МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«ЛИПЕЦКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра прикладной математики

Отчет по индивидуальному домашнему заданию.

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

Вариант 8

Выполнил:

Студент гр. ПМ-21-1 Корнев А.А.

Проверил:

доц., к.п.н. кафедры АСУ Кургасов В. В.

Липецк 2022г.

Оглавление

[Цель работы: 2](#_Toc359241709)

[Задание кафедры: 3](#_Toc629676325)

[Вариант 8: 4](#_Toc2146798534)

[Исходный код: 5](#_Toc1544610195)

[Результат работы программы: 11](#_Toc2109691475)

[Контрольные вопросы: 12](#_Toc1671172892)

[Вывод: 15](#_Toc485984346)

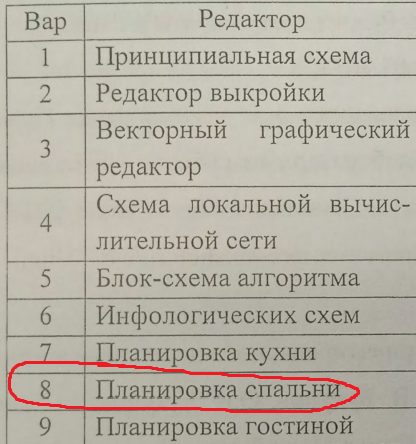
# Цель работы:

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 8:



# Исходный код:

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Планировка\_спальни

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent(); // инициализация компонентов

}

PictureBox pct; //объявление PictureBox

public int sch = 0; //заводим счетчик для того, чтобы была возможность создавать несколько элементов

private void button1\_Click(object sender, EventArgs e)

{

pct = new PictureBox();

pct.Height = 200; //параметры элемента

pct.Width = 300;

pct.Name = "pct" + sch;

pct.Left = 20 + sch \* 110; //координаты появления элемента

pct.Top = 30;

pct.Image = Properties.Resources.cart; //прикрепляем картинку к PictureBox

panel1.Controls.Add(pct); //добавляем элемент на панель

sch += 1;

pct.MouseUp += pct\_MouseUp; //ссылки на события

pct.MouseDown += pct\_MouseDown;

pct.MouseMove += pct\_MouseMove;

}

//реализация движения объектов

private bool Dragging1;

private int xPos1; //координаты элемента

private int yPos1;

private void pct\_MouseUp(object sender, MouseEventArgs e)

{

Dragging1 = false;

}

private void pct\_MouseDown(object sender, MouseEventArgs e)

{

if (e.Button == MouseButtons.Left)

{

Dragging1 = true;

xPos1 = e.X;

yPos1 = e.Y;

}

}

private void pct\_MouseMove(object sender, MouseEventArgs e)

{

Control c = sender as Control;

if (Dragging1 && c != null)

{

c.Top = e.Y + c.Top - yPos1;

c.Left = e.X + c.Left - xPos1;

}

}

PictureBox pictureBox1;

public int sch1 = 0;

private void button3\_Click(object sender, EventArgs e)

{

pictureBox1 = new PictureBox();

pictureBox1.Height = 250;

pictureBox1.Width = 300;

pictureBox1.Name = "pictureBox1" + sch1;

pictureBox1.Left = 20 + sch1 \* 110;

pictureBox1.Top = 30;

pictureBox1.Image = Properties.Resources.money;

panel1.Controls.Add(pictureBox1);

sch1 += 1;

pictureBox1.MouseUp += pictureBox1\_MouseUp\_1;

pictureBox1.MouseDown += pictureBox1\_MouseDown\_1;

pictureBox1.MouseMove += pictureBox1\_MouseMove\_1;

}

private bool Dragging;

private int xPos;

private int yPos;

private void pictureBox1\_MouseUp\_1(object sender, MouseEventArgs e)

{

Dragging = false;

}

private void pictureBox1\_MouseDown\_1(object sender, MouseEventArgs e)

{

if (e.Button == MouseButtons.Left)

{

Dragging = true;

xPos = e.X;

yPos = e.Y;

}

}

private void pictureBox1\_MouseMove\_1(object sender, MouseEventArgs e)

{

Control c = sender as Control;

if (Dragging && c != null)

{

c.Top = e.Y + c.Top - yPos;

c.Left = e.X + c.Left - xPos;

}

}

PictureBox pictureBox2;

public int sch2 = 0;

private void button4\_Click(object sender, EventArgs e)

{

pictureBox2 = new PictureBox();

pictureBox2.Height = 120;

pictureBox2.Width = 120;

pictureBox2.Name = "pictureBox2" + sch2;

pictureBox2.Left = 20 + sch2 \* 110;

pictureBox2.Top = 30;

pictureBox2.Image = Properties.Resources.showcase1;

panel1.Controls.Add(pictureBox2);

sch2 += 1;

pictureBox2.MouseUp += pictureBox2\_MouseUp\_2;

pictureBox2.MouseDown += pictureBox2\_MouseDown\_2;

pictureBox2.MouseMove += pictureBox2\_MouseMove\_2;

