Тема: Разработка приложений с графикой

Цель работы: изучение приемов создания приложений Windows Forms с использованием графических методов, построения графиков с помощью компонента отображения графической информации Chart

Задачи:

– изучение приемов создания приложений Windows Forms с графическими объектами;

– изучить возможности построения графиков с помощью компонента отображения графической информации Chart

– создание проектов с использованием графических методов.

Материально-техническое обеспечение:

Место проведения: Компьютерный класс.

Время на выполнение работы: 2 часа.

Оборудование: ПК

Средства обучения: операционная система, текстовый процессор MS Word, программные средства определенного вида

Исходные данные:

1. Конспект занятия.
2. Задание для практической работы.

Перечень справочной литературы:

1) Программирование на языке высокого уровня. Программирование на языке С++: учеб. пособие / Т. И. Немцова, С. Ю. Голова, А. И. Терентьев ; под ред. Л. Г. Гагариной. – М. : ИД «ФОРУМ» : ИНФРА-М, 2018. – 512 с. – (Среднее профессиональное образование).

Краткие теоретические сведения:

Работа с графикой GDI+

Для работы с графикой в среде .NET предназначен класс **Graphics** пространства имен **System.Drawing.** Для вывода графических примитивов (линий, геометрических фигур), текста, растровых изображений необходимо создать объект класса **Graphics**, например, методом **CreateGraphics**:

Graphics g = this.CreateGraphics();

После создания объекта типа **Graphics** можно применять его свойства и методы. Наиболее часто используются объекты и методы классов **Pen** (рисование линий и геометрических фигур), **Brush** (заполнение областей), **Font** (работа с текстом), **Color** (работа с цветом).

Для интерактивного управления свойствами графических объектов удобно использовать манипулятор мышь. События мыши **MouseDown, MouseUp, MouseMove** и другие работают в сочетании с делегатом **MouseEventHandler**.

Например, при регистрации события движения мыши по форме в методе **InitializeComponent()** (в файле Form1.Designer.cs) появляется строка:

MouseMove += new MouseEventHandler(Form1_MouseMove);

При этом в файле кода Form1.cs создается шаблон метода-обработчика, которому передаются два параметра: объект-источник события и объект класса **MouseEventArgs**, который содержит информацию о событии, например: **X** и **Y** – координаты указателя мыши; **Button** – нажатая кнопка (левая, правая); **Clicks** – количество нажатий и отпусканий кнопки мыши; **Delta** – счетчик (со знаком) щелчков поворота колесика.

Эту информацию можно использовать в обработчике, например, для вывода координат в заголовок формы:

```
public void Form1_MouseMove(object sender, MouseEventArgs e)
{       Text = string.Format("координаты: x={0}, y={1}", e.X, e.Y);
}
```

Для управления графическими объектами нередко используют и клавиатуру. События клавиатуры **KeyUp, KeyDown** работают в сочетании с делегатом **KeyEventHandler.**

Например, при регистрации события нажатия клавиши записывается строка

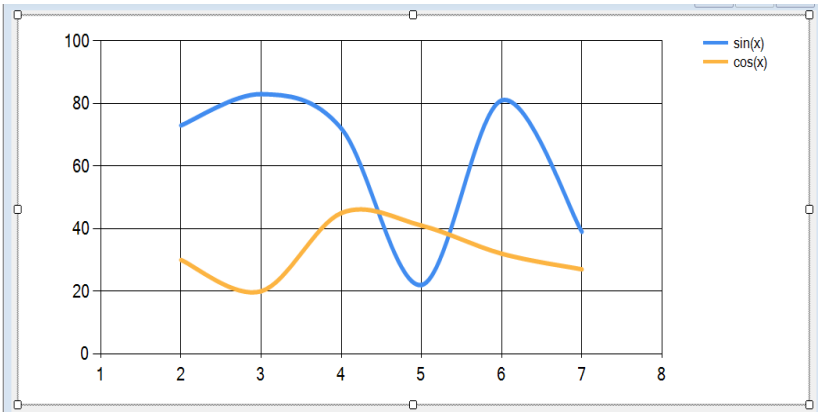
```
KeyDown += new KeyEventHandler(Form1_KeyUp);
```

