Липецкий государственный технический университет

Факультет автоматизации и информатики

Кафедра автоматизированных систем управления

ИНДИВИДУАЛЬНОЕ ДОМАШНЕЕ ЗАДАНИЕ

По объектно-ориентированному программированию

«Разработка графического редактора на языке C++ с применением механизмов ООП»

Студент \_\_\_\_\_\_\_\_\_ Боков Н.Д.

Группа ПМ-21-1

Руководитель

к.т.н., доцент каф. АСУ \_\_\_\_\_\_\_\_\_ Кургасов В. В.

Липецк 2022 г

**Цель работы:** закрепить навыки использования механизмов ООП на примере реализации графического редактора.

**Задание кафедры:** реализовать на языке C++ редактор графической схемы. В ходе выполненной работы обязательно применение объектно-ориентированных возможностей языка C++: наследования и динамического полиморфизма. Каждый тип элемента схемы должен быть представлен в программе в виде отдельного класса, который наследован от базового класса «графический элемент» (имеющего чисто виртуальную функцию прорисовки). Также необходим один класс «поле рисования», который содержит все графические элементы и отвечает за вызов функций прорисовки. Хранение графических элементов осуществляется с использованием контейнеров стандартной библиотеки C++.

**Вариант 1:** Редактор принципиальной схемы.

**Исходный код**

Содержание файла PrincipalScheme.cpp

#include "PrincipalScheme.h"

**using** **namespace** System;

**using** **namespace** System::Windows::Forms;

[STAThreadAttribute]

**void** main(array<String^>^ args)

{

Application::EnableVisualStyles();

Application::SetCompatibleTextRenderingDefault(false);

PrincipalScheme::PrincipalScheme form;

Application::Run(% form);

}

Содержание файла PrincipalScheme.h

#pragma once

#include "DrawingField.h"

#include "Element.h"

#include "Line.h"

#include "Lamp.h"

#include "Fuse.h"

#include "Ammeter.h"

#include "GalvanicCell.h"

#include "Coil.h"

**namespace** PrincipalScheme {

constexpr **int** CURSOR = **0**;

constexpr **int** LINE = **1**;

constexpr **int** LAMP = **2**;

constexpr **int** FUSE = **3**;

constexpr **int** AMMETER = **4**;

constexpr **int** GALVANICCELL = **5**;

constexpr **int** COIL = **6**;

constexpr **int** DELETE = **10**;

**using** **namespace** System;

**using** **namespace** System::ComponentModel;

**using** **namespace** System::Collections;

**using** **namespace** System::Windows::Forms;

**using** **namespace** System::Data;

**using** **namespace** System::Drawing;

/// <summary>

/// Сводка для VecEditor

/// </summary>

/// <summary>

/// Сводка для VecEditor

/// </summary>

**public** ref **class** **PrincipalScheme** : **public** System::Windows::Forms::Form

{

**public:**

PrincipalScheme(**void**)

{

InitializeComponent();

maxnetworkBoxHeight = networkBox->Height;

maxnetworkBoxWidth = networkBox->Width;

isDown = false;

Bitmap^ bitmap = gcnew Bitmap(networkBox->Size.Width, networkBox->Size.Height);

Graphics^ g = Graphics::FromImage(bitmap);

g->FillRectangle(gcnew SolidBrush(Color::White), **0**, **0**, networkBox->Width, networkBox->Height);

networkBox->Image = bitmap;

draw();

cursorType = CURSOR;

}

**protected:**

~PrincipalScheme()

{

**if** (components)

{

**delete** components;

}

}

**private:** System::Windows::Forms::Button^ clearButton;

**private:** System::Windows::Forms::Button^ cursorButton;

**private:** System::Windows::Forms::Button^ lineButton;

**private:** System::Windows::Forms::Button^ deleteButton;

**private:** System::Windows::Forms::PictureBox^ networkBox;

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

**private:** System::Windows::Forms::Button^ lampButton;

**private:** System::Windows::Forms::Button^ fuseButton;

**private:** System::Windows::Forms::Button^ ammeterButton;

**private:** System::Windows::Forms::Button^ galvaniccellButton;

**private:** System::Windows::Forms::PictureBox^ LampImage;

**private:** System::Windows::Forms::PictureBox^ FuseImage;

**private:** System::Windows::Forms::PictureBox^ AmmeterImage;

**private:** System::Windows::Forms::PictureBox^ GalvanicCellImage;

**private:** System::Windows::Forms::Button^ saveButton;

**private:** System::Windows::Forms::Button^ coilButton;

**private:** System::Windows::Forms::PictureBox^ CoilImage;

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

**protected:**

**private:**

System::ComponentModel::Container^ components;

#pragma region Windows Form Designer generated code

**void** **InitializeComponent**(**void**)

{

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

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

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

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

**this**->networkBox = (gcnew System::Windows::Forms::PictureBox());

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

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

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

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

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

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

**this**->LampImage = (gcnew System::Windows::Forms::PictureBox());

**this**->FuseImage = (gcnew System::Windows::Forms::PictureBox());

**this**->AmmeterImage = (gcnew System::Windows::Forms::PictureBox());

**this**->GalvanicCellImage = (gcnew System::Windows::Forms::PictureBox());

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

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

**this**->CoilImage = (gcnew System::Windows::Forms::PictureBox());

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

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

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

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

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

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

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

**this**->SuspendLayout();

//

// clearButton

//

**this**->clearButton->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", **14**, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

