Уважаемые студенты групп!

Вашему вниманию представлена лекция на тему «Изучение приемов работы с графикой в DELPHI». Лекция рассчитана на 8 часов

Задание

1. Прочитать внимательно лекцию.
2. Законспектировать лекцию в рабочую тетрадь не менее 3-6 страницы рукописного текста. В конспекте лекции обязательно должно быть приведены примеры.
3. Ответить письменно в рабочей тетради на контрольные вопросы.
4. Дата предоставления полного фотоотчета будет сообщена дополнительно

С уважением Ганзенко Ирина Владимировна
!!! Если возникнут вопросы обращаться по телефону 0721134803 (вацап), +79591134803 (телеграмм)
disobuch.ganzenko2020@mail.ru

Лекция. Изучение приемов работы с графикой в DELPHI

Целью данного раздела является изучение графических процедур и функций языка Delphi, освоение принципов и методов построения геометрических фигур, работы с цветом.

План

1. Графические элементы
2. Координаты
3. Пиксели и точки
4. Перо Pen
5. Рисование линий
6. Кисть Brush
7. Класс TFont
8. Работа с цветом
9. Классы TGraphic и TPicture
10. Контрольные вопросы

1 Графические элементы

Богатство изобразительных возможностей *Windows* связано с так называемым дескриптором контекста графического устройства DC и тремя входящими в него инструментами − шрифтом, пером и кистью.

В *Delphi* созданы специализированные классы-настройки, существенно упрощающие использование графических инструментов *Windows*: для контекста – класс *TCanvas*, для шрифта – *TFont*, для пера – *TPen*, для кисти – *TBrush*. Связанные с этими классами объекты автоматически создаются для всех видимых элементов и становятся доступны программе через свойства *Canvas, Font, Pen* и *Brush*. Однако перечисленные выше свойства доступны не у всех визуальных компонентов. Например, у геометрической фигуры *Shape* доступны свойства *Font*, *Pen* и *Brush*, а у кнопки *Button* – только *Font* и *Brush.*

Canvas – это область рисунка на форме и многих других графических компонентах. Свойство *Canvas* предоставляет возможность манипуляции областью рисунка во время выполнения программы. Этот класс создает «холст», «канву», на которой можно рисовать чертежными инструментами – пером, кистью и шрифтом. Объекты класса *TCanvas* автоматически создаются для всех видимых компонентов, которые должны уметь нарисовать себя. Они инкапсулируют объекты *Font, Pen, Brush*, а также многочисленные методы, использующие эти объекты. Свойства класса *TCanvas* описаны ниже (табл. 8.1).

Таблица 8.1

Свойства класса *TCanvas*

Свойство	Пояснение
Property Brush: TBrush;	Объект-кисть
Property ClipRect: TRect;	Определяет текущие размеры области, нуждающейся в прорисовке
Property CopyMode: TCopyMode;	Устанавливает способ взаимодействия растрового изображения с цветом фона
Property Font: TFont;	Объект-шрифт
Property Handle: integer;	Дескриптор канвы. Используется при непосредственном обращении к API-функциям Windows
Property LockCount: integer;	Счетчик блокировок канвы. Увеличивается на единицу при каждом обращении к методу Lock и уменьшается на единицу при обращении к Unlock
Property Pen: TPen;	Объект-перо
Property PenPos: TPoint;	Определяет текущее положение пера в пикселях относительно левого верхнего угла канвы

Свойство	Пояснение
Property Pixels [X,Y: integer]: TColor;	Массив пикселей канвы

Свойство *CopyMode* используется при копировании части одной канвы (источника) в другую (приемник) методом *CopyRect* и может иметь одно из следующих значений, описанных ниже (табл. 8.2).

Таблица 8.2

Значения свойства *CopyMode*

Значение	Пояснение
cmBlackness	Заполняет область рисования черным цветом
cmDestInvert	Заполняет область рисования инверсным цветом фона
cmMergeCopy	Объединяет изображение на канве и копируемое изображение операцией AND
cmMergePaint	Объединяет изображение на канве и копируемое изображение операцией OR
cmNotSrcCopy	Копирует на канву инверсное изображение источника
cmNotSrcErase	Объединяет изображение на канве и копируемое изображение операцией OR и инвертирует полученное
cmPatCopy	Копирует образец источника

Окончание табл. 8.2

Значение	Пояснение
crnPatInvert	Комбинирует образец источника с изображением на канве с помощью операции XOR
crnPatPaint	Комбинирует изображение источника с его образцом с помощью операции OR, затем полученное объединяется с изображением на канве также с помощью OR
cmSrcAnd	Объединяет изображение источника и канвы с помощью операции AND
cmSrcCopy	Копирует изображение источника на канву. Этот режим принят по умолчанию
cmSrcErase	Инвертирует изображение на канве и объединяет результат с изображением источника операцией AND
cmSrcInvert	Объединяет изображение на канве и источник операцией XOR
cmSrcPaint	Объединяет изображение на канве и источник операцией OR
cmWhiteness	Заполняет область рисования белым цветом

При выполнении графических операций используются следующие типы:

```
type TPoint=record
     X: Longint;
     Y: Longint;
     end;
```

```
TRect= record
Case integer of
0: (Left, Right, Top, Bottom: integer);
1: (TopLeft, BottonRight: TPoint);
end;
```

2 Координаты

Для построения изображений на экране используется система координат. Отсчёт начинается от верхнего левого угла поверхности рисования, который имеет координаты (0,0). Значение X (столбец) увеличивается слева направо, значение Y (строка) увеличивается сверху вниз.

Все визуальные компоненты имеют свойства *Top* и *Left*. Значения, запоминаемые этими свойствами, определяют местоположение компоненты на форме. Иными словами, компонента размещается в позиции (X,Y), где X относится к свойству *Left*, а Y – к свойству *Top*. Значения X и Y выражаются в пикселах. *Пиксель* – это наименьшая частичка поверхности рисунка, которой можно манипулировать. Для каждой визуальной компоненты, которая наследует объект *Canvas*, характерны свои координаты точек. Верхний угол компоненты имеет координаты (X=0, Y=0), а нижний угол описан в свойствах данной визуальной компоненты полями (X=<Имя компоненты>.*Width*, Y=<Имя компоненты>.*Height*). Например, координаты нижнего угла *Form*1 (*Form*1.*Width, Form*1.*Height*). Для других компонент эти поля аналогичны. Изменяя эти поля, можно изменять рабочее пространство для рисования, и соответственно изменяются размеры компоненты.