}

private bool Dragging2;

private int xPos2;

private int yPos2;

private void pictureBox2\_MouseUp\_2(object sender, MouseEventArgs e)

{

Dragging2 = false;

}

private void pictureBox2\_MouseDown\_2(object sender, MouseEventArgs e)

{

if (e.Button == MouseButtons.Left)

{

Dragging2 = true;

xPos2 = e.X;

yPos2 = e.Y;

}

}

private void pictureBox2\_MouseMove\_2(object sender, MouseEventArgs e)

{

Control c = sender as Control;

if (Dragging2 && c != null)

{

c.Top = e.Y + c.Top - yPos2;

c.Left = e.X + c.Left - xPos2;

}

}

PictureBox pictureBox3;

public int sch3 = 0;

private void button2\_Click(object sender, EventArgs e)

{

pictureBox3 = new PictureBox();

pictureBox3.Height = 120;

pictureBox3.Width = 120;

pictureBox3.Name = "pictureBox3" + sch3;

pictureBox3.Left = 20 + sch3 \* 110;

pictureBox3.Top = 30;

pictureBox3.Image = Properties.Resources.showcase3; //прикрепляем изображение к PictureBox

panel1.Controls.Add(pictureBox3);

sch3 += 1;

pictureBox3.MouseUp += pictureBox3\_MouseUp\_3;

pictureBox3.MouseDown += pictureBox3\_MouseDown\_3;

pictureBox3.MouseMove += pictureBox3\_MouseMove\_3;

}

private bool Dragging3;

private int xPos3;

private int yPos3;

private void pictureBox3\_MouseUp\_3(object sender, MouseEventArgs e)

{

Dragging3 = false;

}

private void pictureBox3\_MouseDown\_3(object sender, MouseEventArgs e)

{

if (e.Button == MouseButtons.Left)

{

Dragging3 = true;

xPos3 = e.X;

yPos3 = e.Y;

}

}

private void pictureBox3\_MouseMove\_3(object sender, MouseEventArgs e)

{

Control c = sender as Control;

if (Dragging3 && c != null)

{

c.Top = e.Y + c.Top - yPos3;

c.Left = e.X + c.Left - xPos3;

}

}

//реализация кнопок удаления элементов

private void button5\_Click(object sender, EventArgs e)

{

pct.Dispose();

sch = 0;

}

private void button6\_Click(object sender, EventArgs e)

{

pictureBox1.Dispose();

sch1 = 0;

}

private void button7\_Click(object sender, EventArgs e)

{

pictureBox2.Dispose();

sch2 = 0;

}

private void button8\_Click(object sender, EventArgs e)

{

pictureBox3.Dispose();

sch3 = 0;

}

private void Form1\_Load(object sender, EventArgs e)

{

}

private void panel1\_Paint(object sender, PaintEventArgs e)

{

}

}

}

# Результат работы программы:

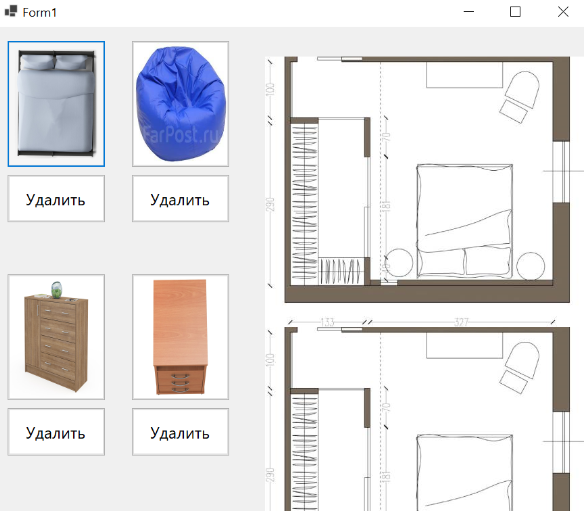


Рисунок 1 - Начальное окно работы программы

# Контрольные вопросы:

1. Зачем нужен перегруженный оператор присваивания?

Операторы копирования-присвоения позволяют **таким классам, как** basic\_string**,** избегать ненужного выделения ресурсов, когда они могут их повторно использовать (в данном случае память). Поэтому, когда вы назначаете базовую строку, перегруженный оператор присваивания копирования избегает выделения памяти и просто копирует строковые данные непосредственно в существующий буфер.

1. Зачем нужен механизм наследования?

Наследование — это механизм создания нового класса на основе уже существующего. При этом к существующему классу могут быть добавлены новые элементы (данные и функции), либо существующие функции могут быть изменены.

Основное назначение механизма наследования — повторное использование кодов, так как большинство используемых типов данных являются вариантами друг друга, и писать для каждого свой класс нецелесообразно.

1. Зачем используются модификаторы при наследовании классов? Какие это модификаторы?