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

**this**->clearButton->Location = System::Drawing::Point(**16**, **13**);

**this**->clearButton->Margin = System::Windows::Forms::Padding(**4**);

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

**this**->clearButton->Size = System::Drawing::Size(**180**, **98**);

**this**->clearButton->TabIndex = **3**;

**this**->clearButton->Text = L"Очистить";

**this**->clearButton->UseVisualStyleBackColor = true;

**this**->clearButton->Click += gcnew System::EventHandler(**this**, &PrincipalScheme::clearButton\_Click);

//

// cursorButton

//

**this**->cursorButton->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", **14**, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

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

**this**->cursorButton->Location = System::Drawing::Point(**16**, **119**);

**this**->cursorButton->Margin = System::Windows::Forms::Padding(**4**);

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

**this**->cursorButton->Size = System::Drawing::Size(**180**, **98**);

**this**->cursorButton->TabIndex = **4**;

**this**->cursorButton->Text = L"Курсор";

**this**->cursorButton->UseVisualStyleBackColor = true;

**this**->cursorButton->Click += gcnew System::EventHandler(**this**, &PrincipalScheme::cursorButton\_Click);

//

// lineButton

//

**this**->lineButton->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", **14**, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

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

**this**->lineButton->Location = System::Drawing::Point(**16**, **264**);

**this**->lineButton->Margin = System::Windows::Forms::Padding(**4**);

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

**this**->lineButton->Size = System::Drawing::Size(**180**, **98**);

**this**->lineButton->TabIndex = **5**;

**this**->lineButton->Text = L"Линия";

**this**->lineButton->UseVisualStyleBackColor = true;

**this**->lineButton->Click += gcnew System::EventHandler(**this**, &PrincipalScheme::lineButton\_Click);

//

// networkBox

//

**this**->networkBox->Location = System::Drawing::Point(**452**, **13**);

**this**->networkBox->Margin = System::Windows::Forms::Padding(**4**);

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

**this**->networkBox->Size = System::Drawing::Size(**1268**, **879**);

**this**->networkBox->TabIndex = **0**;

**this**->networkBox->TabStop = false;

**this**->networkBox->MouseDown += gcnew System::Windows::Forms::MouseEventHandler(**this**, &PrincipalScheme::networkBox\_MouseDown);

**this**->networkBox->MouseMove += gcnew System::Windows::Forms::MouseEventHandler(**this**, &PrincipalScheme::networkBox\_MouseMove);

**this**->networkBox->MouseUp += gcnew System::Windows::Forms::MouseEventHandler(**this**, &PrincipalScheme::networkBox\_MouseUp);

//

// deleteButton

//

**this**->deleteButton->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", **14**, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

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

**this**->deleteButton->Location = System::Drawing::Point(**204**, **119**);

**this**->deleteButton->Margin = System::Windows::Forms::Padding(**4**);

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

**this**->deleteButton->Size = System::Drawing::Size(**180**, **98**);

**this**->deleteButton->TabIndex = **9**;

**this**->deleteButton->Text = L"Удалить";

**this**->deleteButton->UseVisualStyleBackColor = true;

**this**->deleteButton->Click += gcnew System::EventHandler(**this**, &PrincipalScheme::deleteButton\_Click);

//

// label1

//

**this**->label1->AutoSize = true;

**this**->label1->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", **20**, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

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

**this**->label1->Location = System::Drawing::Point(**77**, **221**);

**this**->label1->Margin = System::Windows::Forms::Padding(**4**, **0**, **4**, **0**);

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

**this**->label1->Size = System::Drawing::Size(**291**, **39**);

**this**->label1->TabIndex = **12**;

**this**->label1->Text = L"Элементы схемы";

//

// lampButton

//

**this**->lampButton->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", **14**, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

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

**this**->lampButton->Location = System::Drawing::Point(**16**, **370**);

**this**->lampButton->Margin = System::Windows::Forms::Padding(**4**);

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

**this**->lampButton->Size = System::Drawing::Size(**180**, **98**);

**this**->lampButton->TabIndex = **14**;

**this**->lampButton->Text = L"Лампа";

**this**->lampButton->UseVisualStyleBackColor = true;

**this**->lampButton->Click += gcnew System::EventHandler(**this**, &PrincipalScheme::lampButton\_Click);

//

// fuseButton

//

**this**->fuseButton->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", **14**, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

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

**this**->fuseButton->Location = System::Drawing::Point(**16**, **476**);

**this**->fuseButton->Margin = System::Windows::Forms::Padding(**4**);

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

**this**->fuseButton->Size = System::Drawing::Size(**180**, **98**);

**this**->fuseButton->TabIndex = **15**;

**this**->fuseButton->Text = L"Предохранитель";

**this**->fuseButton->UseVisualStyleBackColor = true;

**this**->fuseButton->Click += gcnew System::EventHandler(**this**, &PrincipalScheme::fuseButton\_Click);

//

// ammeterButton

//

**this**->ammeterButton->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", **14**, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

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

**this**->ammeterButton->Location = System::Drawing::Point(**16**, **582**);

**this**->ammeterButton->Margin = System::Windows::Forms::Padding(**4**);

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

**this**->ammeterButton->Size = System::Drawing::Size(**180**, **98**);

**this**->ammeterButton->TabIndex = **16**;

**this**->ammeterButton->Text = L"Амперметр";

**this**->ammeterButton->UseVisualStyleBackColor = true;

**this**->ammeterButton->Click += gcnew System::EventHandler(**this**, &PrincipalScheme::ammeterButton\_Click);

//

// galvaniccellButton

//

**this**->galvaniccellButton->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", **14**, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

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

**this**->galvaniccellButton->Location = System::Drawing::Point(**16**, **688**);

**this**->galvaniccellButton->Margin = System::Windows::Forms::Padding(**4**);

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

**this**->galvaniccellButton->Size = System::Drawing::Size(**180**, **98**);

**this**->galvaniccellButton->TabIndex = **17**;

**this**->galvaniccellButton->Text = L"Батарея";

**this**->galvaniccellButton->UseVisualStyleBackColor = true;

**this**->galvaniccellButton->Click += gcnew System::EventHandler(**this**, &PrincipalScheme::galvaniccellButton\_Click);