3 Пиксели и точки

Концептуально все графические операции сводятся к установке цвета пикселей на плоскости рисунка. *Delphi* позволяет манипулировать каждым отдельным пикселем, важнейшее свойство которого – его цвет. В сегодняшнем компьютерном мире пиксели могут иметь широкий диапазон цветов.

Свойство *Pixels* − это двумерный массив, соответствующий цветам отдельных пикселей в *Canvas*. *Canvas*.*Pixels* [10, 20] соответствует цвету пикселя, 10-го слева и 20-го сверху. Обращаться с массивом пикселей можно как с любым другим свойством: изменять цвет, задавая пикселю новое значение, или определять его цвет по хранящемуся в нём значению. Изменяя цвет пикселей, можно нарисовать некоторое изображение по отдельным точкам. Например, установить зеленый цвет для пикселя на форме можно следующим образом:

```
Forml .Canvas .Pixels [10, 20] :=clGreen;
```

4 Перо Pen

С помощью класса *TPen* создается объект-перо, служащий для вычерчивания линий. Свойства класса описаны ниже (табл. 8.3).

Таблица 8.3

Свойства класса *TPen*

Свойство	Пояснение
Property Color: TColor;	Цвет вычерчиваемых пером линий. По умолчанию цвет пера clBlack
Property Handle: Integer;	Дескриптор пера. Используется при непосредственном обращении к API-функциям Windows
Property Mode: TPenMode;	Определяет способ взаимодействия линий с фоном (см. ниже)
Property Style: TPenStyle;	Определяет стиль линий. Учитывается только для толщины линий 1 пиксель. Для толстых линий стиль всегда psSolid (сплошная)
Property Width: Integer;	Толщина линий в пикселях экрана

Свойство *Mode* может принимать одно из следующих значений, описанных ниже (табл. 8.4).

Таблица 8.4

Значения свойства *Mode*

Значение	Пояснение
pmBlack	Линии всегда черные. Свойства Color и Style игнорируются
pmWhite	Линии всегда белые. Свойства Color и Style игнорируются
pmNop	Цвет фона не меняется (линии не видны)
pmNot	Инверсия цвета фона. Свойства Color и Style игнорируются
pmCopy	Цвет линий определяется свойством Color пера. Режим по умолчанию
pmNotCopy	Инверсия цвета пера. Свойство Style игнорируется
pmMergePenNot	Комбинация цвета пера и инверсного цвета фона
pmMaskPenNot	Комбинация общих цветов для пера и инверсного цвета фона. Свойство Style игнорируется
pmMergeNotPen	Комбинация инверсного цвета пера и фона
pmMaskNotPen	Комбинация общих цветов для инверсного цвета пера и фона. Свойство Style игнорируется
pmMerge	Комбинация цветов пера и фона
pmNotMerge	Инверсия цветов пера и фона. Свойство Style игнорируется
pmMask	Общие цвета пера и фона
pmNotMask	Инверсия общих цветов пера и фона
pmXor	Объединение цветов пера и фона операцией XOR
pmNotXor	Инверсия объединения цветов пера и фона операцией XOR

Свойство *Style* пера может принимать следующие значения:

1) psSolid − сплошная линия;
2) psDash − штриховая линия;
3) psDot − пунктирная линия;
4) psDashDot − штрихпунктирная линия;

5) psDashDotDot − линия, чередующая штрих и два пунктира;

6) psClear − отсутствие линии;

7) psInsideFrame − сплошная линия (*Width*>1), допускающая цвета, отличные от палитры *Windows*.

Свойство *Width* устанавливает толщину пера в пикселях. Чтобы перо было красным, толщиной в один пиксель и давало пунктирную линию, нужно выполнить следующие команды:

```
Form1.Canvas.Pen.Color := clRed;
Form1.Canvas.Pen.Width:=1;
Form1.Canvas.Pen.Style:= psDot;
```

Для записи составных имен удобно использовать оператор *with*:

```
with Form1.Canvas.Pen do
begin
Color := clRed;
Width:=1;
Style:= psDot;
end;
```

5 Рисование линий

Перо может передвигаться по поверхности рисования без прорисовки линии. При таком способе передвижения пера используется метод *MoveTo*. Следующая строка кода передвигает перо в точку с координатами 23,56:

```
Form1.Canvas.MoveTo(23, 56);
```

Чтобы соединить две позиции пера прямой линией, используется метод *LineTo*. *LineTo* требует просто указания координат конечной точки, и перо чертит прямую линию от текущей позиции до новой позиции:

```
Form1.Canvas.LineTo(100, 150);
```

6 Кисть Brush

Объекты класса *TBrush* (кисти) служат для заполнения внутреннего пространства замкнутых фигур. Свойства класса описаны ниже (табл. 8.5).

Таблица 8.5

Свойства класса *TBrush*

Свойство	Пояснение
Property Bitmap: TBitmap;	Содержит растровое изображение, которое будет использоваться кистью для заполнения. Если это свойство определено, свойства Color и Style игнорируются
Property Color: TColor;	Цвет кисти
Property Handle: Integer;	Дескриптор кисти. Используется при непосредственном обращении к API-функциям Windows

Свойство	Пояснение
Property Style: TbrushStyle;	Стиль кисти

Свойство *Style* кисти может принимать значения, показанные ниже (рис. 8.1).

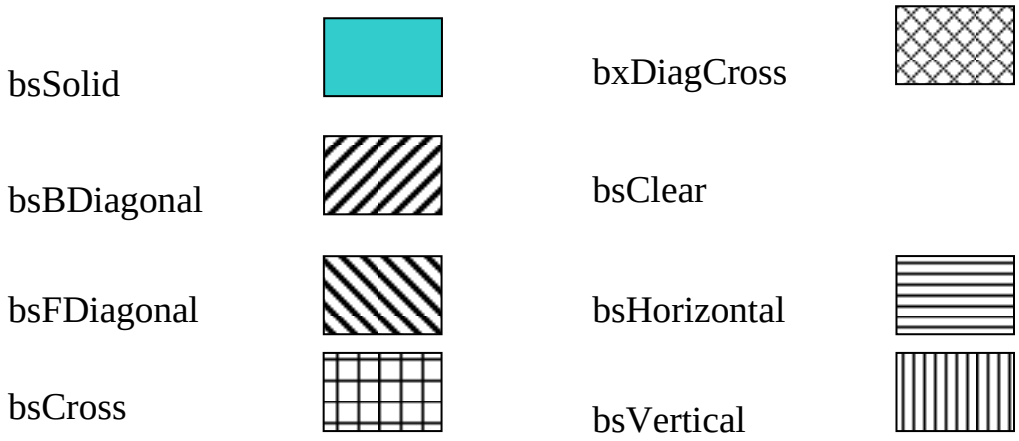

Рис.8.1. Значения свойства *Style* кисти

7 Класс TFont

С помощью класса *TFont* создается объект-шрифт для любого графического устройства (экрана, принтера, плоттера и т.п.). Свойства класса описаны ниже (табл. 8.6).

