МИНЕСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
МОСКОВСКИЙ ЭНЕРГЕТИЧЕСКИЙ ИНСТИТУТ

Отчет к лабораторной работе № 4

Дисциплина: «Системное программирование»

Михаил Чуворкин
28.10.2021

Подготовка к лабораторной работе

Для выполнения лабораторной работы необходимо ознакомиться (по лекциям, в литературе или в сети) и сделать краткое описание следующих функций и параметров, а также используемых событий.

1). Свойство Canvas компонента TForm.

Графика холста (Canvas) C++ Builder, предназначена для рисования графических примитивов на поверхности «холста» формы. Программа может вывести графику на поверхность формы (или компонента Image), которой соответствует свойство Canvas (Canvas – холст для рисования). Для того чтобы на поверхности формы или компонента Image появилась линия, окружность, прямоугольник или другой графический элемент (примитив), необходимо к свойству Canvas применить соответствующий метод. Методы вывода графических примитивов рассматривают свойство Canvas как некоторый абстрактный холст, на котором они могут рисовать (Canvas переводится как "поверхность", "холст для рисования"). Холст состоит из отдельных точек – пикселей. Положение пикселя на поверхности холста характеризуется горизонтальной (X) и вертикальной (Y) координатами. Координаты возрастают сверху вниз и слева направо (см. рис. 1). Левый верхний пиксель поверхности формы (клиентской области) имеет координаты (0,0), правый нижний – (ClientWidth, ClientHeight). Доступ к отдельному пикселю осуществляется через свойство Pixels, представляющее собой двумерный массив, элементы которого содержат информацию о цвете точек холста.

2). Инструменты рисования и их свойства (кисти Brush, карандаши Pen, шрифт Font).

Карандаш и кисть, являясь свойствами объекта Canvas, в свою очередь представляют собой объекты Pen и Brush. Свойства объекта Pen задают цвет, толщину и тип линии или границы геометрической фигуры. Свойства объекта Brush задают цвет и способ закраски области внутри прямоугольника, круга, сектора или замкнутого контура.

Для Brush:

Свойство	Определяет
Color	Цвет закрашивания замкнутой области
Style	Стиль заполнения области bsSolid – сплошная заливка. Штриховка: bsHorizontal – горизонтальная; bsVertical – вертикальная; bsFDiagonal – диагональная с наклоном линий вперед; bsBDiagonal – диагональная с наклоном линий назад; bsCross – в клетку; bsDiagCross – диагональная клетка
Bitmap	Задает внешнее растровое изображение, определяющее узор кисти.
Handle	Обеспечивает доступ к дескриптору системной кисти

Для Pen:

Свойство	Определяет
Color	Цвет линии
Width	Толщину линии (задается в пикселях)
Style	Вид линии psSolid – сплошная; psDash – пунктирная, длинные штрихи; psDot – пунктирная, короткие штрихи; psDashDot – пунктирная, чередование длинного и короткого штрихов;

	psDashDotDot – пунктирная, чередование одного длинного и двух коротких штрихов; psClear – линия не отображается (используется, если не надо изображать границу области – например, прямоугольника)
Mode	Определяет режим вывода: простая линия, с инвертированием, с выполнением исключающего «или» и др.;
Handle	Определяет дескриптор объекта пера Windows

Для Font(определяет параметры шрифта, которым выводится текст на холсте):

Свойство	**Определяет**
Color	Цвет символов. В качестве значения можно использовать константу типа TColor
Height	Размер шрифта в пикселях.
Name	Используемый шрифт. В качестве значения следует использовать название шрифта (например, Arial)
Size	Размер шрифта в пунктах (points). Пункт – это единица измерения размера шрифта, используемая в полиграфии. Один пункт равен 1/72 дюйма
Style	Стиль начертания символов. Может быть: нормальным, полужирным, курсивным, подчеркнутым, перечеркнутым. Стиль задается при помощи следующих констант: fsBold (полужирный), fsItalic (курсив), fsUnderline (подчеркнутый), fsStrikeOut (перечеркнутый)

3). Функции рисования на канве.

Функция	**Действие**
MoveTo (X,Y)	установить перо в точку (X,Y)
LineTo (X,Y)	рисует отрезок от текущей точки до точки (X,Y)
Rectangle (X1,Y1,X2,Y2)	рисует закрашенный прямоугольник. Параметры – это координаты левого верхнего и правого нижнего угла прямоугольника.
FrameRect(rect:TRect)	Выводит рамку вокруг прямоугольника
Polygon(P:array of TPoint)	рисует многоугольник, координаты которого хранятся в массиве P.
Polyline(P:array of TPoint)	рисует ломаную, по точкам, заданным в массиве P.
Ellipse(X1,Y1, X2,Y2)	рисует закрашенный эллипс. Параметры – это координаты прямоугольника, описанного около данного эллипса.
Arc(X1,Y1, X2,Y2, X3,Y3,X4,Y4)	дуга эллипса, лежащая между лучами исходящими из центра эллипса и проходящими через точки (X3,Y3) и (X4,Y4). Точки могут не лежать на эллипсе. Выбирается та дуга, для которой перемещение от точки (X3,Y3) к (X4,Y4) происходило против часовой стрелки.
Pie(X1,Y1, X2,Y2, X3,Y3,X4,Y4)	сектор эллипса (параметры аналогичны дуге)
Chord (X1,Y1, X2,Y2, X3,Y3,X4,Y4)	сегмент эллипса (параметры аналогичны дуге)

RoundRect (Xl,Yl,X2,Y2,X3,Y3)	Рисует прямоугольник со скругленными углами (X3, Y3 – размеры эллипса скругления)
TextOut(X,Y,Text)	выводит строку Text так, чтобы левый верхний угол прямоугольника, охватывающего текст, имел координаты (X,Y).