Создается шаблон метода-обработчика, которому передаются: объект-источник события и объект класса **KeyEventArgs**, который содержит информацию о событии, например: **KeyCode** – код клавиши для событий KeyDown или KeyUp; **Modifiers** – какие модифицирующие клавиши (Shift, Alt, Control) были нажаты; **Handled** – было ли событие полностью обработано.
Эту информацию можно использовать в обработчике, например:

```
private void Form1_KeyDown (object sender, KeyEventArgs e)
{       MessageBox.Show(e.KeyCode.ToString(), "клавиша нажата!");
}
```

Использование элемента управления Chart

Обычно результаты математических расчетов представляются в виде графиков и диаграмм. Библиотека .NET Framework имеет мощный элемент управления **Chart** для отображения на экране графической информации.



Построение графика (диаграммы) производится после вычисления таблицы значений функции y=f(x) на интервале [Xmin, Xmax] с заданным шагом. Полученная таблица передается в специальный массив **Points** объекта **Series** компонента **Chart** с помощью метода **DataBindXY**. Компонент **Chart** относится к группе **Данные** панели элементов **Tools**. Компонент **Chart** осуществляет всю работу по отображению графиков: строит и размечает оси, рисует координатную сетку, подписывает название осей и самого графика, отображает переданную таблицу в виде всевозможных графиков или диаграмм. При необходимости компоненту **Chart** передаются данные о толщине, стиле и цвете линий, параметрах шрифта подписей, шагах разметки координатной сетки и другие настройки. В процессе работы программы изменение параметров возможно через обращение к соответствующим свойствам компонента **Chart**. Так, например, свойство **AxisX** содержит значение максимального предела нижней оси графика и при его изменении во время работы программы автоматически изменяется изображение графика.

Рекурсия

В C# допускается, чтобы метод вызывал самого себя. Этот процесс называется **рекурсией**, а метод, вызывающий самого себя, — **рекурсивным**. Вообще, рекурсия представляет собой процесс, в ходе которого нечто определяет само себя. В этом отношении она чем-то напоминает циклическое определение. Рекурсивный метод отличается главным образом тем, что он содержит оператор, в котором этот метод вызывает самого себя. Рекурсия является эффективным механизмом управления программой.

Как работает рекурсивный вызов метода?

Если метод вызывает сам себя, то в памяти происходят следующие процессы:

- в системном стеке выделяется память для новых локальных переменных и параметров;
- программный код метода выполняется сначала с новыми локальными переменными и параметрами. Эти локальные переменные и параметры получают новые начальные значения. Сам программный код метода не копируется;
- при возврате из рекурсивного метода (оператор return) происходит восстановление старых локальных переменных и параметров а также их значений в точке вызова этого метода.

Классическим примером рекурсии служит вычисление факториала числа

Ход работы:

Требования к содержанию отчета:

- Номер и название практической работы.
- Цель работы.

- По каждой заданию (задаче/примеру) экранные формы (при наличии) и листинг программного кода, показывающие порядок выполнения практической работы, и результаты, полученные в ходе её выполнения.
- Ответы на контрольные вопросы в тетради.

Порядок выполнения работы:

Все задания работы размещать в своей папке проектов в новой папке **Графика_Фамилия**

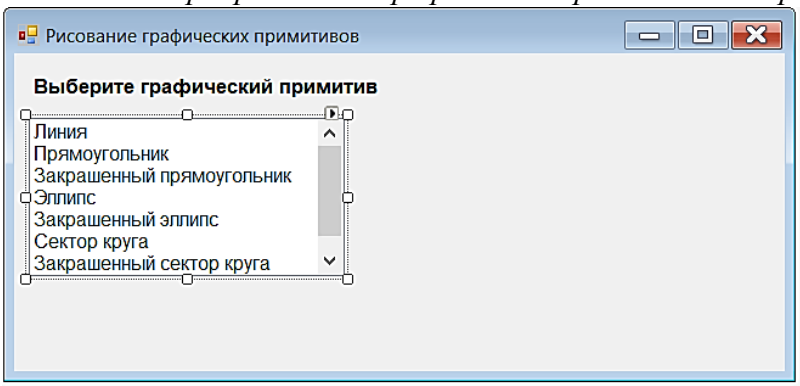
В начале каждого файла проекта установить комментарии: пр.р.№_____ (указать номер), свою Фамилию. Формулировку задания

Задание 1. Вывод графических примитивов на форму.