Таблица 8.6

Свойства класса *TFont*

Свойство	Пояснение
Property Charset: TFontCharSet;	Набор символов. Для русскоязычных программ это свойство обычно имеет значение DEFAULT_CHARSET или RUSSIAN_CHARSET. Используется значение OEM_CHARSET для отображения текста MS-DOS (альтернативная кодировка)
Property Color: TColor;	Цвет шрифта
Property Handle: TFont;	Дескриптор шрифта. Используется при непосредственном обращении к API-функциям Windows
Property Height: Integer;	Высота шрифта в пикселях экрана
Property Name: TfontName;	Имя шрифта. По умолчанию имеет значение MS Sans Serif
Property Pitch: TFontPitch;	Определяет способ расположения букв в тексте: значение fpFixed задает текст, при котором каждая буква имеет одинаковую ширину; значение fpVariabel определяет

Свойство	Пояснение
	пропорциональный текст, при котором ширина буквы зависит от ее начертания, fpDedault определяет ширину, принятую для текущего шрифта
Property PixelPerInch: Integer;	Определяет количество пикселей экрана на один дюйм реальной длины. Это свойство не следует изменять, так как оно используется системой для обеспечения соответствия экранного шрифта шрифту принтера
Property Size: integer;	Высота шрифта в пунктах (1/72 дюйма). Изменение этого свойства автоматически изменяет свойство Height, и наоборот
Property Style: TFontStyles;	Стиль шрифта. Может принимать значение как комбинацию следующих признаков fsBold (жирный), fsItalic (курсив), fsUnderline (подчеркнутый),fsStrikeOut (перечеркнутый)

Для некоторых случаев может оказаться полезным метод

Procedure Assign(Source: TPersistent);

С помощью значения свойств шрифтового объекта *Source* присваиваются свойствам текущего шрифта. Метод не изменяет свойство *PixelPerInch*, поэтому его можно использовать для создания шрифта принтера по экранному шрифту, и наоборот.

8 Работа с цветом

Количество бит, которое поддерживает графическая среда, связано с количеством цветов, которое может быть отображено на экране дисплея в каждый момент времени. Раньше, когда пиксели могли находиться только в состоянии «включено» или «выключено», для поддержки изображения на экран требовалось количество бит, равное количеству пикселей на экране. Если экран имел разрешающую способность 200x300 пикселей, то для сохранения изображения требовалось 60000 бит (или 7500 байт). С каждым добавленным битом на пиксель количество отображаемых на экране уникальных цветов удваивается, однако при этом возрастает объем памяти, необходимой для поддержания изображения на экране.

Возникает вопрос, каково соответствие между цветом и битовым набором. Установление этого соответствия возможно двумя способами. Первый из них использует палитру. Палитра в графике – это множество цветов, которыми можно пользоваться на экране. Количество цветов в палитре определяется числом уникальных битовых наборов (которое равно 2^k, где k – количество бит, поддерживаемых графической средой). Для использования нового, отсутствующего в палитре цвета нужно заменить один из цветов палитры. Это единственная возможность манипуляции палитрой, определяемая графическими режимами в *Delphi*. Палитрой можно манипулировать только в режимах с 256 цветами, что соответствует 8 битам.

Второй способ основан на понятии истинного (*true*) цвета. Истинный цвет определяется как 16 миллионов цветов, которые могут быть получены совместным сложением всех комбинаций зеленого, красного и синего. В компьютерах, которые могут использовать 24-битовый цвет, имеется возможность хранения величин, представляющих цвет каждого пикселя, вместо использования таблицы перекодировки цвета или установки палитры. Каждый основной цвет может иметь интенсивность от 0 до 255. Количество всех возможных комбинаций красного, зеленого и синего равно 256^3= 16 777 216. При этом подходе любой цвет можно определить числом от 0, что соответствует черному, до 16 777 216, что соответствует белому. В таблице 8.7 перечисляются некоторые другие стандартные цвета в системе *RGB* (*Red*, *Green*, *Blue* – красный, зеленый, синий).

Таблица 8.7

Значения цветов *RGB*

Цвет	Значение		
	красного	зеленого	синего
Красный	255	0	0
Зеленый	0	255	0
Синий	0	0	255
Желтый	255	255	0
Фиолетовый	255	0	255

Если ваша среда не поддерживает истинный 24-битный цвет, *Delphi* подбирает цвет, ближайший к выбранному по стандарту *RGB* значению. Большая часть графических систем *PC* использует для своих режимов отображения палитру.

В следующем примере используем три линейки прокрутки, предоставляющие пользователю возможность изменять количество красного, зеленого и синего в цвете фона формы. Используем функцию *RGB* для автоматического построения требуемого цвета исходя из количества красного, зеленого и синего. Добавим дополнительную линейку прокрутки для демонстрации интенсивности шкалы яркости. Движение последней линейки устанавливает значения красного, зеленого и синего, равные одному и тому же значению, соответствующему серому цвету. Каждая линейка прокрутки показывает интенсивность цвета:

```
Unit Main;
Interface
Uses
        SysUtils, WinTypes, WinProcs, Messages, Classes,
        Graphics, Controls, Forms, Dialogs, ExtCtrls,StdCtrls;
Type TForm1 = class (TForm)
        ScrollBar1: TscrollBar;
        ScrollBar2: TscrollBar;
```

```
        ScrollBar3: TscrollBar;
        ScrollBar4: TscrollBar;
        Red:Tlabel;
        Green:Tlabel;
        Blue:Tlabel;
        Clt:Tlabel;
        Procedure ScrollBar1Change(Sender:TObject);
        Procedure UpdateAll;
        Procedure ScrollBar4Change(Sender:TObject);
        Private     {объявления Private }
        Public      { объявления Public }
        End;
Var   Form1:TForm1;
Implementation
{$R*.dfm}
procedure TForm1.UpdateAll;
Begin
Form1.Color:=RGB(ScrollBar1.Position, ScrollBar2.Position, ScrollBar3.Position);
Red.Caption :='Red='+Inttostr (ScrollBar1.Position);
Green.Caption :='Green='+Inttostr (ScrollBar2.Position);
Blue.Caption :='Blue='+Inttostr (ScrollBar3.Position);
Clr.Caption :='RGB Color='+Inttostr (Form1.Color);
End;
procedure TForm1.ScrollBar1Change(Sender:TObject);
Begin
UpdateAll;
End;
procedure TForm1.ScrollBar4Change(Sender:TObject);
Begin
ScrollBar1.Position:= ScrollBar4.Position;
ScrollBar2.Position:= ScrollBar4.Position;
ScrollBar3.Position:= ScrollBar4.Position;
End;
End.
```