Для «рисования» точки можно использовать свойство Pixels: (Canvas->Pixels[10][10] = clRed;)

4). Функции заливки.

Функция	Действие
FloodFill(X,Y, Color, FillStyle)	заливка области, начиная с точки (X,Y). Последний параметр может иметь одно из двух значений: fsSurface – заливка распространяется на все соседние точки цвета Color, fsBorder – заливка прекращается на точках цвета Color.
FillRect(TRect r)	Заполняет прямоугольник, используя кисть Brush, до правой и нижней границ (границы заполняются до предпоследнего пикселя).

5). События: OnClick, OnPaint, OnResize, OnMouseDown, OnMouseMove, OnMouseUp

Обычно событие **OnClick** наступает, если пользователь нажал и отпустил левую кнопку мыши в то время, когда указатель мыши находился на компоненте. Кроме того, это событие происходит в следующих случаях:
- пользователь нажал клавишу пробела, когда кнопка или индикатор были в фокусе;
- пользователь нажал клавишу <Enter>, а активная форма имеет кнопку по умолчанию, указанную свойством **Default**;
- пользователь нажал клавишу <Esc>, а активная форма имеет кнопку прерывания, указанную свойством **Cance**l;
- пользователь нажал клавиши быстрого доступа к кнопке или индикатору;
- пользователь выбрал элемент в сетке, дереве, списке или выпадающем списке, нажав клавишу со стрелкой;
- приложение установило в **true** свойство **Checked** радиокнопки **RadioButton**:
- приложение изменило свойство Checked индикатора **CheckBox**;
- вызван метод **Click** элемента меню.

Событие **OnPaint** наступает, когда возникает сообщение **Windows** о необходимости перерисовать испорченное изображение. Изображение может испортиться из-за временного перекрытия данного окна другим окном того же или постороннего приложения. Обработчик данного события должен перерисовать изображение. При перерисовке изображения канвы **Canvas** можно использовать свойство **ClipRect**, которое указывает область канвы, внутри которой изображение испорчено.

OnResize возникает сразу после изменения размера элемента управления.

Событие **OnMouseDown** наступает в момент нажатия пользователем кнопки мыши над компонентом. Имеется также парное к нему событие **OnMouseUp**, наступающее при отпускании нажатой кнопки мыши над объектом.

Обработка событий **OnMouseDown** и **OnMouseUp** используется для операций, требуемых при нажатии и отпускании пользователем какой-либо кнопки мыши. Обработчики этих событий имеют параметры **Sender**, **Shift**, **Button**, **X** и **Y**. Значения параметра **Button** определяют, какая

кнопка мыши нажата: **mbLeft** — левая, **mbRight** — правая, **mbMiddle** — средняя. Параметры **X** и **Y** определяют координаты указателя мыши в клиентской области компонента.

Событие **OnMouseMove** наступает при перемещении курсора мыши над компонентом. Обработчик события **OnMouseMove** вставляется в программу, если необходимо произвести какие-то операции при перемещении курсора мыши над компонентом. Параметр **Shift**, являющийся множеством, содержит элементы, позволяющие определить, какие кнопки мыши и какие вспомогательные клавиши (<Shift>, <Ctrl> или <Alt>) нажаты в этот момент. Параметры **X** и **Y** определяют координаты указателя мыши в клиентской области компонента.

6). Объекты типа TPoint, TRect

TPoint определяет координаты X и Y целочисленного типа местоположения пикселя на экране с началом координат в верхнем левом углу. X и Y определяют горизонтальную и вертикальную координату точки соответственно.

TRect представляет размеры прямоугольника. Координаты задаются либо в виде четырех отдельных целых чисел, представляющих левую, верхнюю, правую и нижнюю стороны, либо в виде двух точек, представляющих расположение левого верхнего и правого нижнего углов.

7). Методы Form1->Refresh() и :: InvalidateRect(…)

Form1->Refresh() – Запрашивает немедленную перерисовку компонента, стирая перед этим изображение компонента. (Возвращает компонент в первоначальное состояние. Не меняет кисть или шрифт. Берет предыдущее не закрепленное Invalidate-ом или Update-ом состояние и его перерисовывает. Предварительно замазав цветом фона прямоугольник по размерам компонента.)

InvalidateRect(…) - Добавляет прямоугольник в область обновления сцены.

Задание

Программное приложение разрабатывается в среде Borland C++ .

1. По щелчку на кнопке 1 нарисовать на канве формы картинку с использованием всевозможных графических примитивов, карандашей, кистей и заливок (элементы картинки должны составлять единый сюжет).