//

// LampImage

//

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

**this**->LampImage->Location = System::Drawing::Point(**204**, **403**);

**this**->LampImage->Margin = System::Windows::Forms::Padding(**4**);

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

**this**->LampImage->Size = System::Drawing::Size(**95**, **36**);

**this**->LampImage->SizeMode = System::Windows::Forms::PictureBoxSizeMode::AutoSize;

**this**->LampImage->TabIndex = **18**;

**this**->LampImage->TabStop = false;

//

// FuseImage

//

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

**this**->FuseImage->Location = System::Drawing::Point(**204**, **509**);

**this**->FuseImage->Margin = System::Windows::Forms::Padding(**4**);

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

**this**->FuseImage->Size = System::Drawing::Size(**164**, **37**);

**this**->FuseImage->SizeMode = System::Windows::Forms::PictureBoxSizeMode::AutoSize;

**this**->FuseImage->TabIndex = **19**;

**this**->FuseImage->TabStop = false;

//

// AmmeterImage

//

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

**this**->AmmeterImage->Location = System::Drawing::Point(**204**, **616**);

**this**->AmmeterImage->Margin = System::Windows::Forms::Padding(**4**);

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

**this**->AmmeterImage->Size = System::Drawing::Size(**148**, **36**);

**this**->AmmeterImage->SizeMode = System::Windows::Forms::PictureBoxSizeMode::AutoSize;

**this**->AmmeterImage->TabIndex = **20**;

**this**->AmmeterImage->TabStop = false;

//

// GalvanicCellImage

//

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

**this**->GalvanicCellImage->Location = System::Drawing::Point(**204**, **714**);

**this**->GalvanicCellImage->Margin = System::Windows::Forms::Padding(**4**);

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

**this**->GalvanicCellImage->Size = System::Drawing::Size(**121**, **49**);

**this**->GalvanicCellImage->SizeMode = System::Windows::Forms::PictureBoxSizeMode::AutoSize;

**this**->GalvanicCellImage->TabIndex = **21**;

**this**->GalvanicCellImage->TabStop = false;

//

// saveButton

//

**this**->saveButton->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", **14**, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

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

**this**->saveButton->Location = System::Drawing::Point(**204**, **13**);

**this**->saveButton->Margin = System::Windows::Forms::Padding(**4**);

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

**this**->saveButton->Size = System::Drawing::Size(**180**, **98**);

**this**->saveButton->TabIndex = **22**;

**this**->saveButton->Text = L"Сохранить";

**this**->saveButton->UseVisualStyleBackColor = true;

**this**->saveButton->Click += gcnew System::EventHandler(**this**, &PrincipalScheme::saveButton\_Click);

//

// coilButton

//

**this**->coilButton->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", **14**, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

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

**this**->coilButton->Location = System::Drawing::Point(**16**, **794**);

**this**->coilButton->Margin = System::Windows::Forms::Padding(**4**);

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

**this**->coilButton->Size = System::Drawing::Size(**180**, **98**);

**this**->coilButton->TabIndex = **23**;

**this**->coilButton->Text = L"Катушка";

**this**->coilButton->UseVisualStyleBackColor = true;

**this**->coilButton->Click += gcnew System::EventHandler(**this**, &PrincipalScheme::coilButton\_Click);

//

// CoilImage

//

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

**this**->CoilImage->Location = System::Drawing::Point(**204**, **825**);

**this**->CoilImage->Margin = System::Windows::Forms::Padding(**4**);

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

**this**->CoilImage->Size = System::Drawing::Size(**181**, **37**);

**this**->CoilImage->SizeMode = System::Windows::Forms::PictureBoxSizeMode::AutoSize;

**this**->CoilImage->TabIndex = **24**;

**this**->CoilImage->TabStop = false;

//

// PrincipalScheme

//

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

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

**this**->ClientSize = System::Drawing::Size(**1733**, **909**);

**this**->Controls->Add(**this**->CoilImage);

**this**->Controls->Add(**this**->coilButton);

**this**->Controls->Add(**this**->saveButton);

**this**->Controls->Add(**this**->GalvanicCellImage);

**this**->Controls->Add(**this**->AmmeterImage);

**this**->Controls->Add(**this**->FuseImage);

**this**->Controls->Add(**this**->LampImage);

**this**->Controls->Add(**this**->galvaniccellButton);

**this**->Controls->Add(**this**->ammeterButton);

**this**->Controls->Add(**this**->fuseButton);

**this**->Controls->Add(**this**->lampButton);

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

**this**->Controls->Add(**this**->networkBox);

**this**->Controls->Add(**this**->deleteButton);

**this**->Controls->Add(**this**->lineButton);

**this**->Controls->Add(**this**->cursorButton);

**this**->Controls->Add(**this**->clearButton);

**this**->Margin = System::Windows::Forms::Padding(**4**);

**this**->MinimumSize = System::Drawing::Size(**715**, **555**);

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

**this**->Text = L"PrincipalScheme";

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

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

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

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

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

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

**this**->ResumeLayout(false);

**this**->PerformLayout();

}

#pragma endregion

**private:** Point DownPosition;

**private:** Point CurrentPosition;

**private:** Point PrevPosition;

**private:** DrawingField drawingField;

**private:** **int** maxnetworkBoxWidth;

**private:** **int** maxnetworkBoxHeight;

**private:** **int** cursorType;

**private:** Line^ newLine;

**private:** Lamp^ newLamp;

**private:** Fuse^ newFuse;

**private:** Ammeter^ newAmmeter;

**private:** GalvanicCell^ newGalvanicCell;

**private:** Coil^ newCoil;

**private:** Element^ movingElement;

**private:** **bool** isDown;

**private:** Element^ deletingElement;

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

drawingField.clear();

draw();

