Лабораторная работа №4

Инструменты рисования Builder C++

А-13а-19 Кутдусов Р.К.

Подготовка к лабораторной работе

1. Свойство `Canvas` компонента **`TForm`**.
   При разработке проекта, в котором можно рисовать, в распоряжении находится полотно (холст) — свойство `Canvas`, карандаш — свойство `Pen`, и кисть — свойство `Brush`. Канва представляет собой область компонента, на которой можно рисовать или отображать готовые изображения.
   При рисовании на компоненте **`TForm`** сам *компонент* рассматривается как прямоугольная сетка, состоящая из отдельных точек, называемых пикселями. Положение пикселя характеризуется его вертикальной (X) и горизонтальной (Y) координатами. Левый верхний пиксель имеет *координаты* (0,0). Вертикальная координата возрастает сверху вниз, горизонтальная — слева направо. Общее количество пикселей по вертикали определяется свойством `Height`, а по горизонтали — свойством `Weight`. Каждый пиксель может иметь свой цвет.
   Канва имеет свойство `Pixels`. Это свойство представляет собой двумерный массив, который отвечает за цвета канвы. С массивом пикселей можно обращаться как с любым свойством: изменять цвет, задавая пикселю новое значение, или определять его цвет по хранящемуся в нем значению.
2. Инструменты рисования и их свойства (кисти `Brush`, карандаши `Pen`, шрифт `Font`).
   Канва — объект класса `TCanvas` имеет множество методов, которые позволяют рисовать графики, линии, фигуры с помощью свойства `Pen` — перо. Это свойство является объектом, в свою очередь имеющим ряд свойств. Одно из них свойство `Color` — цвет, которым наносится рисунок. Второе свойство — `Width` (ширина линии). Ширина задается в пикселях.
   Свойство `Style` определяет вид линии.
   Фигуры в общем случае рисуются не пустыми, а закрашенными с помощью свойства канвы `Brush` — кисть. Свойство `Brush` является объектом, имеющим в свою очередь ряд свойств. Свойство `Color` определяет цвет заполнения. Свойство `Style` определяет шаблон заполнения (штриховку). По умолчанию значение `Style` равно `bsSolid`, что означает сплошное закрашивание цветом `Color`.
   Наряду с геометрическими фигурами и изображениями канва позволяет выводить текст. Для этого вначале надо установить у канвы свойство `Font`. Свойство `Font` в качестве значения принимает определение шрифта.
   Класс `TFont` включает в себя следующие свойства:

   - `Name` - название шрифта (`Arial, MS Sans Serif, Calibri` и т.д.)
   - `Size` - размер шрифта, который задается в кеглях.
   - `Height` - тоже размер шрифта, но в отличии от `Size` задается уже в пикселях.
   - `Color` - цвет шрифта.
   - `Style` - стиль шрифта. Это свойство принимает следующие значения:

   `fsBold` - полужирный
   `fsItalic` - курсив

3. Функции рисования на канве.
   При помощи `MoveTo` мы перемещаем начало нашей линии из начала координат в ту точку, в которую хотим. При помощи `LineTo` рисуем линию из этой точки в ту точку, которую указали в методе.
   При помощи методов `LineTo` и `MoveTo` можно рисовать не только простые линии, но и такие объекты, как, допустим, прямоугольник или квадрат.
   Для рисования прямоугольников служит функция `Rectangle`, имеющая следующий вид: `Rectangle (x1,y1,x2,y2)`, где переменные `x1,y1` отвечают за положение левого верхнего угла прямоугольника, а переменные `x2,y2` за положение нижнего правого угла.
   Для рисования эллипсов в канве предназначена функция `Ellipse(x1,y1,x2,y2)`. Этот метод имеет точно такие же параметры, как и `Rectangle`, за исключением того, что здесь указываются координаты углов прямоугольника, **описывающего** эллипс.
   `RoundRect (x1,y1,x2,y2,x3,y3)` - данная функция рисует прямоугольник с закругленными углами по заданным координатам.
   `Arc (x1,y1,x2,y2,x3,y3,x4,y4)` - данный метод рисует дугу на канве.
   Переменные `x1,y1,x2,y2` - определяют эллипс, частью которого является дуга.
   Переменные `x3,y3` - определяют начальную точку дуги, а `x4,y4` - конечную точку дуги. Дуга вычерчивается против часовой стрелки из начальной в конечную точку.
   Внешний вид дуги определяет свойство `Pen` канвы.
4. Функции заливки.
   Для заливки (закрашивания) областей на канве определены две функции, это методы `FillRect (TRect* Rect)` и `FloodFill (x,y, Color, Style)`.
   При помощи функции `FillRect` происходит простая заливка указанной прямоугольной области на канве. Вид заливки определяет свойство `Brush`. В качестве единственного параметра у данного метода идет простая структура `TRect& Rect`, представляющая из себя всего лишь набор из 4 переменных, определяющих границы этой области.
   Функция `FloodFill` определяет более сложную заливку, она имеет следующий вид: `FloodFill ( int x, int y, TColor Color, TFillStyle FillStyle)`, где:
   `x,y` - координаты начальной точки заливки
   `Color` - цвет, определяющий область заливки
   `FillStyle` - стиль заливки, о котором я расскажу несколько позже
   Заливка области происходит текущей кистью, как и у метода `FillRect()`.
5. События: `OnClick, OnPaint, OnResize, OnMouseDown, OnMouseMove, OnMouseUp`
   `OnClick` – отслеживает нажатие кнопки мыши на форме
   `OnPaint` – вызывается в момент, когда форме необходимо перерисовать содержимое
   `OnResize` – вызывается, когда форма изменяет свой размер
   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. 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.
6. Объекты типа **`TPoint`**, **`TRect`**
   Тип **`TPoint`** это тип записи, содержащий целочисленные значения X и Y.
   Он обычно используется для хранения 2-х мерных координат.
   Структура **`TRect`** представляет из себя набор из 4 переменных, определяющих границы прямоугольной области.
   «Точка» — тип, позволяющий определить переменную, значением которой являются координаты (x, y) точки на экране. *«Прямоугольник»* — тип, используемый для задания

координат двух точек. Эти точки определяют левый верхний и правый нижний углы прямоугольника со сторонами, параллельными осям координат.

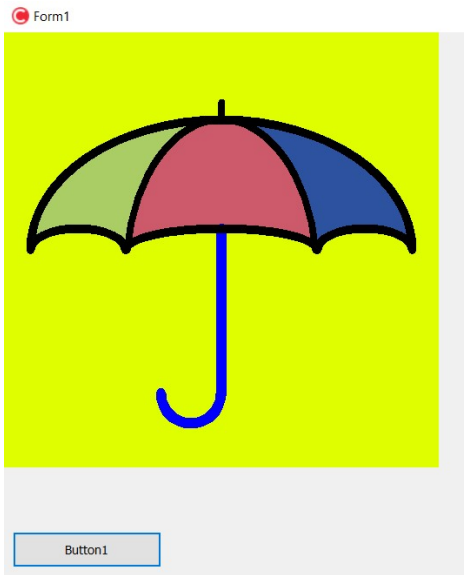
7. Методы `Form1->Refresh()` и `:: InvalidateRect(…)`.
   Функция `InvalidateRect` добавляет прямоугольник к обновляемому региону заданного окна. Обновляемый регион представляет часть рабочей области окна, которая должна быть перерисована.
   `Refresh` – запрашивает немедленную перерисовку компонента, стирая перед этим изображение компонента.