   Нарисуем «пейзаж» со снеговиком и елками, для этого создадим вспомогательные функции:

```
// Рисование круга по центру и радиусу
void Circle(int x, int y, int r) {
        MainForm -> Canvas -> Ellipse(x - r, y - r, x + r, y + r);
}

// Рисование эллипса по центру и двум полуосям
void SimpleEllipse(int x, int y, int r1, int r2) {
        MainForm -> Canvas -> Ellipse(x - r1, y - r2, x + r1, y + r2);
}
```

```
// Рисование треугольника
void Triangle(int x, int y, int size, TBrushStyle style){
        MainForm -> Canvas -> Brush -> Style = style;
        MainForm -> Canvas -> Brush -> Color = TColor(RGB(20, 170, 20));
        MainForm -> Canvas -> Pen -> Color = TColor(RGB(20, 170, 20));
        TPoint points [3] = {{x, y}, {x + size / 2, y + size}, {x - size / 2,
y + size}};
        MainForm -> Canvas -> Polygon(points, 2);
        MainForm -> Canvas -> Brush -> Style = bsSolid;
}
```

И функции для рисования разных элементов картинки:

```
//------------------------------Picture Funcs--------------------------------

// Рисование облака
void Cloud(int x, int y, int scale){
        MainForm -> Canvas -> Brush -> Color = TColor(RGB(240, 240, 255));
        MainForm -> Canvas -> Pen -> Color = TColor(RGB(240, 240, 255));
        int _size = 10 * scale;
        int d1 = _size * 3;
        int d2 = _size * 2;
        MainForm -> Canvas -> Chord(x, y, x + d1, y + d2, x + d1, y + d2 / 2,
x, y + d2 / 2);
        int dx = _size;
        MainForm -> Canvas -> Chord(x, y + d2 / 4,                    x + dx, y
+ 3 * d2 / 4,  x, y + d2 / 2, x + d1, y + d2 / 2);
        MainForm -> Canvas -> Chord(x + dx, y + d2 / 4,       x + 2 * dx,    y
+ 3 * d2 / 4, x, y + d2 / 2, x + d1, y + d2 / 2);
        MainForm -> Canvas -> Chord(x + 2 * dx, y + d2 / 4,   x + 3 * dx,    y
+ 3 * d2 / 4, x, y + d2 / 2, x + d1, y + d2 / 2);
}

// Рисование неба
void Sky(){
        MainForm -> Canvas -> Brush -> Color = TColor(RGB(80, 80, 240));
        MainForm -> Canvas -> Pen -> Color = TColor(RGB(80, 80, 240));
        int sky_width = MainForm -> Width - 20;
        int sky_height = MainForm -> Height * 0.6;
        MainForm -> Canvas -> Rectangle(5, 5, sky_width, sky_height);
        Cloud(sky_width * 0.2, sky_height * 0.1, 3);
        Cloud(sky_width * 0.4, sky_height * 0.2, 4);
        Cloud(sky_width * 0.6, sky_height * 0.1, 3);
}

// Рисование земли
void Ground(){
        MainForm -> Canvas -> Brush -> Color = TColor(RGB(50, 50, 50));
        MainForm -> Canvas -> Pen -> Color = TColor(RGB(50, 50, 50));
        int ground_width = MainForm -> Width - 20;
        int ground_height_top = MainForm -> Height * 0.6;
```