}

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

cursorType = CURSOR;

}

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

cursorType = LINE;

}

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

cursorType = DELETE;

}

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

cursorType = LAMP;

}

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

cursorType = FUSE;

}

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

cursorType = AMMETER;

}

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

cursorType = GALVANICCELL;

}

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

cursorType = COIL;

}

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

SaveFileDialog^ save = gcnew SaveFileDialog();

save->Title = "Сохранить";

save->Filter = "png files (\*.png)|\*.png|All files (\*.\*)|\*.\*";

**if** (save->ShowDialog() == Windows::Forms::DialogResult::OK)

{

networkBox->Image->Save(save->FileName);

}

}

**private:** System::Void draw() {

Graphics^ g = Graphics::FromImage(networkBox->Image);

g->FillRectangle(gcnew SolidBrush(Color::White), **0**, **0**, networkBox->Width, networkBox->Height);

drawingField.drawElement(g);

networkBox->Invalidate();

}

**private:** System::Void networkBox\_MouseDown(System::Object^ sender, System::Windows::Forms::MouseEventArgs^ e) {

isDown = true;

DownPosition = Point(e->X, e->Y);

PrevPosition = Point(e->X, e->Y);

draw();

**switch** (cursorType)

{

**case** CURSOR:

movingElement = drawingField.findElement(e->X, e->Y);

**if** (movingElement != nullptr)

{

**switch** (movingElement->type)

{

**case** LINE:

movingElement = (Line^)movingElement;

**break**;

**case** LAMP:

movingElement = (Lamp^)movingElement;

**break**;

**case** FUSE:

movingElement = (Fuse^)movingElement;

**break**;

**case** GALVANICCELL:

movingElement = (GalvanicCell^)movingElement;

**break**;

**case** COIL:

movingElement = (Coil^)movingElement;

**break**;

}

}

**break**;

**case** LINE:

newLine = gcnew Line(DownPosition, DownPosition);

newLine->draw(Graphics::FromImage(networkBox->Image));

**break**;

**case** DELETE:

deletingElement = drawingField.findElement(e->X, e->Y);

**if** (deletingElement != nullptr)

drawingField.deleteElement(deletingElement);

deletingElement = nullptr;

**break**;

**case** LAMP:

newLamp = gcnew Lamp(DownPosition);

newLamp->draw(Graphics::FromImage(networkBox->Image));

**break**;

**case** FUSE:

newFuse = gcnew Fuse(DownPosition);

newFuse->draw(Graphics::FromImage(networkBox->Image));

**break**;

**case** AMMETER:

newAmmeter = gcnew Ammeter(DownPosition);

newAmmeter->draw(Graphics::FromImage(networkBox->Image));

**break**;

**case** GALVANICCELL:

newGalvanicCell = gcnew GalvanicCell(DownPosition);

newGalvanicCell->draw(Graphics::FromImage(networkBox->Image));

**break**;

**case** COIL:

newCoil = gcnew Coil(DownPosition);

newCoil->draw(Graphics::FromImage(networkBox->Image));

**break**;

}

networkBox->Invalidate();

}

**private:** System::Void networkBox\_MouseMove(System::Object^ sender, System::Windows::Forms::MouseEventArgs^ e) {

**if** (isDown)

{

CurrentPosition = Point(e->X, e->Y);

**if** (movingElement == nullptr) {

draw();

**switch** (cursorType)

{

**case** CURSOR:

**break**;

**case** LINE:

newLine->setFirst(DownPosition);

newLine->setSecond(CurrentPosition);

newLine->draw(Graphics::FromImage(networkBox->Image));

**break**;

**case** LAMP:

newLamp->setLamp(CurrentPosition);

newLamp->draw(Graphics::FromImage(networkBox->Image));

**break**;

**case** FUSE:

newFuse->setFuse(CurrentPosition);

newFuse->draw(Graphics::FromImage(networkBox->Image));

**break**;

**case** AMMETER:

newAmmeter->setAmmeter(CurrentPosition);

newAmmeter->draw(Graphics::FromImage(networkBox->Image));

**break**;

**case** GALVANICCELL:

newGalvanicCell->setGalvanicCell(CurrentPosition);

newGalvanicCell->draw(Graphics::FromImage(networkBox->Image));

**break**;

**case** COIL:

newCoil->setCoil(CurrentPosition);

newCoil->draw(Graphics::FromImage(networkBox->Image));

**break**;

}

}

**else**

{

changemovingElementPosition();

draw();

movingElement->draw(Graphics::FromImage(networkBox->Image));

PrevPosition = CurrentPosition;

}

networkBox->Invalidate();

}

}

**private:** System::Void networkBox\_MouseUp(System::Object^ sender, System::Windows::Forms::MouseEventArgs^ e) {

isDown = false;

**switch** (cursorType)

{

**case** CURSOR:

**if** (movingElement != nullptr) {

drawingField.addElement(movingElement);

movingElement = nullptr;

}

**break**;

**case** LINE:

drawingField.addElement(newLine);

**break**;

**case** LAMP:

drawingField.addElement(newLamp);

**break**;

**case** FUSE:

drawingField.addElement(newFuse);

**break**;

**case** AMMETER:

drawingField.addElement(newAmmeter);

**break**;

**case** GALVANICCELL:

drawingField.addElement(newGalvanicCell);

**break**;

**case** COIL:

drawingField.addElement(newCoil);

**break**;

}

draw();

}

**private:** System::Void changemovingElementPosition()

{

Point p1;

