минобрнауки россии

федеральное государственное бюджетное

образовательное учреждение высшего образования

«ЧЕРЕПОВЕЦКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

|  |  |
| --- | --- |
| Институт (факультет) | Информационных технологий |
| Кафедра | Математическое и программное обеспечение ЭВМ |

КУРСОВАЯ РАБОТА

|  |
| --- |
| по дисциплине Структурное программирование |

|  |  |
| --- | --- |
| на тему | «Программирование на языке высокого уровня» |
|  | |

|  |
| --- |
| Выполнил студент группы 1ПИб-01-1оп-22 |
| *группа* |
| направления подготовки (специальности) |
| 09.03.04 |
| *шифр, наименование* |
| Маслов Владислав Андреевич |
| *фамилия, имя, отчество* |

|  |
| --- |
| Руководитель |
| Пышницкий Константин Михайлович |
| *фамилия, имя, отчество* |
| Старший преподаватель |
| *должность* |

|  |
| --- |
| Дата представления работы |
| «\_\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20 \_\_\_ г. |
|  |
| Заключение о допуске к защите |
|  |
|  |
|  |
|  |
| Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| количество баллов |
| Подпись преподавателя\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

Череповец, 2023

*Год*

Аннотация

Курсовая работа по дисциплине: «Структурное программирование».

Автор: Маслов Владислав Андреевич – студент Череповецкого государственного университета, института информационных технологий, группы 1ПИб-02-1оп-22.

Тема: программное построение геометрического орнамента.

Курсовая работа была создана на основе Технического задания, описанного в Приложении 1.

В курсовой работе представлено описание разработки программы, которая рисует геометрический орнамент.

Оглавление

[Введение 4](#_Toc162504443)

[1. Описание предметной области 5](#_Toc162504444)

[2. Описание классов Graphics, Pen и Brush 9](#_Toc162504445)

[2.1. Описание класса Graphics 12](#_Toc162504446)

[2.2. Описание класса Pen 14](#_Toc162504447)

[2.3. Описание класса Brush 16](#_Toc162504448)

[3. Описание созданного приложения 17](#_Toc162504449)

[3.1. Постановка задачи 17](#_Toc162504450)

[3.2. Логическое проектирование 18](#_Toc162504451)

[3.3. Физическое проектирование 22](#_Toc162504452)

[3.4. Структуры данных и спецификация функций программы 23](#_Toc162504453)

[3.5. Тестирование 24](#_Toc162504454)

[3.6. Результаты работы 26](#_Toc162504455)

[Заключение 27](#_Toc162504456)

[Источники 28](#_Toc162504457)

[Приложение 1. Техническое задание 29](#_Toc162504458)

[Приложение 2. Руководство пользователя 36](#_Toc162504459)

[Приложение 3. Программный код 40](#_Toc162504460)

# Введение

В современном обществе компьютерные технологии играют важную роль и активно влияют на различные области жизни человека. Они создаются с целью облегчить и помочь человеку, используя разнообразные инструменты.

Один из таких инструментов – язык программирования C++ – это высокоуровневый язык программирования, который используется для разработки программного обеспечения.

В языке C++ используются различные конструкции, включая переменные, функции, классы и объекты, условные операторы, операторы цикла и т.д. Эти конструкции используются для создания алгоритмов программы и решения задач. [1].

Программирование на языке высокого уровня в современном мире играет ключевую роль в автоматизации и оптимизации различных процессов. В данной курсовой работе мы рассмотрим создание программы на языке C++ в среде разработки Visual Studio, которая будет рисовать геометрический орнамент с помощью библиотеки Windows Forms.

Целью данной работы является разработка приложения, которое будет создавать орнамент с возможностью изменения цвета, количества итераций и качества рендера, на языке программирования С++ с использованием Windows Forms.

# Описание предметной области

Орнамент – это разновидность декоративного оформления предметов или объектов посредством чередования и сочетания различных узоров. Орнамент широко используется в архитектуре и вышивке, монументальной скульптуре и гончарном деле, книжной графике и производстве текстильных изделий, ювелирном деле и кузнечном ремесле. [2]

Существует множество различных видов орнамента, каждый из которых имеет свои характеристики и исторические корни:

1. Геометрический орнамент – узор, который состоит из геометрических фигур. Постоянными элементами геометрического орнамента являются отдельные фигуры в виде точек, прямых и ломаных линий, кругов, крестов, треугольников, квадратов (рис. 1).

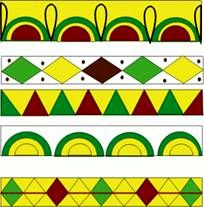


Рис. 1. Геометрический орнамент

1. Растительный орнамент – это орнамент, созданный с использованием повторяющегося многократно растительного мотива (пучка листьев, виноградной грозди, цветка и т. д.) (рис. 2).



Рис. 2. Растительный орнамент

1. Животный орнамент – это орнамент, созданный изображениями животных (рис. 3).

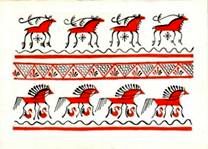


Рис. 3. Животный орнамент

1. Антропоморфный орнамент – декор, в котором используются повторяющиеся изображения женских и мужских стилизованных фигур либо лиц (рис. 4).



Рис. 4. Антропоморфный орнамент

1. Абстрактный орнамент – это орнамент, созданный без использования конкретных объектов или форм, а скорее на основе абстрактных линий, фигур и цветов (рис. 5).

Изображение выглядит как рисунок, искусство, шаблон, зарисовка

Автоматически созданное описание

Рис. 5. Абстрактный орнамент

1. Культурный орнамент – это орнамент, который имеет особенности определенной культуры или этнической группы, такие как азтекский орнамент, кельтский орнамент (рис. 6).

Изображение выглядит как шаблон, одежда, текстиль, Выкройка (дизайн одежды)

Автоматически созданное описание

Рис. 6. Культурный орнамент

1. Астральный орнамент – утверждал культ неба. Основными его элементами были изображения неба, солнца, облаков, звезд (рис. 8).

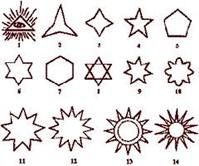


Рис. 7. Культурный орнамент

1. Каллиграфический орнамент – это орнамент, который составляется из отдельных букв или элементов текста, выразительных по своему пластическому рисунку и ритму (рис. 8).



Рис. 8. **Каллиграфический орнамент**

Кроме этого, существует ещё множество видов орнаментов. [3].

# 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++ [4].

Инициализация и обработка ошибок:

• detectgraph() – определяет графический драйвер и графический режим при проверке аппаратного обеспечения;

• initgraph() – установка видеорежима;

• restorecrtmode() – восстановление текстового режима;

• graphresult() – возвращает код ошибки выполнения последней графической операции;

• grapherrormsg() – возвращает указатель на строку сообщения об ошибке;

• closegraph() – выход из графического режима;

• getdrivername() – возвращает указатель на строку, содержащую имя текущего графического драйвера;

• getmaxmode() – возвращает максимальный номер режима для текущего драйвера;

• getaspectratio() – возвращает текущее характеристическое отношение ширины изображения к его высоте графического режима;

• setaspectratio() – установка коэффициента сжатия;

• getgraphmode() – возвращает текущий графический режим;

• getmodename() – возвращает указатель на строку, содержащую имя указанного графического режима;

• getmoderange() – получает диапазон режимов для данного графического драйвера;

• graphdefaults() – переустанавливает все графические параметры в их начальные значения;

• installuserdriver() – добавляет дополнительные драйверы устройств в таблицу драйверов устройств BGI;

• installuserfont() – загружает файл со шрифтом.