Чтобы показать все возможные интенсивности серого цвета, можно использовать следующую процедуру:

```
procedure TForml.DrawGreyClick(Sender: TObject);
var   Count: Integer;
begin
For Count := 0 to 255 do
        begin
Form1.Canvas.Pen.Color:=RGB(Count,Count,Count);          Form1.Canvas.MoveTo
(Count, 0) ;
        Form1 .Canvas.LineTo (Count, 100) ;
        end;
end;
```

Для присваивания значения свойству цвета *Color* можно использовать предопределенный цвет. Предопределенный цвет – это либо постоянный, часто используемый цвет (такой, как *clBlue, clGreen, clRed*), либо стандартный цвет среды *Windows* (такой, как *clBackGround*), который указывает цвет фона рабочего стола.

Delphi предлагает богатый набор графических компонентов и методов. Методы рисования геометрических фигур описаны ниже (табл. 8.8). Цвет, стиль линий и заливки замкнутых фигур устанавливается с помощью свойств пера *Pen* и кисти *Brush*.

Таблица 8.8

Методы рисования геометрических фигур

Метод	Пояснение
procedure Arc(x1, yl, x2, y2, x3, y3, x4, y4: Integer);	Чертит дугу эллипса в охватывающем прямоугольнике (x1, yl) − (x2, y2). Начало дуги лежит на пересечении эллипса и луча, проведенного из его центра в точку (x3, y3), а конец − на пересечении с лучом из центра в точку (x4, y4). Дуга чертится против часовой стрелки
procedure Chord (x1, yl, x2, y2, x3, y3, x4, y4: Integer) ;	Чертит сегмент эллипса в охватывающем прямоугольнике (x1, yl) − (x2, y2). Начало дуги сегмента лежит на пересечении эллипса и луча, проведенного из его центра в точку (x3, y3), а конец − на пересечении с лучом из центра в точку (x4, y4). Дуга сегмента чертится против часовой стрелки, а начальная и конечная точки дуги соединяются прямой

Окончание табл. 8.8

Метод	Пояснение
procedure Ellipse (x1, yl, x2,. y2: Integer);	Чертит эллипс в охватывающем прямоугольнике (x1, y1) − (x2, y2). Если задать координаты квадрата, то будет нарисована окружность
procedure FrameRect (const Rect: TRect);	Очерчивает границы прямоугольника Rect текущей кистью толщиной в 1 пиксель без заполнения внутренней части прямоугольника
procedure LineTo(x, y: Integer) ;	Чертит линию от текущего положения пера до точки (x, y)
procedure MoveTo(x, y: Integer) ;	Перемещает перо в положение (x, y) без вычерчивания линий

procedure Pie (x1, yl, x2, y2, x3, y3, x4, y4: LongInt) ;	Рисует сектор эллипса в охватывающем прямоугольнике (x1, yl) − (x2, y2). Начало дуги лежит на пересечении эллипса и луча, проведенного из его центра в точку (x3, y3), а конец − на пересечении с лучом из центра в точку (x4, y4). Дуга чертится против часовой стрелки. Начало и конец дуги соединяются отрезками прямых линий с центром эллипса
procedure Polygon (Points: array of TPoint);	Вычерчивает пером многоугольник по точкам, заданным в массиве Points. Конечная точка соединяется с начальной точкой и многоугольник заполняется кистью
procedure Polyline (Points: array of TPoint);	Вычерчивает пером ломаную линию по точкам, заданным в массиве Points
procedure Rectangle (x1, yl, x2, y2: integer);	Вычерчивает и заполняет прямоугольник (x1,yl) − (x2,y2). Для вычерчивания без заполнения используйте FrameRect или Polyline
procedure RoundRect (x1, yl, x2, y2, x3, y3): Integer);	Вычерчивает и заполняет прямоугольник (x1, yl) − (x2, y2) со скругленными углами. Прямоугольник (x1,y1) − (x3,y3) определяет дугу эллипса для округления углов

Помимо рисования отдельных примитивов, с помощью методов можно выполнять операции копирования, закраски произвольных фигур и др. (табл. 8.9).

Таблица 8.9

Методы для работы с изображениями

Метод	Пояснение
procedure BrushCopy (const Dest: TRect; Bitmap: TBitmap; const Source: TRect; Color: TColor) ;	Копирует часть изображения Source на участок канвы Dest. Color указывает цвет в Dest, который должен заменяться на цвет кисти канвы. Метод введен для совместимости с ранними версиями Delphi. Вместо него следует пользоваться классом TImageList
procedure CopyRect (Dest: TRect; Canvas: TCanvas; Source: TRect);	Копирует изображение Source канвы Canvas в участок Dest текущей канвы. При этом разнообразные специальные эффекты достигаются с помощью свойства CopyMode
procedure Draw (X, Y: Integer; Graphic: TGraphic) ;	Осуществляет прорисовку графического объекта Graphic так, чтобы левый верхний угол объекта расположился в точке (x, y)
procedure DrawFocusRect (const Rect: TRect);	Прорисовывает прямоугольник с помощью операции XOR, поэтому повторная прорисовка

Метод	Пояснение
	уничтожает ранее вычерченный прямоугольник. Используется в основном для прорисовки нестандартных интерфейсных элементов при получении ими фокуса ввода и при потере его
procedure Lock;	Блокирует канву в многопоточных приложениях для предотвращения использования канвы в других потоках команд
procedure Refresh;	Устанавливает в канве умалчиваемые шрифт, перо и кисть
procedure StretchDraw (const Rect: TRect; Graphic: Tgraphic);	Вычерчивает и при необходимости масштабирует графический объект Graphic так, чтобы он полностью занял прямоугольник Rect
function TextExtent (const Text: String): TSize;	Возвращает ширину и высоту прямоугольника, охватывающего текстовую строку Text
function TextHeight (const.Text: String): Integer;	Возвращает высоту прямоугольника, охватывающего текстовую строку Text
procedure TextOut(x, y: Integer; const Text: String);	Выводит текстовую строку Text так, чтобы левый верхний угол прямоугольника, охватывающего текст, располагался в точке (x, y)