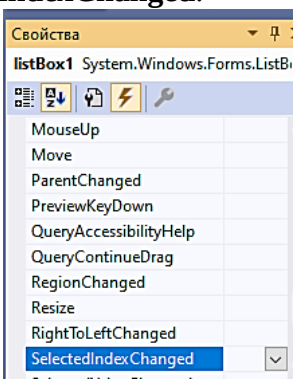
✓ Создайте новый проект **prGR_1_Фамилия** типа Windows Forms.

✓ Разместите на форме размером **650*350** поле выбора со списком **listBox1**. В пункте **Item** окна свойств введите список графических примитивов (рис. 1.1):

Рис. 1.1. Выбор и рисование графических примитивов на форме



✓ Зарегистрируйте событие выбора из списка **SelectedIndexChanged**.



✓ Проверьте подключение пространства имен **System.Drawing**.

✓ В шаблон обработчика введите код выбора и вывода примитивов на форму:

```
private void listBox1_SelectedIndexChanged(object sender, EventArgs e)
{
    Graphics g = this.CreateGraphics(); // создание графического объекта
    Pen pn = new Pen(Color.Red, 4); // создание пера
    Brush br = new SolidBrush(Color.Green); // создание кисти
    g.Clear(SystemColors.Control);  // очистка области цветом формы

    switch (listBox1.SelectedIndex) // выбор и рисование примитива
    {
        case 0: g.DrawLine(pn, 450, 50, 350, 180); break;
        case 1: g.DrawRectangle(pn, 350, 50, 200, 150); break;
        case 2: g.FillRectangle(br, 350, 50, 250, 150); break;
        case 3: g.DrawEllipse(pn, 350, 50, 250, 150); break;
        case 4: g.FillEllipse(br, 350, 50, 250, 150); break;
        case 5: g.DrawPie(pn, 300, 50, 300, 200, 180, 225); break;
        case 6: g.FillPie(br, 350, 50, 150, 150, 0, 45); break;
    }
}
```

✓ Протестируем программу. При необходимости откорректируйте код.

Задание 2. Построение графиков с помощью компонента отображения графической информации **Chart.**

Постановка базовой задачи: создать программу, отображающую графики функций **sin(x)** и **cos(x)** на интервале **[Xmin, Xmax].** Предусмотреть возможность изменения разметки координатных осей, а также шага построения таблицы.

1) Создайте новый проект **prGR_2_Фамилия** типа Windows Forms. Измените название формы.
2) Разместите на форме размером **800*500** компонент **Chart**

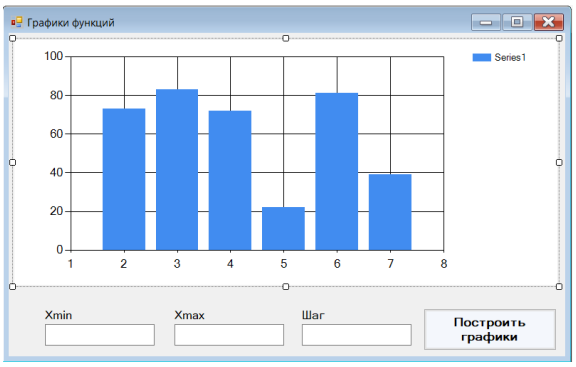
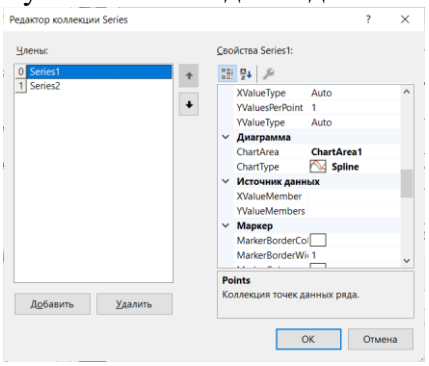


Chart по умолчанию настроен на отображение столбчатых диаграмм.

3) Далее выполните необходимые настройки.