Фигуры:

• line() – рисует линию между двумя указанными точками;

• linerel() – рисует линию на заданное расстояние от текущей позиции CP;

• lineto() – рисует линию от текущей позиции;

• circle() – рисует окружность заданного радиуса;

• ellipse() – эллиптическая дуга;

• arc() – дуга окружности;

• getarccoords() – выдает координаты последнего обращения к функции arc;

• ellipse() – эллиптическая дуга;

• rectangle() – рисует прямоугольник;

• drawpoly() – рисует контур многоугольника;

• getx() – возвращает координату x текущей графической позиции;

• gety() – возвращает координату y текущей графической позиции;

• moveto() – изменение значения текущего положения пера (CR).

Заполненные фигуры:

• bar() – прямоугольник;

• fillpoly() – рисует и закрашивает многоугольник;

• fillellipse() – рисует и закрашивает эллипс;

• sector() – рисует заполненный эллиптический сектор;

• pieslice() – рисует заполненный сектор круга.

Цвет и заливка:

• setcolor() – установка цвета;

• getcolor() – возвращает текущий цвет рисунка;

• getmaxcolor() – возвращает максимальное значение цвета, которое может быть передано функции setcolor;

• setbkcolor() – установка цвета фона;

• getbkcolor() – возвращает текущий цвет фона;

• putpixel() – установить цвет пикселя;

• getpixel() – возвращает цвет заданной точки.

Работа с палитрой:

• setpalette() – изменение цвета в палитре;

• getpalette() – получает информацию о текущей палитре;

• setrgbpalette() – изменение цвета в палитре по системе RGB;

• getdefaultpalette() – возвращает структуру определения палитры;

• getpalettesize() – возвращает размер справочной таблицы цветов палитры.

Стиль и заливка:

• floodfill() – заполняет ограниченную область;

• setfillstyle() — устанавливает цвет и шаблон заполнения;

• getfillsettings() – получает информацию о текущем шаблоне и цвете заполнения;

• setfillpattern() – задаёт шаблон заполнения, определяемый пользователем;

• getfillpattern() – возвращает определенный пользователем шаблон заполнения;

• setlinestyle() – установка ширины и стиля линии;

• getlinesettings() – получает текущий цвет линии, шаблон и толщину.

Окна и страницы:

• cleardevice() – очистка активной страницы;

• setactivepage() – устанавливает номер активной страницы;

• setvisualpage() – устанавливает номер выводимой страницы;

• clearviewport() –очищает текущее окно;

• setviewport() – создаёт графическое окно;

• getviewsettings() – вызывает информацию об активном окне;

• getmaxx() – возвращает максимальную координату х экрана;

• getmaxy() – возвращает максимальную координату y экрана.

Графические образы:

• getimage() – запоминает в битовый образ некоторой области экрана;

• imagesize() – возвращает количество байтов, требуемых для хранения битового образа;

• putimage() – помещает на экран ранее записанный в память образ.

Текст:

• outtext() – выводит строку на экран с текущей позиции;

• outtextxy() – выводит строку на экран с заданной позицией;

• settextstyle() – устанавливает шрифт, стиль и коэффициент увеличения шрифта;

• setusercharsize() – устанавливает пользовательский размер шрифта;

• settextjustify() – устанавливает режим выравнивания текста;

• textheight() – возвращает ширину строки в пикселях;

• textwidth() – возвращает высоту строки в пикселях.

Память:

• \_graphfreemem() – освобождает графическую память;

• \_graphgetmem() – захватывает графическую память.

## Описание класса Graphics

Graphics – это класс, предоставляющий функционал для рисования на поверхности. Данный класс входит в пространство имен System.Drawing, как и большинство классов для работы с графикой. Класс Graphics используется для работы с графическим контекстом, который представляет поверхность, на которую рисуют [5].

Ниже приведены методы данного класса (табл.1):

Таблица 1

Основные методы класса Graphics

|  |  |
| --- | --- |
| Имя метода | Описание |
| Clear(Color) | Очищает всю поверхность рисования и выполняет заливку поверхности указанным цветом фона. |
| CopyFromScreen(Point, Point, 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, Rectangle, Single, Single) | Рисует дугу, которая является частью эллипса, заданного структурой Rectangle. |
| DrawBezier(Pen, Point, Point, Point, Point) | Рисует кривую Безье, определяемую четырьмя структурами Point. |
| DrawBeziers(Pen, Point[]) | Рисует несколько (N) кривых Безье, определяемых массивом из (3N+1) структур Point. |
| DrawCloseCurve(Pen, Point[ ]) | Рисует замкнутый фундаментальный сплайн. |
| DrawEllipse(Pen, Rectangle) | Рисует эллипс. |
| DrawIcon(Icon, Rectangle) | Рисует значок. |
| DrawImage(Image image, int x, int y) | Рисует заданное изображение image, используя его фактический размер в месте с координатами (x,y). |
| DrawLine(Pen, Point, Point) | Проводит линию, соединяющую две структуры Point. |

Продолжение табл. 1

|  |  |
| --- | --- |
| DrawPolygon(Pen, Point[]) | Рисует многоугольник, определяемый массивом структур Point. |
| DrawRectangle(Pen, Rectangle) | Рисует прямоугольник, определяемый структурой Rectangle. |
| DrawString(String, Font, Brush, PointF) | Создает указываемую текстовую строку в заданном месте с помощью определяемых объектов Brush и Font. |
| Equals(Object) | Определяет, равен ли заданный объект текущему объекту. |
| FillClosedCurve(Brush, Point[]) | Заполняет внутреннюю часть замкнутой фундаментальной кривой, определяемой массивом структур Point. |
| FillEllipse(Brush, Rectangle) | Заполняет внутреннюю часть эллипса, определяемого ограничивающим прямоугольником, который задан структурой Rectangle. |
| Flush() | Вызывает принудительное выполнение всех отложенных графических операций и немедленно возвращается, не дожидаясь их окончания. |
| IntersectClip(Region) | Обновляет вырезанную область данного объекта, включая в нее пересечение текущей вырезанной области и указанной структуры |
| ResetClip() | Сбрасывает выделенную область g, делая ее бесконечной. |

## Описание класса Pen

Для контура используется перо. Класс Pen представляет собой инструмент для письма, находящийся в группе Draw, класса Graphics. Класс Pen включает в себя методы для нанесения линий, для изменения цвета и толщины линии [6].