Важнейшим свойством объектно-ориентированного программирования является наследование. Наследование – это способ повторного использования программного обеспечения, при котором новые производные классы (наследники) создаются на базе уже существующих базовых классов (родителей). При создании новый класс является наследником членов и методов ранее определенного базового класса. Создаваемый путем наследования класс является производным (derived class), который в свою очередь может выступать в качестве базового класса (based class) для создаваемых классов. Если имена методов производного и базового классов совпадают, то методы производного класса перегружают методы базового класса.

При описании класса в его заголовке перечисляются все классы, являющиеся для него базовыми. Возможность обращения к элементам этих классов регулируется с помощью модификаторов наследования private, protected и public:

class имя : [private | protected | public] базовый\_класс{тело класса};

Если базовых классов несколько, они перечисляются через запятую. Перед каждым может стоять свой модификатор наследования. По умолчанию он private.

Если задан модификатор наследования public, наследование называется *открытым*. Использование модификатора protected делает наследование *защищенным*, а модификатора private - *закрытым*.

До сих пор мы рассматривали только спецификаторы доступа private и public, применяемые к элементам класса. Для любого элемента класса может также использоваться спецификатор protected, который для одиночных классов, не входящих в иерархию, равносилен private. Разница между ними проявляется при наследовании. Возможные сочетания модификаторов и спецификаторов доступа приведены в таблице:

|  |  |  |
| --- | --- | --- |
| Модификатор наследования | Спецификатор базового класса | Доступ в производном классе |
| private | private protected public | нет private private |
| protected | private protected public | нет protected protected |
| public | private protected public | нет protected public |

Как видно из таблицы, private элементы базового класса в производном классе недоступны вне зависимости от ключа. Обращение к ним может осуществляться только через методы базового класса.

Элементы protected при наследовании с ключом private становятся в производном классе private, в остальных случаях права доступа к ним не изменяются.

Доступ к элементам public при наследовании становится соответствующим ключу доступа.

1. Зачем нужен механизм полиморфизма?

Полиморфизм позволяет писать более абстрактные, расширяемые программы, один и тот же код используется для объектов разных классов, улучшается читабельность кода. Полиморфизм позволяет избавить разработчика от написания, чтения и отладки множества if-else/switch-case конструкций.

Полиморфизм может применяться также и к операторам. Фактически во всех языках программирования ограниченно применяется полиморфизм, например, в арифметических операторах. В С++ вы можете применить эту концепцию к заданным вами типам данных. Такой тип полиморфизма называется перегрузкой операторов.

1. Что понимается под динамическим полиморфизмом?

**Динамический** **полиморфизм**– выбор функции производится по используемому при выводе имени, типу вызываемого объекта и типу используемый параметров. Если объявленный тип не совпадает с типом использованного объекта, тип этого объекта идентифицируется и проверяется во время исполнения программы, что позволяет осуществить правильный выбор соответствующей функции. Это и называется **динамическим** **полиморфизмом** (поздним связыванием).

1. Что такое интерфейс класса?

Интерфейс — это совокупность методов и правил взаимодействия элементов системы. Другими словами, интерфейс определяет как элементы будут взаимодействовать между собой.

* Интерфейс двери — наличие ручки;
* Интерфейс автомобиля — наличие руля, педалей, рычага коробки передач;
* Интерфейс дискового телефона — трубка + дисковый набиратель номера.

Когда вы используете эти "объекты", вы уверены в том, что вы сможете использовать их подобным образом. Благодаря тому, что вы знакомы с их интерфейсом.

Графическую часть программы часто называют интерфейсом, потому что она определяет каким образом вы сможете использовать основной функционал, заложенный в программе.

Интерфейс определяет каким образом мы можем использовать объект.

1. Зачем нужен чисто виртуальный метод?

**виртуальная функция** или **виртуальный метод** - это наследуемая и переопределяемая функция или метод для которых облегчается динамическая отправка. Эта концепция является важной частью (во время выполнения) части полиморфизма объектно-ориентированного программирования (ООП). Виртуальная функция определяет целевую функцию, которая должна выполняться, но цель может быть неизвестна во время компиляции.

Виртуальные функции разрешаются "с опозданием". Если рассматриваемая функция является "виртуальной" в базовом классе, реализация функции наиболее производного класса вызывается в соответствии с фактическим типом объекта, на который ссылается, независимо от объявленного типа указателя или ссылки. Если она не является "виртуальной", метод разрешается "досрочно" и выбирается в соответствии с объявленным типом указателя или ссылки.

Виртуальные функции позволяют программе вызывать методы, которые необязательно даже существуют на момент компиляции кода.

1. Какой класс называется абстрактным?

Абстрактные классы в объектно-ориентированном программировании — это базовые классы, которые можно наследовать, но нельзя реализовывать. То есть на их основе нельзя создать объект.

Пример такого класса из жизни — животное. Рыбы, пауки и насекомые являются животными, но каждое из них также принадлежит своему подклассу с набором специфических свойств. При этом, просто «животных» в природе не существует.

# Вывод:

В ходе выполнения данного индивидуального домашнего задания я закрепил навыки использования механизмов ООП на примере реализации графического редактора.