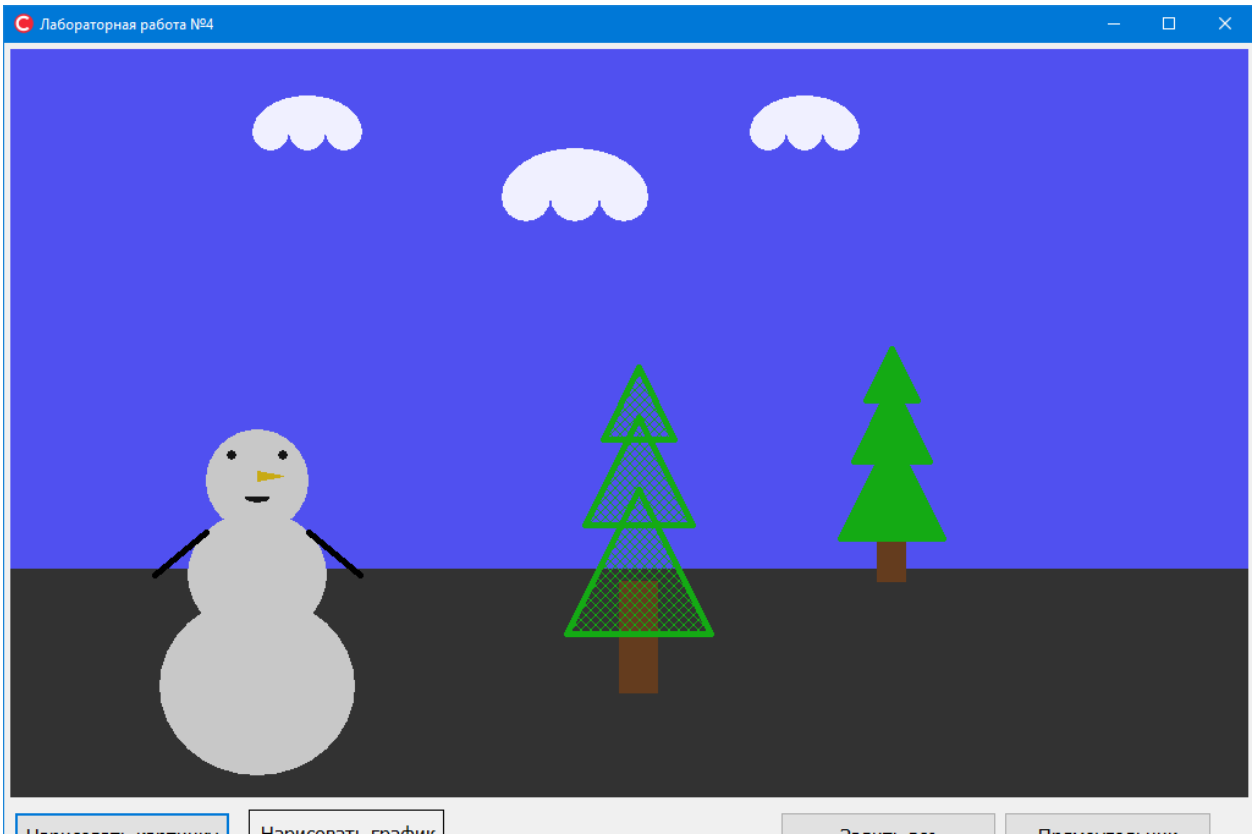
```
        int ground_height_bottom = MainForm -> Height - 100;
        MainForm -> Canvas -> Rectangle(5, ground_height_top, ground_width,
ground_height_bottom);
}

// ===== ### Рисание снеговика ### =====

// Рисование глаз
void Eyes(int x, int y, int _size){
    MainForm -> Canvas -> Brush -> Color = TColor(RGB(20, 20, 20));
        MainForm -> Canvas -> Pen -> Color = TColor(RGB(20, 20, 20));
        int eye_size = _size / 10;
        Circle(x - _size / 2, y - _size / 2, eye_size);
        Circle(x + _size / 2, y - _size / 2, eye_size);
}

// Рисование носа
void Nose(int x, int y, int _size){
        MainForm -> Canvas -> Brush -> Color = TColor(RGB(200, 170, 20));
        MainForm -> Canvas -> Pen -> Color = TColor(RGB(200, 170, 20));
        TPoint points [3] = {{x, y - _size / 5}, {x + _size / 2, y - _size /
10}, {x, y}};
        MainForm -> Canvas -> Polygon(points, 2);
}

// Рисование рта
void Mouth(int x, int y, int _size){
        MainForm -> Canvas -> Brush -> Color = TColor(RGB(20, 20, 20));
        MainForm -> Canvas -> Pen -> Color = TColor(RGB(20, 20, 20));
        int offset = _size / 4;
        MainForm -> Canvas -> Chord(x - offset, y + offset, x + offset, y +
offset + _size / 5, x - offset, y + offset + _size / 10, x + offset, y +
offset + _size / 10);

}

// Рисование головы
void DrawHead(int x, int y, int _size){
        MainForm -> Canvas -> Brush -> Color = TColor(RGB(200, 200, 200));
        MainForm -> Canvas -> Pen -> Color = TColor(RGB(200, 200, 200));
        SimpleEllipse(x, y, _size, _size);
        Eyes(x, y, _size);
        Nose(x, y, _size);
        Mouth(x, y, _size);
}

// Рисование рук
void DrawArms(int x, int y, int _size, int y_body_center){
        MainForm -> Canvas -> Brush -> Color = TColor(RGB(0, 0, 0));
        MainForm -> Canvas -> Pen -> Color = TColor(RGB(0, 0, 0));
        MainForm -> Canvas -> Pen -> Width = 5;
```

```
        MainForm -> Canvas -> MoveTo(x - _size, y + _size);
        MainForm -> Canvas -> LineTo(x - _size * 2, y_body_center);
        MainForm -> Canvas -> MoveTo(x + _size, y + _size);
        MainForm -> Canvas -> LineTo(x + _size * 2, y_body_center);
}

// Рисование тела (средней части)
void DrawBody(int x, int y, int _size){
        MainForm -> Canvas -> Brush -> Color = TColor(RGB(200, 200, 200));
        MainForm -> Canvas -> Pen -> Color = TColor(RGB(200, 200, 200));
        int r1 = _size * 1.25;
        int y_body_center = y + _size * 0.6 + r1;
        SimpleEllipse(x, y_body_center, r1 * 1.1, r1);
        DrawArms(x, y, _size, y_body_center);
}

// Рисование тела (нижней части)
void DrawBottom(int x, int y, int _size){
        MainForm -> Canvas -> Brush -> Color = TColor(RGB(200, 200, 200));
        MainForm -> Canvas -> Pen -> Color = TColor(RGB(200, 200, 200));
        int r1 = _size * 1.7;
        int y_bottom_center = y + _size * 4;
        SimpleEllipse(x, y_bottom_center, r1 * 1.1, r1);
}

// Весь снеговик
void SnowMan(){
        int size = MainForm -> Height / 17;
        int x_start = MainForm -> Width / 5;
        int y_start = MainForm -> Height / 2;
        DrawHead(x_start, y_start, size);
        DrawBody(x_start, y_start, size);
        DrawBottom(x_start, y_start, size);
}


// Рисование елочки
void FirTree(int x, int y, int size, TBrushStyle style){
        MainForm -> Canvas -> Brush -> Color = TColor(RGB(100, 60, 30));
        MainForm -> Canvas -> Pen -> Color = TColor(RGB(100, 60, 30));
        MainForm -> Canvas -> Rectangle(x - size / 4, y + size * 3, x + size /
4, y + size * 4.5);
        Triangle(x, y, size, style);
        Triangle(x, y + size * 0.7, size * 1.5, style);
        Triangle(x, y + size * 1.7, size * 2, style);
}

// Вся картинка
void DrawPicture(){
        Sky();
        Ground();
```

```
        SnowMan();
        int size = MainForm -> Height / 17;
        int x_start = MainForm -> Width * 0.7;
        int y_start = MainForm -> Height * 0.35;
        FirTree(x_start, y_start, size, bsSolid);
        size = MainForm -> Height / 12;
        x_start = MainForm -> Width * 0.5;
        y_start = MainForm -> Height * 0.37;
        FirTree(x_start, y_start, size, bsDiagCross);
}
```

По кнопке будем вызывать функцию рисования всей картинки:

```
void __fastcall TMainForm::btnPictureClick(TObject *Sender)
{
        DrawPicture();
}
```



2. Нарисовать на канве формы кнопку (это должен быть рисунок, выполненный карандашом, кистью и графическими функциями). Пользователь должен воспринимать рисунок как элемент управления. При щелчке мышью по кнопке она должна мигнуть и должен выполниться код пункта 3.

