Графика.

Построение графиков функций.

Общий алгоритм рисования с помощью программных средств библиотеки .NET Framework:

Подготовка:

1. вынести на форму компонент, который может отображать изображение (мы будем работать с компонентом *PictureBox*);
2. создать через *gcnew* объект, представляющий изображение (мы будем использовать объекты класса *Bitmap*), и загрузить ссылку на этот объект в компонент из п. 1 (*PictureBox*);
3. создать объекты «кисть» и «перо» (мы будем использовать экземпляры классов *SolidBrush* и *Pen*) и сохранить ссылки на них в соответствующих переменных;
4. из объекта-изображения (*Bitmap*, см. п. 2) создать объект класса *Graphics* и сохранить ссылку на него в соответствующую переменную;

Рисование:

5. рисовать геометрические примитивы с использованием функций созданного в п. 4 экземпляра класса *Graphics*, передавая в качестве параметров созданные в п. 3 «кисть» (где нужно) и «перо» (где нужно);
6. после того, как изображение сформировано (см. п. 5), вызвать принудительную перерисовку компонента отображения изображения (*PictureBox*, см. п. 1).

Далее по порядку.

1. Компонент PictureBox

Компонент *PictureBox* находится на Панели Элементов, во вкладке «Стандартные». Этот компонент может хранить ссылку на изображение и выводить это изображение на экран. В данном примере на форму был вынесен компонент класса PictureBox, и его свойство Name было изменено ***pbPlot***.

2. Создание объекта «изображение»

После вынесения компонента pbPlot класса *PictureBox* на форму необходимо создать объект «изображение» и сохранить ссылку на это

изображение в свойство Image компонента pbPlot. Правильнее всего это делать в конструкторе класса MyForm (функцию-конструктор формы можно найти в самом верху класса), как показано на рисунке:

```
namespace LabTest {

    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();

            pbPlot->Image = gcnew Bitmap(pbPlot->Width, pbPlot->Height);
        }
```

На этом рисунке pbPlot – переменная, соответствующая компоненту PictureBox, pbPlot->Image – свойство этого компонента, отвечающее за то, какое изображение он должен выводить на экран. *Bitmap* – один из классов, экземпляры которых могут хранить в себе изображения.

На рисунке выше при помощи *gcnew* создаётся новый экземпляр класса *Bitmap* (проще говоря, создаётся новое «изображение») шириной pbPlot->Width и высотой pbPlot->Height, то есть, совпадающее по размерам с pbPlot, и ссылка на это «изображение» (пока пустое) сохраняется в поле Image объекта pbPlot, чтобы pbPlot в дальнейшем выводил на экран именно это изображение.

3. Объекты «кисть» и «перо»

За некоторые параметры рисования отвечают объекты «кисть» (экземпляры классов-потомков класса *Brush*) и «перо» (экземпляры класса *Pen*). При этом «кисти» отвечают за то, как рисовать сплошные области, а «перья» - за то, как рисовать линии и контуры.

Сначала нужно объявить переменные для хранения ссылок на «кисть» и «перья». Если необходимо, чтобы эти переменные были видны всем функциям класса MyForm, то, как и раньше, их нужно объявить где-то внутри класса MyForm, но не внутри функций. Например, можно их объявить там, где объявляли другие переменные – после #pragma endregion:

```
#pragma endregion

        Brush^ br;
        Pen ^pn_line,
            ^pn_axes;
```

Обратите внимание, символ «^» ставится не после типа, а перед каждой переменной, которая к нему относится, иначе этот символ будет относиться только к первой переменной из перечисленных через запятую. Это касается всех ссылок и указателей.

В данном примере переменная br будет (в дальнейшем) хранить ссылку на «кисть», отвечающую за очистку поля рисования, а переменные pn_line и pn_axes – ссылки на «перья», отвечающие за рисование кривой графика функции и координатных осей соответственно.

Теперь, имея переменные, необходимо создать сами «кисть» и «перья», и сохранить в объявленные переменные ссылки на них. Это снова лучше сделать в конструкторе класса MyForm:

```
namespace LabTest {

    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();

            pbPlot->Image = gcnew Bitmap(pbPlot->Width, pbPlot->Height);
            br = gcnew SolidBrush(Color::White);
            pn_line = gcnew Pen(Color::Blue, 3);
            pn_axes = gcnew Pen(Color::Black, 3);
        }
```

В качестве кисти br здесь создаётся экземпляр класса SolidBrush. Такие «кисти» закрашивают сплошные области сплошным цветом (есть другие варианты «кистей»). Конструктор SolidBrush принимает один аргумент – цвет кисти.

