МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«ЧЕРЕПОВЕЦКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Институт (факультет) Институт информационных технологий
Кафедра МПО ЭВМ

КУРСОВАЯ РАБОТА

по дисциплине Структурное программирование

на тему Программирование на языке высокого уровня

Выполнил студент группы 1ПИб-02-2оп-22
группа

Направления подготовки (специальности)
09.03.04 Программная инженерия
шифр, наименование

Зернов Владислав Александрович
фамилия, имя, отчество

Руководитель
Пышницкий Константин Михайлович
фамилия, имя, отчество

Старший преподаватель
должность

Дата представления работы
«___»_____2024 г.

Заключение о допуске к защите

Оценка _____, _____
количество баллов

Подпись преподавателя _____

2024 год

Оглавление

Введение

Программирование на языке высокого уровня, является одной из ключевых областей современной информатики, позволяющей разработчикам создавать сложные и функциональные программы с помощью абстрактных и удобных для восприятия инструментов. В отличие от низкоуровневых языков программирования, он может использовать элементы естественного языка, быть более простым в использовании или может автоматизировать значительные области вычислительных систем таких как, управление памятью, делая процесс разработки программы проще и понятнее, чем при использовании языка более низкого уровня [1].

Язык C++ является одним из самых популярных высокоуровневых языков программирования. Он является развитием языка C и добавляет возможности объектно-ориентированного программирования, что делает его мощным инструментом для создания сложных программных систем [2].

Задача данной курсовой работы заключается в разработке программы, способной генерировать графическое изображение орнамента из шестиугольников и обеспечивать возможность изменения его размера и цвета, а также реализовывать сохранение получившегося графического изображения.

Разработанная программа, способная создавать и изменять орнамент, позволит пользователям создавать уникальные и привлекательные графические элементы, придавая им индивидуальный стиль.

1.Описание предметной области

Орнамент — рисунок, состоящий из повторяющихся или чередующихся элементов. Это понятие имеет латинские корни и дословно означает «украшение». Орнаменты используются в архитектуре, графическом дизайне, ювелирном деле и других областях искусства и промышленности.

Орнаменты делятся на несколько видов:

- Технический орнамент — воспроизведение на керамике либо декора и фактуры не керамических изделий ;
- Символические орнаменты — рисунки, которые человек наблюдал в живой природе и упрощал до символов;
- Геометрический орнамент — чередование декоративных элементов из простых фигур и цветовых сочетаний;
- Растительные и животные орнаменты — чередование растительности или животных .

Разработка программы для создания орнаментов в Windows Forms позволит пользователям создавать уникальный орнамент, настраивать его параметры, масштабировать и менять цвета. Вместо ручного рисования орнаментов или использования сложных графических инструментов, программа позволит пользователю выбирать и настраивать орнаменты с помощью удобного интерфейса. Таким образом, разработка программы значительно упрощает и ускоряет процесс создания орнаментов и повышает эффективность работы с графическими элементами в Windows Forms.

Предметной областью является орнамент множества пересекающихся шестиугольников (рис.1)..

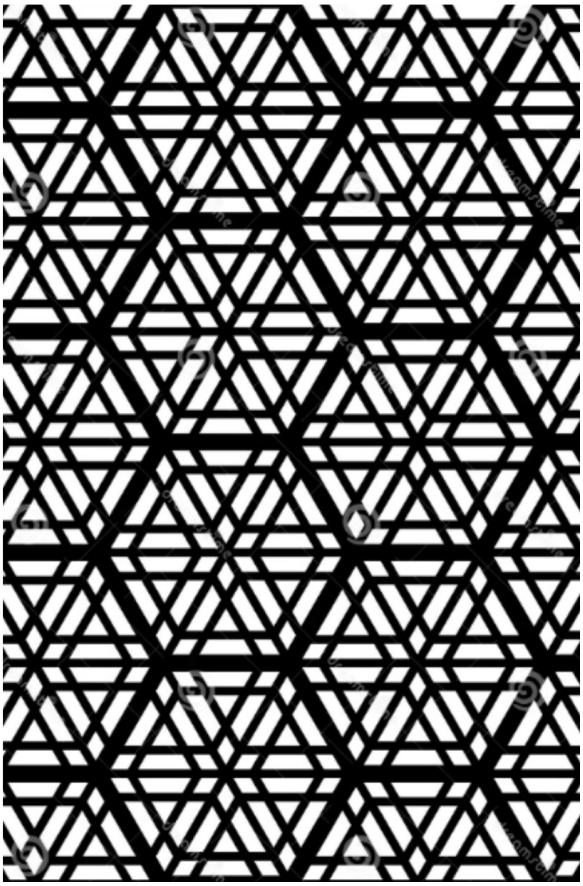

Рис.1. Орнамент множества пересекающихся шестиугольников

2.Описание классов Graphics, Pen и Brush

Рисование фигур является одним из важных аспектов в разработке графических приложений. В Windows Forms это может быть реализовано с помощью классов Graphics, Pen и Brush в языке программирования C++. Пространство имен System.Drawing (Рисование) обеспечивает доступ к функциональным возможностям графического интерфейса GDI+ , используя около 50 классов, в том числе класс Graphics, Pen и Brush.

2.1. Класс Graphics

Класс Graphics предоставляет методы рисования на устройстве отображения. Данный класс входит в пространство имен System.Drawing, как и большинство классов для работы с графикой. Класс Graphics определяет набор методов для вывода текста, изображений и геометрических фигур [4].

Методов в этом классе огромное количество, поэтому рассмотрим некоторую часть из них в табл. 1:

Таблица 1

Методы класса Graphics

Имя	Описание
AddMetafileComment	Добавляет комментарий к текущему объекту Metafile.
BeginContainer()	Сохраняет графический контейнер, содержащий текущее состояние данного объекта Graphics, а затем открывает и использует новый графический контейнер.
Clear	Очищает всю поверхность рисования и выполняет заливку поверхности указанным цветом фона.
CopyFromScreen(Int32, Int32, Int32, Int32, Size)	Выполняет передачу данных о цвете, соответствующих прямоугольной области пикселей, блоками битов с экрана на поверхность рисования объекта Graphics.
Dispose	Освобождает все ресурсы, используемые данным объектом Graphics.
DrawArc(Pen, Int32, Int32, Int32, Int32, Int32, Int32)	Рисует дугу, которая является частью эллипса, заданного парой координат, шириной и высотой.
DrawBezier(Pen, Point, Point, Point, Point)	Рисует кривую Безье, определяемую четырьмя структурами Point.
DrawLine(Pen, Int32, Int32, Int32, Int32)	Проводит линию, соединяющую две точки, задаваемые парами координат.
FillClosedCurve(Brush, Point[], FillMode)	Заполняет внутреннюю часть замкнутой фундаментальной сплайновой кривой, определяемой массивом структур Point, используя указанный режим заливки.
FillEllipse(Brush, Int32, Int32, Int32, Int32)	Заполняет внутреннюю часть эллипса, определяемого ограничивающим прямоугольником, заданным с помощью пары координат, ширины и высоты.
FillPath	Заполняет внутреннюю часть объекта GraphicsPath.
FillPie(Brush, Rectangle, Single, Single)	Заполняет внутреннюю часть сектора, определяемого эллипсом, который задан структурой RectangleF, и двумя радиальными линиями.
FillPie(Brush, Int32, Int32, Int32, Int32, Int32, Int32)	Заполняет внутреннюю часть сектора, определяемого эллипсом, который задан парой координат, шириной, высотой и двумя радиальными линиями.
FillPolygon(Brush, Point[])	Заполняет внутреннюю часть многоугольника, определяемого массивом точек, заданных структурами Point.
FillRectangle(Brush, Rectangle)	Заполняет внутреннюю часть прямоугольника, определяемого структурой Rectangle.