   Объявим прямоугольник для кнопки и напишем функцию обновления положения кнопки:

```
TRect btnRect;
void UpdateBtnPos(){
        MainForm -> Canvas -> Brush -> Color = clBtnFace;
        MainForm -> Canvas -> FillRect(btnRect);
        btnRect.Top = MainForm -> Height - 90;
        btnRect.Left = 200;
        btnRect.Bottom = btnRect.Top + 40;
        btnRect.Right = btnRect.Left + 160;
}
```

Напишем функцию отрисовки кнопки:

```
void DrawPlotButton(TColor color){
        UpdateBtnPos();
        MainForm -> Canvas -> Brush -> Color = color;
        MainForm -> Canvas -> Pen -> Color = clBlack;
        MainForm -> Canvas -> Pen -> Width = 1;
        MainForm -> Canvas -> Rectangle(btnRect);
        MainForm -> Canvas -> Font -> Name = "Tahoma";
        MainForm -> Canvas -> Font-> Color = clBtnText;
        MainForm -> Canvas -> Font -> Size = 12;
        MainForm -> Canvas -> TextOut(btnRect.Left + 10, btnRect.Top + 10,
"Нарисовать график");
}
```

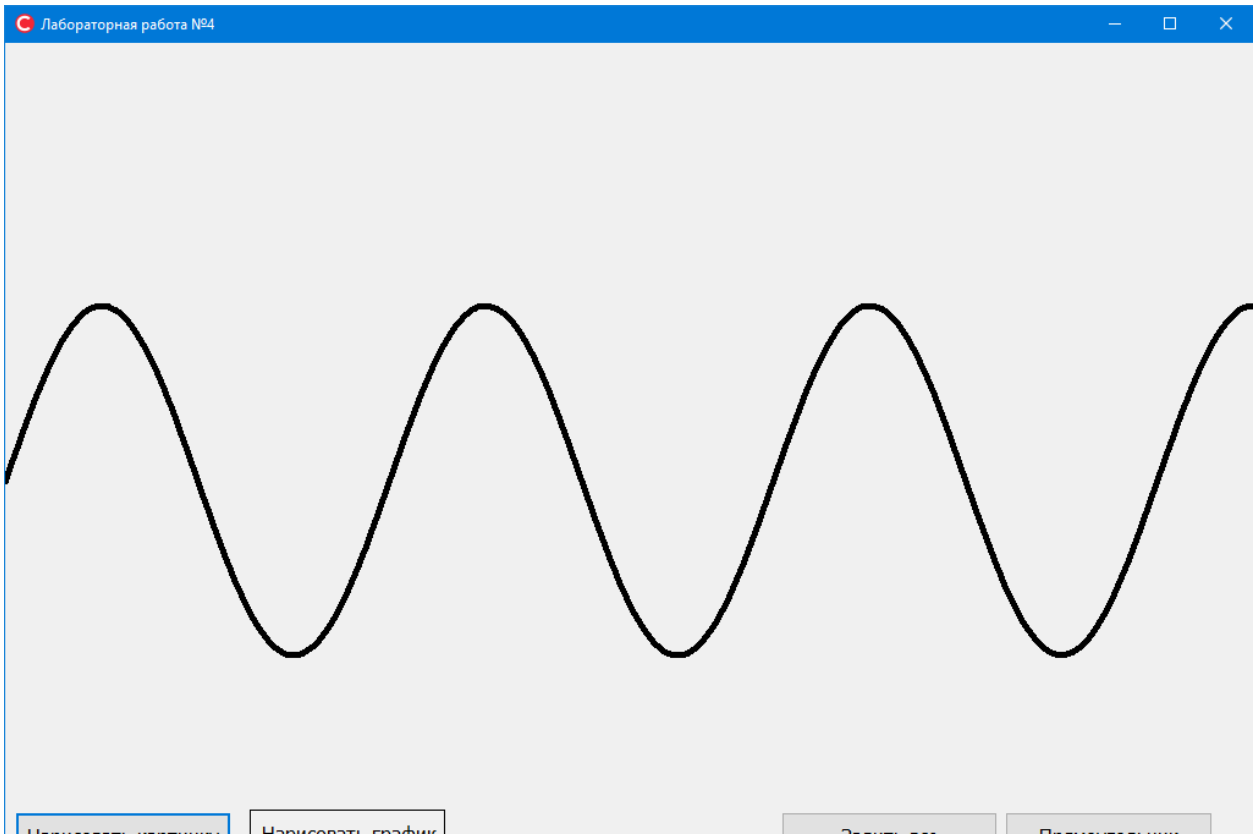
Напишем обработчики нажатия на область кнопки:

```
//---------------------------------------------------------------------------
void __fastcall TMainForm::FormMouseDown(TObject *Sender, TMouseButton
Button, TShiftState Shift,
                  int X, int Y)
{
        if((X > btnRect.Left && X < btnRect.Right) && (Y > btnRect.Top && Y <
btnRect.Bottom) && Button == mbLeft){
                DrawPlotButton(clBtnShadow);
        }
}
//---------------------------------------------------------------------------
void __fastcall TMainForm::FormMouseUp(TObject *Sender, TMouseButton Button,
TShiftState Shift,
                  int X, int Y)
{
        if((X > btnRect.Left && X < btnRect.Right) && (Y > btnRect.Top && Y <
btnRect.Bottom) && Button == mbLeft){
                btnPictureClicked = false;
                btnPlotClicked = true;
                MainForm -> Refresh();
                DrawPlotButton(clBtnFace);
                DrawPlot();
        }
}
```