Ниже приведены методы данного класса (табл.2):

Таблица 2

Основные методы класса Pen

|  |  |
| --- | --- |
| Имя метода | Описание |
| Clone() | Создает точную копию данного объекта Pen. |
| Dispose() | Освобождает все ресурсы, используемые этим объектом Pen. |
| Equals(Object) | Определяет, равен ли указанный объект текущему объекту. |
| Finalize() | Позволяет объекту попытаться освободить ресурсы и выполнить другие операции очистки, перед тем как он будет уничтожен во время сборки мусора. |
| GetHashCode() | Служит хэш-функцией по умолчанию. |
| GetLifetimeService() | Извлекает объект обслуживания во время существования, который управляет политикой времени существования данного экземпляра. |
| GetType() | Возвращает объект Type для текущего экземпляра. |
| InitializeLifetimeService() | Получает объект службы времени существования для управления политикой времени существования для этого экземпляра. |
| MemberwiseClone() | Создает неполную копию текущего объекта Object. |
| MemberwiseClone(Boolean) | Создает неполную копию текущего объекта MarshalByRefObject. |
| MultiplyTransform(Matrix) | Умножает матрицу преобразования объекта Pen на заданный объект Matrix. |
| MultiplyTransform(Matrix, MatrixOrder) | Умножает матрицу преобразования объекта Pen на заданный объект Matrix в заданном порядке. |
| ResetTransform() | Возвращает матрице геометрического преобразования объекта Pen единичное значение. |

Продолжение табл. 2

|  |  |
| --- | --- |
| RotateTransform(Single) | Поворачивает локальное геометрическое преобразование на заданный угол. Этот метод добавляет поворот перед преобразованием. |
| RotateTransform(Single, MatrixOrder) | Поворачивает локальное геометрическое преобразование на заданный угол в заданном порядке. |
| ScaleTransform(Single, Single) | Выполняет изменение масштаба локального геометрического преобразования при помощи заданных коэффициентов. Этот метод вставляет изменение масштаба перед преобразованием. |
| SetLineCap(LineCap, LineCap, DashCap) | Устанавливает значения, определяющие стиль завершения, используемый в конце линий, нарисованных при помощи объекта Pen. |
| TranslateTransform(Single, Single) | Выполняет перевод локального геометрического преобразования на заданные размеры. Этот метод вставляет перевод перед преобразованием. |
| TranslateTransform(Single, Single, MatrixOrder) | Выполняет перевод локального геометрического преобразования на заданные размеры в заданном порядке. |

## Описание класса Brush

Класс Brush имеет несколько конструкторов, которые позволяют создавать кисти различных типов. Также класс содержит методы для изменения цвета, продолжительности и других параметров кисти. Кисти могут быть использованы с объектами классов Graphics, Pen и SolidBrush.

Класс Brush является абстрактным, что означает, что нельзя создать его экземпляр напрямую. Вместо этого необходимо создать объекты классов-наследников, таких как SolidBrush, TextureBrush, LinearGradientBrush [6].

Ниже приведены методы данного класса (табл.3):

Таблица 3

Основные методы класса Brush

|  |  |
| --- | --- |
| Имя метода | Описание |
| Clone() | Создает изменяемый клон данного объекта Brush, делая глубокие копии значений этого объекта. При копировании свойств зависимостей этот метод копирует ссылки на ресурсы и привязки данных (которые могут уже не быть разрешимыми), но не копирует анимации и их текущие значения. |
| Dispose() | Освобождает все ресурсы, используемые этим объектом Pen. |
| CloneCurrentValue() | Создает модифицируемый клон данного объекта Brush, делая глубокие копии текущих значений этого объекта. Ссылки на ресурсы, привязки данных и анимации не копируются, но копируются их текущие значения. |
| GetHashCode() | Возвращает хэш-код объекта Brush. |
| GetHashCode() | Служит хэш-функцией по умолчанию. |
| GetType() | Возвращает объект Type для текущего экземпляра. |
| ToString(IFormatProvider) | Создает строковое представление данного объекта на основе сведений о форматировании, связанных с определенным языком и региональными параметрами. |
| ScaleTransform() | Применяет масштабирование к матрице преобразования кисти. |
| TranslateTransform() | Применяет сдвиг к матрице преобразования кисти. |

# Описание созданного приложения

## Постановка задачи

Требуется написать программу, реализующую построение геометрического орнамента.

Функционал, который должен присутствовать в программе:

* Изменение масштаба;
* Изменение цвета;
* Сохранение изображения в файле и в открывшемся окне указание его названия и пути к папке, в которой он лежит.

Орнамент был выбран и нарисован разработчиком и согласован с преподавателем (рис. П1.1).

Дополнительным функционалом, присутствующим в программе, является изменение качества рендеринга.

## Логическое проектирование

В ходе анализа составляющей фигуры геометрического орнамента выясняется, что изображение состоит из треугольников, полукругов и круга.

Чтобы построить орнамент, нужно найти координаты фигур. В C++ координатная плоскость начинается в левом верхнем углу в точке, где x=0 и y=0, и заканчивается в правом нижнем углу в (width, height), где width - ширина холста, а height - высота.

Для построения треугольников использовались координаты трёх вершин:

1. Первый треугольник:
   * Первая точка: (x, y + height /3);
   * Вторая точка: (x, y);
   * Третья точка: (x + width/3, y).
2. Второй треугольник:
   * Первая точка: (x, y + 2\*height / 3);
   * Вторая точка: (x, y + height);
   * Третья точка: (x + width / 3, y + height).
3. Третий треугольник:
   * Первая точка: (x + 2\*width / 3, y + height);
   * Вторая точка: (x + width, y + height);
   * Третья точка: (x + width, y + 2\*height / 3).
4. Четвертый треугольник:
   * Первая точка: (x + width, y + height / 3);
   * Вторая точка: (x + width, y);
   * Третья точка: (x + 2\*width / 3, y).
5. Пятый треугольник:
   * Первая точка: (x + width / 3, y + 2\*height / 3);
   * Вторая точка: (x + width / 2, y + height / 3);
   * Третья точка: (x + 2\*width / 3, y + 2 \* height / 3).
6. Шестой треугольник:
   * Первая точка: (x + width / 3, y + height / 3);
   * Вторая точка: (x + width / 2, y + 2 \* height / 3);
   * Третья точка: (x + 2 \* width / 3, y + height / 3).

Для построения четырёх полукругов использовались координаты:

1. Первый полукруг: (x - 2 \* width / 3 , y - 2 \* height / 3, width, height);
2. Второй полукруг: (x + 2 \* width / 3, y - 2 \* height / 3, width, height);
3. Третий полукруг: (x + 2 \* width / 3, y + 2 \* height / 3, width, height);
4. Четвёртый круг: (x - 2 \* width / 3, y + 2 \* height / 3, width, height).

Для построения круга использовались координаты:

(x + width / 8 \* 3, y + height / 8 \* 3, width / 4, height / 4).

В итоге, используя эти значения, если количество итераций равно единице, получится изображение (рис. 9).

Изображение выглядит как круг, снимок экрана, дизайн

Автоматически созданное описание

Рис. 9. Итоговое изображение

Блок-схема, описывающая работу приложения описана на рисунках 10-11.