2.2. Класс Pen

Класс Pen определяет объект, используемый для рисования прямых линий и кривых [4].

Методы представлены в табл. 2:

Таблица 2

Методы класса Pen

Имя	Описание
Pen(Color)	Инициализирует новый экземпляр класса Pen с указанным цветом.
Pen(Color, Single)	Инициализирует новый экземпляр класса Pen с указанными свойствами Color и Width. (Width - устанавливает ширину пера Pen, в единицах объекта Graphics, используемого для рисования)

2.3. Класс Brush

Класс Brush определяет объекты, которые используются для заливки внутри графических фигур, таких как прямоугольники, эллипсы, круги, многоугольники и дорожки.

Это абстрактный базовый класс, который не может быть реализован. Для создания объекта "кисть" используются классы, производные от Brush, такие как SolidBrush, TextureBrush и LinearGradientBrush [4].

3.Описание созданного приложения

В данном разделе описаны основные этапы разработки приложения для построения орнамента пересекающихся шестиугольников.

3.1. Постановка задачи

Для корректной работы программы требуется выполнить следующие задачи:

- Программа должна создавать графическое изображение орнамента при нажатии соответствующей кнопки;
- программа должна позволять выбирать цвет заднего фона;
- программа должна позволять менять масштаб и цвет орнамента;
- программа должна позволять сохранять в файловом виде изображение

при нажатии соответствующей кнопки;

- программа должна иметь возможность повторно создать графическое изображение при изменении масштаба и цвета.

3.2. Логическое проектирование

Для построения орнамента (рис.1) необходимо определиться с алгоритмом.

Принимаем значение размера элемента орнамента, то есть шестиугольника, и цвет, которым будет реализовано рисование, эти параметры предусмотрены для того, чтобы можно было реализовать изменение масштаба и цвета рисунка. Затем определяются значения ширины и высоты картинки исходя из размеров окна приложения, значение количества шестиугольников путём отношения ширины окна, на котором происходит рисование, к половине размера шестиугольника, это необходимо для того, чтобы шестиугольники находились вплотную друг к другу. Задаются начальные координаты по оси x и по оси y, равные нулю. Позже вычисляется количество рядов шестиугольников по оси y отношением высоты окна, на котором будет производится реализация рисунка, к половине размера шестиугольника. Эти шестиугольники будут рисоваться сначала по оси x, затем дублироваться по оси y. Для начала необходимо создать цикл, который будет выполняться пока переменная, отвечающая за счётчик цикла, меньше количества рядов по оси y, которое было вычислено ранее. В теле цикла задаются координаты для ряда по оси y, путём суммы начальных координат и разности половины размера шестиугольника с одной четырнадцатой размера шестиугольника, координаты вычисляются таким образом, для того чтобы не было отступов между рядами шестиугольников. Для правильного пересечения рядов шестиугольников нам необходимо условие, которое будет изменять координаты ряда по оси x, прибавляя сумму половины размера шестиугольника и четверти его размера. Тем самым координаты начала каждого второго ряда будут находится так, что верхняя вершина правильного шестиугольника окажется в центре предыдущего

шестиугольника. Далее реализуем рисование ряда больших шестиугольников, в которых будут находится маленькие. Но в конце рисования необходимо вернуть координаты ряда по оси x, если этот ряд чётный.

В самом рисовании зависим от общего количества шестиугольников в ряду, размер шестиугольника, координаты ряда по оси x и y, поверхность для рисования, и объект для рисования, и цвет, которым будут рисоваться шестиугольники. Для реализации рисования ряда шестиугольников по оси x, необходим цикл, который будет выполняться пока счётчик цикла меньше общего количества шестиугольников в ряду. В цикле вычисляется координата x, для текущего шестиугольника через сумму координаты ряда и произведения половины размера и переменной счётчика, тем самым каждый шестиугольник будет вплотную находится с предыдущим. Для построения правильного шестиугольника нам необходимо создать массив шести точек. Заполнение этого массива происходит по формулам: angle = Math::PI / 3 * j; pointX = x + (int)(size / 2 * Math::Cos(angle)); pointY = rowY + (int)(size / 2 * Math::Sin(angle)). Где angle – угол, то есть точка на числовой окружности, pointX, pointY – значения координат точки вершины по оси x и y соответственно, size – размер шестиугольника , x – координата x центра текущего шестиугольника , rowY– координата y, текущего ряда шестиугольников. Обратив внимание на исходный рисунок, можно заметить, что каждый третий большой шестиугольник имеет большую толщину, чем другие. Для реализации этого, необходимо условие которое будет рисовать каждый третий шестиугольник с толщиной 3 по массиву точек вычисленному ранее, иначе рисуется обычный шестиугольник по массиву точек. После рисования больших шестиугольников переходим к маленьким.

Рисование маленьких шестиугольников, как и большие зависят от значения размера шестиугольника, координаты ряда по оси y, поверхность для рисования, и объект, используемый для рисования. Рассчитываем размер малого шестиугольника, он будет на 20% меньше большого. Далее, как и в рисовании большого шестиугольника, вычисляется массив точек по тем же

формулам, однако вместо переменной размера шестиугольника, применяется вычисленная ранее переменная, означающая размер малого шестиугольника. Тем самым, координаты центров двух этих шестиугольников совпадают, и малый находится в центре большого.

Алгоритм работы программы представлен на блок-схеме (рис. 2):