Класс *Color* содержит множество предопределённых цветов, список которых можно увидеть, если написать *Color::* и немного подождать. Кроме того, есть функция *Color::FromArgb*, позволяющая сформировать цвет из значений байтов целого числа или из отдельных цветовых компонент.

В конструкторе класса *Pen* (см. рисунок выше) в качестве второго параметра передаётся целое число, задающее толщину пера в пикселях.

4. Создание экземпляра класса Graphics из изображения

Для рисования геометрических примитивов используется экземпляр класса *Graphics*, однако, чтобы нарисованные примитивы отображались на определённом изображении, экземпляр класса *Graphics* нужно создать именно из этого изображения. Это делается при помощи функции *Graphics::FromImage*(*изображение*). Например, создание экземпляра класса *Graphics* из изображения, которое было создано в пункте 2, и ссылка на которое была помещена в свойство Image объекта pbPlot, будет выглядеть так:

```
Graphics^ gr = Graphics::FromImage(pbPlot->Image);
```

После этого переменная *gr*, хранящая экземпляр «графики», созданной из «изображения», «хранящегося» в pbPlot, может использоваться для рисования на этом изображении.

Обратите внимание, что на рисунке переменная *gr* объявлена в том же месте, где она инициализируется (т.е. где ей присваивается значение). Это можно сделать внутри собственной функции рисования или внутри обработчика события. Однако возможен вариант, когда переменная *gr* объявляется отдельно, вне функций (например, после #pragma endregion), а инициализируется в конструкторе MyForm (там, где инициализировались pbPlot, br, pn_line и pn_axes).

Построение графика функции

Для примера рисования далее рассматривается построение графика произвольной вещественной функции одного вещественного аргумента (т.е. функций y = f(x), где y и x – вещественные числа), не имеющей разрывов в области построения.

В примере ниже предполагается, что п. 1-3 уже выполнены, поэтому повторяться они не будут.

Задача построения графика разбита на несколько подзадач, каждая из которых реализована в виде отдельной функции. Эти функции можно писать где-нибудь в классе MyForm (только не внутри других функций). Например, после переменных, объявленных после #pragma endregion.

Очистка области построения

Первая функция выполняет «очистку» области построения, закрашивая её сплошным цветом:

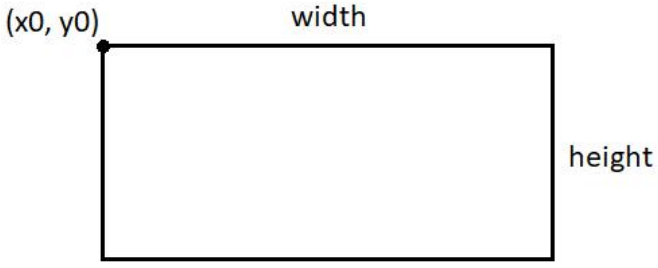
```
void clear(Image^ img, Brush^ br) {
    Graphics^ gr = Graphics::FromImage(img);
    gr->FillRectangle(br, 0, 0, img->Width, img->Height);
}
```

Функция clear, когда её вызовут, принимает два параметра. Первый параметр – ссылка img на экземпляр класса *Image* (“изображение”) или его потомка (например, класс *Bitmap* является потомком класса *Image*), который необходимо очистить. *Благодаря тому, что функция clear принимает на вход ссылку на изображение, которое нужно очистить, можно использовать одну эту функцию для очистки разных изображений (например, если в программе несколько PictureBox’ов, в каждом из которых своё изображение).*

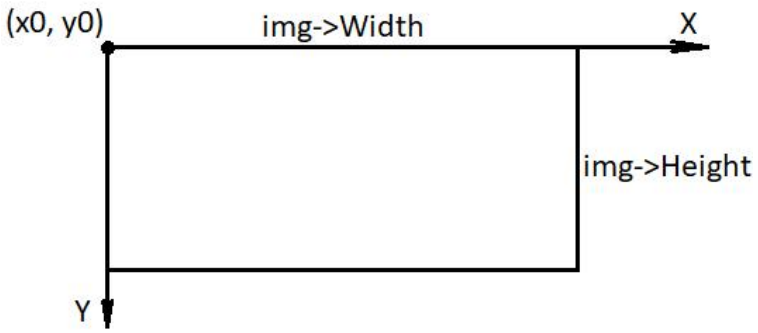
Второй параметр – «кисть», которой будет выполнена закраска области изображения.

Первое, что делает функция clear – создаёт экземпляр «графики» из изображения img и сохраняет ссылку на этот экземпляр в переменную gr (см. п. 4), чтобы объекты, рисуемые с помощью gr, отображались на img.