Окончание табл. 8.9

Метод	Пояснение
procedure TextRect (Rect: TRect; x, y: Integer; const Text: String) ;	Выводит текстовую строку Text так, чтобы левый верхний угол прямоугольника, охватывающего текст, располагался в точке (x, y). Если при этом какая-либо часть надписи выходит из границ прямоугольника Rect, она отсекается и не будет видна
function TextWidth (const Text: String): Integer;	Возвращает ширину прямоугольника, охватывающего текстовую строку Text
function TryLock: Boolean;	Пытается заблокировать канву. Если канва не была заблокирована другим потоком команд, возвращает True, в противном случае ничего не делает и возвращает False
Procedure Unlock;	Уменьшает на 1 счетчик блокировок канвы
Procedure FillRect (const Rect: TRect);	Заполняет текущей кистью прямоугольную область Rect, включая ее левую и верхнюю границы, но, не затрагивая правую и нижнюю границы

Procedure FloodFill (x, y: Integer; Color: TColor; FillStyle: TfillStyle);	Производит заливку канвы текущей кистью. Заливка начинается с точки (x, y) и распространяется во все стороны от нее. Если FillStyle = fsSurface, заливка распространяет на все соседние точки с цветом Color. Если FillStyle = fsBorder, наоборот, заливка прекращается на точке с этим цветом

9 Классы TGraphic и TPicture

Важное место в графическом инструментарии *Delphi* занимают классы *TGraphic* и *TPicture*.

TGraphic − это абстрактный класс, инкапсулирующий общие свойства и методы трех своих потомков: пиктограммы (*TIcon*), метафайла (*TMetafile*) и растрового изображения (*TBitmap*). Общей особенностью потомков *TGraphic* является то, что обычно они сохраняются в файлах определенного формата. Пиктограммы представляют собой небольшие растровые изображения, снабженные специальными средствами, регулирующими их прозрачность. Для файлов пиктограмм обычно используется расширение *1CO*. Метафайл − это изображение, построенное на графическом устройстве с помощью специальных команд, которые сохраняются в файле с расширения *WMF* или *EMF*. Растровые изображения − это произвольные графические изображения в файлах со стандартным расширением *BMP*. Свойства класса *TGraphic* описаны ниже (табл. 8.10).

Таблица 8.10

Свойства класса *TGraphic*

Свойство	Пояснение
Property Empty: Boolean;	Содержит True, если с объектом не связано графическое изображение
Property Height: Integer;	Содержит высоту изображения в пикселях
Property Modified: Boolean;	Содержит True, если графический объект изменялся
Property Palette: HPALETTE;	Содержит цветовую палитру графического объекта
Property PaletteModified: Boolean;	Содержит True, если менялась цветовая палитра графического объекта
Property Transparent: Boolean;	Содержит True, если объект прозрачен для фона, на котором он изображен
Property Width: Integer;	Содержит ширину изображения в пикселях

Методы класса *TGraphic* описаны ниже (табл. 8.11).

Таблица 8.11

Методы класса *TGraphic*

Метод	Пояснение
procedure	Ищет в буфере межпрограммного обмена

Метод	Пояснение
LoadFrornClipboardFormat (AFormat: Word; AData: THandle; Apalette: HPALETTE);	Clipboard зарегистрированный формат AFormat и, если формат найден, загружает из буфера изображение AData и его палитру APalette
procedure LoadFromFile(ñonst FileName: String);	Загружает изображение из файла FileName
procedure LoadFromStream (Stream: TStream);	Загружает изображение из потока данных Stream
procedure SaveToClipboardFormat (var AFormat: Word; var AData: THandle; var APalette: HPALETTE);	Помещает графическое изображение AData и его цветовую палитру APalette в буфер межпрограммного обмена в формате AFormat
procedure SaveToFile(const FileName: String) ;	Сохраняет изображение в файле FileName
procedure SaveToStream (Stream: TStream);	Сохраняет изображение в потоке Stream

Класс *TPicture* инкапсулирует в себе все необходимое для работы с готовыми графическими изображениями: пиктограммой, растром или метафайлом. Его свойство *Graphic* может содержать объект любого из этих типов, обеспечивая нужный полиморфизм методов класса. Свойства класса *TPicture* описаны ниже (табл. 8.12).

Таблица 8.12

Свойства класса *TPicture*

Свойство	Пояснение
Property Bitmap: TBItmap;	Интерпретирует графический объект как растровое изображение
Property Graphic: TGraphic;	Содержит графический объект
Property Height: Integer;	Содержит высоту изображения в пикселях
Property Icon: TIcon;	Интерпретирует графический объект как пиктограмму
Property Metafile: TMetafile;	Интерпретирует графический объект как метафайл
Property Width: Integer;	Содержит ширину изображения в пикселях

Методы класса *TPicture* описаны ниже (табл. 8.13).

Таблица 8.13

Методы класса *TPicture*

Метод	Пояснение
procedure Assign (Source: TPersistent);	Связывает собственный графический объект Graphic с объектом Source

Метод	Пояснение
procedure LoadFrornClipboardFormat (AFormat: Word; AData: THandle; APalette: HPALETTE);	Ищет в буфере межпрограммного обмена Clipboard зарегистрированный формат AFormat и, если формат найден, загружает из буфера изображение AData и его палитру APalette
procedure LoadFromFile(const FileName: String);	Загружает изображение из файла FileName
class procedure RegisterClipboardFormat(AFormat:Word; AGraphicClass: TGraphicClass) ;	Используется для регистрации в Clipboard нового формата изображения
class procedure RegisterFileFormat (const AExtension, ADescription: String; AGraphicClass: TGraphicClass);	Используется для регистрации нового файлового формата

Окончание табл. 8.13

Метод	Пояснение
class procedure RegisterFileFormatRes (const AExtension: String; ADescriptionReslD: integer; AGraphicClass: TGraphicClass);	Используется для регистрации нового формата ресурсного файла
procedure SaveToClipboardFormat (var AFormat: Word; var AData: THandle; var APalette: HPALETTE);	Помещает графическое изображение AData и его цветовую палитру APalette в буфер межпрограммного обмена в формате AFormat
procedure SaveToFile(const FileName: String);	Сохраняет изображение в файле FileName
class function SupportsClipboardFormat (AFormat: Word): Boolean;	Возвращает True, если формат AFormat зарегистрирован в буфере межпрограммного обмена Clipboard
class procedure UnregisterGraphicClass (Aclass: TGraphicClass);	Делает недоступными любые графические объекты класса AClass

Пример программы с использованием графики

Задание. Изобразить на экране полет стрекозы (рис. 8.2).

