangle;
        path->AddArc(rect.X, rect.Y, radius, radius, 180, 90);
        path->AddArc(rect.X + rect.Width - radius, rect.Y, radius, radius, 270, 90);
        path->AddArc(rect.X + rect.Width - radius, rect.Y + rect.Height - radius, radius, radius, 0, 90);
        path->AddArc(rect.X, rect.Y + rect.Height - radius, radius, radius, 90, 90);
        path->CloseFigure();

        // Создаем Region и применяем его к контейнеру
        System::Drawing::Region^ region = gcnew System::Drawing::Region(path);
        this->Region = region;

        // Рисуем обводку контейнера
        g->DrawPath(pen, path);
    }
```

# ТЕСТИРОВАНИЕ

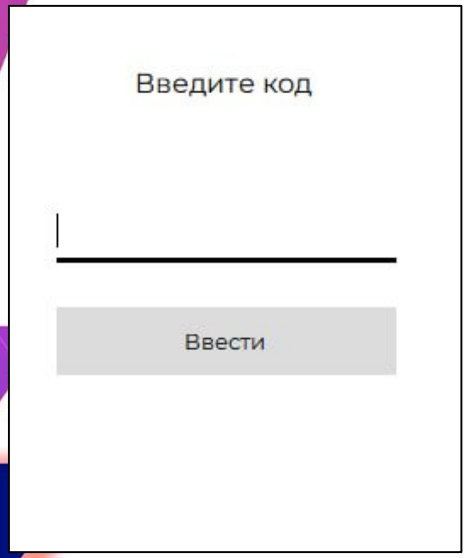

*Основной экран программы (тестового редактора) со скрывающимся меню*




*Экран визуального модуля с добавленными в рабочую область узлами*

# ТЕСТИРОВАНИЕ

*Слева направо скриншоты: экрана блокировки; экрана настройки таймера блокировки экрана ; экрана настройки авто сохранения файла*





# ЗАКЛЮЧЕНИЕ

В ходе тестирования созданного программного продукта был выявлен следующие **недостатки**:

- Сложность кода для ознакомления с принципами написания подобных программ;
- Наличие нерешённой ошибки с блокировочным экраном;
- Отсутствие единого дизайна интерфейса.

---

При этом были отмечены такие **преимущества**, как:

- Простота интерфейса;
- Портативность программы;
- Малый вес;
- Низкие системные требования.

# ЗАКЛЮЧЕНИЕ

В перспективе планируется расширить функционал программы:

- Доработка основных функции добавления текстовых файлов в рабочую область визуального модуля в виде фигурного узла;
- Добавление функции связывания узлов с помощью векторов;
- Добавление функции настройки внешнего вида узлов и векторов связей;
- Добавление функции позволяющей осуществлять поиск по документу.

Подводя итоги, можно сказать, что разработанный программный продукт может быть полезен для индивидуального использования в ведении структурированных записей.