Второе, что делает функция clear – вызывает функцию gr->FillRectangle. Функция FillRectangle рисует закрашенный прямоугольник (только сплошную область, без контуров) со сторонами, параллельными осям координат. Такой прямоугольник задаётся четырьмя параметрами – координатами своего левого верхнего угла, а также шириной и высотой:



Однако нужно учесть, что ось Y в экранных координатах идёт **сверху вниз**, поэтому, чтобы нарисовать закрашенный прямоугольник размером со всё изображение img, и тем самым «стереть» всё, что было изображено ранее, необходимо указать левый верхний угол прямоугольника в точке (0, 0), а ширину и высоту – равными ширине и высоте изображения img:



Функция FillRectangle принимает 5 аргументов. Первый аргумент – кисть, с помощью которой будет закрашиваться прямоугольная область. Затем идут две координаты левого верхнего угла прямоугольника, высота прямоугольника и его ширина.

В классе Graphics есть множество функций, названия которых начинаются со слова “Fill…” (FillRectangle, FillEllipse, FillPie, …), и множество функций, чьи названия начинаются со слова “Draw…” (DrawRectangle, DrawEllipse, DrawPie, DrawLine, …). Функции, начинающиеся с “Fill…”, принимают в качестве параметра кисть и выводят закрашенные области (без контуров). Функции, начинающиеся с “Draw…” – наоборот, принимают в качестве параметра перо и выводят линии или контуры (без закрашивания областей). Если нужно изобразить закрашенную область с контуром, то нужно использовать и функцию первого вида, и функцию второго вида.

Обратите внимание, изображение по сути – это просто двумерный массив пикселей. Примитивы выводятся на изображение в том порядке, в котором вызываются функции их рисования. Поэтому примитивы, изображенные позже, могут частично или полностью «перекрыть» собой примитивы, нарисованные раньше (на самом деле – заменить своими пикселями пиксели более ранних примитивов).

Построение осей координат

Вторая функция при её вызове построит оси координат:

```
void plot_axes(Image^ img, Point origin, Pen^ pn) {
    Graphics^ gr = Graphics::FromImage(img);
    gr->DrawLine(pn, 0, origin.Y, img->Width - 1, origin.Y);
    gr->DrawLine(pn, origin.X, 0, origin.X, img->Height);
}
```

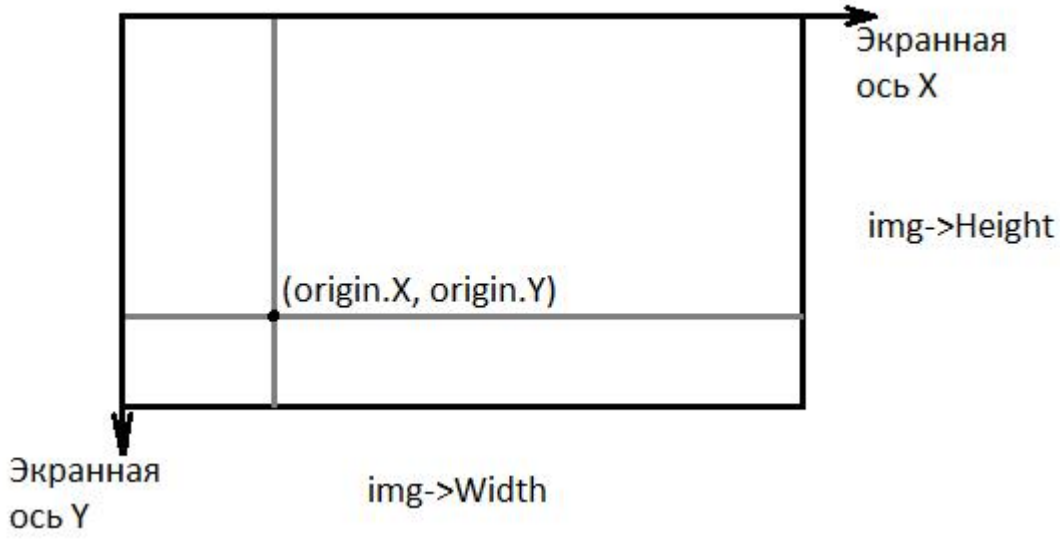
Функция plot_axes принимает на вход три параметра. Первый параметр, как и прежде – ссылка на то изображение, на котором нужно нарисовать оси координат.

Второй параметр – origin – координаты пикселя на изображении, в котором должно оказаться начало координат (т.е. точка пересечения осей). Тип данных Point – это класс (ближе к структуре), который содержит **целочисленные** координаты X и Y и может использоваться для хранения координат точки на плоскости.

Обратите внимание, доступ к полям origin осуществляется не через «->», а через «.» (origin.X, origin.Y), как, например, при работе со структурами. Дело в том, что origin является не ссылкой на экземпляр класса, а самим экземпляром класса (тип origin – не Point^, а просто Point). А оператор «->» используется только для доступа к полям экземпляров структур и классов *через указатели/ссылки CLR*.