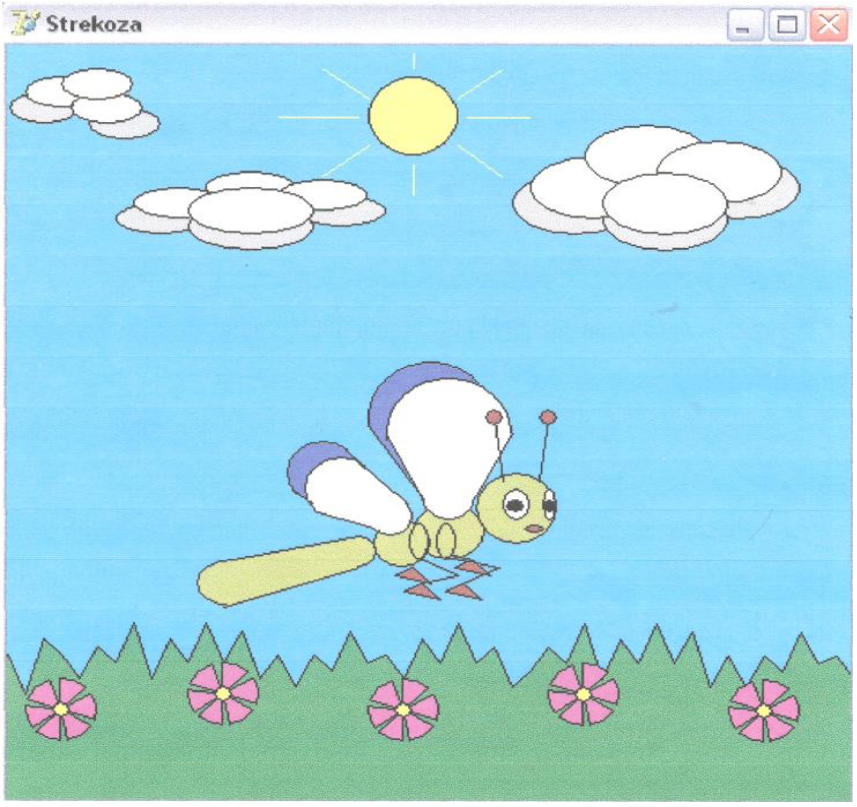

Рис. 8.2. Изображение стрекозы

1. Разработка алгоритма (рис. 8.3, 8.4, 8.5).

Рис. 8.3. Схема алгоритма процедуры TForml.FormActivate(Sender: TObject);

Рис. 8.4. Схема алгоритма процедуры Strekoza (x,y: integer)

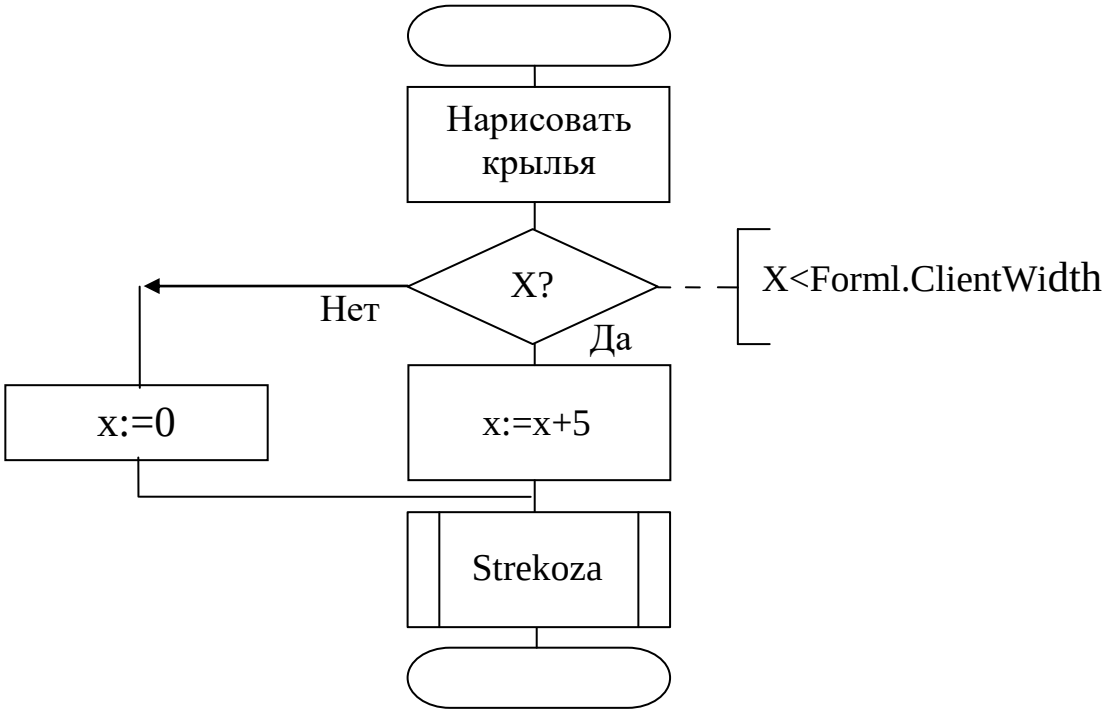

Рис. 8.5. Схема алгоритма процедуры TForm1.TimerlTimer(Sender: Object)