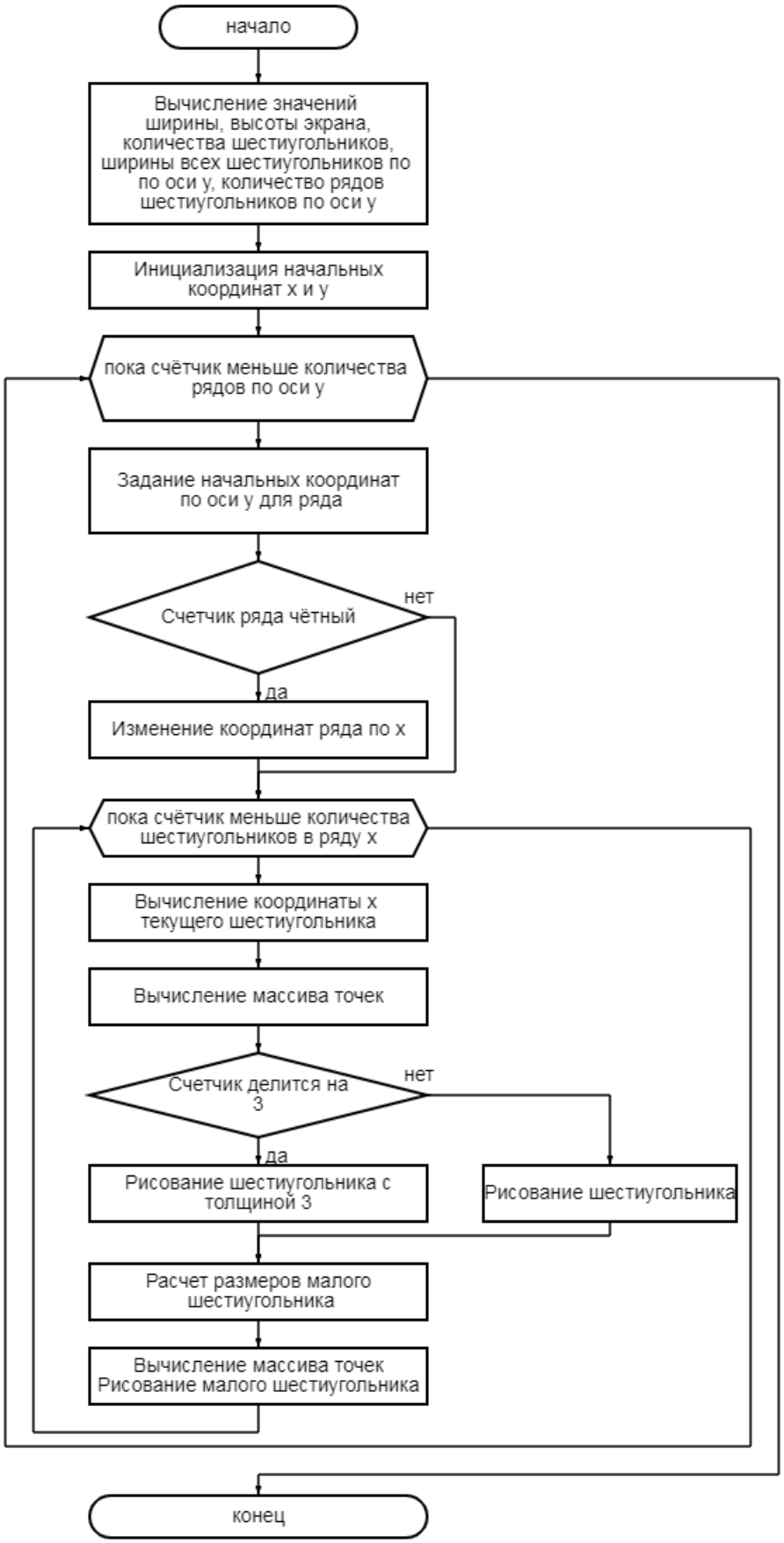

Рис. 2. Алгоритм работы программы в блок-схеме

3.3. Физическое проектирование

Переменные, примененные для реализации программы, представлены в табл. 3:

Таблица 3

Переменные

Наименование	Обозначение	Тип данных
Значение размера шестиугольника	size	int
Ширина блока рисования	width	int
Высота блока рисования	height	int
Количество шестиугольников по оси x	totalHexagons	int
Начальная координата x	startX	int
Начальная координата y	startY	int
Количество шестиугольников по оси y	numHexagonY	int
Координата ряда шестиугольников по оси x	rowX	int
Координата ряда шестиугольников по оси y	rowY	int
Значение угла	angle	double
Координаты вершины шестиугольника по оси x	pointX	int
Координаты вершины шестиугольника по оси y	pointY	int
Размер малого шестиугольника	smallSize	int
Координата x центра малого шестиугольника	smallX	int
Координата y центра малого шестиугольника	smallY	int
Координата x центра большого шестиугольника	x	Int
Цвет линий орнамента	color	Color

Спецификация функций представлена в табл. 4:

Таблица 4

Спецификация функций

Имя модуля	Заголовок процедуры или функции	Формальные параметры	Выполняемое действие
cursStruct.sln	startProcess_Click_1	System::Object^ sender, System::EventArgs^ e	Функция выполняет вызов функции построения орнамента.
	SaveButton_Click	System::Object^ sender, System::EventArgs^ e	Функция сохраняет полученное изображение.
	numericUpDown1_ValueChanged	System::Object^ sender, System::EventArgs^ e	Функция задаёт масштаб шестиугольника и

			обновляет поле для рисования.
	RandomColorButton_Click	System::Object^ sender, System::EventArgs^ e	Функция задаёт орнаменту рандомный цвет и строит орнамент с новым цветом.
	colorButton_Click	System::Object^ sender, System::EventArgs^ e	Функция вызывает диалоговое окно для выбора пользователю цвета орнамента и вызывает функцию построения орнамента.
	backgroundColorButton_Click	System::Object^ sender, System::EventArgs^ e	Функция меняет цвет заднего фона.
	SmallHexagonDraw	int size, int x, int rowY, Graphics^ graghics, Pen^ pen	Функция рисует малый шестиугольник внутри большого
	BigHexagonDraw	int totalHexagons,int size, int rowX, int rowY, Graphics^ graghics, Pen^ pen, Color color	Функция рисует ряд больших шестиугольников и вызывает функцию для рисования маленьких шестиугольников.
	OrnamentDraw	int size, Color color	Функция рисует полноценный орнамент.

4.4. Тестирование

Было проведено тестирование для проверки работоспособности программы и выявления возможных ошибок. В результате было подтверждено, что программа функционирует корректно, и никаких проблем не было обнаружено. Тестовые данные представлены в табл. 5.

Таблица 5

Исходные данные	Тестируемый модуль	Ожидаемые результаты
Проверка работы кнопки построения орнамента	cursStruct.sln	Корректная работа кнопки и орнамент построится
Проверка работы сохранения	cursStruct.sln	Корректное сохранение

изображения орнамента		изображение
Изменение цвета	cursStruct.sln	Корректное изменение цвета
Масштабирование	cursStruct.sln	Корректное работа масштабирования
Изменение фона	cursStruct.sln	Корректное изменение фона
Проверка всех функций работы программы	cursStruct.sln	Корректное работа всего функционала

Результаты тестирования представлены в табл. 6:

Таблица 6

Результаты тестирования

Дата тестирования	Тестируемый модуль	Кто проводил тестирование	Описание теста	Результат тестирования
20.05.2023	cursStruct.sln	Зернов В.А.	Проверка работы кнопки построения орнамента	Успех
21.05.2023	cursStruct.sln	Зернов В.А.	Проверка работы сохранения изображения орнамента	Успех
22.05.2023	cursStruct.sln	Зернов В.А.	Изменение цвета	Успех
22.05.2023	cursStruct.sln	Зернов В.А.	Масштабирование	Успех
25.05.2023	cursStruct.sln	Зернов В.А.	Изменение фона	Успех
30.05.2023	cursStruct.sln	Зернов В.А.	Проверка всех функций работы программы	Успех

4.5. Результаты работы

В результате работы было создано приложение для построения орнамента множества пересекающихся шестиугольников в Windows Forms на языке C++. Орнамент строится по изначальным значениям, присутствует возможность изменения масштаба, задания цвета орнамента и цвета заднего фона, построения орнамента случайного цвета, а также возможность сохранения изображения.

Интерфейс понятен и функционален (рис. 3).

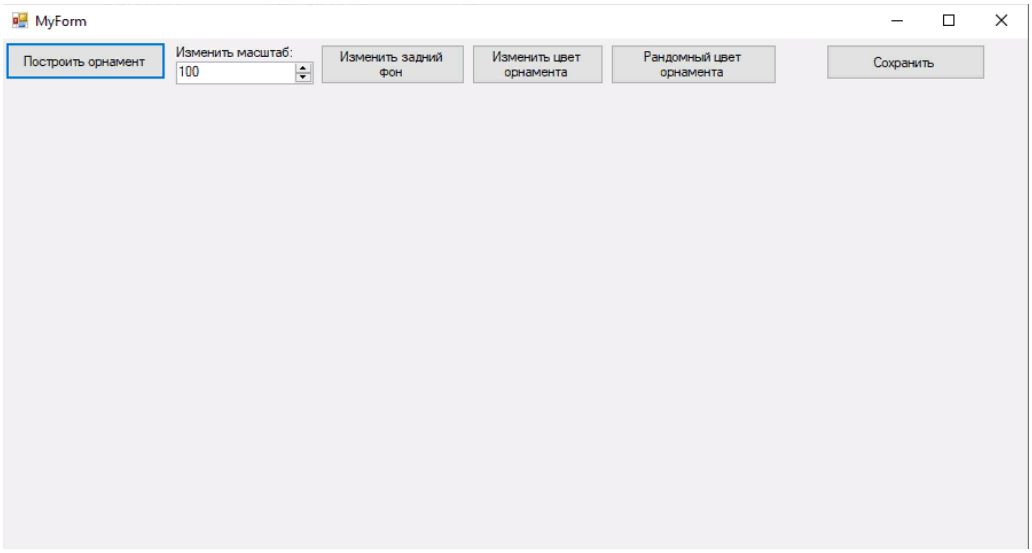

Рис. 3. Интерфейс программы

Заключение

В рамках данного проекта была разработана программа в Windows Forms на языке C++, реализующая графическое изображение орнамента множества пересекающихся шестиугольников.