Point p2;

**int** dX = CurrentPosition.X - PrevPosition.X;

**int** dY = CurrentPosition.Y - PrevPosition.Y;

**switch** (movingElement->type)

{

**case** LINE:

p1 = ((Line^)movingElement)->getFirst();

p2 = ((Line^)movingElement)->getSecond();

**if** (p1.X + dX < **0** || p1.X + dX > networkBox->Width || p1.Y + dY < **0** || p1.Y + dY > networkBox->Height)

**return**;

**if** (p2.X + dX < **0** || p2.X + dX > networkBox->Width || p2.Y + dY < **0** || p2.Y + dY > networkBox->Height)

**return**;

((Line^)movingElement)->setFirst(Point(p1.X + dX, p1.Y + dY));

((Line^)movingElement)->setSecond(Point(p2.X + dX, p2.Y + dY));

**break**;

**case** LAMP:

p1 = ((Lamp^)movingElement)->getLamp();

**if** (p1.X + dX < **0** || p1.X + dX > networkBox->Width || p1.Y + dY < **0** || p1.Y + dY > networkBox->Height)

**return**;

p1.X = p1.X + dX;

p1.Y = p1.Y + dY;

((Lamp^)movingElement)->setLamp(p1);

**break**;

**case** FUSE:

p1 = ((Fuse^)movingElement)->getFuse();

**if** (p1.X + dX < **0** || p1.X + dX > networkBox->Width || p1.Y + dY < **0** || p1.Y + dY > networkBox->Height)

**return**;

p1.X = p1.X + dX;

p1.Y = p1.Y + dY;

((Fuse^)movingElement)->setFuse(p1);

**break**;

**case** AMMETER:

p1 = ((Ammeter^)movingElement)->getAmmeter();

**if** (p1.X + dX < **0** || p1.X + dX > networkBox->Width || p1.Y + dY < **0** || p1.Y + dY > networkBox->Height)

**return**;

p1.X = p1.X + dX;

p1.Y = p1.Y + dY;

((Ammeter^)movingElement)->setAmmeter(p1);

**break**;

**case** GALVANICCELL:

p1 = ((GalvanicCell^)movingElement)->getGalvanicCell();

**if** (p1.X + dX < **0** || p1.X + dX > networkBox->Width || p1.Y + dY < **0** || p1.Y + dY > networkBox->Height)

**return**;

p1.X = p1.X + dX;

p1.Y = p1.Y + dY;

((GalvanicCell^)movingElement)->setGalvanicCell(p1);

**break**;

**case** COIL:

p1 = ((Coil^)movingElement)->getCoil();

**if** (p1.X + dX < **0** || p1.X + dX > networkBox->Width || p1.Y + dY < **0** || p1.Y + dY > networkBox->Height)

**return**;

p1.X = p1.X + dX;

p1.Y = p1.Y + dY;

((Coil^)movingElement)->setCoil(p1);

**break**;

**break**;

}

}

};

}

Содержание файла DrawingField.cpp

#include "DrawingField.h"

**void** DrawingField::addElement(Element^ element)

{

elements.push\_front(element);

}

**void** DrawingField::drawElement(Graphics^ g)

{

**if** (!elements.empty()) {

**for** each (Element ^ element **in** elements) {

element->draw(g);

}

}

}

Element^ DrawingField::findElement(**int** x, **int** y)

{

Element^ result = nullptr;

**for** **each** (Element ^ element **in** elements)

**if** (element->find(x, y) != nullptr)

{

result = element;

elements.remove(element);

**break**;

}

**return** result;

}

**void** DrawingField::clear() {

elements.clear();

}

**void** DrawingField::deleteElement(Element^ element)

{

elements.remove(element);

}

Содержание файла DrawingField.h

#pragma once

#include <cliext/list>

#include "Element.h"

ref **class** **DrawingField**

{

**private:**

cliext::list<Element^> elements;

**public:**

**void** **addElement**(Element^ element);

**void** **drawElement**(Graphics^ g);

Element^ findElement(**int** x, **int** y);

**void** **deleteElement**(Element^ element);

**void** **clear**();

};

Содержание файла Element.h

#pragma once

**using** **namespace** System::Drawing;

ref **class** **Element** abstract

{

**public:**

**int** type;

**virtual** **void** draw(Graphics^ g) = **0**;

**virtual** Element^ find(**int** x, **int** y) = **0**;

};

Содержание файла GalvanicCell.cpp

#include "GalvanicCell.h"

**using** **namespace** System::Drawing;

GalvanicCell::GalvanicCell(Point point) {

**this**->p = point;

type = **5**;

}

Point GalvanicCell::getGalvanicCell() {

**return** p;

}

**void** GalvanicCell::setGalvanicCell(Point point) {

**this**->p = point;

}

**void** GalvanicCell::draw(Graphics^ g) {

Image^ img = Image::FromFile("galvaniccell.png");

width = img->Width;

height = img->Height;

g->DrawImage(img, p);

}

Element^ GalvanicCell::find(**int** x, **int** y)

{

**if** (x >= p.X && x <= p.X + width) {

**if** (y >= p.Y && y <= p.Y + height)

**return** **this**;

}

**return** nullptr;

}

Содержание файла GalvanicCell.h

#pragma once

#include "Element.h"

ref **class** **GalvanicCell** :

**public** Element

{

**private:**

Point p;

**int** width, height;

**public:**

GalvanicCell(Point point);

Point **getGalvanicCell**();

**void** **setGalvanicCell**(Point point);

**virtual** **void** draw(Graphics^ g) override;

**virtual** Element^ find(**int** x, **int** y) override;

};

Содержание файла Line.cpp

#include "Line.h"