Ход работы

1. При нажатии на кнопку 1 на канве формы рисуется картинка с использованием всевозможных графических примитивов, карандашей, кистей и заливок (элементы картинки должны составлять единый сюжет).
   Нарисуем простой зонтик:

```
void TForm1::DrawPic() {
        Form1->Canvas->Brush->Color = TColor(0x00FFDF);
        Form1->Canvas->FillRect(TRect(0, 0, 500, 500));
        Form1->Canvas->Pen->Color = clBlue;
        Form1->Canvas->Pen->Width = 12;
        Form1->Canvas->Arc(180, 380, 250, 450, 180, 415, 250, 415);
        Form1->Canvas->MoveTo(250, 415);
        Form1->Canvas->LineTo(250, 225);
        Form1->Canvas->Pen->Color = clBlack;
        Form1->Canvas->Pen->Width = 10;
        Form1->Canvas->Arc(30, 100, 470, 400, 470, 250, 30, 250);
        Form1->Canvas->Arc(30, 225, 140, 275, 140, 250, 30, 250);
        Form1->Canvas->Arc(140, 225, 360, 275, 360, 250, 140, 250);
        Form1->Canvas->Arc(360, 225, 470, 275, 470, 250, 360, 250);
        Form1->Canvas->Pen->Width = 8;
        Form1->Canvas->MoveTo(250, 100);
        Form1->Canvas->LineTo(250, 80);
        Form1->Canvas->Arc(140, 100, 360, 450, 360, 250, 140, 250);
        Form1->Canvas->Brush->Color = TColor(0x66CDAA);
        Form1->Canvas->FloodFill(105, 200, TColor(0x00FFDF), fsSurface);
        Form1->Canvas->Brush->Color = TColor(0x6A5ACD);
        Form1->Canvas->FloodFill(250, 140, TColor(0x00FFDF), fsSurface);
        Form1->Canvas->Brush->Color = TColor(0xA0522D);
        Form1->Canvas->FloodFill(375, 180, TColor(0x00FFDF), fsSurface);}
void __fastcall TForm1::Button1Click(TObject *Sender)
        { DrawPic();}
```

   Получилось:



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.
   Рисунок кнопки 2:


```
void TForm1::DrawBott(){
         Form1->Canvas->Pen->Color = (TColor)RGB(0, 120, 215);
         Form1->Canvas->Pen->Width = 5;
         Form1->Canvas->Rectangle(240, 575, 410, 610);
         Form1->Canvas->Brush->Color = TColor(0xE2E2E2);
         TRect rect = TRect(240, 575, 410, 610);
         Form1->Canvas->FillRect(rect);
         Form1->Canvas->Font->Name = "Tahoma";
         Form1->Canvas->Font->Size = 9;
         Form1->Canvas->Font->Color = clBlack;
         Form1->Canvas->TextRect(rect, 300, 584, "Button2");}
void __fastcall TForm1::FormPaint(TObject *Sender)
        {DrawBott();}
```

   Мигание кнопки (при вызове события формы `OnMouseDown`):

```
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState
Shift,
                  int X, int Y) {
        if (X <= 410 && X >= 240 && Y <= 610 && Y >= 575) {
                 TRect rect = TRect(240, 575, 410, 610);
                ::InvalidateRect(Form1->Handle, &rect, true);
                Form1->Canvas->Pen->Color = (TColor)RGB(0, 120, 215);
                Form1->Canvas->Pen->Width = 2;
                Form1->Canvas->Rectangle(240, 575, 410, 610);
                Form1->Canvas->Brush->Color = (TColor)RGB(230, 240, 250);
                Form1->Canvas->FillRect(rect);
                ::Sleep(100);
                ::InvalidateRect(Form1->Handle, &rect, true);
                DrawBott();
                PrintFonts();}}
```

   Кнопка в “мигающем” состоянии:

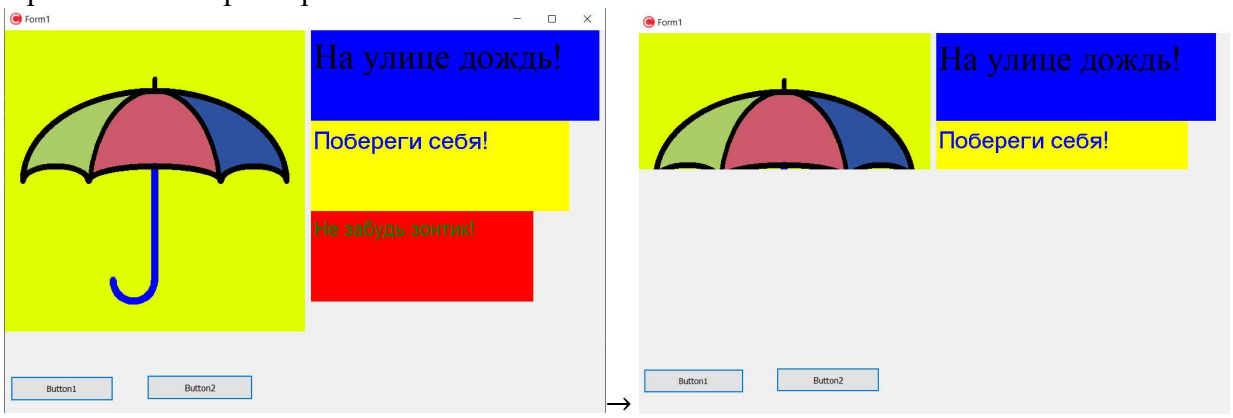
3. По щелчку по кнопке 2 создать на канве формы другое изображение (вывод текста различными шрифтами и цветами).

```
void TForm1::PrintFonts(){
         TRect rects[3];
         rects[0] = TRect(510, 0, 990, 150);
         rects[1] = TRect(510, 150, 940, 300);
         rects[2] = TRect(510, 300, 880, 450);
         Form1->Canvas->Brush->Color = clBlue;
         Form1->Canvas->FillRect(rects[0]);
         Form1->Canvas->Font->Name = "Times New Roman";
         Form1->Canvas->Font->Size = 35;
         Form1->Canvas->Font->Color = clBlack;
         Form1->Canvas->TextRect(rects[0], 515, 10, "На улице дождь!");
         Form1->Canvas->Brush->Color = clYellow;
         Form1->Canvas->FillRect(rects[1]);
         Form1->Canvas->Font->Name = "MS Sans Serif";
```

```
    Form1->Canvas->Font->Size = 25;
    Form1->Canvas->Font->Color = clBlue;
    Form1->Canvas->TextRect(rects[1], 515, 160, "Побереги себя!");
    Form1->Canvas->Brush->Color = clRed;
    Form1->Canvas->FillRect(rects[2]);
    Form1->Canvas->Font->Name = "Arial";
    Form1->Canvas->Font->Size = 20;
    Form1->Canvas->Font->Color = clGreen;
    Form1->Canvas->TextRect(rects[2], 515, 310, "Не забудь зонтик!");}
```

Что вышло:

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

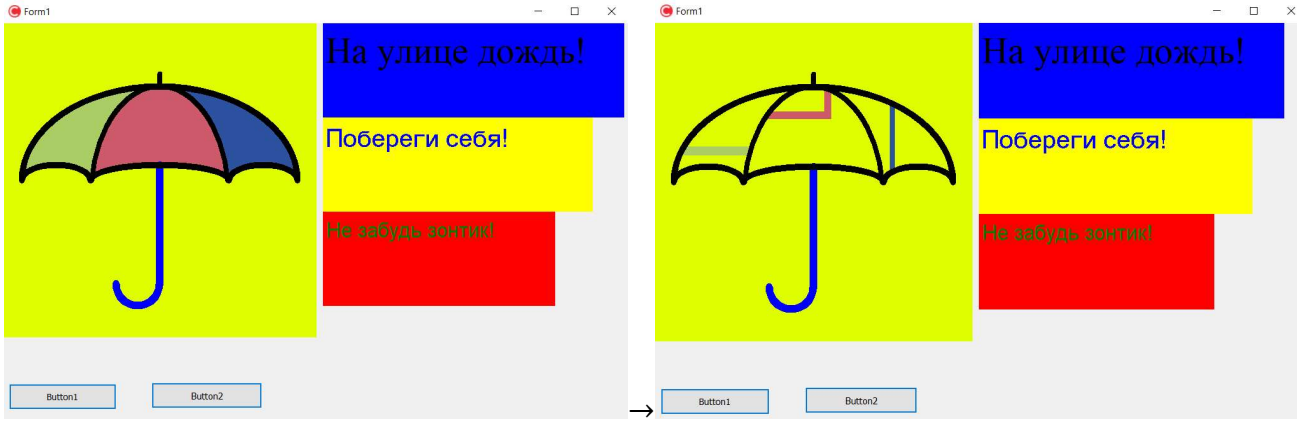


   При сворачивании/разворачивании:

5. Перераспределить код приложения таким образом, чтобы картинки не исчезали с экрана. Заметим, что изображение кнопки не исчезает, так как оно записано в `OnPaint`. Проделаем тоже самое для двух рисунков:

```
bool pic = false;      // для зонтика
bool fonts = false;    // для текста
…
void __fastcall TForm1::Button1Click(TObject *Sender) {
        DrawPic();
        pic = true; // включаем
}
…
PrintFonts();
fonts = true; // включаем
…
void __fastcall TForm1::FormPaint(TObject *Sender) {
        DrawBott();
        if (pic) DrawPic();
        if (fonts) PrintFonts();
}
```

До/после изменения размера:



Сворачивание/разворачивание формы не меняет картинок.

Добавим код в `OnResize`.

```
void __fastcall TForm1::FormResize(TObject *Sender){
        if (pic) DrawPic();}
```

Тогда и изменение размера формы не будет менять картинок.

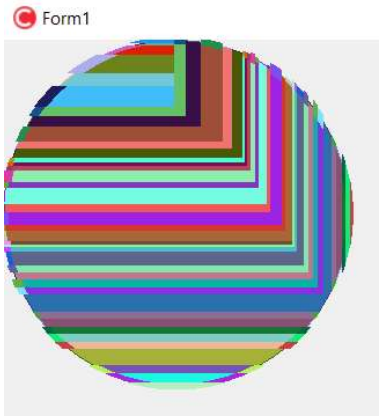
6. Добавить к программе функции:

- 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.

```
void __fastcall TForm1::Butt3Click(TObject *Sender){
        DrawEllipse();
        el = true; // включаем фигуру
}
void __fastcall TForm1::FormPaint(TObject *Sender)
{
        DrawBott();
        if (pic) DrawPic();
        if (fonts) PrintFonts();
        if (el) DrawEllipse(); // чтобы не исчезала
}
void TForm1::DrawEllipse(){
        Form1->Canvas->Brush->Color = (TColor)RGB(Random(255), Random(255),
Random(255));
        Form1->Canvas->Pen->Color = (TColor)RGB(Random(255), Random(255), Random(255));
        Form1->Canvas->Pen->Width = Random(10);
```

```
        TRect rect = TRect(0, 0, 300, 300);
        Form1->Canvas->Ellipse(rect);
        Form1->Canvas->FloodFill(150, 150, clGray, fsSurface);}
```

Полосы при уменьшении размера:



Чтобы избавится от полос, добавим код в `OnResize`:

```
void __fastcall TForm1::FormResize(TObject *Sender){
        if (pic) DrawPic();
        if (el) DrawEllipse();
}
```

Полосы появляются из-за того, что обработчик события `OnPaint` вызывается много раз, круг рисуется не целиком, система пропускает на экран только ту часть, которая нуждается в перерисовке (скрытая пользователем).

- ❖ Окрасить случайным цветом всю клиентскую область окна и, убедившись, что при изменении размеров окна появляются полоски, избавиться от подобного эффекта.

```
void TForm1::DrawClient() {
        TRect rect;
        ::GetWindowRect(Form1->Handle, &rect);
        Form1->Canvas->Brush->Color = (TColor)RGB(Random(255), Random(255),
Random(255));
        Canvas->FillRect(rect);
}
```

Как мы убедились, от полос избавляет только добавление кода в `OnResize`.