✓ список графиков хранится в свойстве **Series**

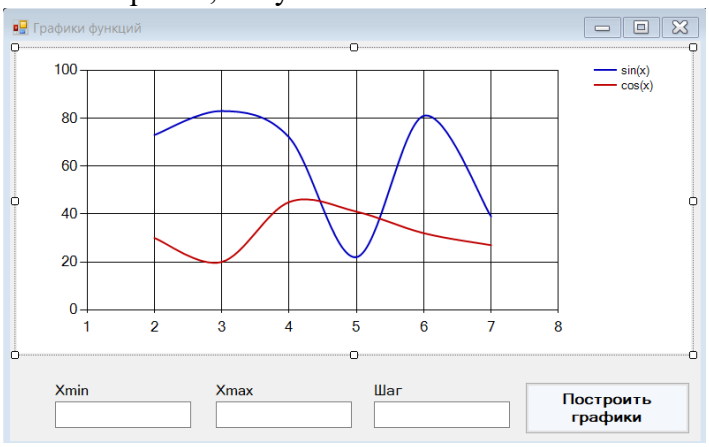
✓ на одном поле требуется вывести два отдельных графика функций, нужно добавить еще один элемент, нажав на «**Добавить**»



✓ Для обоих элементов выполните следующие настройки в правой части окна:

- тип диаграммы **ChartType** заменить на **Spline**.
- изменить подписи к графикам с абстрактных **Series1** и **Series2** на **sin(x)** и **cos(x)** – за это отвечает свойство **LegendText**.
- с помощью свойства **BorderWidth** можно увеличить толщину линию

Таким образом, получим:



4) Реализуйте обработчик события **Click** нажатия на кнопку:

✓ Прежде всего, следует определить в коде класса все необходимые переменные и константы.

Конечно, можно обойтись и без этого, вставляя значения в виде чисел прямо в формулы, но это, во-первых, снизит читабельность кода программы, а во-вторых, значительно усложнит изменение каких-либо параметров программы, например, интервала построения графика

```
public partial class Form1 : Form
{

    private double XMin;    // Левая граница графика
    private double XMax;    // Правая граница графика
    private double Step;    // Шаг графика

    // Массив значений X - общий для обоих графиков
    private double[] x;
    // Два массива Y - по одному для каждого графика
    private double[] y1;
    private double[] y2;
```

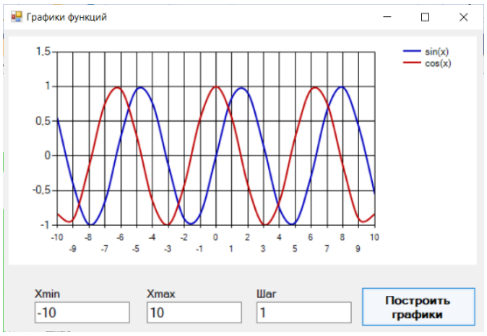
✓ Далее следует определить метод, который будет рассчитывать количество шагов и вычислять значения функций в каждой точке, внося вычисленные значения в массивы x, y1 и y2:

```
// Расчёт значений графика
ссылка: 1
private void CalcFunction()
{
    // Количество точек графика
    int count = (int)Math.Ceiling((XMax - XMin) / Step) + 1;
    // Создаём массивы нужных размеров
    x = new double[count];
    y1 = new double[count];
    y2 = new double[count];
    // Расчитываем точки для графиков функции
    for (int i = 0; i < count; i++)
    {
        // Вычисляем значение X
        x[i] = XMin + Step * i;
        // Вычисляем значение функций в точке X
        y1[i] = Math.Sin(x[i]);
        y2[i] = Math.Cos(x[i]);
    }
}
```

✓ После расчёта значений нужно отобразить графики на форме с помощью элемента **Chart** и настроить внешний вид осей. Реализуйте обработчик события **Click** нажатия на кнопку:

```
private void button1_Click(object sender, EventArgs e)
{
    //считываем с формы требуемые значения
    XMin = double.Parse(textBox1.Text);
    XMax = double.Parse(textBox2.Text);
    Step = double.Parse(textBox3.Text);
    CalcFunction();
    //настриваем оси графика
    chart1.ChartAreas[0].AxisX.Minimum = XMin;
    chart1.ChartAreas[0].AxisX.Maximum = XMax;

    //определяем шаг сетки
    chart1.ChartAreas[0].AxisX.MajorGrid.Interval = Step;
    //добавляем вычисляемы значения в графики
    chart1.Series[0].Points.DataBindXY(x, y1);
    chart1.Series[1].Points.DataBindXY(x, y2);
}
```