В ходе работы, были получены навыки в работе с Windows Forms, освоены возможности классов Graphics, Pen и Brush.

Поставленные задачи были выполнены в полном объеме.

Источники

1. Высокоуровневые языки программирования [электронный ресурс]: https://en.wikipedia.org/wiki/High-level_programming_language. Дата доступа: 19.03.2024

2. Язык программирования C++ [электронный ресурс]: https://metanit.com/cpp/tutorial/1.1.php Дата доступа: 19.03.2024

3. Windows Forms [электронный ресурс]: https://learn.microsoft.com/ru-ru/dotnet/desktop/winforms/overview/?view=netdesktop-6.0 Дата доступа: 19.03.2024

4. Основы библиотеки System.Drawing [электронный ресурс]: https://c-sharp.pro/общие-замечания-классы-graphics-pen-и-brush/. Дата доступа: 19.03.2024

5. Орнамент [электронный ресурс]: https://media.contented.ru/glossary/ornament/ Дата доступа: 19.03.2024

Приложение 1. Техническое задание

МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«ЧЕРЕПОВЕЦКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Институт информационных технологий
наименование института (факультета)

Кафедра математического и программного обеспечения ЭВМ
наименование кафедры

Структурное программирование
наименование дисциплины в соответствии с учебным планом

УТВЕРЖДАЮ
Зав. кафедрой_____
д.т.н., профессор___Ершов Е.В.
«___»_____2024г.

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ ВЫСОКОГО УРОВНЯ

Техническое задание на курсовую работу
Листов 5

Руководитель: старший преподаватель
Пышницкий К.М.
Исполнитель: студент гр. 1ПИб-02-2оп-22
Зернов В.А.

2024 год

Введение

Программа для построения орнамента пересекающихся шестиугольников в Windows Forms рисует изображение и сохраняет полученный результат.

Данный проект предназначен для формирования знаний и навыков по дисциплине «Структурное программирование».

1. Основания для разработки

Основанием для разработки является задание на курсовую работу по дисциплине «Структурное программирование», выданное на кафедре МПО ЭВМ ИИТ ЧГУ.

Наименование темы разработки: программирование на языке высокого уровня.

2. Назначение разработки

Назначение разработки этой программы состоит в создании приложения для генерации и визуализации орнамента пересекающихся шестиугольников. Программа предоставляет пользователю возможность изменения размеров элементов орнамента через графический интерфейс, также возможность выбора цвета заднего фона и цвета линий орнамента. Кроме того, в программе присутствует возможность построения орнамента с случайным цветом. Также программа позволяет сохранить полученный орнамент в файл для дальнейшего использования в форматах JPEG,PNG,BMP. Программа может использоваться для создания уникального орнамента и использования его изображения в дизайне.

3. Требования к программе

3.1. Требования к функциональным характеристикам

Требования к функциональным характеристикам программы:

- Генерация орнамента: Программа должна позволять пользователю генерировать орнамент множества шестиугольников. Сгенерированный орнамент должен быть наглядно отображен на экране, чтобы пользователь мог визуально оценить результат.
- Изменение параметров орнамента: Пользователь должен иметь возможность изменять параметры орнамента, такие как радиус и количество окружностей, цвет и масштаб, и видеть соответствующие изменения визуализации.
- Сохранение орнамента: Пользователь должен иметь возможность сохранить сгенерированный орнамент в файле, чтобы иметь возможность использовать его в дальнейшем.
- Понятный интерфейс: Интерфейс программы должен быть интуитивно понятным и удобным в использовании, чтобы пользователь мог легко настраивать параметры орнамента и видеть результаты своих действий.

3.2. Требования к надежности

Программа должна быть стабильной и не вызывать непредвиденных ошибок. Она должна обрабатывать любые возможные ситуации и ошибки. Программа должна корректно рисовать пересекающиеся шестиугольники при любых допустимых параметрах.

3.3. Условия эксплуатации

Программа должна быть совместима с операционной системой пользователя. Условия эксплуатации программы зависят от условий эксплуатации персонального компьютера.

3.4. Требования к составу и параметрам технических средств

Минимальные требования к техническим характеристикам компьютера для работы программы:

- процессор с тактовой частотой 1,2 ГГц;
- оперативная память объемом не менее 1 ГБ;
- жесткий диск с объемом не менее 2 ГБ;
- монитор;
- мышь;
- клавиатура.

3.5. Требования к информационной и программной совместимости

Для обеспечения программной совместимости необходим Visual Studio 2019 ли новее на ОС Windows 7 или выше.

3.6. Требования к маркировке и упаковке

Требования к маркировке и упаковке программы, как правило, не являются применимыми, так как программа представляет собой цифровой продукт, который распространяется в электронном формате.

3.7. Требования к транспортированию и хранению

Файлы, требуемые для корректной работы программы, необходимо расположить на флеш-носителе либо памяти компьютера. Рекомендуется сохранить программу на внешний носитель, чтобы избежать потери информации.

3.8. Специальные требования

Программное обеспечение должно иметь дружественный интерфейс, рассчитанный на пользователя средней квалификации. Ограничение возрастного порога с 6 лет.

4. Требования к программной документации

4.1. Содержание расчетно-пояснительной записки

Программная документация должна содержать расчётно-пояснительную записку с содержанием:

Титульный лист

Оглавление

Введение

Описание предметной области

Описание классов Graphics, Pen и Brush

Описание созданного приложения

1. Постановка задачи
2. Логическое проектирование
3. Физическое проектирование
4. Тестирование
5. Результаты работы

Заключение

Источники

Приложения

4.2. Требования к оформлению

Программная документация должна удовлетворять следующему оформлению (табл. П1.1):

Таблица П1.1

Требования к оформлению

Документ	Печать на отдельных листах формата А4 (210х297 мм); оборотная сторона не заполняется; листы нумеруются. Печать возможна ч/б. Файлы предъявляются на компакт-диске: РПЗ с ТЗ; программный код. Листы и диск в конверте вложены в пластиковую папку скоросшивателя.
Страницы	Ориентация – книжная; отдельные страницы, при необходимости, альбомная. Поля: верхнее, нижнее – по 2 см, левое – 3 см , правое – 1 см.
Абзацы	Межстрочный интервал – 1.5, перед и после абзаца – 0.

Шрифты	Кегль – 14. В таблицах шрифт 12. Шрифт листинга – 10 (возможно в 2 колонки).
Рисунки	Подписывается под ним по центру: Рис.Х. Название В приложениях: Рис.П1.3. Название
Таблицы	Подписывается: над таблицей, выравнивание по правому: «Таблица Х». В следующей строке по центру Название Надписи в «шапке» (имена столбцов, полей) – по центру. В теле таблицы (записи) текстовые значения – выровнены по левому краю, числа, даты – по правому.

5. Технико-экономические показатели

Технико-экономические показатели к данной программе не предъявляются.