2. Текст программы:

```
unit Strekoza_;
interface
uses
Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs, StdCtrls, ExtCtrls;
type
```

```
TForml = class(TForm)
Timer1: TTimer;
PaintBoxl. TPaintBox;
procedure TimerlTimer(Sender: TObject);
procedure FormActivate(Sender: TObject);
private
{ Private declarations }
public
{ Public declarations }
end;
var     Form1: TForml;
        x,y: integer;
implementation
{$R*.dfm}
procedure Strekoza(x,y: integer);
type Tpol= array[1..3]of TPoint,
const dx = 5;
var pol,pol2,pol3,pol4:Tpol; i: integer;
begin
with Form1.Canvas do begin
//рисуем крылья
Pen.Color: =clBlack;
arc(x+5+dx,200,x+75+dx,270,x+75+dx,208,x+5+dx,240);
moveto(x+5+dx,240); lineto(x+37+dx,288);
arc(x-40+dx,250,x+dx,290,x-10+dx,260,x-30+dx,280);
moveto(x-8+dx,255); lineto(x+7+dx,270); moveto(x-37+dx,281);
lineto(x-25+dx,290);
Brush.Color:=clWhite;
arc(x+35+dx,270,x+65+dx,300,x+35+dx,285,x+65+dx,285);
arc(x+15+dx,210,x+85+dx,280,x+85+dx,245,x+15+dx,250);
moveto(x+15+dx,245);lineto(x+35+dx,285);
moveto(x+85+dx,245);lineto(x+65+dx,285),
floodfill(x+45+dx,245,clblack,fsborder);
arc(x+5+dx,285,x+30+dx,310,x+10+dx,305,x+25+dx,290);
arc(x-30+dx,260,x+10+dx,300,x+dx,270,x-20+dx,290);
moveto(x+8+dx,305);lineto(x-26+dx,294);
moveto(x+27+dx,289);lineto(x-4+dx,260);
floodfill(x+17+dx,297,clblack,fsborder);
Brush.Color:=clBlue;
floodfill(x+9+dx,228,clblack,fsborder);
floodfill(x-28+dx,270,clblack,fsborder);
//рисуем ноги
Brush.Color:=clMaroon;
pol3[l].x:=x+30+dx;
pol3[l].y:=330;
pol3[2].x:=x+40+dx;
pol3[2].y:= 340;
```

```
pol3[3].x:=x+20+dx;
pol3[3].y:=335;
polygon(pol3);
moveto(x+63+dx,325); lineto(x+45+dx,320);
pol4[l].x:=x+63+dx;
pol4[l].y:=325;
pol4[2] x:=x+73+dx;
pol4[2] y:= 335;
pol4[3].x:=x+53+dx;
pol4[3].y:=330;
polygon(pol4);
//рисуем тело стрекозы
Brush.Color:=clOlive;
arc(x-90+dx,325,x-50+dx,355,x-70+dx,325,x-70+dx,355);
moveto(x-70+dx,325); lineto(x+dx,310);
moveto(x-75+dx,3S5); lineto(x+dx,330);
arc(x-10+dx,310,x+10+dx,330,x+dx,330,x+dx,310);
floodfill(x-67+dx,338,clblack,fsborder);
arc(x+8+dx,300,x+40+dx,330,x+10+dx,305,x+31+dx,290);
arc(x+30+dx,290,x+70+dx,325,x+40+dx,295,x+60+dx,295);
ellipse(x+28+dx,302,x+39+dx,325);
ellipse(x+43+dx,302,x+54+dx,325);
floodfill(x+55+dx,305,clblack,fsborder);
floodfill(x+20+dx,320,clblack,fsborder);
ellipse(x+65+dx,270,X+110+dx,315);
//рисуем глаза
Brush.Color:=clWhite,
ellipse(x+80+dx,280,x+95+dx,300);
eIlipse(x+102+dx,280,x+110+dx,300);
Brush.Color:=clBlack;
eilipse(x+82+dx,287,x+92+dx,295);
ellipse(x+102+dx,287,x+110+dx,295);
Brush.Color:=clMaroon;
ellipse(x+92+dx,303,x+103+dx,308);
moveto(x+80+dx,275); lineto(x+75+dx,235); moveto(x+100+dx,275);
lineto(x+105+dx,235); ellipse(x+70+dx,230,x+80+dx,240);
ellipse(x+100+dx,230,x+110+dx,240);
moveto(x+35+dx,325); lineto(x+55+dx,335); lineto(x+35+dx,340);
moveto(x+48+dx,323); lineto(x+78+dx,330); lineto(x+58+dx,340);
pol[l].x:=x+35+dx;
pol[l].y:=340;
pol[2].x:=x+45+dx;
pol[2].y:= 350;
pol[3].x:=x+25+dx;
pol[3].y:=345;
polygon(pol);
pol2[l].x:=x+58+dx;
```

```
pol2[l].y:=340;
pol2[2].x:=x+68+dx;
pol2[2].y:= 350;
pol2[3].x:=x+48+dx;
pol2[3].y:=345;
polygon(poI2);
end;
end;
```
{Процедура перемещения изображения с учетом координат окна. Данная процедура вызывается через определенный промежуток времени, который устанавливается в свойстве *Timer1.Interval* в миллисекундах}
```
procedure TForm1.TimerlTimer(Sender: TObject);
begin
with Forml.Canvas do
begin
Pen.Color:=clAqua;
Brush.Color:=clAqua;
Rectangle(0,150,600, 360);
end,
if x < Forml.ClientWidth then x := x+5
    else  x :=0;
Strekoza(x,y);
end;
procedure TForml.FormActivate(Sender: TObject);
begin
with Canvas do begin
Pen.Color:=clBlack;
//рисуем фон
Brush.Color:=clGreen;
moveto(0,385); lineto(10,410); lineto(20,375); lineto(30,385); lineto(40,400);
lineto(50,380); lineto(60,390); lineto(70,365), lineto(80,400); lineto(90,375);
lineto(100,390); lineto(l10,365); lineto(120,390); lineto(130,370); lineto(140,400);
lineto(150,385); lineto(160,405); lineto(170,385); lineto(180,395); lineto(190,370);
lineto(200,390); lineto(210,385); lineto(220,400); lineto(230,375); lineto(240,390),
lineto(250,365); lineto(260,390); lineto(270,380); lineto(280,405), lineto(290,395);
lineto(300,380); lineto(310,390); lineto(320,365); lineto(330,400); lineto(340,375);
lineto(350,390); lineto(360,365); lineto(370,390); lineto(380,370); lineto(390,405);
lineto(400,385); lineto(410,405); lineto(420,385); lineto(430,395); lineto(440,370);
lineto(450,390); lineto(460,385); Iineto(470,400); lineto(480,375); lineto(490,390);
lineto(500,365); lineto(510,390), lineto(520,380), Iineto(530,405); lineto(540,395);
floodfill(5,415,clblack,fsborder);
//рисуем цветы
Brush.Color:=clFuchsia;
pie(400,400,440,440,415,405,405,415);
pie(400,400,440,440,425,415,420,410);
pie(400,400,440,440,435,425,425,417);
```

```
pie(400,400,440,440,415,425,420,435),
pie(400,400,440,440,403,418,405,431),
pie(400,400,440,440,423,437,438,431);
pie(300,390,340,430,315,395,305,405);        pie(300,390,340,430,325,405,320,400);
pie(300,390,340,430,335,415,325,407);        pie(300,390,340,430,315,415,320,425);
pie(300,390,340,430,303,408,305,421); pie(300,390,340,430,323,427,338,421);
pie(200,400,240,440,215,405,205,415);        pie(200,400,240,440,225,415,220,410);
pie(200,400,240,440,235,425,225,417);        pie(200,400,240,440,215,425,220,435);
pie(200,400,240,440,203,418,205,431); pie(200,400,240,440,223,437,238,431),
pie(100,390,140,430,l15,395,105,405);        pie(100,390,140,430,125,405,120,400);
pie(100,390,140,430,135,415,125,407);
pie(100,390,140,430,l15,415,120,425);         pie(100,390,140,430,103,408,I05,421);
pie(100,390,140,430,123,427,138,421);
pie(10,400,50,440,25,405,15,415);
pie(10,400,50,440,35,415,30,415);
pie(10,400,50,440,45,425,35,417);
pie(10,400,50,440,25,425,30,435);
pie(10,400,50,440,13,418,15,431);
pie(10,400,50,440,33,437,48,431);
Brush.Color:=clYellow;
ellipse(415,415,425,425); ellipse(315,405,325,415); ellipse(215,415,225,425);
ellipse(115,405,125,415);
ellipse(25,415,35,425);
//рисуем небо
Brush.Color:=clAqua;
floodfill(l,270,clblack,fsborder);
Brush.Color:=clYellow;
ellipse(200,20,250,70);
Pen.Color:=clYellow;
moveto(255,45); llneto(290,45); moveto(150,45); lineto(195,45); moveto(225,5);
lineto(225,15); moveto(225,75); lineto(225,95); moveto(200,33); lineto(174,14);
moveto(250,33); lineto(275,14); moveto(200,63); lineto(174,83); moveto(250,63);
lineto(275,83);
Pen.Color:=clBlack;
Brush.Color:=clSilver;
ellipse(280,80,350,120); ellipse(370,70,440,110); ellipse(330,90,400,130);
ellipse(60,l 00,110,120), ellipse(160,95,210,115); ellipse(100,100,170,130);
ellipse(l, 30,40,50); ellipse(45,40,85,60);
Brush.Color:=clWhite;
ellipse(290,70,360,110); ellipse(320,50,390,90); ellipse(360,60,430,100);
ellipse(330,80,400,120); ellipse(70,90,120,110); ellipse(l10,80,160,100);
ellipse(150,85,200,105); ellipse(100,90,170,120), ellipse(10,20,50,40);
ellipse(35,30,75,50); ellipse(30,15,70,35);
end;
x:=0; y:=100;
```

```
Timerl.Interval:=100;
Timerl.Enabled:=true;
end;
end.
```