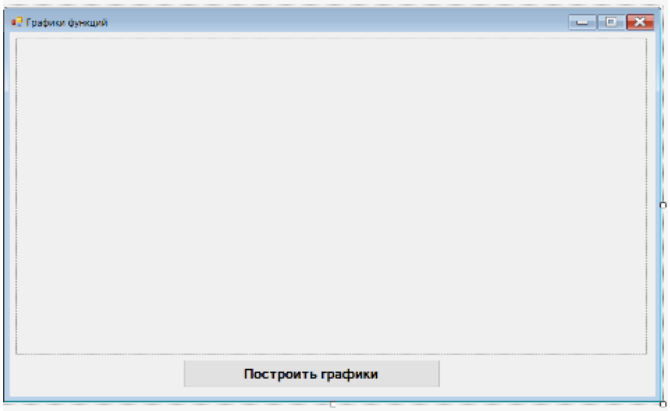


✓ Протестируйте приложение с корректными исходными данными. При необходимости измените свойства компонентов и код.

Задание 3. Построение графиков функций с использованием Graphics.

Необходимо построить на декартовой системе координат графики трех функций:

1) **Синусоида (y = sin(x))**
2) **Парабола (y = x^2)**
3) **Гипербола (y = k/x,** k - коэффициент)



Для этого:

✓ Создайте новый проект **prGR_3_Фамилия** типа Windows Forms.

✓ Разместите на форме размером **1000*600** кнопку **Button**, а также элемент **PictureBox**.

✓ В шаблон обработчика события **button1_Click** нажатия кнопки введите коды:

```
private void button1_Click(object sender, EventArgs e)
{
    // холст для рисования
    Bitmap canvas = new Bitmap(pictureBox1.Width, pictureBox1.Height);
    Graphics graphCanvas = Graphics.FromImage(canvas);

    // центр графика точка 0,0
    PointF centerDrawing = new PointF(canvas.Width / 2, canvas.Height / 2);

    // границы расчета функции
    int minx = -10, maxx = 10;

    // шаг масштабирования на графике
    float stepX = canvas.Width / Math.Abs(maxx - minx);
    float stepY = canvas.Height / Math.Abs(maxx - minx);

    // рисуем оси графика
    DrawAxis(graphCanvas, Color.Black, centerDrawing, canvas.Size, minx, maxx, stepX, stepY);

    // генерим нужные функции
    List<PointF> parabolic = CreateParabolic(1, 0, -5, minx, maxx, 0.01f);
    List<PointF> sinus = CreateSinus(5, 5, minx, maxx, 0.001f);
    List<PointF> hyper = CreateHyper(5, minx, maxx, 0.001f);

    // рисуем нужные функции
    DrawCurve(graphCanvas, Brushes.Magenta, parabolic, centerDrawing, stepX, stepY);
    DrawCurve(graphCanvas, Brushes.Cyan, sinus, centerDrawing, stepX, stepY);
    DrawCurve(graphCanvas, Brushes.CornflowerBlue, hyper, centerDrawing, stepX, stepY);

    // показываем пользователю
    pictureBox1.Image = canvas;
}
```

✓ Метод построения осей:

```
void DrawAxis(Graphics canvas, Color Color, PointF Center, Size drawingField, float min, float max, float stepX, float stepY)
{
    // Оси
    Pen axisPen = new Pen(Color, 2);
    canvas.DrawLine(axisPen, new PointF(0, drawingField.Height / 2.0f), new PointF(drawingField.Width, drawingField.Height / 2.0f));
    canvas.DrawLine(axisPen, new PointF(drawingField.Width / 2.0f, 0), new PointF(drawingField.Width / 2.0f, drawingField.Height));

    for (float x = min; x < max; x += 0.5f)
    {
        float gx = x * stepX + drawingField.Width / 2.0f;
        float gy = x * stepY + drawingField.Height / 2.0f;
        canvas.DrawLine(axisPen, new PointF(gx, (drawingField.Height / 2.0f) + 2), new PointF(gx, (drawingField.Height / 2.0f) - 2.0f));
        canvas.DrawLine(axisPen, new PointF(drawingField.Width / 2.0f - 2.0f, gy), new PointF(drawingField.Width / 2.0f + 2.0f, gy));
    }
}
```