6. Стадии и этапы разработки

Стадии и этапы разработки представлены в табл. П1.2:

Таблица П1.2

Стадии и этапы разработки

Наименование этапа разработки	Сроки разработки	Результат выполнения	Отметка о выполнении
Разработка ТЗ	до 15.02.2024	Разработанное ТЗ	
Разработка программы	25.02.2024-20.03.2024	Разработанная программа	
Разработка руководства пользователя	15.03.2024-22.03.2024	Разработанное руководство пользователя	
Разработка РПЗ	15.05.2024-22.03.2024	Разработанная РПЗ	

7. Порядок контроля и приемки

Порядок контроля и приемки представлен в табл. П1.3:

Таблица П1.3

Порядок контроля и приемки

Наименование контрольного этапа	Сроки контроля	Результат выполнения	Отметка о приемке результата

выполнения курсовой работы			контрольного этапа
Оформление ТЗ	15.02.2024	Оформленное ТЗ	
Разработка программы	20.03.2024	Неконечная версия программы	
Оформление руководство пользователя	21.03.2024	Оформленное руководство пользователя	
Доработка программы	21.03.2024	Конечная версия программы	
Оформление РПЗ	21.03.2024	Оформленная РПЗ	
Сдача РПЗ	22.03.2024	Оценка за курсовую работу	

Приложение 2. Руководство пользователя

1. Общие сведения о программе

Программа предназначена для графического изображения орнамента множества пересекающихся шестиугольников (рис. П2.1).

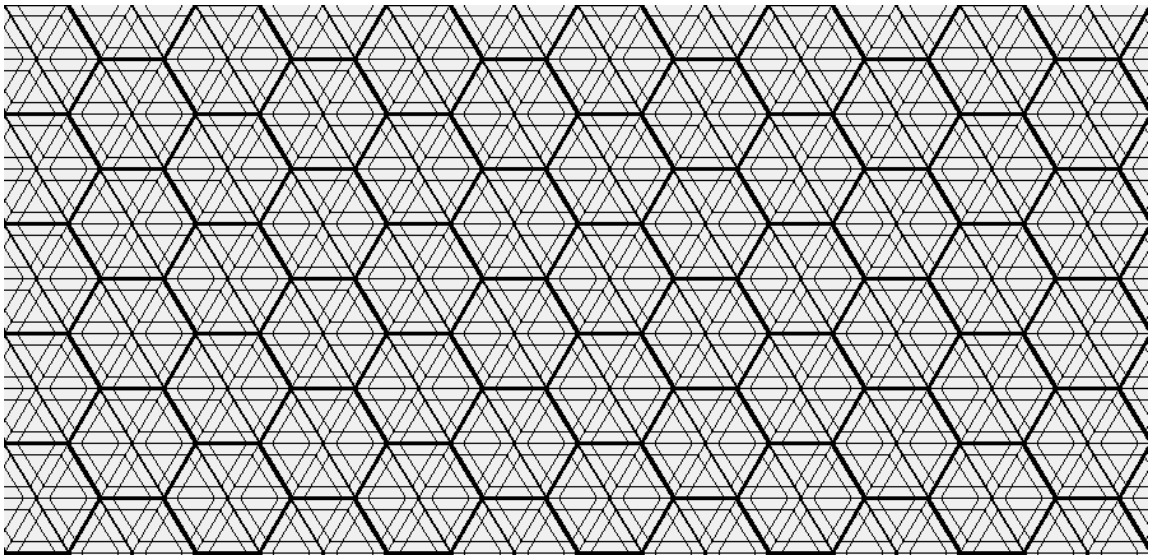

Рис. П2.1. Орнамент множества пересекающихся шестиугольников

Программа создает орнамент по параметрам: масштаб и цвет изображения.

2. Описание установки

Установка программы не требуется. Чтобы запустить программу, нужно иметь доступ к файлу программы и Visual Studio 2019 или выше.

3. Описание запуска

1. Открыть папку, в которой находится файл программы.
2. Открыть программу.
3. Появляется окно программы, в котором можно работать (рис. П2.2).

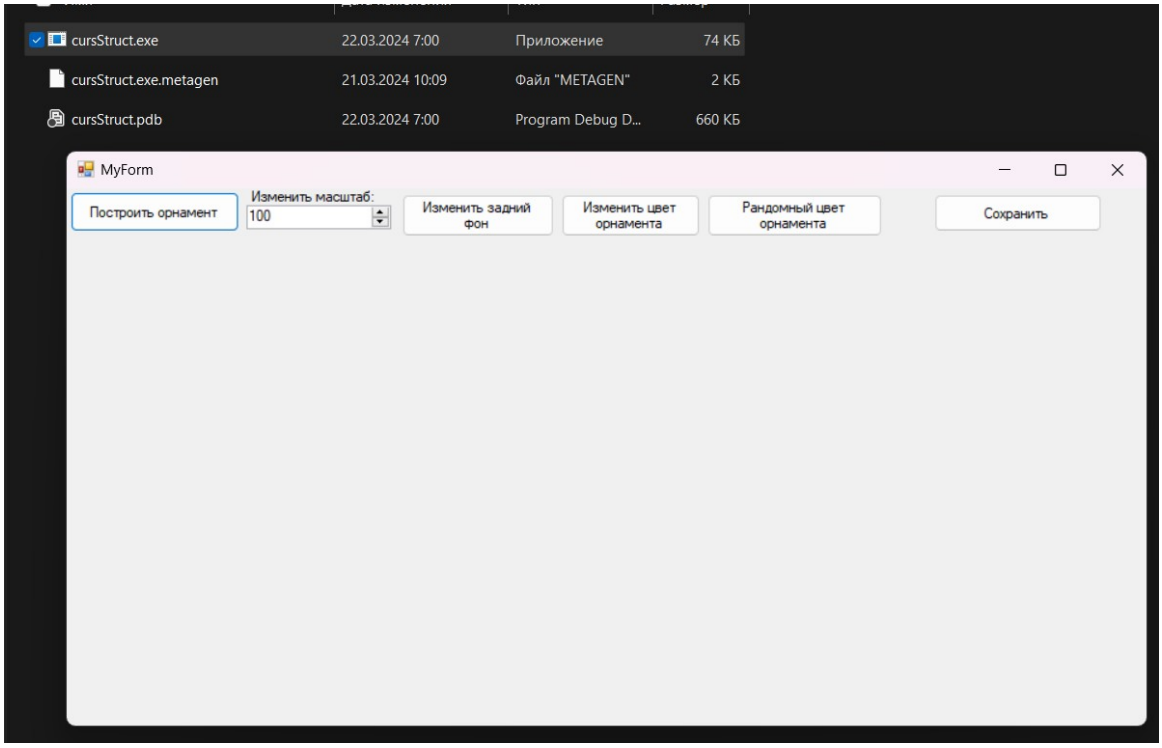

Рис. П2.2. Окно программы

4. Инструкция по работе

Параметры построения орнамента окружности по окружности задаются с помощью элементов интерфейса управления (рис. П2.3).

Рис. П2.3. Элементы управления

1. Построение изначального орнамента, цвет по стандарту чёрный, размер зависит от значения поля размера.

2. Выбор масштаба. Для выбора масштаба, можно ввести в соответствующее окно значение размера элемента орнамента(от 50 до 300) либо нажимать на треугольник и изменять значение на 1.

3. Выбор заднего фона. В этом случае всё поле для рисования заполнится одним цветом.

4. Выбор цвета орнамента. При нажатии на эту кнопку появляется диалоговое окно и при выборе необходимого цвета построится орнамет с соответствующим цветом.

5. Построение орнамента с случайным цветом. Для построения орнамента с случайным цветом нужно нажать на кнопку «Рандомный цвет».