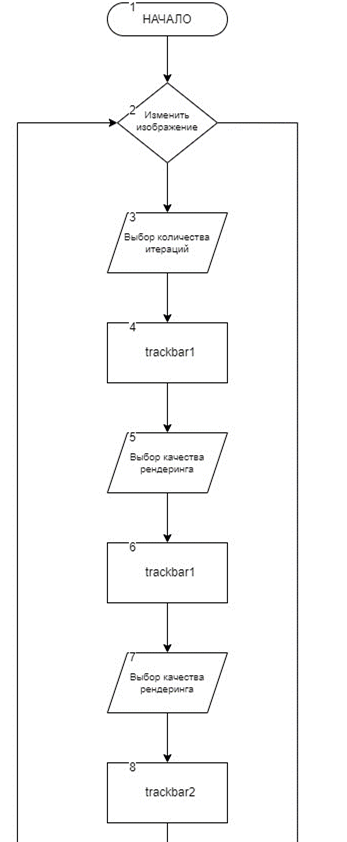
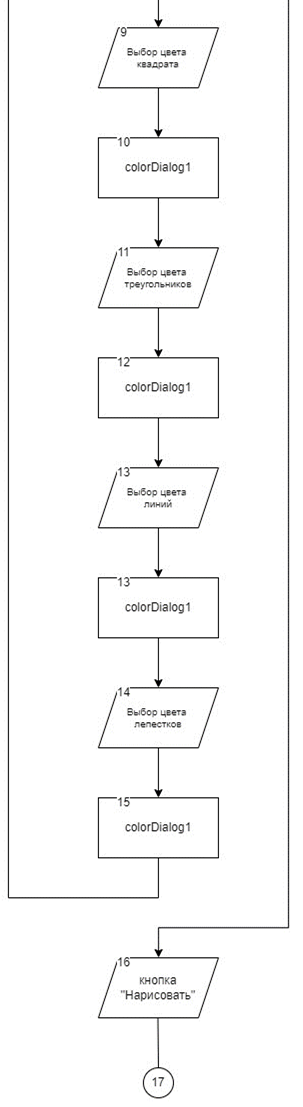
 

Рис. 10. Блок-схема программы

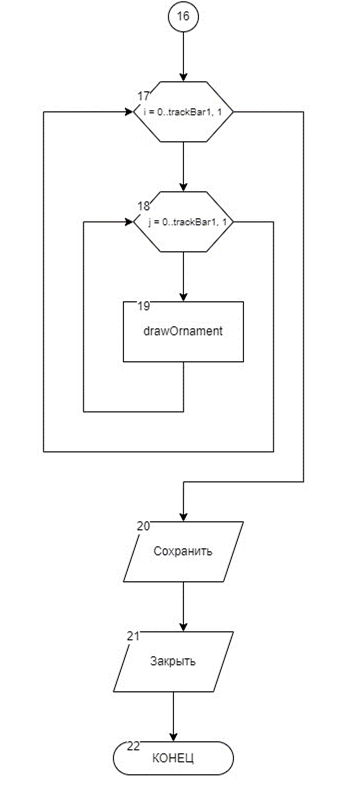


Рис. 11. Блок схема программы

## Физическое проектирование

Структуры данных и функции, использовавшиеся в программе по изображению графика, представлены ниже (табл.4-5).

Таблица 4

Структуры данных

|  |  |  |
| --- | --- | --- |
| Обозначение | Тип данных | Описание |
| ellipseColor | Color | Хранит цвет круга |
| triangleColor | Color | Хранит цвет треугольника |
| arcColor | Color | Хранит цвет линий |
| backColor | Color | Хранит цвет фона |
| bitmap | Bitmap^ | Хранит отрисованное изображение |
| gr | Graphics^ | Объект для отрисовки в img |
| button1 | Button | Нажатие приводит к отображению орнамента |
| button2 | Button | Нажатие приводит к выбору цвета |
| button3 | Button | Нажатие приводит к выбору цвета |
| button4 | Button | Нажатие приводит к выбору цвета |
| button5 | Button | Нажатие приводит к выбору цвета |
| button6 | Button | Нажатие приводит к сохранению картинки |
| trackBar1 | TrackBar | Выбор количества итераций орнамента |
| trackBar2 | TrackBar | Выбор качества рендеринга орнамента |
| pictureBox1 | PictureBox | Холст с орнаментом |
| label1 | Label | Надпись “Количество итераций”, находящаяся над трек-баром. |
| label2 | Label | Надпись “Качество рендера”, находящаяся над трек-баром. |
| colorDialog1 | ColorDialog | После нажатия любой из кнопок выбора |

## Структуры данных и спецификация функций программы

В визуальном приложении была использована одна функция drawOrnament типа void, которая рисует весь орнамент. В неё передаётся объект для отрисовки и координаты верхнего левого и нижнего правого углов pictureBox1.

У данной разработки при определённых действиях будет определённый исход, всё это показано в табл. 3.2.

Таблица 3.2

Обработчики событий формы MyForm

|  |  |  |
| --- | --- | --- |
| Название | Параметры | Результат действия |
| button1\_Click | System::Object^ sender, System::EventArgs^ e | Рисует орнамент |
| button2\_Click | System::Object^ sender, System::EventArgs^ e | Задаёт цвет круга |
| button3\_Click | System::Object^ sender, System::EventArgs^ e | Задаёт цвет треугольникам |
| button4\_Click | System::Object^ sender, System::EventArgs^ e | Задаёт цвет линиям |
| button5\_Click | System::Object^ sender, System::EventArgs^ e | Задаёт цвет лепесткам |
| Button6\_Click | System::Object^ sender, System::EventArgs^ e | Сохраняет изображение, если оно нарисовано, если не нарисовано, то сообщает об ошибке |

## Тестирование

Для корректной работы программы и устранению всех ошибок необходимо проводить её тестирование. Тестовые данные и результаты тестов приведены ниже (табл. 6-7).

Таблица 6

Тестовые данные

|  |  |  |
| --- | --- | --- |
| Исходные данные | Тестируемый модуль или подпрограмма | Ожидаемый результат |
| Использовались различные параметры количества итераций | ornament.sln | Успех |
| Использовались различные параметры качества рендера | ornament.sln | Успех |
| Использовались различные параметры цвета | ornament.sln | Провал |
| Закрытие окна с выбором цвета | ornament.sln | Провал |
| Множественное использование построения графика | ornament.sln | Провал |
| Производилось сохранение графиков | ornament.sln | Провал |
| Множественные сохранения графиков | ornament.sln | Успех |

Таблица 7

Результаты тестирования

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Дата тестирования | Тестируемый модуль или подпрограмма | Кто проводил тестирование | Описание тестирования | Результаты тестирования |
| 01.06.2023 | ornament.sln | Маслов В.А. | При изменении ползунка количества итераций программа корректно меняла масштаб изображения. | Успех |
| 01.06.2023 | ornament.sln | Маслов В.А. | При изменении ползунка качества рендера программа корректно меняла масштаб изображения. | Успех |
| 01.06.2023 | ornament.sln | Маслов В.А. | При изменении параметра было выявлено неправильное построение изображения. | Провал |
| 01.06.2023 | ornament.sln | Маслов В.А. | При закрытии окна выбора цвета без подтверждения программа выдавала ошибку. | Провал |
| 02.06.2023 | ornament.sln | Маслов В.А. | При нажатии на кнопку “Нарисовать” происходило построение изображения. | Провал |
| 03.06.2023 | ornament.sln | Маслов В.А. | При изменении ползунка количества итераций программа корректно меняла масштаб изображения. | Успех |
| 05.06.2023 | ornament.sln | Маслов В.А. | При сохранении файла выявлена ошибка. | Провал |

## Результаты работы

В ходе работы была написана программа, которая создаёт орнамент, которому можно задавать масштаб, цвет и параметры функции. Программа создана на языке программирования С++ и с использованием Windows Forms. Программа соответствует поставленной задаче.

Интерфейс программы разделён на две части - левая и правая. Слева рисуется изображение, а справа расположено меню настроек параметров орнамента. Пользователь с помощью трек-баров изменяет количество итераций фигуры и качество её рендера в орнаменте. Кнопками изменяет цвет, рисует и сохраняет изображение. Интерфейс программы представлен ниже (рис. 13).