#include <cmath>

Line::Line(Point one, Point two)

{

**this**->one = one;

**this**->two = two;

type = **1**;

}

Point Line::getFirst()

{

**return** one;

}

Point Line::getSecond()

{

**return** two;

}

**void** Line::setFirst(Point p)

{

one = p;

}

**void** Line::setSecond(Point p)

{

two = p;

}

**void** Line::draw(Graphics^ g)

{

g->DrawLine(gcnew Pen(Color::Black, **6**), one, two);

}

Element^ Line::find(**int** x, **int** y)

{

**if** (x < fmin(one.X, two.X) - **3** || x > fmax(one.X, two.X) + **3** || y < fmin(one.Y, two.Y) - **3** || y > fmax(one.Y, two.Y) + **3**)

**return** nullptr;

**int** ab = abs((one.Y - two.Y) \* x + (two.X - one.X) \* y + (one.X \* two.Y - two.X \* one.Y));

**if** (sqrt(abs((one.Y - two.Y) \* x + (two.X - one.X) \* y + (one.X \* two.Y - two.X \* one.Y))) <= **49**)

**return** **this**;

**return** nullptr;

}

Содержание файла Line.h

#pragma once

#include "Element.h"

ref **class** **Line** :

**public** Element

{

**private:**

Point one;

Point two;

**public:**

Line(Point one, Point two);

Point **getFirst**();

Point **getSecond**();

**void** **setFirst**(Point p);

**void** **setSecond**(Point p);

**virtual** **void** draw(Graphics^ g) override;

**virtual** Element^ find(**int** x, **int** y) override;

};

Содержание файла Coil.cpp

#include "Coil.h"

**using** **namespace** System::Drawing;

Coil::Coil(Point point) {

**this**->p = point;

type = **6**;

}

Point Coil::getCoil() {

**return** p;

}

**void** Coil::setCoil(Point point) {

**this**->p = point;

}

**void** Coil::draw(Graphics^ g) {

Image^ img = Image::FromFile("coil.png");

width = img->Width;

height = img->Height;

g->DrawImage(img, p);

}

Element^ Coil::find(**int** x, **int** y)

{

**if** (x >= p.X && x <= p.X + width) {

**if** (y >= p.Y && y <= p.Y + height)

**return** **this**;

}

**return** nullptr;

}

Содержание файла Coil.h

#pragma once

#include "Element.h"

ref class Coil :

public Element

{

private:

Point p;

int width, height;

public:

Coil(Point point);

Point getCoil();

void setCoil(Point point);

virtual void draw(Graphics^ g) override;

virtual Element^ find(int x, int y) override;

};

Содержание файла Lamp.cpp

#include "Lamp.h"

**using** **namespace** System::Drawing;

Lamp::Lamp(Point point) {

**this**->p = point;

type = **2**;

}

Point Lamp::getLamp() {

**return** p;

}

**void** Lamp::setLamp(Point point) {

**this**->p = point;

}

**void** Lamp::draw(Graphics^ g) {

Image^ img = Image::FromFile("lamp.png");

width = img->Width;

height = img->Height;

g->DrawImage(img, p);

}

Element^ Lamp::find(**int** x, **int** y)

{

**if** (x >= p.X && x <= p.X + width) {

**if** (y >= p.Y && y <= p.Y + height)

**return** **this**;

}

**return** nullptr;

}

Содержание файла Lamp.h

#pragma once

#include "Element.h"

**using** **namespace** System::Drawing;

ref **class** **Lamp** :

**public** Element

{

**private:**

Point p;

**int** width, height;

**public:**

Lamp(Point point);

Point **getLamp**();

**void** **setLamp**(Point point);

**virtual** **void** draw(Graphics^ g) override;

**virtual** Element^ find(**int** x, **int** y) override;

};

Содержание файла Fuse.cpp

#include "Fuse.h"

**using** **namespace** System::Drawing;

Fuse::Fuse(Point point) {

**this**->p = point;

type = **3**;

}

Point Fuse::getFuse() {

**return** p;

}

**void** Fuse::setFuse(Point point) {

**this**->p = point;

}

**void** Fuse::draw(Graphics^ g) {

Image^ img = Image::FromFile("fuse.png");

width = img->Width;

height = img->Height;

g->DrawImage(img, p);

}

Element^ Fuse::find(**int** x, **int** y)

{

**if** (x >= p.X && x <= p.X + width) {

**if** (y >= p.Y && y <= p.Y + height)

**return** **this**;

}

**return** nullptr;

}

Содержание файла Fuse.h

#pragma once

#include "Element.h"

**using** **namespace** System::Drawing;

ref **class** **Fuse** :

**public** Element

{

**private:**

Point p;

**int** width, height;

**public:**

Fuse(Point point);

Point **getFuse**();

**void** **setFuse**(Point point);

**virtual** **void** draw(Graphics^ g) override;

**virtual** Element^ find(**int** x, **int** y) override;

};

Содержание файла Ammeter.cpp

#include "Ammeter.h"

**using** **namespace** System::Drawing;

Ammeter::Ammeter(Point point) {

**this**->p = point;

type = **4**;

}

Point Ammeter::getAmmeter() {

**return** p;

}

**void** Ammeter::setAmmeter(Point point) {

**this**->p = point;

}

**void** Ammeter::draw(Graphics^ g) {

Image^ img = Image::FromFile("ammeter.png");

width = img->Width;

height = img->Height;

g->DrawImage(img, p);

}

Element^ Ammeter::find(**int** x, **int** y)

{

**if** (x >= p.X && x <= p.X + width) {

**if** (y >= p.Y && y <= p.Y + height)

**return** **this**;

}

**return** nullptr;

}

Содержание файла Ammeter.h

#pragma once

#include "Element.h"

ref **class** **Ammeter** :