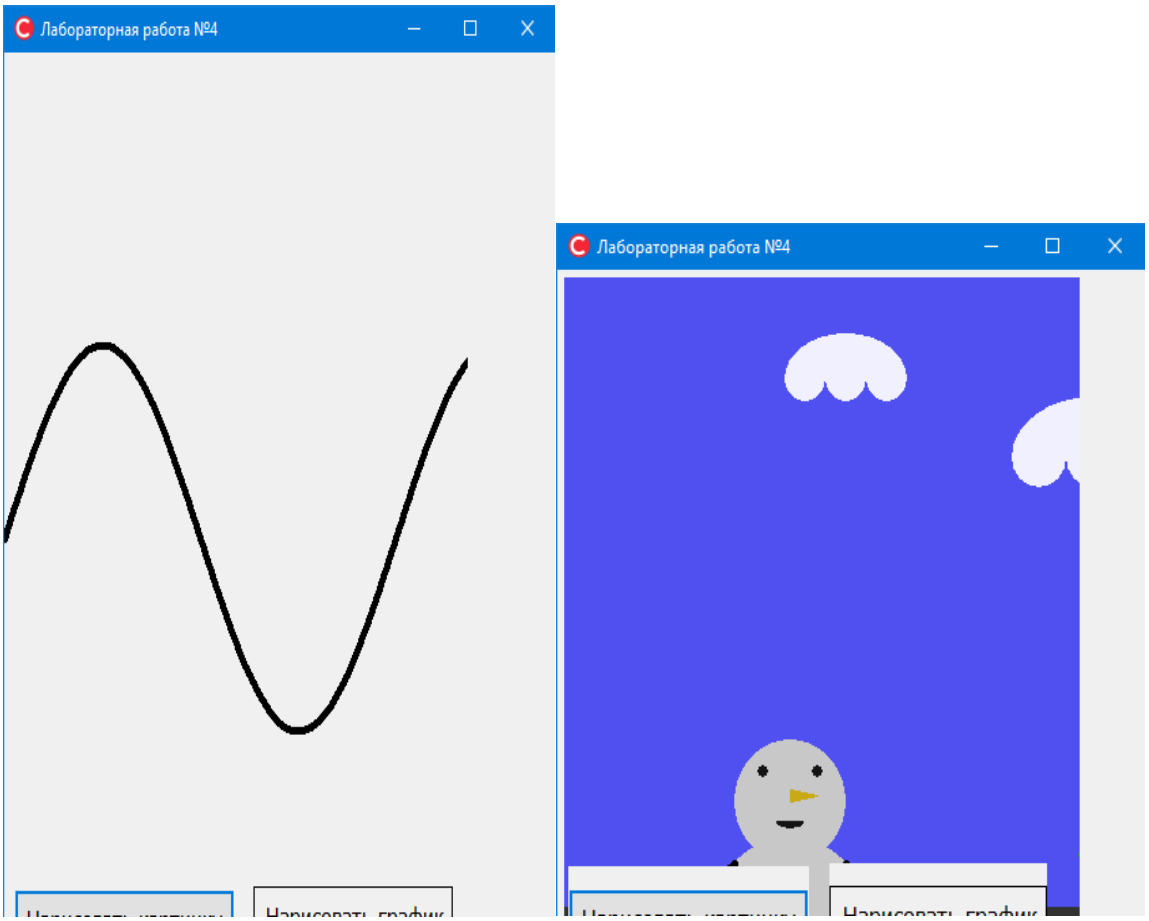
3. По щелчку на кнопке создать на канве формы другое изображение (график функции или вывод текста различными шрифтами и цветами).

Будем выводить график синуса:

```
void DrawPlot(){
        MainForm -> Canvas -> Brush -> Color = TColor(RGB(0, 0, 0));
        MainForm -> Canvas -> Pen -> Color = TColor(RGB(0, 0, 0));
        MainForm -> Canvas -> Pen -> Width = 5;

        int yoff = MainForm -> Height / 2;
        float amp = MainForm -> Height / 5;
        float freq = 0.02;
        int h = 3;
        for(int i = 0 ; i < MainForm -> Width; i+=h){
                int xFrom = i;
                int xTo = i + h;
                MainForm -> Canvas -> MoveTo(xFrom, - amp * sin(freq * xFrom)
+ yoff);
                MainForm -> Canvas -> LineTo(xTo,     - amp * sin(freq * xTo)
+ yoff);
        }
        MainForm -> Canvas -> Pen -> Width = 1;
}
```



4. Убедиться, что после изменения размеров окна (а также при сворачивании/разворачивании) картинка исчезает.



5. Перераспределить код приложения таким образом, чтобы картинки не исчезали с экрана.