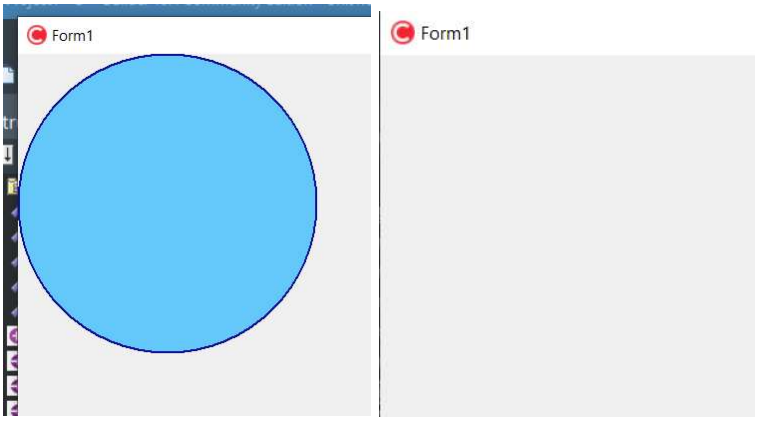
```
void __fastcall TForm1::Button4Click(TObject *Sender)
{
        DrawClient();
        client = true;
}
void __fastcall TForm1::FormResize(TObject *Sender)
{
        //DrawBott();
        if (pic) DrawPic();
        if (el) DrawEllipse();
        if (client) DrawClient();}
```

7. Добавить к приложению следующие функции:
    1) По щелчку на кнопке рисуем круг и заставляем его исчезнуть, если щелкаем мышью внутри круга.
    Изменим нашу функцию (сделаем цвет определённым):

```
TRect el_rect; // глобальная переменная
…
void TForm1::DrawEllipse(){
        Form1->Canvas->Brush->Color = (TColor)RGB(100, 200, 250);
        Form1->Canvas->Pen->Color = (TColor)RGB(13, 14, 156);
        Form1->Canvas->Pen->Width = 2;
        TRect rect = el_rect;
        Form1->Canvas->Ellipse(rect);
        Form1->Canvas->FloodFill(el_rect.left + 150, el_rect.top + 150, clGray,
fsSurface);}
void __fastcall TForm1::Butt3Click(TObject *Sender)
{
        el_rect = TRect(0, 0, 300, 300);
        DrawEllipse();
        el = true; // el – флажок для отрисовки в OnPaint
}
```

Заставим наш круг исчезнуть по щелчку внутри:

```
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState
Shift, int X, int Y)
{
        if (X <= 300 && Y <= 300 && (((X - 150) * (X - 150) +  (Y - 150) * (Y - 150))
                <= 150 * 150))
                {
                el = false;
                ::InvalidateRect(Form1->Handle, &el_rect, true);}
}
```



    2) Круг рисуется по событию `OnPaint`. С помощью мыши круг перемещаем по поверхности формы.
    Переделаем немного предыдущий пункт: сделаем так, чтобы круг пропадал при нажатии ПКМ.

```
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState
Shift,
                  int X, int Y)
{
        if (X <= 300 && Y <= 300 && (((X - 150) * (X - 150) +  (Y - 150) * (Y - 150))
<= 150 * 150))
        {
                if (Button == mbRight) {
                        el = false;
                        ::InvalidateRect(Form1->Handle, &el_rect, true);
                }
                else if (Button == mbLeft)
                        move_el = true;
}}
```

```
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
{
        if (move_el && X >= el_rect.left && Y >= el_rect.top && X <= el_rect.right && Y
<= el_rect.bottom &&
           (((X - 150 - el_rect.left) * (X - 150 - el_rect.left) +
           (Y - 150 - el_rect.top) * (Y - 150 - el_rect.top)) <= 150 * 150))
        {
                ::InvalidateRect(Form1->Handle, &el_rect, true);
                el_rect.top += 1;
                el_rect.bottom += 1;
                el_rect.right += 1;
                el_rect.left += 1;
                ::InvalidateRect(Form1->Handle, &el_rect, true);
        }
}
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button, TShiftState
Shift,
                  int X, int Y)
{
        if (move_el) move_el = false; // снимаем флажок
}
```

Движение: