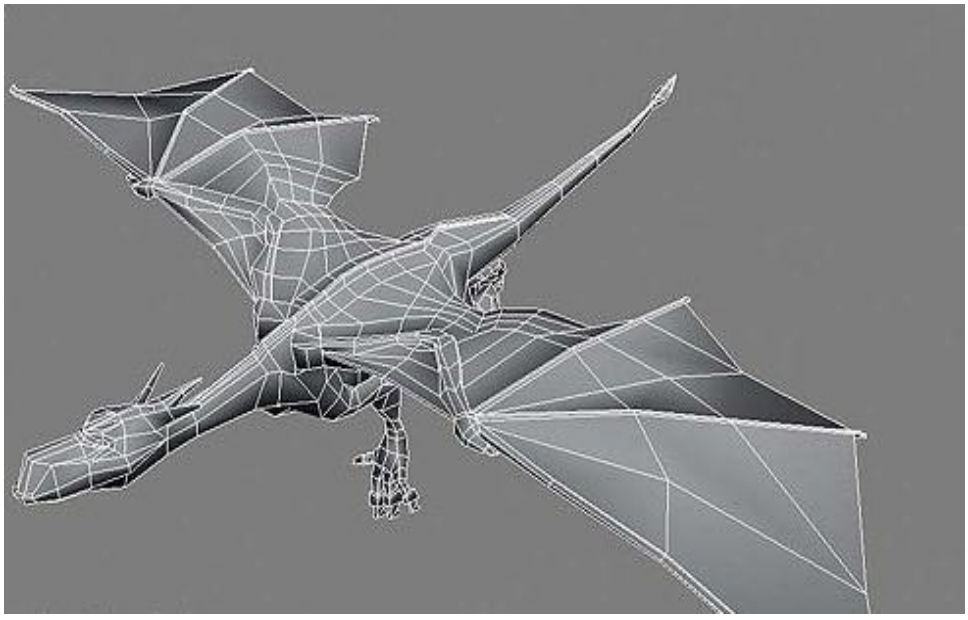
ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ
СЕВЕРО-КАВКАЗСКИЙ ФИЛИАЛ
ФЕДЕРАЛЬНОГО ГОСУДАРСТВЕННОГО ОБРАЗОВАТЕЛЬНОГО БЮДЖЕТНОГО
УЧРЕЖДЕНИЯ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
МОСКОВСКОГО ТЕХНИЧЕСКОГО УНИВЕРСИТЕТА СВЯЗИ И ИНФОРМАТИКИ

Дизайн графических и пользовательских интерфейсов

Методическое пособие по выполнению лабораторных работ



Ростов-на-Дону
2019

УДК 004.925

Дизайн графических и пользовательских интерфейсов . Методическое пособие по выполнению лабораторных работ . / Моск. техн. ун-т связи и информатики, Сев.-Кавк. филиал. – Ростов н/Д, 2019, 6 с.

В пособии даются организационно-методические указания на лабораторный работы практикум, приводятся достаточно подробная методика выполнения заданий и порядок выполнения и оформления отчёта.

Предназначено для студентов обеих форм обучения, изучающих дисциплину «Дизайн графических и пользовательских интерфейсов », а также может быть полезно всем остальным студентам, желающим самостоятельно освоить программирование графики.

Обсуждено и утверждено на заседании кафедры ИВТ (протокол заседания кафедры №1 от 26.08.2019).

СОДЕРЖАНИЕ

Лабораторная работа 1. Создание Windows-приложения с различными элементами управления и отображения информации

Постановка задачи

В данной лабораторной работе нам предстоит написать *приложение* – для просмотра графических файлов. Это *приложение* должно обладать всеми необходимыми для нашей задачи качествами:

- выбирать необходимый файл;
- просматривать в необходимом размере;

Реализация

Для начала откройте **Lazarus** с новым проектом. Форму называем **fMain**, для этого в инспекторе объектов находим свойство **Name**, меняем его на **fMain,** на вкладке **Проект** выбираем пункт «**Сохранить проект как**…» и сохраняем проект в папку **Prototip02** под именем **MyImageBrowser**, модулю главной формы присваиваем имя Main вместо предлагаемого по умолчанию unit1.

Для повышения быстродействия и сокращения требуемого объёма памяти необходимо сразу же отключить от проекта вставку отладочной информации.

Нужно выбрать команду "**Проект -> Параметры проекта**", в разделе "**Параметры компилятора**" перейти на подраздел "**Отладка**" и убрать флажок "**Генерировать отладочную информацию для GDB**". Это позволит нашей программе сразу "похудеть" - от 15 *мегабайт* готового продукта, до менее чем 2-х *мегабайт*.

Компонент TImage и диалоги TOpenPictureDialog, TSavePictureDialog

Данную лабораторную работу посвятим работе с графикой, у Lazarus есть и такая возможность.

Сделаем следующие настройки свойств у главной формы:

Caption = Просмотр изображений

Height = 450

Position = poDesktopCenter

Width = 700

На вкладке **Additional Палитры компонентов** найдите *компонент* TImage:

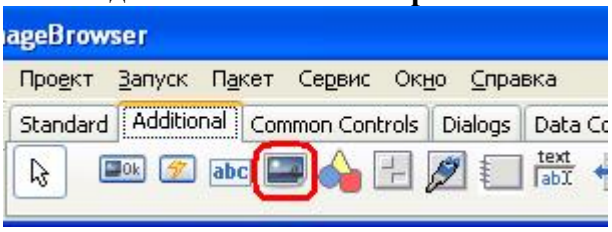


Рис. 1. Компонент TImage

Установите его на форму. *Компонент* TImage является контейнером для показа изображений - графических файлов. Сам *компонент* по умолчанию имеет такой же цвет, как форма, а его границы обозначены пунктирной линией. TImage позволяет загружать графические файлы как на этапе проектирования, так и в процессе работы программы. В первом случае изображение считывается, и в дальнейшем становится частью проекта, его ресурсом. Во втором случае изображение не становится частью проекта, оно занимает *память* с момента его открытия пользователем, и до закрытия программы, либо пока *пользователь* не загрузит другое изображение. Мы опробуем оба варианта, но для начала настроим положение и размеры компонента. Имя оставим по умолчанию - Image1. Обратим внимание на свойства компонента

в **Инспекторе объектов**.

AutoSize - автоматический размер. Если равно True, компонент TImage будет подгонять свой размер под реальный размер изображения.

Center - при значении True выводит изображение по центру компонента.

Picture - основное свойство, имеет специальный тип TPicture и содержит само изображение. С помощью этого свойства можно загрузить изображение из файла в компонент и во время проектирования, и во время выполнения программы. Можно также сохранить изображение в файл, хотя это имеет смысл только для графических редакторов, позволяющих создавать и изменять изображения.

Proportional - при значении True изображение будет изменять размеры, сохраняя пропорции изображения - отношение высоты к ширине. Если значение False, то при изменении размеров изображение может быть искажено: чрезмерно вытянуто в длину или в ширину.

Stretch - при значении True размер изображения будет подстраиваться под размер TImage. Обычно либо картинку подгоняют под размер контейнера, либо контейнер под размер картинки. В первом случае True будет у свойства AutoSize, во втором - у Stretch.

Transparent - прозрачность. Действует только на битовые матрицы, на файлы с форматом bmp. Применяется, когда нужно спрятать фоновый цвет рисунка. Допустим, в редакторе **Paint** вы на белом фоне нарисовали синий круг. Сохранили в bmp-файл, который затем загрузили в TImage. Так вот, если Transparent = False, то вы получите картинку, как есть - синий круг в белом прямоугольном фоне. Если же Transparent = True, то фоновый цвет будет заменяться на цвет компонента под TImage, то есть, станет прозрачным. Фоновым считается цвет самого нижнего левого пиксела - если он белый, то этот цвет отображаться не будет.

Нужными нам методами обладает сложное свойство Picture, которое само является объектом. Подобно массивам строк, свойство Pictureимеет такие методы, как LoadFromFile и SaveToFile, хотя последний метод применяется в основном, в графических редакторах.

Установим следующие свойства компонента Image1:

Left, Top = 5
Height = 385
Width = 690
AutoSize = False
Center = True
Proportional = True
Stretch = True

Остальные свойства изменять не будем. Попробуем загрузить картинку. Выделите свойство Picture и щелкните по кнопке "**…**" справа от него. Откроется диалог загрузки изображений. Нажав кнопку "**Загрузить**", вы откроете стандартный диалог "**Открыть файл изображений**". Здесь вы можете выбрать желаемый *файл*, просматривая в правой части диалога миниатюрное изображение файла. Выбрав нужный *файл*, нажмите кнопку "**Открыть**". Выбранный *файл* отобразится в **Диалоге загрузки изображений**, после чего нажмите кнопку "**ОК**". Выбранное изображение попадет в TImage, причем будет отображено по центру контейнера, сохранит пропорции и подгонит свой размер под размеры контейнера.

Однако нам не нужен фон для формы, нам нужна *программа* для просмотра различных изображений, а такая *программа* загружает картинки по требованию пользователя. Поэтому удалим изображение. Для этого снова выделите Picture и откройте **Диалог загрузки изображений**. Там нажмите на кнопку "**Очистить**", после чего нажмите на "**ОК**". Диалог закроется, *контейнер* Image1 снова будет пуст.

Теперь нам нужен диалог открытия графических файлов - TOpenPictureDialog, который находится на вкладке **Dialogs Палитры компонентов**:

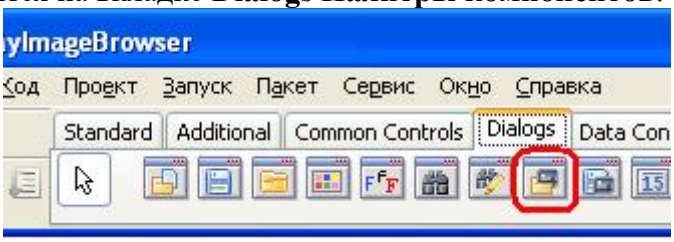


Рис. 2. Компонент TOpenPictureDialog

Подобно другим диалогам, TOpenPictureDialog является невизуальным, его можно установить на любое *место*, например, прямо посреди Image1. Свойств у него также немного, причём все нужные свойства уже заполнены. Откройте, например, **Редактор фильтров** в свойстве Filter, и посмотрите на то обилие графических форматов, с которыми вы можете работать! Однако имя диалога слишком длинное, так что переименуем его свойство Name на OPD.

Также внизу нам понадобится кнопка **TButton**, которой мы дадим имя bOpen, а в свойстве Caption напишем **Загрузить изображение**. Чтобы надпись поместилась на кнопке, установите её свойство Width равным 160, или просто растяните кнопку с помощью мыши или кнопкой со стрелкой вправо, удерживая нажатой кнопку **<Shift>**.

Давайте воспользуемся примером предыдущего приложения, и подкорректируем свойство Anchors у контейнера и кнопки: Image1 "привяжите" ко всем сторонам формы, чтобы он мог менять размер вместе с формой, а кнопку "отвяжите" от верхней границы и "привяжите" к нижней.

Сгенерируйте событие OnClick для кнопки, его код:

```
procedure TfMain.bOpenClick(Sender: TObject);
begin
  if OPD.Execute then Image1.Picture.LoadFromFile(OPD.FileName);
end;
```

Как видите, ничего сложного, работа диалога абсолютно похожа на работу других диалогов. Сохраните проект, запустите его, попробуйте загружать различные графические файлы и изменять размеры формы - размеры картинки также должны меняться, сохраняя пропорции:

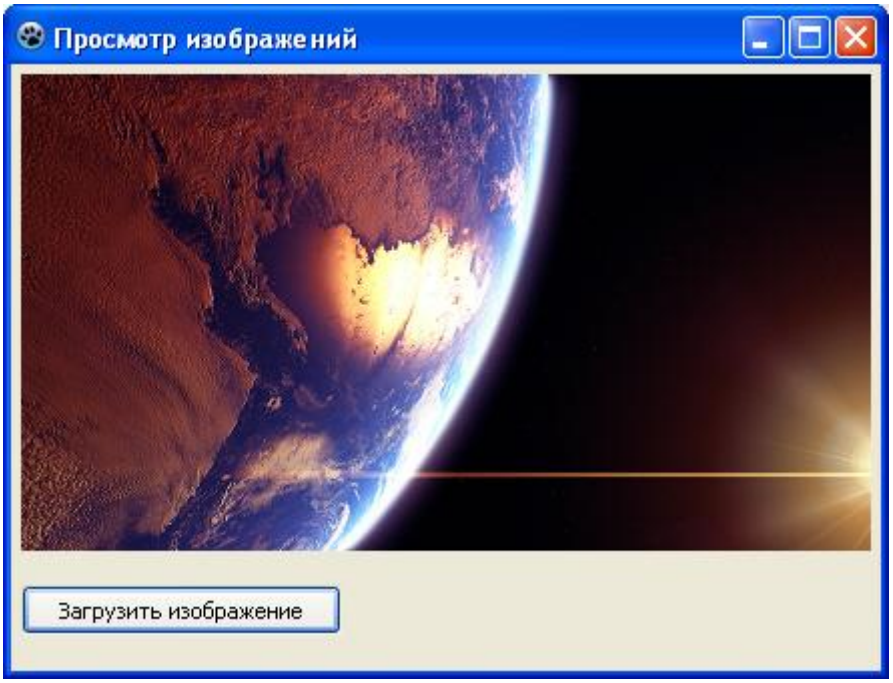


Рис. 3. Программа MyImageBrowser с загруженным изображением и уменьшенным размером

Диалог TSavePictureDialog имеет те же свойства, что и TOpenPictureDialog и предназначен для сохранения изображения в файл. Наша программа только просматривает файлы, а не редактирует их, поэтому здесь надобности в TSavePictureDialog нет. Однако на следующей лекции мы все же используем его, чтобы сохранять файл с изображением под другим именем.

Помимо пройденных диалогов вкладка Dialogs содержит и другие, более специфичные диалоги. Например, диалоги поиска и замены текста, настройки принтера и печати. Однако все диалоги работают схожим образом, так что при необходимости, вы сможете самостоятельно разобраться с работой любого из этих диалогов.

Лабораторная работа 2. Изучение объектной модели холста (Canvas).

Задание

1. Изучить методы создания графических образов средствами Lazarus.
2. Создать приложение – форму с четырьмя кнопками, и полем для рисования, по нажатию первой кнопки на поле появляется движущееся слева направо облако, которое после выхода за левый край формы вновь появляется слева и вновь движется (см. п.2), по нажатию второй кнопки облако останавливается. По нажатию третьей кнопки на поле последовательно, элемент за элементом (не менее 5 элементов), с задержкой 0,5 секунд прорисовывается объект, заданный индивидуальным заданием. По нажатию четвёртой кнопки объект исчезает. Объект прорисовать схематично с использованием процедур рисования и закрашивания. Варианты индивидуальных задний в п.3.

3. По результатам работы сдать отчёт о проделанной работе в формате MS Word и Файл с заархивированным проектом Lazarus, само приложение в архив не включать с целью сокращения размеров архива. В отчёт описать ход выполнения работы, текст разработанного модуля на языке freePascal и скриншоты работы приложения. Дать письменные ответы на контрольные вопросы.

Контрольные вопросы к Лабораторной работе 2

1. Перечислите основные свойства объекта Canvas.
2. На каких компонентах можно создавать графическое изображение в Lazarus?
3. Назовите методы создания графических примитивов в среде Lazarus.Приведите примеры.
4. Что понимают под компьютерной графикой?
5. На какие категории делятся графические изображения?
6. Приведите пример категорий графических изображений и соответствующих им объектов.

1. Средства рисования в Lazarus

При разработке проекта, в котором можно рисовать, в распоряжении программиста находится полотно (холст) свойство Canvas, карандаш свойствоPen, и кисть свойствоBrush.

Свойством Canvas обладают следующие компоненты:

•форма (класс Tform);

•таблица (класс TSringGrid);

•растровая картинка (класс Timage);

•принтер (класс TPrinter).

При рисовании компонента, обладающего свойством Canvas, 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 свойствомWeight. Каждый пиксель может иметь свой цвет. Для доступа к любой точке полотна используется свойство Pixels[X,Y]:TColor. Это свойство определяет цвет пикселя с координатами

X(integer), Y(integer).

Изменить цвета любого пикселя полотна можно с помощью следующего оператора присваивания:

Компонент . Canvas . P i x e l s [X,Y]:= Color ;
где Color переменная или константа типаTcolor.

Таблица 2.1. Значение свойств Color

Константа	Цвет	Константа	Цвет
clBlack	Чёрный	clSilve	Серебристый
clMaroon	Каштановый	clRed	Красный
clGreen	Зелёный	clLime	Салатовый
clOlive	Оливковый	clBlue	Синий
clNavy	Тёмно-синий	clFuchsia	Ярко-розовый
clPurple	Розовый	clAqua	Бирюзовый
clTeal	Лазурный	clWhite	Белый
clGray	Серый		

Определены следующие константы цветов (табл. 2.1).
Цвет любого пикселя можно получить с помощью следующего оператора присваивания:
Color :=Компонент . Canvas . P i x e l s [X,Y] ;
где Color переменная типаTcolor.
Класс цвета точки Tcolor определяется как длинное целоelongint. Переменные этого типа занимают в памяти четыре байта. Четыре байта переменных этого типа содержат информацию о долях синего (B), зелёного (G) и красного
(R) цветов и устроены следующим образом: $00BBGGRR.
Для рисования используются методы класса TCanvas, позволяющие изобразить фигуру (линию, прямоугольник и т. д.) или вывести текст в графическом режиме, и три класса, определяющие инструменты вывода фигур и текстов:
•TFont (шрифты);
•TPen (карандаш, перо);
•TBrush (кисть).
Класс TFONT. Можно выделить следующие свойства соответствующего объекта Canvas.TFont:
•Name (тип string) имя используемого шрифта.
•Size (тип integer) размер шрифта в пунктах (points). Пункт это единица измерения шрифта, равная0,353 мм или1/72 дюйма.
•Style стиль начертания символов, который может быть обычным, полужирным (fsBold), курсивным (fsItalic), подчёркнутым (fsUnderline) и перечёркнутым (fsStrikeOut). В программе можно комбинировать необ-
ходимые стили, например, чтобы установить стиль ¾полужирный курсив¿ необходимо написать следующий оператор:
Объект . Canvas . Font . Style :=[f s I t a l i c , fsBold]
•Color (типTcolor) цвет символов.
•Charset (тип0..255) набор символов шрифта. Каждый вид шрифта,
определяемый его именем, поддерживает один или более наборов символов. В табл. 2.2 приведены некоторые значения Charset.

Таблица 2.2. Значения свойства Charset

Константа	Значение	Описание
ANSI_CHARSET	0	Символы ANSI
DEFAULT_CHARSET	1	Задаётся по умолчанию. Шрифт выбирается только по его имени Name и раз меру Size. Если описанный шрифт недо ступен в системе, он будет заменён дру гим - - -
SYMBOL_CHARSET	2	Стандартный набор символов
MAC_CHARSET	77	Символы Macintosh
GREEK_CHARSET	161	Греческие символы
RUSSIAN_CHARSET	204	Символы кириллицы
EASTEUROPE_CHARSET	238	Включает диалектические знаки (знаки, добавляемые к буквам и характеризую- щие их произношение) для восточно-ев- ропейских языков

Класс TPEN. Карандаш (перо) используется как инструмент для рисования точек, линий, контуров геометрических фигур. Основные свойства объекта Canvas.TPen:

•Color (типTcolor) определяет тип линии;

•Width (типInteger) задаёт толщину линии в пикселях;

•Style даёт возможность выбрать вид линии. Это свойство может принимать значение, указанное в таблице 2.3.

•Mode определяет, каким образом взаимодействуют цвета пера и полотна.

Таблица 2.3. Виды линий

Значение	Описание
psSolid	Сплошная линия
psDash	Штриховая линия
psDot	Пунктирная линия
psDashDot	Штрих-пунктирнаялиния
psDashDodDot	Линия, чередующая штрих и два пунктира
psClear	Нет линии

Таблица 2.4. Возможные значения свойства Mode

Режим	Операция	Цвет пикселя
pmBlack	Black	Всегда чёрный
pmWhite	White	Всегда белый
pmNop		Неизменный
pmNot	Not Screen	Инверсный цвет по отношению к цвету фона
pmCopy	Pen	Цвет, указанный в свойствах Color пераPen (это значение принято по умолчанию)
pmNotCopy	Not Pen	Инверсия цвета пера
pmMergePenNot	Pen or Not Pen	Дизъюнкция цвета пера и инверсного цвета фона
pmMaskPenNot	Pen and Not Screen	Конъюнкция цвета пера и инверсного цвета фона
pmMergeNotPen	Not Pen or Screen	Дизъюнкция цвета фона и инверсного цвета пера
PmMaskNotPen	Not Pen and Screen	Конъюнкция цвета фона и инверсного цвета пера
pmMerge	Pen or Screen	Дизъюнкция цвета пера и цвета фона
pmNotMerge	Not (Pen or Screen)	Инверсия режима pmMerge
pmMask	Pen and Screen	Конъюнкция цвета пера и цвета фона
pmNotMask	Not (Pen and Screen)	Инверсия режима pmMask
pmXor	Pen xor Screen	Операция xor над цветом пера и цветом фона
pmNotXor	Not (Pen xor Screen)	Инверсия режима pmXor

Выбор значения этого свойства позволяет получать различные эффекты. Возможные значения Mode приведены в табл.2.4. По умолчанию вся линия вычерчивается цветом, определяемым значениемPen.Color, но можно определять инверсный цвет линии по отношению к цвету фона. В этом случае независимо от цвета фона, даже если цвет линии и фона одинаков, линия будет видна.

Класс TBRUSH. Кисть (Canvas.Brush) используется методами, обеспечивающими вычерчивание замкнутых фигур для заливки. Кисть обладает двумя основными свойствами:
•Color (типTcolor) цвет закрашивания замкнутой области;
•Style (типTBrushStyle) определяет стиль заполнения области (bsSolid сплошное заполнение, bsClear прозрачное, bsHorizontal горизонтальные линии, bsVertical вертикальные линии, bsFDiagonal, bsBDiagonal диагональные линии, bsCross решётка, bsDiagCross диагональная решетка).

Класс TCANVAS. Этот класс является основным инструментом для рисования графики. Рассмотрим его наиболее часто используемые методы.

Procedure MoveTo(X, Y : Integer) ;

Метод MoveTo изменяет текущую позицию пера на позицию, заданную точкой(X,Y). Текущая позиция хранится в переменнойPenPos типаTpoint. Определение типаTPoint следующее:

type TPoint =record X : Longint ;

Y: Longint ; end ;

Текущую позицию пера можно считывать с помощью свойства PenPos следующим образом:

X:=PenPos .X;

Y:=PenPos .Y;

Procedure LineTo (X, Y : Integer) ;

Метод LineTo соединяет прямой линией текущую позицию пера и точку с координатами(X,Y). При этом текущая позиция пера перемещается в точку с координатами(X,Y).

Рассмотрим работу процедуры на примере. Расположим на форме кнопку и рассмотрим процедуру обработки события TForm1.Button1Click, которая рисует прямые линии:

Procedure TfForm1 . Button1Click (Sender : TObject) begin

Form1 . Canvas . LineTo (3 0 , 5 0) ; end ;

В результате щелчка по кнопке на форме будет нарисована прямая линия, соединяющая точку с координатами (0,0) и точку с координатами(30,50).