✓ Метод построения графика по точкам:

```
/// Рисует график по точкам на указанном холсте
Ссылок: 3
void DrawCurve(Graphics canvas, Brush Color, List<PointF> dataPoint, PointF Center, float stepX, float stepY)
{
    dataPoint.ForEach(new Action<PointF>((o) =>
    {
        canvas.FillEllipse(Color, new RectangleF(new PointF((o.X * stepX + Center.X) - 1, (o.Y * stepY + Center.Y) - 1), new SizeF(2, 2)));
    }));
}

Ссылок: 3
List<PointF> CalcFunc(Func<float, float> func, float min, float max, float step)
{
    List<PointF> _result = new List<PointF>();
    for (float x = min; x < max; x += step)
    {
        _result.Add(new PointF(x, func(x)));
    }
    return _result;
}
```

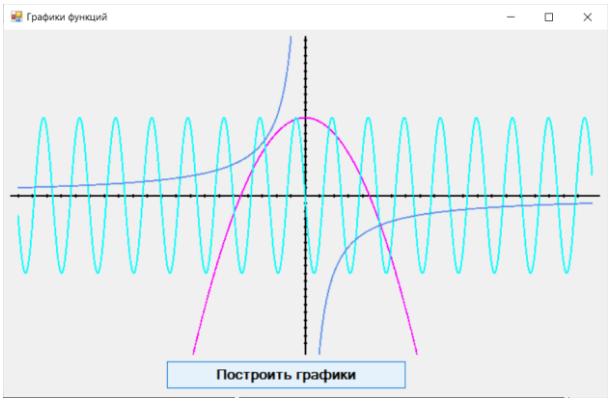
✓ Методы формирования точек для каждого графика:

```
/// Создает точки функции параболы
ссылка: 1
List<PointF> CreateParabolic(float a, float b, float c, float min, float max, float step)
{
    return CalcFunc(new Func<float, float>((x) => { return (x * x) * a + (b * x) + c; }), min, max, step);
}

/// Создает точки функции синуса
ссылка: 1
List<PointF> CreateSinus(float a, float b, float min, float max, float step)
{
    return CalcFunc(new Func<float, float>((x) => { return (float)(a * Math.Sin(b * x)); }), min, max, step);
}

/// Создает точки функции гиперболы
ссылка: 1
List<PointF> CreateHyper(float k, float min, float max, float step)
{
    return CalcFunc(new Func<float, float>((x) => { return (float)(k / x); }), min, max, step);
}
```

✓ Протестируйте программу.



Задание 4. Создать приложение с использованием рекурсивных функций, позволяющее построить *Дерево Пифагора*.

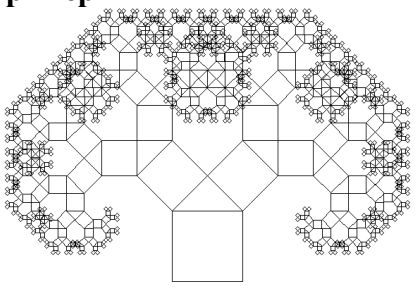
Дерево Пифагора — разновидность фрактала, основанная на фигуре, известной как «Пифагоровы штаны». Одним из свойств дерева Пифагора является то, что если площадь первого квадрата равна единице, то на каждом уровне сумма площадей квадратов тоже будет равна единице.

Если в классическом дереве Пифагора угол равен 45 градусам, то также можно построить и обобщённое дерево Пифагора при использовании других углов. Такое дерево часто называют обдуваемое ветром дерево Пифагора. Если изображать только отрезки, соединяющие каким-либо образом выбранные «центры» треугольников, то получается обнаженное дерево Пифагора.