Для начала заведем флажки на нажатия кнопок

```
bool btnPictureClicked, btnPlotClicked = false;
```

Перенесем отрисовку в FormPaint() в соответствии с состоянием флажков:

```
void __fastcall TMainForm::FormPaint(TObject *Sender)
{
        DrawPlotButton(clBtnFace);
        if (btnPictureClicked){
                DrawPicture();
        }
        else if (btnPlotClicked){
                DrawPlot();
        }
}
```

И будем изменять состояния флажков при нажатии на кнопки, а также будем вызывать функцию Refresh() для перерисовки всей формы при регистрации нажатий на кнопки и при изменении размера формы:

```
void __fastcall TMainForm::btnPictureClick(TObject *Sender)
{
        btnPictureClicked = true;
        btnPlotClicked = false;
        MainForm -> Refresh();
}
void __fastcall TMainForm::FormResize(TObject *Sender)
{
        MainForm -> Refresh();
        DrawPlotButton(clBtnFace);
}
void __fastcall TMainForm::FormMouseUp(TObject *Sender, TMouseButton Button,
TShiftState Shift,
                  int X, int Y)
{
        if((X > btnRect.Left && X < btnRect.Right) && (Y > btnRect.Top && Y <
btnRect.Bottom)){
                btnPictureClicked = false;
                btnPlotClicked = true;
                MainForm -> Refresh();
        }
}
```

6. Добавить к программе функции (можно добавить еще две кнопки):

- ❖ Нарисовать геометрическую фигуру случайным цветом (так чтобы фигура не исчезала) и проверить, что происходит. Объяснить, почему при изменении размеров окна появляются цветные полосы. Продумать, как избавиться от такого эффекта.
- ❖ Окрасить случайным цветом всю клиентскую область окна и, убедившись, что при изменении размеров окна появляются полоски, избавиться от подобного эффекта.

Добавим две кнопки и флажки для каждой:

```
bool btnRandomBackgroundClicked, btnRandomRectClicked = false;
```

В качестве фигуры будет прямоугольник:

```
TRect randomColorRect(200, 200, 800, 400);
```

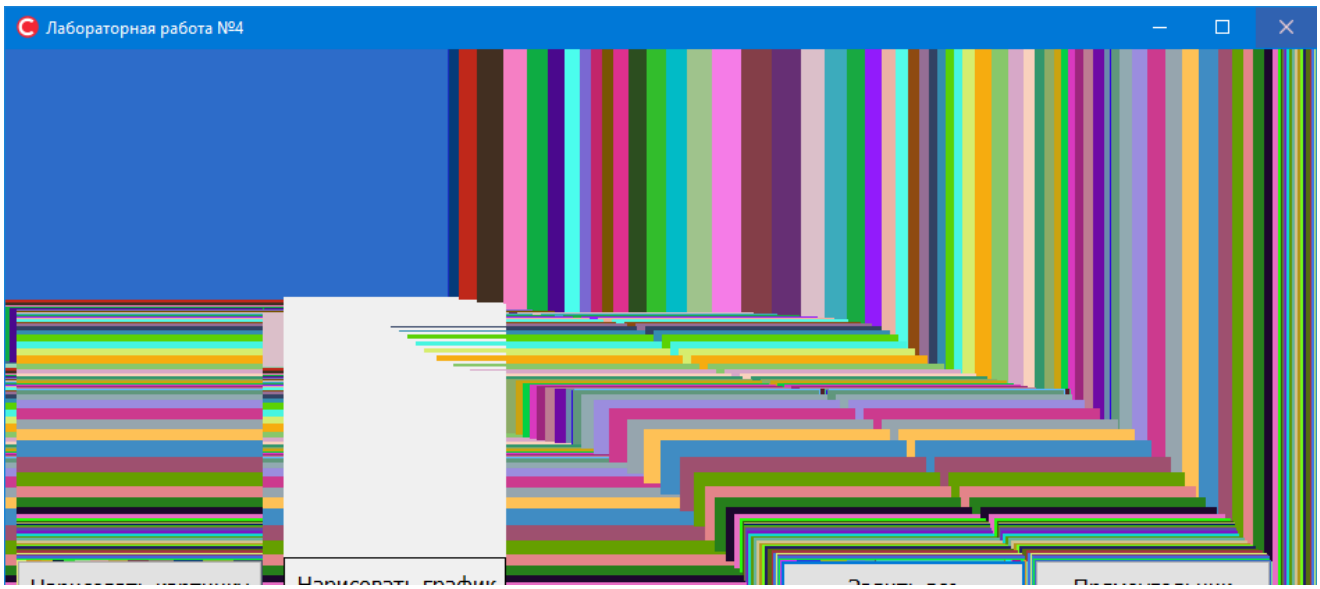
При нажатии на кнопки будем поднимать соответствующий флажок и запрашивать перерисовку нужной области:

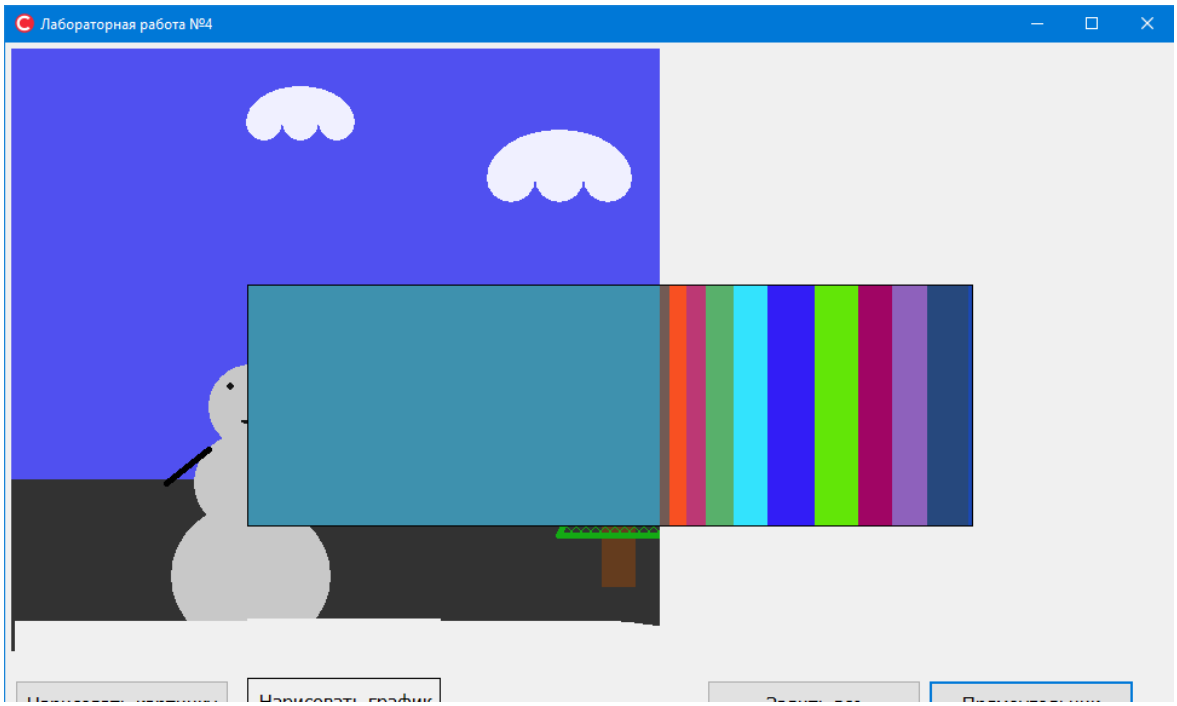
```
void __fastcall TMainForm::btnRandomBackgroundClick(TObject *Sender)
{
        btnPictureClicked = btnPlotClicked = btnRandomRectClicked = false;
        btnRandomBackgroundClicked = true;
        ::InvalidateRect(MainForm->Handle, 0, false);
}
//---------------------------------------------------------------------------
void __fastcall TMainForm::btnRandomRectClick(TObject *Sender)
{
        btnPictureClicked = btnPlotClicked = btnRandomBackgroundClicked =
false;
        btnRandomRectClicked = true;
        ::InvalidateRect(MainForm->Handle, &randomColorRect, false);
}
```

Функции закрашивания всей области и рисования фигуры:

```
void RandomBackground() {
        MainForm -> Canvas -> Brush -> Style = bsSolid;
        MainForm -> Canvas -> Brush -> Color = TColor(RGB(random(255),
random(255), random(255)));
        TRect rect(0, 0, MainForm -> Width, MainForm -> Height);
        MainForm -> Canvas -> FillRect(rect);
}

void RandomColorRect(){
        MainForm -> Canvas -> Brush -> Color = TColor(RGB(random(255),
random(255), random(255)));
        MainForm -> Canvas -> Brush -> Style = bsSolid;
        MainForm -> Canvas -> Rectangle(randomColorRect);
}
```

Для наблюдения эффекта полос уберем из FormResize функцию Refresh(), тогда при изменении размера окна зарисовываться будут только появившиеся участки области, указанной в последнем вызванном InvalidateRect:





Для избавления от этого эффекта вернем Refresh() в FormResize.

Теперь цвет всей формы и всего прямоугольника меняется при изменении размера окна.

7. Добавить к приложению следующие функции:

1). По щелчку на кнопке рисуем круг и заставляем его исчезнуть, если щелкаем мышью внутри круга.

Флажок для кнопки и прямоугольник для круга:

```
bool btnCircleClicked = false;
TRect circleRect(200, 200, 300, 300);
```

При щелчке по кнопке:

```
void __fastcall TMainForm::btnCircleClick(TObject *Sender)
{
        btnPictureClicked = btnPlotClicked = false;
        btnRandomBackgroundClicked = btnRandomRectClicked = false;
        btnCircleClicked = true;
        ::InvalidateRect(MainForm->Handle, &circleRect, false);
}
```

В FormMouseUp() добавим проверку попадания мыши в круг:

```
int r = (circleRect.Right - circleRect.Left) / 2;
int xrel = X - (circleRect.Right + circleRect.Left) / 2;
int yrel = Y - (circleRect.Bottom + circleRect.Top) / 2;

if ( (xrel * xrel + yrel * yrel) < r * r) {
        btnCircleClicked = false;
        ::InvalidateRect(MainForm->Handle, &circleRect, true);
}
```

Функция рисования круга:

```
void DrawCircle() {
        MainForm -> Canvas -> Brush -> Color = clLime;
        MainForm -> Canvas -> Ellipse(circleRect);
}
```

2). Круг рисуется по событию OnPaint. С помощью мыши круг перемещаем по поверхности формы.

Для того, чтобы части круга не обрезались создадим квадрат, больший того, в который вписан круг:

```
TRect circleRectBB(150, 150, 350, 350);
```

Будем двигать круг в обработчике события FormMouseMove()

```
void __fastcall TMainForm::FormMouseMove(TObject *Sender, TShiftState Shift,
int X,
                int Y)
{
        if (Shift.Contains(ssLeft) && btnCircleClicked) {
                ::InvalidateRect(MainForm->Handle, &circleRectBB, true);
                int r = (circleRect.Right - circleRect.Left) / 2;
                circleRect = TRect(X-r, Y-r, X+r, Y+r);
                r = (circleRectBB.Right - circleRectBB.Left) / 2;
```

```
                circleRectBB = TRect(X-r, Y-r, X+r, Y+r);
        }
}
```

Полный код программы доступен здесь: https://pastebin.com/wW3dGnAH