6. Сохранение изображения. Для сохранения изображения нужно нажать на кнопу «Сохранить».Появится диалоговое окно, где пользователь может выбрать путь и формат сохранения файла.

После изменения масштаба и цвета программа построит орнамент (рис. П2.4) и сохранит его изображение (рис. П2.5).

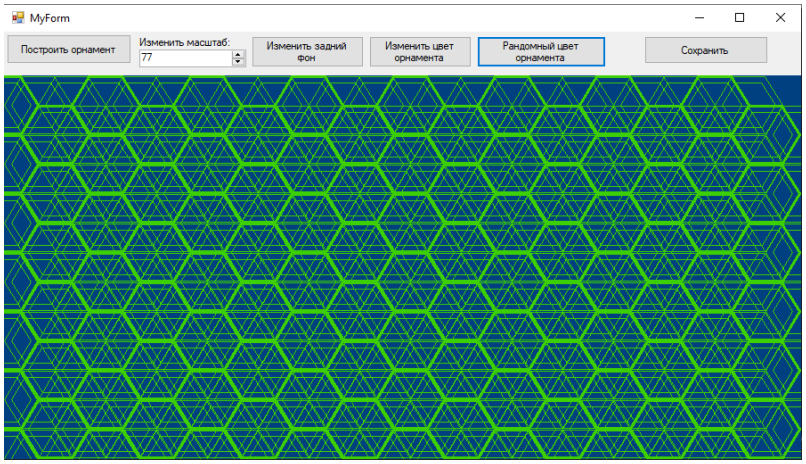

Рис. П2.4. Построенный орнамент

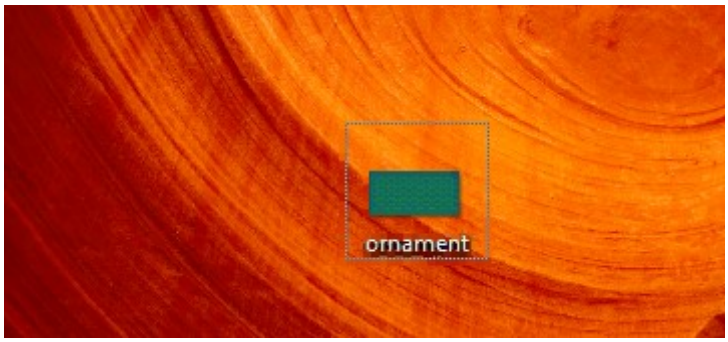

Рис. П2.5. Сохраненное изображение

5. Сообщения пользователю

При нажатии на кнопку изменить цвет орнамента появится диалоговое окно, представленное на рис. П2.6:

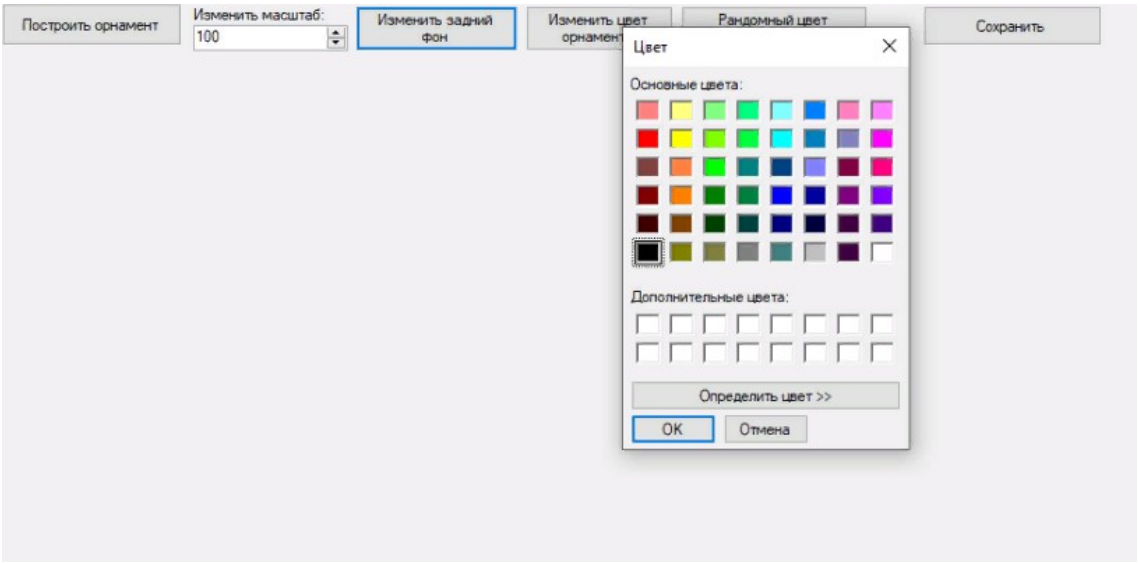

Рис. П2.6. Диалоговое окно выбора цвета

При нажатии на кнопку сохранить появится диалоговое окно, представленное на рис. П2.7:

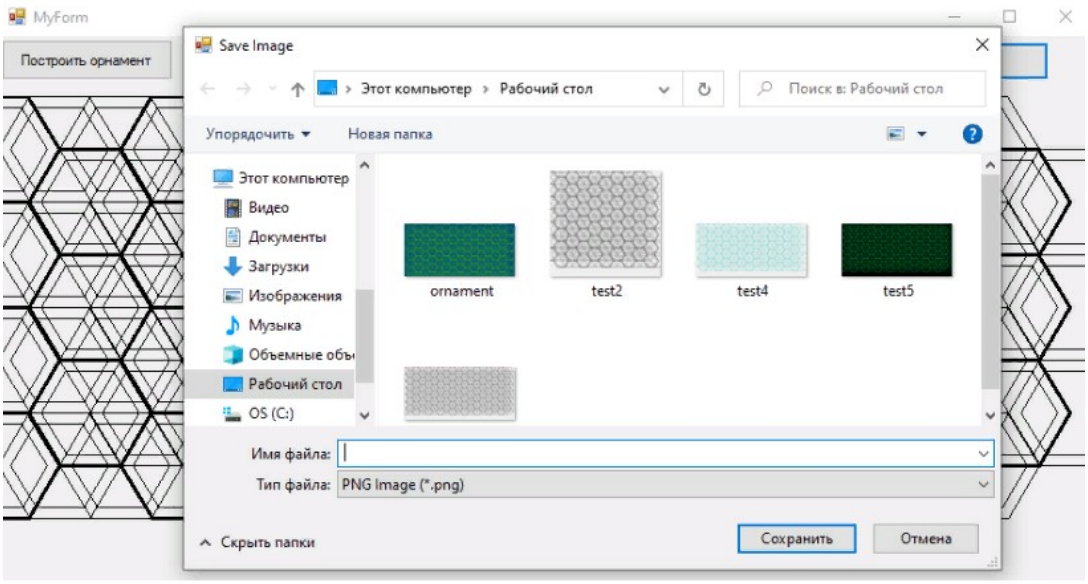

Рис. П2.7. Диалоговое окно сохранения изображения

Приложение 3. Программный код

Код MyForm.cpp:

```
#include "MyForm.h"

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

[STAThread]
int main(array<String^>^ args) {
   Application::EnableVisualStyles();
   Application::SetCompatibleTextRenderingDefault(false);
   cursStruct::MyForm form;
   Application::Run(% form);
   return 0;
}
```

Код MyForm.h:

```
#pragma once
#include <ctime>
#include <cstdlib>
namespace cursStruct {

   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 MyForm
   /// </summary>
   public ref class MyForm : public System::Windows::Forms::Form
   {
   public:
      MyForm(void)
      {
         InitializeComponent();
         //
         //TODO: добавьте код конструктора
         //
      }

   protected:
      /// <summary>
      /// Освободить все используемые ресурсы.
      /// </summary>
      ~MyForm()
      {
         if (components)
         {
            delete components;
         }
      }
   private: System::Windows::Forms::PictureBox^ pictureBox1;
   protected:
   private: System::Windows::Forms::Panel^ panel1;
   private: System::Windows::Forms::Button^ startProcess;
```

```
        private: System::Windows::Forms::Button^ SaveButton;
        private: System::Windows::Forms::SaveFileDialog^ saveFileDialog1;
        private: System::Windows::Forms::Button^ RandomColorButton;
        private: System::Windows::Forms::Label^ label1;
        private: System::Windows::Forms::NumericUpDown^ numericUpDown1;
        private: System::Windows::Forms::Button^ colorButton;
        private: System::Windows::Forms::ColorDialog^ colorDialog1;
        private: System::Windows::Forms::Button^ backgroundColorButton;

        private:
            /// <summary>
            /// Обязательная переменная конструктора.
            /// </summary>
            System::ComponentModel::Container^ components;

    #pragma region Windows Form Designer generated code
            /// <summary>
            /// Требуемый метод для поддержки конструктора — не изменяйте
            /// содержимое этого метода с помощью редактора кода.
            /// </summary>
            void InitializeComponent(void)
            {
                this->pictureBox1 = (gcnew System::Windows::Forms::PictureBox());
                this->panel1 = (gcnew System::Windows::Forms::Panel());
                this->backgroundColorButton = (gcnew System::Windows::Forms::Button());
                this->colorButton = (gcnew System::Windows::Forms::Button());
                this->label1 = (gcnew System::Windows::Forms::Label());
                this->numericUpDown1 = (gcnew System::Windows::Forms::NumericUpDown());
                this->RandomColorButton = (gcnew System::Windows::Forms::Button());
                this->SaveButton = (gcnew System::Windows::Forms::Button());
                this->startProcess = (gcnew System::Windows::Forms::Button());
                this->saveFileDialog1 = (gcnew System::Windows::Forms::SaveFileDialog());
                this->colorDialog1 = (gcnew System::Windows::Forms::ColorDialog());
                (cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this->pictureBox1))->BeginInit();
                this->panel1->SuspendLayout();
                (cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this->numericUpDown1))-
>BeginInit();
                this->SuspendLayout();
                //
                // pictureBox1
                //
                this->pictureBox1->Anchor =
static_cast<System::Windows::Forms::AnchorStyles>((((System::Windows::Forms::AnchorStyles::Top |
System::Windows::Forms::AnchorStyles::Bottom)
                    | System::Windows::Forms::AnchorStyles::Left)
                    | System::Windows::Forms::AnchorStyles::Right));
                this->pictureBox1->Location = System::Drawing::Point(0, 60);
                this->pictureBox1->Margin = System::Windows::Forms::Padding(4);
                this->pictureBox1->Name = L"pictureBox1";
                this->pictureBox1->Size = System::Drawing::Size(1193, 489);
                this->pictureBox1->TabIndex = 0;
                this->pictureBox1->TabStop = false;
                //
                // panel1
                //
                this->panel1->Anchor = System::Windows::Forms::AnchorStyles::Top;
                this->panel1->Controls->Add(this->backgroundColorButton);
                this->panel1->Controls->Add(this->colorButton);
                this->panel1->Controls->Add(this->label1);
                this->panel1->Controls->Add(this->numericUpDown1);
                this->panel1->Controls->Add(this->RandomColorButton);
                this->panel1->Controls->Add(this->SaveButton);
```

```
            this->panel1->Controls->Add(this->startProcess);
            this->panel1->Location = System::Drawing::Point(0, 0);
            this->panel1->Margin = System::Windows::Forms::Padding(4);
            this->panel1->Name = L"panel1";
            this->panel1->Size = System::Drawing::Size(1193, 53);
            this->panel1->TabIndex = 1;
            //
            // backgroundColorButton
            //
            this->backgroundColorButton->Location = System::Drawing::Point(371, 6);
            this->backgroundColorButton->Margin = System::Windows::Forms::Padding(4);
            this->backgroundColorButton->Name = L"backgroundColorButton";
            this->backgroundColorButton->Size = System::Drawing::Size(168, 43);
            this->backgroundColorButton->TabIndex = 6;
            this->backgroundColorButton->Text = L"Изменить задний фон";
            this->backgroundColorButton->UseVisualStyleBackColor = true;
            this->backgroundColorButton->Click += gcnew System::EventHandler(this,
&MyForm::backgroundColorButton_Click);
            //
            // colorButton
            //
            this->colorButton->Location = System::Drawing::Point(547, 6);
            this->colorButton->Margin = System::Windows::Forms::Padding(4);
            this->colorButton->Name = L"colorButton";
            this->colorButton->Size = System::Drawing::Size(153, 43);
            this->colorButton->TabIndex = 5;
            this->colorButton->Text = L"Изменить цвет орнамента";
            this->colorButton->UseVisualStyleBackColor = true;
            this->colorButton->Click += gcnew System::EventHandler(this, &MyForm::colorButton_Click);
            //
            // label1
            //
            this->label1->AutoSize = true;
            this->label1->Location = System::Drawing::Point(200, 0);
            this->label1->Margin = System::Windows::Forms::Padding(4, 0, 4, 0);
            this->label1->Name = L"label1";
            this->label1->Size = System::Drawing::Size(134, 16);
            this->label1->TabIndex = 4;
            this->label1->Text = L"Изменить масштаб:";
            //
            // numericUpDown1
            //
            this->numericUpDown1->Location = System::Drawing::Point(199, 17);
            this->numericUpDown1->Margin = System::Windows::Forms::Padding(4);
            this->numericUpDown1->Maximum = System::Decimal(gcnew cli::array< System::Int32 >(4) { 300,
0, 0, 0 });
            this->numericUpDown1->Minimum = System::Decimal(gcnew cli::array< System::Int32 >(4) { 50, 0,
0, 0 });
            this->numericUpDown1->Name = L"numericUpDown1";
            this->numericUpDown1->Size = System::Drawing::Size(160, 22);
            this->numericUpDown1->TabIndex = 3;
            this->numericUpDown1->Value = System::Decimal(gcnew cli::array< System::Int32 >(4) { 100, 0, 0,
0 });
            this->numericUpDown1->ValueChanged += gcnew System::EventHandler(this,
&MyForm::numericUpDown1_ValueChanged);
            //
            // RandomColorButton
            //
            this->RandomColorButton->Location = System::Drawing::Point(708, 6);
            this->RandomColorButton->Margin = System::Windows::Forms::Padding(4);
            this->RandomColorButton->Name = L"RandomColorButton";
            this->RandomColorButton->Size = System::Drawing::Size(193, 43);
```

```
            this->RandomColorButton->TabIndex = 2;
            this->RandomColorButton->Text = L"Рандомный цвет орнамента";
            this->RandomColorButton->UseVisualStyleBackColor = true;
            this->RandomColorButton->Click += gcnew System::EventHandler(this,
&MyForm::RandomColorButton_Click);
            //
            // SaveButton
            //
            this->SaveButton->Location = System::Drawing::Point(959, 6);
            this->SaveButton->Margin = System::Windows::Forms::Padding(4);
            this->SaveButton->Name = L"SaveButton";
            this->SaveButton->Size = System::Drawing::Size(185, 38);
            this->SaveButton->TabIndex = 1;
            this->SaveButton->Text = L"Сохранить";
            this->SaveButton->UseVisualStyleBackColor = true;
            this->SaveButton->Click += gcnew System::EventHandler(this, &MyForm::SaveButton_Click);
            //
            // startProcess
            //
            this->startProcess->Location = System::Drawing::Point(4, 4);
            this->startProcess->Margin = System::Windows::Forms::Padding(4);
            this->startProcess->Name = L"startProcess";
            this->startProcess->Size = System::Drawing::Size(187, 41);
            this->startProcess->TabIndex = 0;
            this->startProcess->Text = L"Построить орнамент";
            this->startProcess->UseVisualStyleBackColor = true;
            this->startProcess->Click += gcnew System::EventHandler(this, &MyForm::startProcess_Click_1);
            //
            // MyForm
            //
            this->AutoScaleDimensions = System::Drawing::SizeF(8, 16);
            this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;
            this->ClientSize = System::Drawing::Size(1193, 549);
            this->Controls->Add(this->panel1);
            this->Controls->Add(this->pictureBox1);
            this->Margin = System::Windows::Forms::Padding(4);
            this->Name = L"MyForm";
            this->Text = L"MyForm";
            (cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this->pictureBox1))->EndInit();
            this->panel1->ResumeLayout(false);
            this->panel1->PerformLayout();
            (cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this->numericUpDown1))->EndInit();
            this->ResumeLayout(false);

        }
    #pragma endregion
    private:
        int hexagonSize;
        int hexagonColor;
        Color GetRandimColor() {
            srand(time(NULL));
            int r = rand() % 256;
            int g = rand() % 256;
            int b = rand() % 256;
            return Color::FromArgb(r, g, b);
        }
    private: System::Void numericUpDown1_ValueChanged(System::Object^ sender, System::EventArgs^ e) {
        pictureBox1->Refresh();
    }
    private: System::Void startProcess_Click_1(System::Object^ sender, System::EventArgs^ e) {
        // Очищаем box
        pictureBox1->Refresh();
```

```
            OrnamentDraw((int)numericUpDown1->Value, Color::Black);
        }
        private: System::Void SaveButton_Click(System::Object^ sender, System::EventArgs^ e) {
            // Определяем область, которую хотим сохранить (в данном случае - область pictureBox)
            Rectangle rect = pictureBox1->Bounds;
            // Создаем Bitmap с размерами области
            Bitmap^ bitmap = gcnew Bitmap(rect.Width, rect.Height);
            // Создаем Graphics для Bitmap
            Graphics^ graphics = Graphics::FromImage(bitmap);
            // Копируем область pictureBox на Graphics
            Point location = pictureBox1->PointToScreen(Point(0, 0));
            graphics->CopyFromScreen(location, Point(0, 0), rect.Size);
            // Освобождаем ресурсы Graphics
            delete graphics;
            // Открываем диалоговое окно для выбора места сохранения файла
            SaveFileDialog^ saveFileDialog = gcnew SaveFileDialog();
            saveFileDialog->Filter = "PNG Image (*.png)|*.png|JPEG Image(*.jpg)|*.jpg|Bitmap Image(*.bmp)|
*.bmp";
            saveFileDialog->Title = "Save Image";
            if (saveFileDialog->ShowDialog() == System::Windows::Forms::DialogResult::OK) {
                // Если пользователь выбрал место сохранения файла
                if (saveFileDialog->FileName != "") {
                    // Сохраняем Bitmap в выбранном месте
                    bitmap->Save(saveFileDialog->FileName, Imaging::ImageFormat::Png);
                }
            }
            // Освобождаем ресурсы Bitmap
            delete bitmap;

        }
        private: System::Void RandomColorButton_Click(System::Object^ sender, System::EventArgs^ e) {
            // Очищаем box
            pictureBox1->Refresh();
            OrnamentDraw((int)numericUpDown1->Value, GetRandimColor());
        }
            void OrnamentDraw(int size, Color color) {
                Graphics^ graphics = pictureBox1->CreateGraphics();
                Pen^ pen = gcnew Pen(color);
                //размеры picture box
                int width = pictureBox1->Width;
                int height = pictureBox1->Height;
                // Размер каждого шестиугольника
                int totalHexagons = width / (size / 2);
                // Общая ширина всех шестиугольников и промежутков между ними
                int startX = 0; // Начальная позиция по X
                int startY = 0;
                int numHexagonsY = height / (size / 2);
                int rowY = startY;
                int    rowX = startX;
                for (int ii = 0; ii < numHexagonsY; ii++) {
                    // координаты y для текущего ряда
                    rowY += size / 2 - size / 14;
                    // условие чередования местоположения
                    if (ii % 2 != 0) {
                        rowX += (size / 2 + size / 4);
                    }
                    BigHexagonDraw(totalHexagons, size, rowX, rowY, graphics, pen, color);
                    //при чередовании возвращениие в предыдущее
                    if (ii % 2 != 0) {
                        rowX = startX;
                    }
                }
```

```
        }
        void BigHexagonDraw(int totalHexagons, int size, int rowX, int rowY, Graphics^ graphics, Pen^ pen,
Color color) {
            int i = 0;
            while (i < totalHexagons || size * i + size <= pictureBox1->Width) {
                int x = rowX + (size / 2) * i; // Позиция X текущего шестиугольника
                // Рассчитываем координаты точек для текущего big шестиугольника
                array<Point>^ bighexagonPoints = gcnew array<Point>(6);
                for (int j = 0; j < 6; j++) {
                    double angle = Math::PI / 3 * j;
                    int pointX = x + (int)(size / 2 * Math::Cos(angle));
                    int pointY = rowY + (int)(size / 2 * Math::Sin(angle));
                    bighexagonPoints[j] = Point(pointX, pointY);
                }
                // Рисуем текущий big шестиугольник
                if (i % 3 == 0) {
                    Pen^ thickPen = gcnew Pen(color, 3);
                    graphics->DrawPolygon(thickPen, bighexagonPoints);
                }
                else {
                    graphics->DrawPolygon(pen, bighexagonPoints);
                }
                SmallHexagonDraw(size, x, rowY, graphics, pen);
                i++;
            }
        }
        void SmallHexagonDraw(int size, int x, int rowY, Graphics^ graphics, Pen^ pen) {
            // Рассчитываем размеры и координаты для внутреннего малого шестиугольника
            int smallSize = (int)(size * 0.8);
            int smallX = x;
            int smallY = rowY;

            // Рассчитываем координаты точек для текущего малого шестиугольника
            array<Point>^ smallHexagonPoints = gcnew array<Point>(6);
            for (int jj = 0; jj < 6; jj++) {
                double angle = 2 * Math::PI / 6 * jj;
                int pointX = smallX + (int)(smallSize / 2 * Math::Cos(angle));
                int pointY = smallY + (int)(smallSize / 2 * Math::Sin(angle));
                smallHexagonPoints[jj] = Point(pointX, pointY);
            }       // Рисуем текущий малый шестиугольник
            graphics->DrawPolygon(pen, smallHexagonPoints);
        }
    private: System::Void colorButton_Click(System::Object^ sender, System::EventArgs^ e) {
        ColorDialog^ colorDialog1 = gcnew ColorDialog();
        if (colorDialog1->ShowDialog() == System::Windows::Forms::DialogResult::OK) {
            Color selectedcolor = colorDialog1->Color;
            OrnamentDraw((int)numericUpDown1->Value, selectedcolor);
        }
    }
    private: System::Void backgroundColorButton_Click(System::Object^ sender, System::EventArgs^ e) {
        ColorDialog^ colorDialog2 = gcnew ColorDialog();
        if (colorDialog2->ShowDialog() == System::Windows::Forms::DialogResult::OK) {
            Color selectedcolor = colorDialog2->Color;
            pictureBox1->BackColor = selectedcolor;
        }
    }
};
}
```