**public** Element

{

**private:**

Point p;

**int** width, height;

**public:**

Ammeter(Point point);

Point **getAmmeter**();

**void** **setAmmeter**(Point point);

**virtual** **void** draw(Graphics^ g) override;

**virtual** Element^ find(**int** x, **int** y) override;

};

**Результаты работы алгоритма**

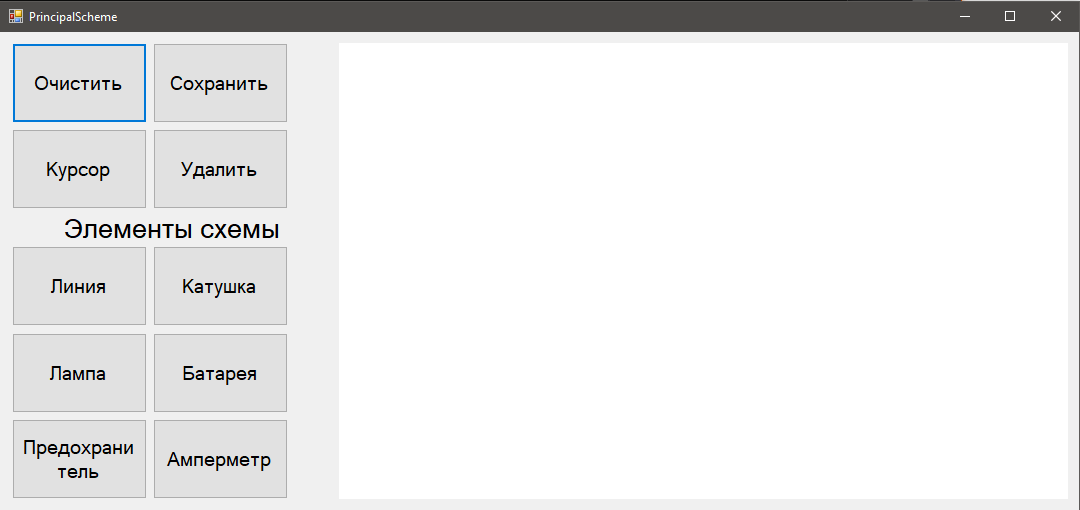


Рисунок 1. Начальное окно программы.

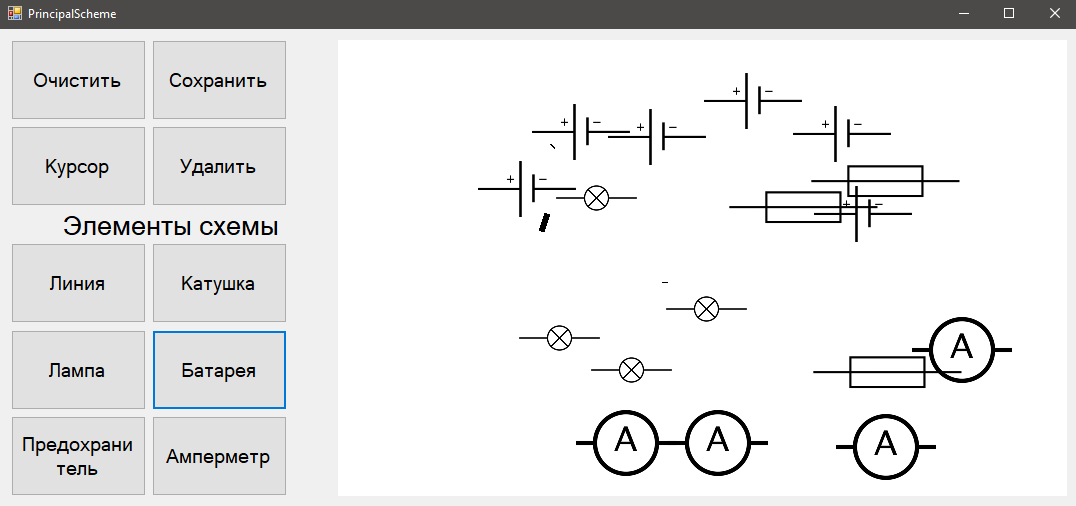


Рисунок 2. Пример расстановки элементов.

Для того чтобы поставить элемент на экран нужно выбрать её и кликнуть ЛКМ на экран