Третий параметр – ссылка на перо, которым будут рисоваться оси координат. Поскольку оси координат – это линии, а не сплошные области, для их рисования используется перо, а не кисть.

Первым делом, функция, как и предыдущая, создаёт объект «графика», после чего с помощью его функции DrawLine проводит горизонтальную и вертикальную линии от края до края изображения, проходящие через origin.



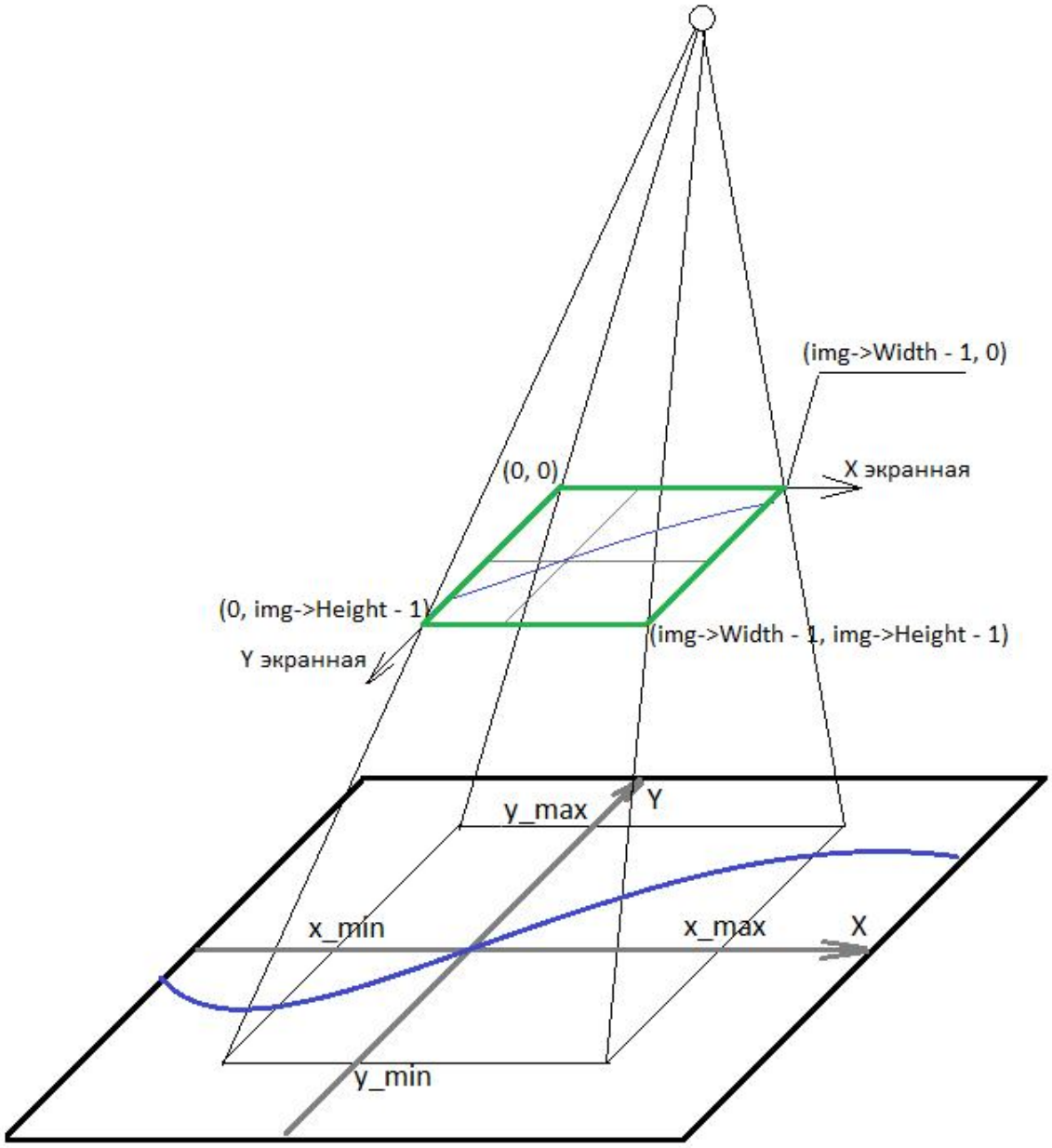
На рисунке чёрным прямоугольником схематично изображено пространство изображения img ширины img->Width и высоты img->Height и «экранные координаты» - система координат изображения img.

Серыми линиями внутри прямоугольника изображены (для примера) координатные оси, которые могут быть построены функцией plot_axes (см. предыдущий рисунок).

Функция DrawLine строит отрезок прямой. Она принимает на вход перо, которым необходимо начертить линию, координаты начала отрезка и координаты его конца.

Вычисление масштаба

Проблема: изображение img, в котором необходимо построить график функции, имеет конечный фиксированный размер img->Width на img->Height пикселей. При этом, в зависимости от нужд пользователя, в этот фиксированный размер может понадобиться отобразить различные участки бесконечной координатной плоскости. Можно **представлять** себе изображение img как «окно», через которое пользователь смотрит на некую «истинную» координатную плоскость:



На этом рисунке большой чёрной рамкой схематично изображена некая «истинная» координатная плоскость, область которой (маленькая черная рамка внутри) необходимо отобразить на изображении img (зелёная рамка). Размер и положение отображаемой области (маленькая чёрная рамка) можно задать её границами по осям: x_min, x_max, y_min и y_max.

Очевидно, что чем больший участок «реальной» координатной плоскости необходимо отобразить на изображении img фиксированного размера, тем большему участку координатной плоскости будет соответствовать каждый пиксель img. То есть, в зависимости от размера отображаемого участка координатной плоскости и размера изображения img, необходим разный **масштаб**.

Функция вычисления масштаба в этом случае может выглядеть следующим образом:

```
void scale(double real_width, double real_height,
           double screen_width, double screen_height,
           double& scale_x, double& scale_y) {
    scale_x = real_width / screen_width;
    scale_y = real_height / screen_height;
}
```

Функция принимает «истинные» ширину и высоту отображаемой области (real_width и real_height; например это могут быть x_max – x_min и y_max – y_min соответственно) и ширину и высоту изображения в пикселях (screen_width и screen_height), а так же две ссылки scale_x и scale_y, в которые будет записан результат вычисления масштаба (*Примечание: про передачу параметров по значению и по ссылке см. семинары прошлого семестра*).

От того, делится ли в функции «истинный» размер на «экранный», как на рисунке, или же наоборот, будет зависеть, в каких случаях для перевода из одного масштаба в другой нужно будет на полученные коэффициенты масштаба умножать, а в каких – делить.

Определение экранных координат центра «истинных» координат

Чтобы определить, в каком пикселе на изображении должны пересечься оси координат, можно написать такую функцию:

```
Point get_origin_pixel(double x_min, double x_max, double y_min, double y_max, Image^ img) {
    double x_scale, y_scale;
    scale(x_max - x_min, y_max - y_min, img->Width, img->Height, x_scale, y_scale);
    return Point(-x_min / x_scale, y_max / y_scale);
}
```

Функция принимает на вход «истинные» границы отображаемой области координатной плоскости и ссылку на изображение. Функция вычисляет масштаб (с помощью предыдущей функции), после чего вычисляет координаты пикселя, в котором должно оказаться «истинное» начало координат.

Искомый пиксель находится на расстоянии (0 - x_min) от левого края изображения и на расстоянии (y_max - 0) от верхнего края (необходимо помнить, что ось Y на изображении перевёрнута). При этом нужно учитывать масштаб, поэтому эти расстояния делятся на соответствующие коэффициенты.

Функция построения графика функции

Функция построения самого графика произвольной вещественной функции вещественного аргумента, не имеющей разрывов в области построения, может выглядеть таким образом:

```
void plot(double (*f)(double),
          double x_min,
          double x_max,
          double y_min,
          double y_max,
          Image^ img,
          Pen^ pn) {

    double x_scale, y_scale;
    scale(x_max - x_min, y_max - y_min, img->Width, img->Height, x_scale, y_scale);

    Graphics^ gr = Graphics::FromImage(img);
    int y_pix = (y_max - f(x_min)) / y_scale;
    for (int x_pix1 = 1; x_pix1 < img->Width; x_pix1++) {
        double x = x_min + x_pix1 * x_scale,
               y = f(x);

        int y_pix1 = (y_max - y) / y_scale;
        gr->DrawLine(pn, x_pix1 - 1, y_pix, x_pix1, y_pix1);

        y_pix = y_pix1;
    }
}
```

Функция plot принимает на вход в качестве первого аргумента **указатель на функцию**, график которой необходимо построить. Запись *double (*f)(double)* означает, что первый параметр будет называться «f», и что он примет указатель на функцию, принимающую один аргумент типа double и возвращающую результат типа double. Под этот прототип подходит любая

вещественная функция одного вещественного аргумента. *Подробнее об указателях на функции смотрите в материалах семинаров прошлого семестра*.

Следующие четыре аргумента функции plot – границы «истинной» области построения. Затем идёт ссылка изображение, на котором необходимо построить график, и ссылка на перо, которым необходимо рисовать линию графика.

Первым делом функция вычисляет коэффициенты масштаба по осям с помощью написанной выше функции scale, затем создаёт экземпляр «Графики» из изображения img для рисования на этом изображении.

Сам график функции строится последовательностью отрезков, и алгоритм построения графика может выглядеть так:

1. Вычислить **экранную** y-координату начальной точки отрезка (y_pix).
2. Для всех **экранных** x-координат конечных точек отрезков (x_pix1), начиная с 1:
   2.1. Вычислить **«истинную»** x-координату конечной точки отрезка.
   2.2. Вычислить **«истинную»** y-координату конечной точки отрезка.
   2.3. Вычислить **экранную** y-координату конечной точки отрезка (y_pix1).
   2.4. Провести отрезок из **начальной** точки (x_pix1 – 1, y_pix) в **конечную** точку (x_pix1, y_pix1).
   2.5. Сделать **экранную** y-координату конечной точки отрезка новой **экранной** y-координатой начальной точки отрезка.

Некоторые пояснения.

Хотя график и строится ломаной линией, за счёт малого смещения по x за один шаг (всего на 1 пиксель) – линия выглядит более-менее гладкой. Большей гладкости можно достичь, только используя алгоритмы сглаживания.

Алгоритм смещается за 1 шаг на один пиксель вдоль OX, вычисляет, какому «истинному» значению x соответствует текущая экранная x-координата (с учётом масштаба и того, что нулю экранной x-координаты должна соответствовать «истинная» координата x_min), вычисляет «истинное» значение функции y = f(x) от этой «истинной» координаты x, затем переводит полученное значение y обратно в экранные координаты (с учётом переворота оси Y, масштаба и того, что верхней границе изображения должна соответствовать «истинная» координата y_max) и проводит отрезок

из предыдущего пикселя в новый. Затем делает этот новый пиксель начальным для следующего отрезка и так далее.

Построение графика функции и вывод на экран

Теперь, написав все эти функции, можно, используя их, построить график произвольной функции и вывести его на экран. В обработчике какого-нибудь события (например, нажатия кнопки) можно написать следующее:

```
clear(pbPlot->Image, br);

Point origin = get_origin_pixel(x_min, x_max, y_min, y_max, pbPlot->Image);
plot_axes(pbPlot->Image, origin, pn_axes);

plot(&sin, x_min, x_max, y_min, y_max, pbPlot->Image, pn_line);
pbPlot->Refresh();
```

То есть:

1. Очистить область рисования.
2. Найти пиксель origin, соответствующий началу координат.
3. Построить на изображении pbPlot->Image координатные оси, проходящие через пиксель origin, при помощи пера pn_axes.
4. Построить график заданной функции (в данном случае, sin из библиотеки cmath), отобразив участок координатной плоскости от x_min до x_max и от y_min до y_max на изображении pbPlot->Image при помощи пера pn_line.
5. **Обновить pbPlot.** (*pbPlot->Refresh();*)

Некоторые пояснения.

Во-первых, если используете функции из библиотеки cmath (здесь – синус), не забудьте её подключить.

Во-вторых, переменных x_min, x_max, y_min и y_max, как видно, в этом коде нет. Эти переменные необходимо объявить выше и заполнить значениями (например, из TextBox’ов или NumericUpDown’ов, или же, например, вычислять автоматически по движению курсора мыши и колесика)

В-третьих, обратите внимание на последнюю строчку: *pbPlot->Refresh();* Эта команда вызывает принудительную перерисовку компонента pbPlot. Дело в том, что формирование изображения не значит обновление экрана. Когда функции clear, plot_axes и plot отработают, они заполнят определённые пиксели Bitmap’а, ссылка на который хранится в pbPlot->Image, но результата на экране видно не будет, пока система не обновит

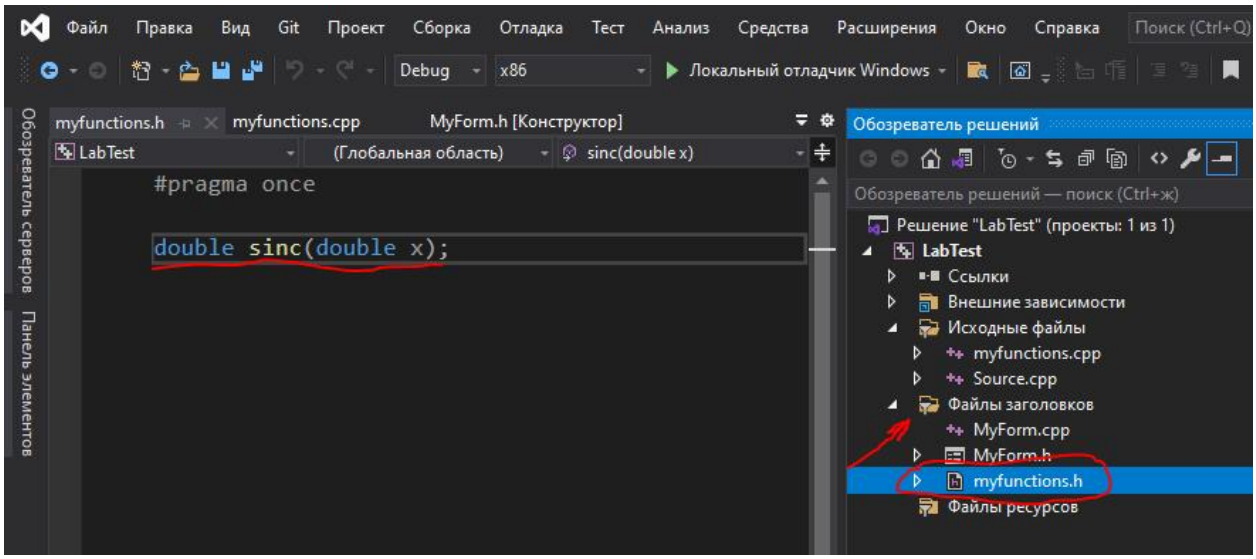
внешний вид pbPlot. Естественным образом это произойдёт, например, при сворачивании и разворачивании окна или некоторых других событиях, однако, чтобы увидеть изображение сразу после построения, необходимо вызвать принудительную перерисовку.

При этом не стоит вызывать принудительную перерисовку после рисования каждого отрезка: обновление изображения на экране – процесс очень долгий, и построение графика в этом случае будет заметно тормозить.

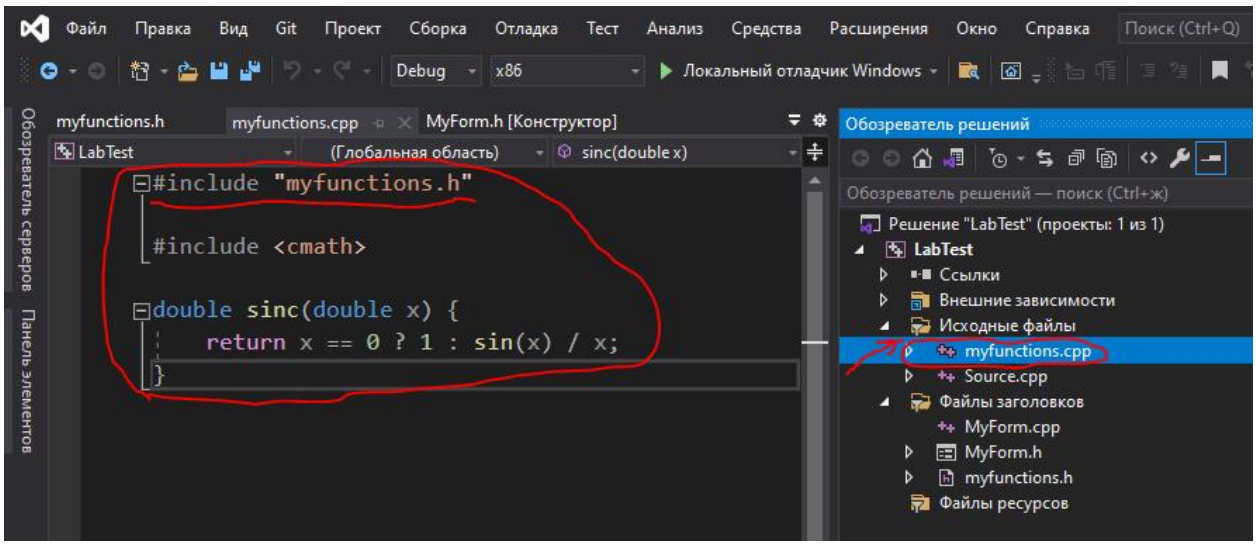
Добавление своих собственных функций для построения их графиков

Функция plot принимает в качестве первого аргумента указатель на любую подходящую функцию, поэтому её возможности, конечно, не ограничиваются построением одних лишь синусов. Однако она не сможет принять на вход функцию-член класса (например, функцию, которую Вы напишете в коде класса MyForm), потому что функции-члены класса, даже вещественные и от одного вещественного аргумента, имеют другой прототип, не подходящий под описание **double (*f)(double)**.

Чтобы добавить свою функцию для рисования и не вызвать конфликта с тем, как VisualStudio рисует форму, можно добавить заголовочный файл и написать там прототип Вашей функции (или нескольких):



Затем добавить ещё один файл исходного кода, подключить в него добавленный заголовочный файл, и написать в этом файле исходного кода реализацию Вашей функции:

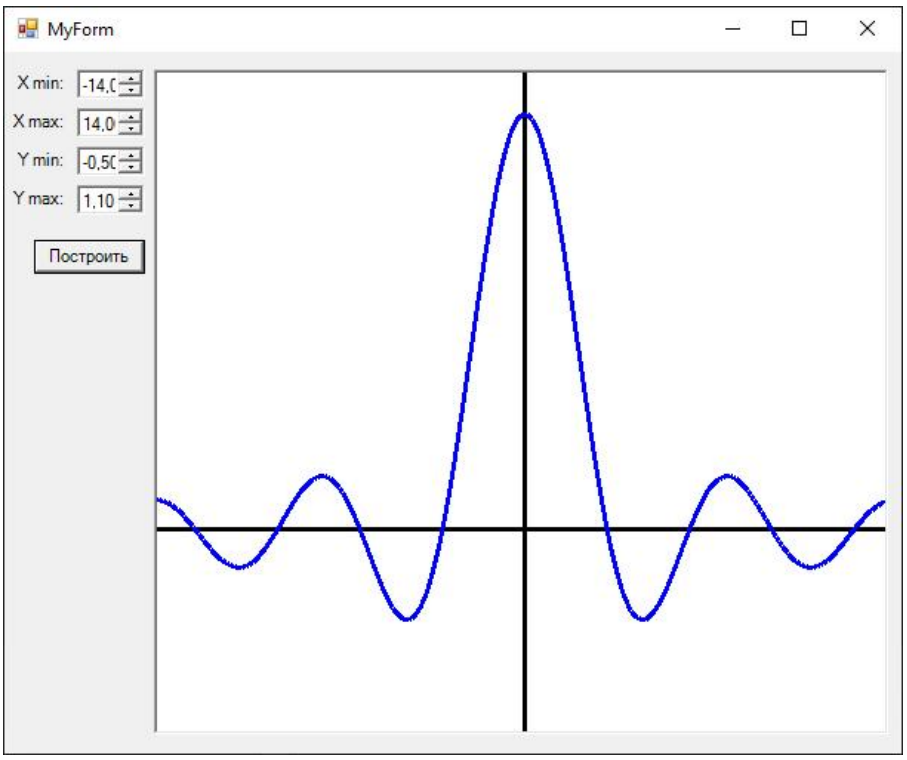


Теперь, имея собственные функции, подходящие под прототип **double (*f)(double)**, можно подключить добавленный заголовочный файл (в примере – файл myfunctions.h) в файл с кодом формы (т.е. в файл MyForm.h) и передавать функции plot для построения любую написанную Вами функцию, чей прототип Вы вынесете в myfunctions.h (не забывая использовать &, т.к. нужен **указатель** на функцию):

```
clear(pbPlot->Image, br);

Point origin = get_origin_pixel(x_min, x_max, y_min, y_max, pbPlot->Image);
plot_axes(pbPlot->Image, origin, pn_axes);

plot(&sinc, x_min, x_max, y_min, y_max, pbPlot->Image, pn_line);
pbPlot->Refresh();
```



Задание на лабораторную работу.

Реализовать построение графика функции. Выбор функции для построения осуществляется из списка (например, ListBox или ComboBox), не менее 5 вариантов.

Необходимо предоставить возможность перемещать и масштабировать область построения (в простейшем случае можно сделать через изменение границ области) – по двум осям **независимо**.

Построенное изображение должно обязательно содержать:

1. координатные оси (если попадают в «окно»);
2. график функции;
3. координатную сетку;
4. подписи значений координатной сетки (можно с краю изображения);
5. подписи осей координат («X» и «Y»);
6. подписи точек пересечения графика с осями координат (т.е. в точках пересечения функции с осью X необходимо вывести значения X, а в точке пересечения с Y – вывести значение Y).

Для подписей можно использовать функцию «Графики» DrawString – она выводит необходимую строку нужным шрифтом в нужном месте изображения и закрашивает фон нужной кистью.

Дополнительно, при желании:

Прокрутку графика реализовать «перетаскиванием» графика при помощи мыши (вам понадобятся события MouseDown, MouseUp, MouseMove и их параметр типа EventArgs, на который мы раньше не обращали внимания), масштабирование реализовать «перетаскиванием» графика с зажатой управляющей клавишей (например, Ctrl. Понадобится включить в форме свойство KeyPreview и назначить обработчики событиям KeyDown и KeyUp формы и использовать их параметр типа EventArgs).

Экстра-сложное (при большом желании):

Вместо выбора функции из списка реализовать построение графика произвольной функции, вводимой пользователем с клавиатуры в виде строки (см. Обратную польскую запись из прошлого семестра – в неё надо будет добавить функции и переменную)