10 Контрольные вопросы

1. Какие Вы знаете классы в *Delphi* для работы с изображениями?
2. С помощью какого инструмента можно нарисовать изображение по отдельным точкам?
3. Каким образом можно установить нужный цвет?
4. Какие основные свойства пера Вы знаете?
5. Какие основные свойства кисти Вы знаете?
6. Какие основные свойства шрифта Вы знаете?
7. Каким образом определяются координаты области рисования?
8. Какими методами можно нарисовать линию?
9. С помощью каких методов можно закрасить область?
10. Какие методы нужно использовать, чтобы нарисовать геометрические фигуры?
11. Какие основные свойства и методы имеет класс *TGraphic*?
12. Какие основные свойства и методы имеет класс *TPicture*?

Задания к лабораторной работе №8

1. Написать программу, выводящую на экран изображение движущегося поезда. Фоном могут быть придорожные строения, столбы, деревья.
2. Написать программу, выводящую на экран изображение движущейся грузовой машины. Фоном могут быть придорожные строения, столбы, деревья.
3. Написать программу, выводящую на экран изображение движущегося снегохода. Фоном могут быть снежная равнина, столбы, деревья.
4. Написать программу, выводящую на экран изображение движущейся легковой машины. Фоном могут быть придорожные строения, столбы, деревья.
5. Написать программу, выводящую на экран изображение плывущей яхты. Фоном могут быть море, берег, деревья.
6. Написать программу, выводящую на экран изображение плывущего корабля. Фоном могут быть море, берег, деревья.
7. Написать программу, выводящую на экран изображение плывущей подводной лодки. Фоном может быть изображение морского дна.
8. Написать программу, выводящую на экран изображение плывущей рыбы. Фоном может быть изображение морского дна.
9. Написать программу, выводящую на экран изображение летящего самолета. Фоном может быть изображение неба, поверхности земли, облаков.
10. Написать программу, выводящую на экран изображение летящей ракеты. Фоном может быть изображение неба, поверхности земли, облаков.
11. Написать программу, выводящую на экран изображение летающего дельтоплана. Фоном может быть изображение неба, поверхности земли, облаков.
12. Написать программу, выводящую на экран изображение летящего вертолета. Фоном может быть изображение неба, поверхности земли, облаков.
13. Написать программу, выводящую на экран изображение летящего космического корабля. Фоном может быть изображение неба, земли, луны, звезд.
14. Написать программу, выводящую на экран изображение летящего воздушного шара. Фоном может быть изображение неба, поверхности земли, облаков.
15. Написать программу, выводящую на экран изображение спускающегося на землю парашютиста. Фоном может быть изображение неба, поверхности земли, облаков.
16. Написать программу, выводящую на экран изображение плывущей парусной лодки. Фоном могут быть море, берег, деревья.
17. Написать программу, выводящую на экран изображение работающей ветряной мельницы. Фоном могут быть дома, деревья.
18. Написать программу, выводящую на экран изображение движущегося по дороге подъемного крана. Фоном могут быть придорожные строения, столбы, деревья.
19. Написать программу, выводящую на экран изображение поздравительной открытки к празднику Нового года с динамическими элементами (мигающие надписи, гирлянды, салют и т.п.).

20. Написать программу, выводящую на экран изображение поздравительной открытки к празднику Победы с динамическими элементами (мигающие надписи, салют и т.п.).

21. Написать программу, выводящую на экран изображение экологического плаката с динамическими (мигающими или движущимися) элементами.

22. Изобразить на экране действующие песочные часы. Учесть законы физики: количество песка, вытекающего из верхней колбы, равно количеству песка, притекающего в нижнюю колбу. Установку времени перетекания песка выполнить после запуска программы.

23. Изобразить на экране работающие часы со стрелочными индикаторами (часовая, минутная, секундная стрелки). Вывести также дату и день недели.

24. Изобразить на экране картину праздничного салюта: взлеты, взрывы, падения пиротехнических ракет.

25. Изобразить на экране бухгалтерские счеты и реализовать на них демонстрацию операций сложения и вычитания. Числа и знак операции вводить с клавиатуры.

26. Изобразить на экране действующий конвейер, транспортирующий какие-либо однотипные предметы.

27. Написать программу, выводящую на экран изображение плаката с динамическими (мигающими или движущимися) эффектами, рекламирующего профессию программиста.