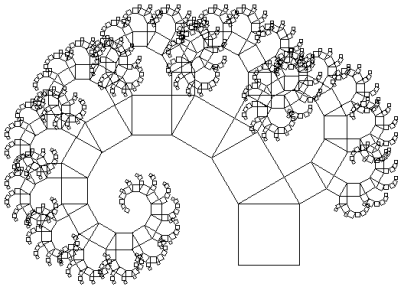
Алгоритм:

1) Строим вертикальный отрезок
2) Из верхнего конца этого отрезка рекурсивно строим еще 2 отрезка под определенными углами
3) Вызываем функцию построения двух последующих отрезков для каждой ветви дерева

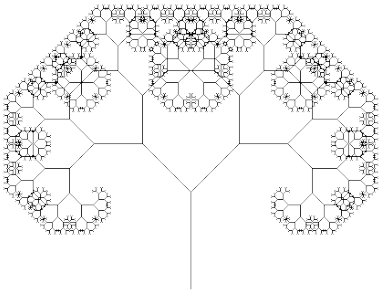
Примеры



Классическое дерево Пифагора



Обдуваемое ветром дерево Пифагора



Обнаженное дерево Пифагора

1) Создайте новый проект **prGR_4_Фамилия** типа Windows Forms. Измените название формы.
2) Создайте программный код решения задачи:

✓ определить в коде класса все необходимые переменные и константы.

```
public partial class Form1 : Form
{
    public Graphics g; //Графика
    public Bitmap map; // объект для работы с изображениями, определяемыми данными пикселей
    public Pen p; //Ручка
    public double angle = Math.PI / 2; //Угол поворота на 90 градусов
    public double ang1 = Math.PI / 4;  //Угол поворота на 45 градусов
    public double ang2 = Math.PI / 6;  //Угол поворота на 30 градусов
```

✓ Реализуйте обработчик события **Load** при загрузке формы:

```
private void Form1_Load(object sender, EventArgs e)
{
    map = new Bitmap(pictureBox1.Width, pictureBox1.Height);//Подключаем Bitmap
    g = Graphics.FromImage(map); //Подключаем графику
    g.SmoothingMode = System.Drawing.Drawing2D.SmoothingMode.AntiAlias;//Включаем сглаживание
    p = new Pen(Color.Green);   //Зеленая ручка

    //Вызов рекурсивной функции отрисовки дерева
    DrawTree(300, 450, 200, angle);

    //Переносим картинку из Bitmap на picturebox
    pictureBox1.BackgroundImage = map;
}

//Рекурсивная функция отрисовки дерева
//x и y - координаты родительской вершины
//a - параметр, который фиксирует количество итераций в рекурсии
//angle - угол поворота на каждой итерации
Ссылок: 3
public int DrawTree(double x, double y, double a, double angle)
{

    if (a > 2)
    {
        a *= 0.7; //Меняем параметр a

        //Считаем координаты для вершины-ребенка
        double xnew = Math.Round(x + a * Math.Cos(angle)),
               ynew = Math.Round(y - a * Math.Sin(angle));

        //рисуем линию между вершинами
        g.DrawLine(p, (float)x, (float)y, (float)xnew, (float)ynew);

        //Переприсваеваем координаты
        x = xnew;
        y = ynew;

        //Вызываем рекурсивную функцию для левого и правого ребенка
        DrawTree(x, y, a, angle + ang1);
        DrawTree(x, y, a, angle - ang2);
    }
    return 0;
}
```

✓ Протестируйте приложение с корректными исходными данными. При необходимости измените свойства компонентов и код.

Задания для самостоятельной работы

Создать проект, содержащий форму. Расположить на форме 2 кнопки, при нажатии на которых

отображаются **Герб** и **Флаг** государства. Флаг строить из прямоугольных элементов соответствующего цвета. Флаг и Герб отображаются на разных формах.
Номер задания определяется по номеру студента в списке группы.

1. Англия	2. Армения
3. Белоруссия	4. Болгария
5. Германия	6. Греция
7. Грузия	8. Дания
9. Индия	10. Испания
11. Италия	12. Казахстан
13. Китай	14. Латвия
15. Литва	16. Нидерланды
17. Норвегия	18. Польша
19. Россия	20. Румыния
21. Таджикистан	22. Туркменистан
23. Узбекистан	24. Украина
25. Финляндия	26. Франция
27. Чехия	28. Швейцария
29. Швеция	30. Япония

Контрольные вопросы:

1) Какие способы построения графиков функций существует в C#.
2) Компонент Chart: назначение, в каких случаях используется.
3) Какое свойство элемента Chart отвечает за толщину линий?
4) Какое свойство элемента Chart отвечает за подписи на легенде графиков?
5) Какое свойство элемента Chart отвечает за максимальный предел нижней оси графика?