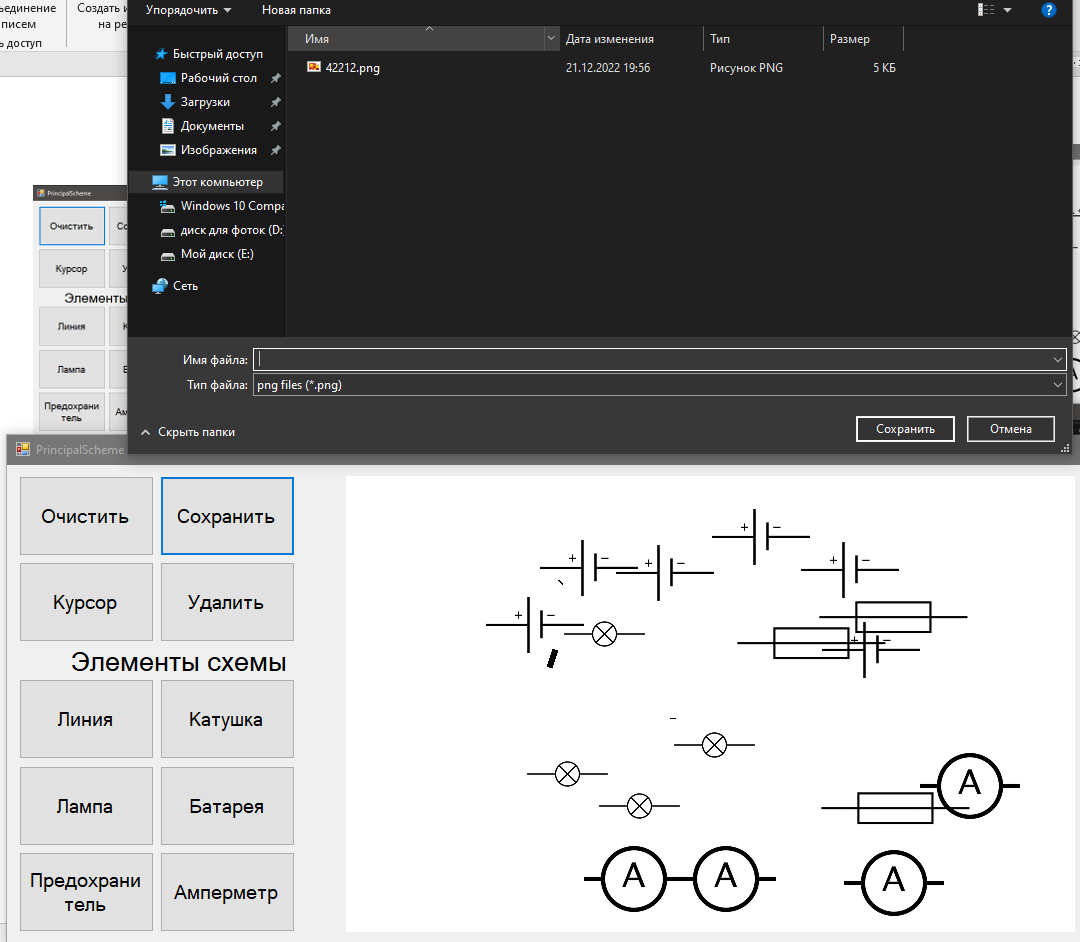


Рисунок 3. Пример сохранения изображения.

После того, как пользователь расставит элементы, он может сохранить изображение в удобную для него папку

**Ответы на контрольные вопросы**

**1. Зачем нужен перегруженный оператор присваивания?**

Перегруженный оператор присваивания используется для присваивания объектов одного конкретного класса друг другу, если объекты класса не являются обычными типами данных.

**2. Зачем нужен механизм наследования?**

Наследование – это механизм создания нового класса на основе уже существующего. Основное назначение механизма наследования — повторное использование кодов, так как большинство используемых типов данных являются вариантами друг друга, и писать для каждого свой класс нецелесообразно.

Объекты разных классов и сами классы могут находиться в отношении наследования, при котором формируется иерархия объектов, соответствующая заранее предусмотренной иерархии классов.

**3. Зачем используются модификаторы при наследовании классов? Какие это модификаторы?**

Существуют модификаторы доступа и наследования - public, private, protected.

В модификаторах доступа:

Public – доступ открыт всем, кто видит определение данного класса.

Protected – доступ открыт классам, производным от данного. То есть, производные классы получают свободный доступ к таким свойствам или метода. Все другие классы такого доступа не имеют.

Private – доступ открыт самому классу (т.е. функциям-членам данного класса) и друзьям (friend) данного класса – как функциям, так и классам. Однако производные классы не получают доступа к этим данным совсем. И все другие классы такого доступа не имеют.

В C++ существует public-наследование, private-наследование и protected-наследование. В зависимости от того, какой тип используется, изменяется доступ к членам базового класса для клиентов производного.

**4. Зачем нужен механизм полиморфизма?**

Полиморфизм – это свойство, которое позволяет одно и то же имя использовать для решения двух или более схожих, но технически разных задач. Целью полиморфизма, применительно к объектно-ориентированному программированию, является использование одного имени для задания общих для класса действий. Выполнение каждого конкретного действия будет определяться типом данных. Тип данных, который используется при вызове функции, определяет, какая конкретная версия функции действительно выполняется. В С++ можно использовать одно имя функции для множества различных действий. Это называется перегрузкой функций. В более общем смысле, концепцией полиморфизма является идея «один интерфейс, множество методов».

**5. Что понимается под динамическим полиморфизме?**

Динамический полиморфизм предстает перед нами в форме классов с виртуальными функциями и объектов, работа с которыми осуществляется косвенно – через указатели или ссылки, в то время как статический полиморфизм включает шаблоны классов и функций.

**6. Что такое интерфейс класса?**

Интерфейс – это класс, который не имеет переменных-членов, и все методы которого являются чистыми виртуальными функциями.

**7. Зачем нужен чисто виртуальный метод?**

Чисто виртуальный метод является методом, который объявляется в базовом классе, но не имеет в нем определения. Поскольку она не имеет определения, то есть тела в этом базовом классе, то всякий производный класс обязан иметь свою собственную версию определения данного метода. При введении чисто виртуальной функции в производном классе обязательно необходимо определить свою собственную реализацию этой функции. Если класс не будет содержать определения этой функции, то компилятор выдаст ошибку.

**8. Какой класс называется абстрактным?**

Если класс имеет хотя бы одну чисто виртуальную функцию, то такой класс называется абстрактным. Важной особенностью абстрактных классов является то, что он не имеет объектов. Вместо этого абстрактный класс служит в качестве базового для других производных классов. Причина, по которой абстрактный класс не может быть использован для объявления объекта, заключается в том, что одна или несколько его функций-членов не имеют определения. Тем не менее, даже если базовый класс является абстрактным, все равно можно объявлять указатели или ссылки на него, с помощью которых затем поддерживается полиморфизм времени исполнения.

**Вывод:** в данном индивидуальном задании я закрепил навыки ООП и создал графический редактор элементов принципиальной схемы.