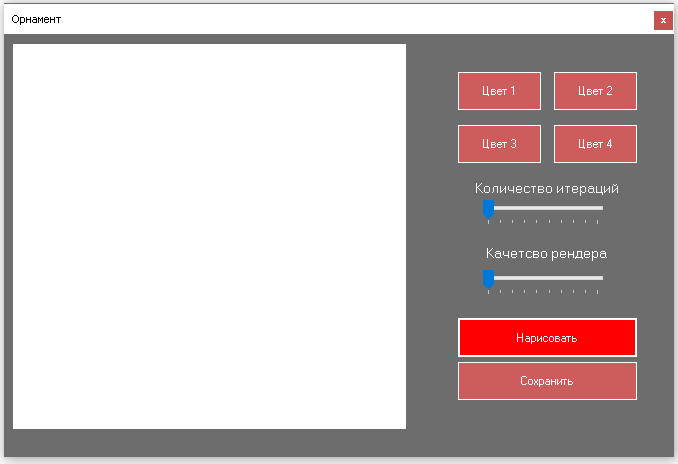


Рис. 13. Интерфейс программы

Когда пользователь задаёт параметры и нажимает кнопку “Нарисовать”, в окне приложения появляется орнамент (рис. 14).

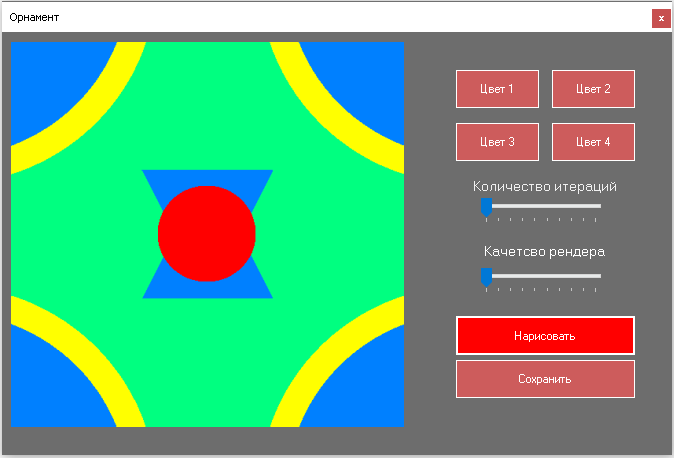


Рис. 14. Орнамент в окне приложения

# Заключение

Во время работы над курсовой было разработано приложение на языке высокого уровня, помогающие пользователям создавать геометрический орнамент. Программа разрешает пользователю выбирать количество итераций и качество рендера. Данная работа наглядно показывает построение соответствующего графика и приводит алгоритм её работы.

Благодаря программированию высокого уровня существует возможность создавать уникальные и привлекательные рисунки, используя различные алгоритмы и математические формулы. Для создания визуального приложения была изучена библиотека Graphics класса, которая позволяет рисовать геометрические фигуры с помощью Brush и Pen в Windows Form.

# Источники

1. Язык программирования С++ [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/C%2B%2B (дата обращения: 06.05.2023).
2. Орнамент – что это такое: суть орнаментов, виды, история возникновения и развития. Национальные особенности орнаментов и самые известные из них [Электронный ресурс]. URL: https://veryimportantlot.com/ru/news/blog/ornament (дата обращения: 06.05.2023).
3. Орнаменты и их виды [Электронный ресурс]. URL: https://obuchonok.ru/node/2262#:~:text=Орнаменты%20подразделяются%20на%20следующие%20виды%3A,пейзажный%2C%20животный%2C%20предметный%20(или%20вещный) (дата обращения: 07.05.2023).
4. Описание класса Graphics [Электронный ресурс]. URL: <http://mycpp.ru/cpp/scpp/cppd_graphics.h.htm> (дата обращения: 09.05.2023).
5. Описание классов Pen и Brush [Электронный ресурс]. [Электронный ресурс]. URL: <http://mycpp.ru/cpp/scpp/cppd_graphics.h.htm> (дата обращения: 09.05.2023).

# Приложение 1. Техническое задание

МИНОБОРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

«ЧЕРЕПОВЕЦКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Институт информационных технологий

наименование института (факультета)

Математическое и программное обеспечение ЭВМ

наименование кафедры

Структурное программирование

наименование дисциплины в соответствии с учебным планом

УТВЕРЖДАЮ

Зав. кафедрой МПО ЭВМ

д.т.н., профессор Ершов Е.В.

« » 2023 г.

Программирование на языке высокого уровня

Техническое задание на курсовую работу

Листов 15

Руководитель: доцент Пышницкий К.М.

Исполнитель: студент гр. 1ПИб-02-1оп-22

Маслов В.А.

2023 год

Введение

Программа позволит пользователю создать геометрический орнамент. Данная разработка наглядно показывает, как работает графическое приложение Windows Form на C++.

1. Основания для разработки

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

Дата утверждения: 23 февраля 2023 года.

Наименование темы разработки: Программирование на языке высокого уровня.

2. Назначение разработки

Разрабатываемая программа предназначена для студентов, так как разработка подобной программы может обладать академической ценностью, поскольку позволяет изучить работу с графикой и алгоритмами отрисовки. Может быть полезна для пользователей, которые хотят разработать иллюстрации, арт-работы, текстуры и другие графические объекты для использования в различных проектах.

3. Требования к программе

3.1. Требования к функциональным характеристикам

Приложение должно выполнять следующие функции:

1. Рисовать орнамент по фигуре, утверждённой ранее с преподавателем (рис. П1.1);
2. Изменять цвет фигур созданного орнамента;
3. Возможность выбирать количество итераций;
4. Сохранять его в файл с указанием его названия и пути к папке, в которой он находится.

Изображение выглядит как шаблон, Симметрия, дизайн, снимок экрана

Автоматически созданное описание

Рис. П1.1. Фигура

3.2. Требования к надежности

В ходе проектирования и разработки обеспечить защиту при попытках пользователя совершить не предусмотренное программой действие, которое может привести к нарушению работы приложения или его закрытию.

3.3. Условия эксплуатации

Правила эксплуатации компьютера в закрытом помещении:

1. Компьютер должен находиться на ровной поверхности, защищенной от пыли и вибрации.
2. Окружающая температура не должна превышать 35 градусов и не должна падать ниже 10 градусов.
3. Влажность в помещении должна быть не более 60%.
4. Компьютер должен быть подключен к надежному источнику питания.
5. В помещении не должно быть никаких вспышек или сильных электромагнитных помех.

3.4. Требования к составу и параметрам технических средств

Программа должна корректно работать при соблюдении минимальных системных требований:

1. Процессор с тактовой частотой не менее 1 ГГц.
2. Оперативная память не менее 2 Гбайт.
3. Видеокарта, поддерживающая DirectX 9 совместимость и минимум 1024 Мб видеопамяти.
4. Свободное место на жестком диске не менее 100 Мб.
5. Устройства ввода: клавиатура и мышь.

3.5. Требования к информационной и программной совместимости

Код программы реализуется на языке C++ в среде разработки Visual Studio 2022. Данный программный продукт может использоваться в операционных системах Windows 7 / 8 / 10 / 11 при наличии установленного наборf библиотек и системных компонентов - .NET Framework 4.5 или выше.

3.6. Требования к маркировке и упаковке

Распространение данного приложения будет происходить бесплатно, через разные интернет-платформы для возможности его использования.

3.7. Требования к транспортированию и хранению

Созданный проект будет помещен в папку, где будет храниться, а также будет записан на отдельный информационный накопитель для возможности восстановления.

3.8. Специальные требования

Для работы с приложением пользователь должен иметь минимальные навыки в работе с компьютером, быть старше 12 лет и не иметь психологических отклонений

4. Требование к программной документации

4.1. Содержание расчётно-пояснительной записки

Программная документация должна содержать расчётно-пояснительную записку с содержанием:

Введение

1. Описание предметной области.
2. Описание классов Graphics, Pen и Brush.
3. Описание созданного приложения: постановка задачи; логическое проектирование - алгоритм работы программы; физическое проектирование - структуры данных и спецификация функций; тестирование; результаты работы.
4. Заключение
5. Список литературы.
6. Техническое задание.
7. Руководство пользователя.
8. Программный код.

4.2. Требования к оформлению

Оформление документации пишется по правилам перечисленным в таблице (табл.П1.1).

Таблица П1.1

Правила оформления документации

|  |  |
| --- | --- |
| 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. |
| 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. Стадии и этапы разработки

В данном пункте в табличном виде представлена информация о стадиях и этапах разработки приложения (табл.П1.2).

Таблица П1.2.

Стадии и этапы разработки

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование  этапа разработки | Сроки разработки | Результат выполнения | Отметка о выполнении |
| Определение темы для курсового проекта | 23.02.2023 | Утверждена тема для разработки |  |
| Программирование | 22.05.2023-01.06.2023 | Созданная программа |  |
| Тестирование | 01.06.2023-09.06.2023 | Проверка программы на возможные ошибки и их устранение |  |

6. Порядок контроля и приемки

В данном пункте в табличном виде представлена информация о порядке и контроля приемки курсовой работы (табл.П1.3).

Таблица П1.3.

Порядок контроля и приемки

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование  контрольного этапа  выполнения  курсовой работы | Сроки  контроля | Результат выполнения | Отметка о приемке  результата  контрольного этапа |
| Создание технического задания | 20.04.2023 | Готовый документ «Техническое задание» |  |
| Начало создания РПЗ | 16.05.2023 | Частично заполненные разделы РПЗ |  |
| Окончательное оформление РПЗ | 06.06.2023 | Готовый документ РПЗ |  |
| Сдача РПЗ. Оценка качества РПЗ | 09.06.2023 | Итоговая оценка за курсовую работу |  |

# Приложение 2. Руководство пользователя

1. Общие сведения о программе

Это компьютерная программа, которая может создавать графическое представление геометрического орнамента. Орнамент – это разновидность декоративного оформления предметов или объектов посредством чередования и сочетания различных узоров.

1. Описание установки

Для работы программы нужно установить Visual Studio 2022. Для этого необходимо сделать следующие шаги:

1. Скачать установщик с официального сайта Microsoft.
2. Запустить установщик.
3. Принимаем условия лицензионного соглашения (рис.П2.2).



Рис. П2.2 Окно принятия лицензионного соглашения

1. Далее в появившемся окне в разделе “Классические и мобильные приложения” выбираем “Разработка классических приложений .NET” и “Разработка классических приложений на C++”. Дополнительно в правой части устанавливаем “C++/CLI support for v143…” (рис.П2.3).

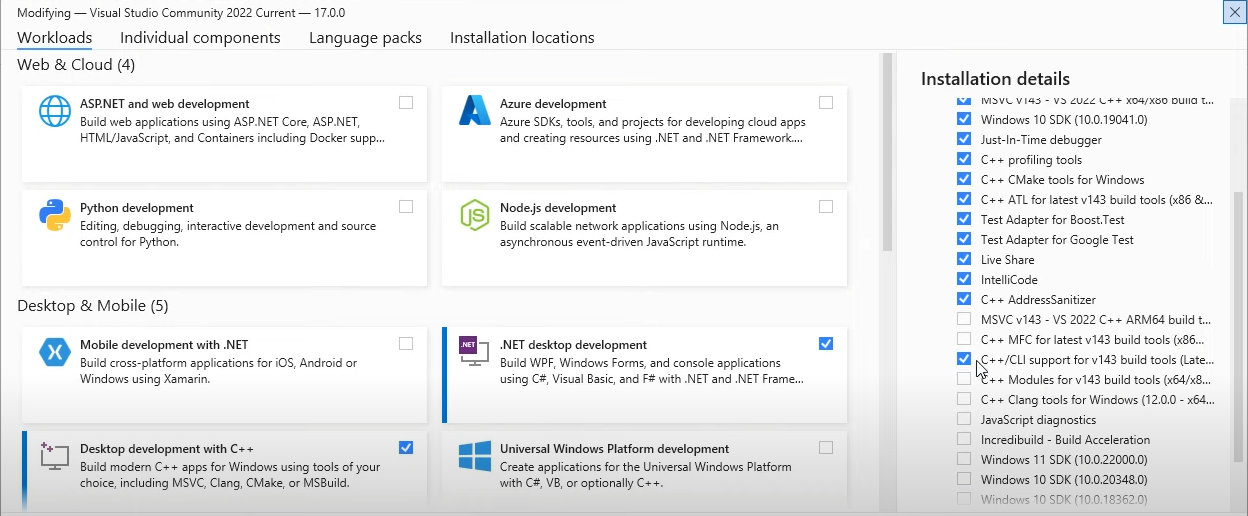


Рис. П2.3 Окно установки компонентов

1. Ждём установки программы.
2. Описание запуска

Для запуска программы необходимо открыть файл “Kursovaya.exe”. После чего появится окно программы.

1. Инструкция по работе

После запуска приложения мы видим его интерфейс (рис.П2.4).

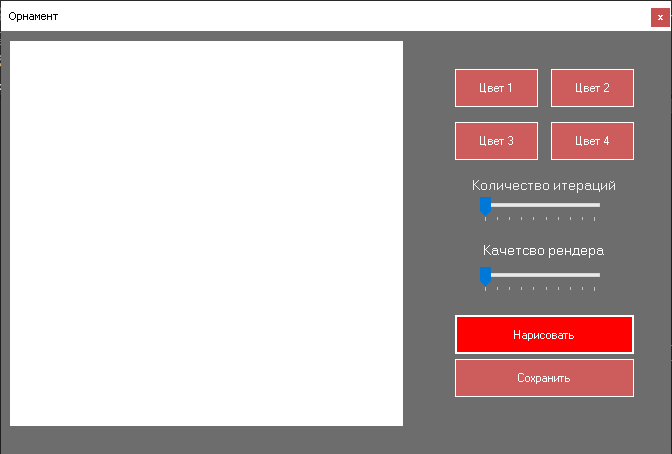


Рис. П2.4. Окно программы

Открывшееся окно делится на несколько частей, самая левая – это полотно, на котором будет расположен орнамент (рис. П2.5).

Изображение выглядит как белый, дизайн

Автоматически созданное описание

Рис. П2.5. Полотно

Правая часть состоит из трек-баров, отвечающих за количество итераций и качества рендера, цветов в которые будет окрашены элементы орнамента, кнопка сохранить, и кнопка нарисовать (рис. П2.6).

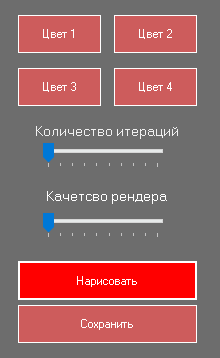


Рис. П2.6. Меню

У каждой кнопки цвета есть небольшое окно, в котором, показан цвет по умолчанию при запуске программы, а также в них будет отображён выбранный цвет (рис. П2.4)

Кнопка “Нарисовать” отвечает за отображение орнамента на полотне (рис. П2.3) при выбранных параметрах.

Кнопка “Сохранить” сохраняет орнамент в jpg расширении в папке проекта. При попытке сохранить пустое полотно, скриншот не появится в папке проекта. Для правильного сохранения рисунка, нужно, чтобы заранее были выбраны параметры и нарисован орнамент.

# Приложение 3. Программный код

#pragma once

namespace Project1 {

using namespace System;

using namespace System::ComponentModel;

using namespace System::Collections;

using namespace System::Windows::Forms;

using namespace System::Data;

using namespace System::Drawing;

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

{

public:

MyForm(void)

{

InitializeComponent();

}

protected:

~MyForm()

{

if (components)

{

delete components;

}

}

private: Color ellipseColor = Color::FromArgb(133, 108, 94);

private: Color arcColor = Color::FromArgb(133, 108, 94);

private: Color backColor = Color::FromArgb(70, 82, 76);

private: Color triangleColor = Color::FromArgb(252, 245, 232);

private: System::Windows::Forms::PictureBox^ pictureBox1;

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

private: System::Windows::Forms::TrackBar^ trackBar1;

private: System::Windows::Forms::ColorDialog^ colorDialog1;

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

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

private: System::Windows::Forms::Button^ button4;

private: System::Windows::Forms::Button^ button5;

private: System::Windows::Forms::Button^ button6;

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

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

private: System::Windows::Forms::TrackBar^ trackBar2;

protected:

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

this->trackBar1 = (gcnew System::Windows::Forms::TrackBar());

this->colorDialog1 = (gcnew System::Windows::Forms::ColorDialog());

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

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

this->button4 = (gcnew System::Windows::Forms::Button());

this->button5 = (gcnew System::Windows::Forms::Button());

this->button6 = (gcnew System::Windows::Forms::Button());

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

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

this->trackBar2 = (gcnew System::Windows::Forms::TrackBar());

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->pictureBox1))->BeginInit();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->trackBar1))->BeginInit();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->trackBar2))->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->BackColor = System::Drawing::Color::White;

this->pictureBox1->Location = System::Drawing::Point(9, 10);

this->pictureBox1->Margin = System::Windows::Forms::Padding(2, 2, 2, 2);

this->pictureBox1->MaximumSize = System::Drawing::Size(1200, 1100);

this->pictureBox1->Name = L"pictureBox1";

this->pictureBox1->Size = System::Drawing::Size(393, 385);

this->pictureBox1->SizeMode = System::Windows::Forms::PictureBoxSizeMode::StretchImage;

this->pictureBox1->TabIndex = 0;

this->pictureBox1->TabStop = false;

//

// button1

//

this->button1->Anchor = static\_cast<System::Windows::Forms::AnchorStyles>((System::Windows::Forms::AnchorStyles::Bottom | System::Windows::Forms::AnchorStyles::Right));

this->button1->BackColor = System::Drawing::Color::Red;

this->button1->FlatStyle = System::Windows::Forms::FlatStyle::Flat;

this->button1->ForeColor = System::Drawing::SystemColors::ButtonHighlight;

this->button1->Location = System::Drawing::Point(454, 284);

this->button1->Margin = System::Windows::Forms::Padding(2, 2, 2, 2);

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

this->button1->Size = System::Drawing::Size(179, 39);

this->button1->TabIndex = 1;

this->button1->Text = L"Нарисовать";

this->button1->UseVisualStyleBackColor = false;

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

//

// trackBar1

//

this->trackBar1->Anchor = static\_cast<System::Windows::Forms::AnchorStyles>((System::Windows::Forms::AnchorStyles::Top | System::Windows::Forms::AnchorStyles::Right));

this->trackBar1->LargeChange = 1;

this->trackBar1->Location = System::Drawing::Point(471, 164);

this->trackBar1->Margin = System::Windows::Forms::Padding(2, 2, 2, 2);

this->trackBar1->Minimum = 1;

this->trackBar1->Name = L"trackBar1";

this->trackBar1->Size = System::Drawing::Size(136, 37);

this->trackBar1->TabIndex = 2;

this->trackBar1->Value = 1;

//

// button2

//

this->button2->Anchor = static\_cast<System::Windows::Forms::AnchorStyles>((System::Windows::Forms::AnchorStyles::Top | System::Windows::Forms::AnchorStyles::Right));

this->button2->BackColor = System::Drawing::Color::IndianRed;

this->button2->FlatStyle = System::Windows::Forms::FlatStyle::Flat;

this->button2->ForeColor = System::Drawing::SystemColors::ButtonHighlight;

this->button2->Location = System::Drawing::Point(454, 38);

this->button2->Margin = System::Windows::Forms::Padding(2, 2, 2, 2);

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

this->button2->Size = System::Drawing::Size(83, 38);

this->button2->TabIndex = 3;

this->button2->Text = L"Цвет 1";

this->button2->UseVisualStyleBackColor = false;

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

//

// button3

//

this->button3->Anchor = static\_cast<System::Windows::Forms::AnchorStyles>((System::Windows::Forms::AnchorStyles::Top | System::Windows::Forms::AnchorStyles::Right));

this->button3->BackColor = System::Drawing::Color::IndianRed;

this->button3->FlatStyle = System::Windows::Forms::FlatStyle::Flat;

this->button3->ForeColor = System::Drawing::SystemColors::ButtonHighlight;

this->button3->Location = System::Drawing::Point(550, 38);

this->button3->Margin = System::Windows::Forms::Padding(2, 2, 2, 2);

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

this->button3->Size = System::Drawing::Size(83, 38);

this->button3->TabIndex = 4;

this->button3->Text = L"Цвет 2";

this->button3->UseVisualStyleBackColor = false;

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

//

// button4

//

this->button4->Anchor = static\_cast<System::Windows::Forms::AnchorStyles>((System::Windows::Forms::AnchorStyles::Top | System::Windows::Forms::AnchorStyles::Right));

this->button4->BackColor = System::Drawing::Color::IndianRed;

this->button4->FlatStyle = System::Windows::Forms::FlatStyle::Flat;

this->button4->ForeColor = System::Drawing::SystemColors::ButtonHighlight;

this->button4->Location = System::Drawing::Point(454, 91);

this->button4->Margin = System::Windows::Forms::Padding(2, 2, 2, 2);

this->button4->Name = L"button4";

this->button4->Size = System::Drawing::Size(83, 38);

this->button4->TabIndex = 5;

this->button4->Text = L"Цвет 3";

this->button4->UseVisualStyleBackColor = false;

this->button4->Click += gcnew System::EventHandler(this, &MyForm::button4\_Click);

//

// button5

//

this->button5->Anchor = static\_cast<System::Windows::Forms::AnchorStyles>((System::Windows::Forms::AnchorStyles::Top | System::Windows::Forms::AnchorStyles::Right));

this->button5->BackColor = System::Drawing::Color::IndianRed;

this->button5->FlatStyle = System::Windows::Forms::FlatStyle::Flat;

this->button5->ForeColor = System::Drawing::SystemColors::ButtonHighlight;

this->button5->Location = System::Drawing::Point(550, 91);

this->button5->Margin = System::Windows::Forms::Padding(2, 2, 2, 2);

this->button5->Name = L"button5";

this->button5->Size = System::Drawing::Size(83, 38);

this->button5->TabIndex = 6;

this->button5->Text = L"Цвет 4";

this->button5->UseVisualStyleBackColor = false;

this->button5->Click += gcnew System::EventHandler(this, &MyForm::button5\_Click);

//

// button6

//

this->button6->Anchor = static\_cast<System::Windows::Forms::AnchorStyles>((System::Windows::Forms::AnchorStyles::Bottom | System::Windows::Forms::AnchorStyles::Right));

this->button6->BackColor = System::Drawing::Color::IndianRed;

this->button6->FlatStyle = System::Windows::Forms::FlatStyle::Flat;

this->button6->ForeColor = System::Drawing::SystemColors::ButtonHighlight;

this->button6->Location = System::Drawing::Point(454, 328);

this->button6->Margin = System::Windows::Forms::Padding(2, 2, 2, 2);

this->button6->Name = L"button6";

this->button6->Size = System::Drawing::Size(179, 38);

this->button6->TabIndex = 7;

this->button6->Text = L"Сохранить";

this->button6->UseVisualStyleBackColor = false;

this->button6->Click += gcnew System::EventHandler(this, &MyForm::button6\_Click);

//

// label1

//

this->label1->Anchor = static\_cast<System::Windows::Forms::AnchorStyles>((System::Windows::Forms::AnchorStyles::Top | System::Windows::Forms::AnchorStyles::Right));

this->label1->AutoSize = true;

this->label1->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 9.75F, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

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

this->label1->ForeColor = System::Drawing::SystemColors::Control;

this->label1->Location = System::Drawing::Point(468, 146);

this->label1->Margin = System::Windows::Forms::Padding(2, 0, 2, 0);

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

this->label1->Size = System::Drawing::Size(152, 16);

this->label1->TabIndex = 8;

this->label1->Text = L"Количество итераций";

//

// label2

//

this->label2->Anchor = static\_cast<System::Windows::Forms::AnchorStyles>((System::Windows::Forms::AnchorStyles::Top | System::Windows::Forms::AnchorStyles::Right));

this->label2->AutoSize = true;

this->label2->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 9.75F, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

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

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

this->label2->Location = System::Drawing::Point(479, 211);

this->label2->Margin = System::Windows::Forms::Padding(2, 0, 2, 0);

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

this->label2->Size = System::Drawing::Size(129, 16);

this->label2->TabIndex = 10;

this->label2->Text = L"Качетсво рендера";

//

// trackBar2

//

this->trackBar2->Anchor = static\_cast<System::Windows::Forms::AnchorStyles>((System::Windows::Forms::AnchorStyles::Top | System::Windows::Forms::AnchorStyles::Right));

this->trackBar2->LargeChange = 1;

this->trackBar2->Location = System::Drawing::Point(471, 234);

this->trackBar2->Margin = System::Windows::Forms::Padding(2, 2, 2, 2);

this->trackBar2->Minimum = 1;

this->trackBar2->Name = L"trackBar2";

this->trackBar2->Size = System::Drawing::Size(136, 37);

this->trackBar2->TabIndex = 9;

this->trackBar2->Value = 1;

//

// MyForm

//

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

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

this->BackColor = System::Drawing::SystemColors::GrayText;

this->ClientSize = System::Drawing::Size(670, 423);

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

this->Controls->Add(this->trackBar2);

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

this->Controls->Add(this->button6);

this->Controls->Add(this->button5);

this->Controls->Add(this->button4);

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

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

this->Controls->Add(this->trackBar1);

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

this->Controls->Add(this->pictureBox1);

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

this->Margin = System::Windows::Forms::Padding(2, 2, 2, 2);

this->MinimumSize = System::Drawing::Size(678, 407);

this->Name = L"MyForm";

this->ShowIcon = false;

this->Text = L"Орнамент";

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->pictureBox1))->EndInit();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->trackBar1))->EndInit();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->trackBar2))->EndInit();

this->ResumeLayout(false);

this->PerformLayout();

}

#pragma endregion

private: System::Void drawOrnament(Graphics^ gr, int x, int y, int width, int height) {

array<array<Point>^>^ triangles = {

{

Point(x, y + height /3),

Point(x, y),

Point(x + width/3, y)

},

{

Point(x, y + 2\*height / 3),

Point(x, y + height),

Point(x + width / 3, y + height)

},

{

Point(x + 2\*width / 3, y + height),

Point(x + width, y + height),

Point(x + width, y + 2\*height / 3)

},

{

Point(x + width, y + height / 3),

Point(x + width, y),

Point(x + 2\*width / 3, y)

},

{

Point(x + width / 3, y + 2\*height / 3),

Point(x + width / 2, y + height / 3),

Point(x + 2\*width / 3, y + 2 \* height / 3)

},

{

Point(x + width / 3, y + height / 3),

Point(x + width / 2, y + 2 \* height / 3),

Point(x + 2 \* width / 3, y + height / 3)

}

};

for (int i = 0; i < triangles->Length; i++) {

gr->FillPolygon(gcnew SolidBrush(triangleColor), triangles[i]);

};

gr->DrawArc(gcnew Pen(arcColor, width / 14), x - 2 \* width / 3 , y - 2 \* height / 3, width, height, 17, 56);

gr->DrawArc(gcnew Pen(arcColor, width / 14), x + 2 \* width / 3, y - 2 \* height / 3, width, height, 108, 55);

gr->DrawArc(gcnew Pen(arcColor, width / 14), x + 2 \* width / 3, y + 2 \* height / 3, width, height, 198, 55);

gr->DrawArc(gcnew Pen(arcColor, width / 14), x - 2 \* width / 3, y + 2 \* height / 3, width, height, 288, 54);

gr->FillEllipse(gcnew SolidBrush(ellipseColor), x + width / 8 \* 3, y + height / 8 \* 3, width / 4, height / 4);

}

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

Bitmap^ bitmap = gcnew Bitmap(500 \* trackBar2->Value, 500 \* trackBar2->Value);

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

gr->FillRectangle(gcnew SolidBrush(backColor), 0, 0, bitmap->Width, bitmap->Height);

for (int i = 0; i < trackBar1->Value; i++)

for (int j = 0; j < trackBar1->Value; j++)

drawOrnament(

gr,

bitmap->Width / trackBar1->Value \* i,

bitmap->Height / trackBar1->Value \* j,

bitmap->Width / trackBar1->Value,

bitmap->Height / trackBar1->Value

);

pictureBox1->Image = bitmap;

}

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

colorDialog1->ShowDialog();

ellipseColor = colorDialog1->Color;

}

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

colorDialog1->ShowDialog();

triangleColor = colorDialog1->Color;

}

private: System::Void button4\_Click(System::Object^ sender, System::EventArgs^ e) {

colorDialog1->ShowDialog();

arcColor = colorDialog1->Color;

}

private: System::Void button5\_Click(System::Object^ sender, System::EventArgs^ e) {

colorDialog1->ShowDialog();

backColor = colorDialog1->Color;

}

private: System::Void button6\_Click(System::Object^ sender, System::EventArgs^ e) {

if (pictureBox1->Image) {

String^ filename = "image.jpg";

pictureBox1->Image->Save(filename, System::Drawing::Imaging::ImageFormat::Jpeg);

MessageBox::Show("Орнамент сохранён под именем " + filename, "Файл сохранён");

}

else {

MessageBox::Show("Орнамент отсутствует", "Ошибка!");

}

}

};

}