При повторном щелчке по кнопке процедура продолжит рисовать эту же линию. Теперь перепишем процедуру обработки события следующим образом:

Procedure TForm1 . Button1Click (Sender : TObject) begin

Form1 . Canvas . LineTo (Canvas . PenPos . x+30,Canvas . PenPos . y+50); end ;

При первом щелчке по кнопке на экране прорисуется аналогичная линия. Но при повторном щелчке процедура рисует линию, которая соединяет текущую точку с точкой, получившейся из текущей добавлением к координате X числа 30, а к координатеY числа 50. Т. е. при повторном щелчке по кнопке процедура соединяет прямой линией точки(30,50) и(60,100). При третьем щелчке по кнопке будут соединяться прямой линией точки(60,100) и(90,150) и т. д.

Procedure PolyLine (const Points array of TPoint) ;

Метод PolyLine рисует ломаную линию, координаты вершин которой определяются массивомPoints.

Рассмотрим работу процедуры на примере. Расположим на форме кнопки Рисовать и Выход и запишем следующие операторы процедур обработки события:

```
procedure TForm1.Button2Click(Sender: TObject);
  var temp: array [ 1 .. 25 ] of TPoint ;
i: byte ;
j: integer ;
begin
j :=1 ;
for i :=1 to 25 do
begin
//вычисление координат вершин ломаной линии
temp [ i ] . x:=25+( i - 1) * 10;
temp [ i ] . y:=150 - j * ( i - 1) * 5;
j :=-j ;
end ;
Form1.Canvas . Polyline ( temp ) ;

end;

end.
```

После запуска программы и щелчка по кнопке Рисовать окно формы будет выглядеть, как на рисунке 2.1.

Procedure E l l i p s e (X1 , Y1 , X2 , Y2 : Integer) ;

Метод Ellipse вычерчивает на холсте эллипс или окружность.X1, Y1, X2, Y2 это координаты прямоугольника, внутри которого вычерчивается эллипс.

Если прямоугольник является квадратом, то вычерчивается окружность.

Procedure Arc (X1 , Y1 , X2 , Y2 , X3 , Y3 , X4 , Y4 : Integer) ;

Метод Arc вычерчивает дугу эллипса.X1, Y1, X2, Y2 это координаты, определяющие эллипс, частью которого является дуга;X3, Y3 координаты, определяющие начальную точку дуги;X4, Y4 координаты, определяющие конечную точку дуги. Дуга рисуется против часовой стрелки.

Procedure Rectangle (X1 , Y1 , X2 , Y2 : Integer) ;

Метод Rectangle рисует прямоугольник.X1, Y1, X2, Y2 координаты верхнего левого и нижнего правого углов прямоугольника.

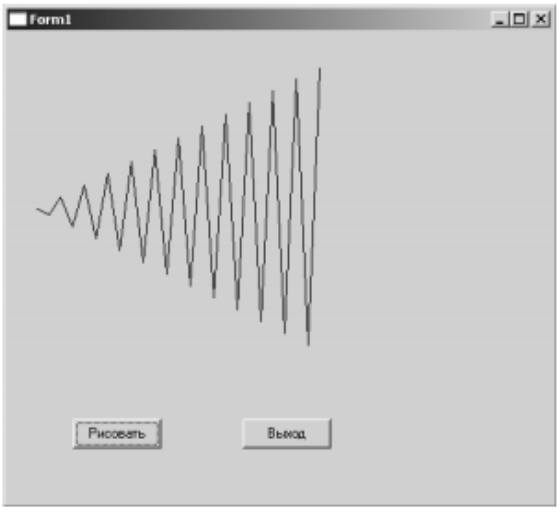


Рис. 2.1. Пример использования процедуры PolyLine

Procedure RoundRect (X1 , Y1 , X2 , Y2 , X3 , Y3 : Integer) ;

Это метод вычерчивания прямоугольника со скруглёнными углами. X1, Y1, X2, Y2 координаты верхнего левого и нижнего правого углов прямоугольника, аX3, Y3 размер эллипса, одна четверть которого используется для вычерчивания скругленного угла.

Procedure PolyGon (const Points array of TPoint) ;

Метод PolyGon рисует замкнутую фигуру (многоугольник) по множеству угловых точек, заданному массивомPoints. При этом первая точка соединяется прямой линией с последней. Этим методPolyGon отличается от методаPoliline, который не замыкает конечные точки. Рисование осуществляется текущим пером Pen, а внутренняя область фигуры закрашивается текущей кистьюBrush.

Procedure Pie (X1 , Y1 , X2 , Y2 , X3 , Y3 , X4 , Y4 : Integer) ; Метод Pie рисует замкнутую фигуру сектор окружности или эллипса с помощью текущих параметров пера Pen, внутренняя область закрашивается текущей кистью Brush. Точки(X1,Y1) и(X2,Y2) задают прямоугольник, описывающий эллипс. Начальная точка дуги определяется пересечением эллипса с прямой, проходящей через его его центр и точку (X3,Y3). Конечная точка дуги определяется пересечением эллипса с прямой, проходящей через его центр и точку (X4,Y4). Дуга рисуется против часовой стрелки от начальной до конечной точки. Рисуются прямые, ограничивающие сегмент и проходящие через центр эллипса и точки (X3,Y3) и(X4,Y4).

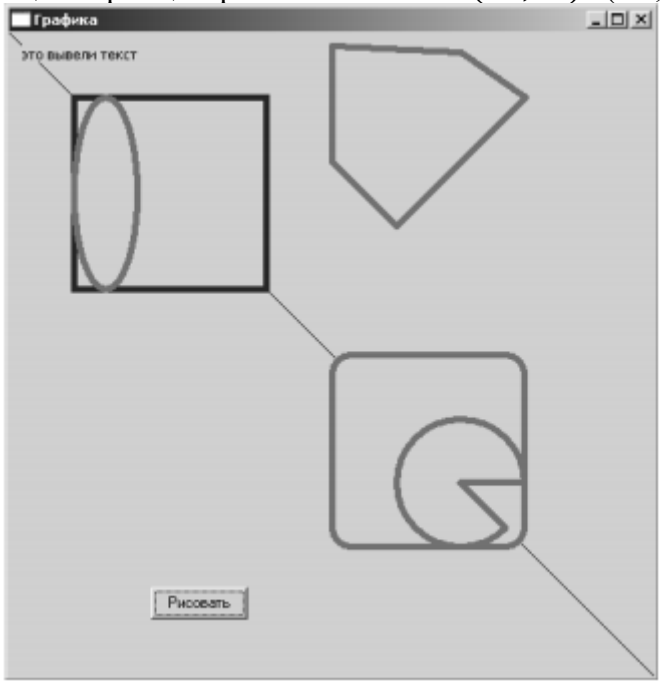


Рис. 2.2. Пример использования методов рисования фигур

Создадим форму, установим ей размеры Heigth 500,Weight 500. Внизу разместим кнопку, зададим ей свойствоCaption ¾Рисовать¿. При запуске программы и щелчке по этой кнопке на форме прорисуются различные фигуры (см. рис. 2.2). Ниже приведён листинг программы, демонстрирующий работу перечисленных методов. Результат работы программы приведен на рис.2.2.

```
procedure TForm1.Button3Click(Sender: TObject);
  var t : array [ 1 .. 5 ] of TPoint ;
begin
//рисование линии
Form1 . Canvas . LineTo ( 500 ,500 ) ;
```

```
//изменяем цвет и толщину линии
Form1 . Canvas . Pen . Color := clMaroon ;
Form1 . Canvas . Pen . Width:= 5 ;
//рисование прямоугольника
Form1 . Canvas . Rectangle ( 50 ,50 ,200 ,200 ) ;
Form1 . Canvas . Pen . Color := clolive ;
//рисование эллипса
Form1 . Canvas . Ellipse ( 50 ,50 ,100 ,200 ) ;
//рисование прямоугольника со скруглёнными углами
Form1 . Canvas . RoundRect ( 250 ,250 ,400 ,400 ,30 ,30 ) ;
//рисование сектора окружности
Form1 . Canvas . Pie ( 300 ,300 ,400 ,400 ,350 ,350 ,500 ,500 ) ;
//формирование массива координат вершин пятиугольника
t [ 1 ] . x :=250 ; t [ 1 ] . y :=10 ;
t [ 2 ] . x :=350 ; t [ 2 ] . y :=15 ;
t [ 3 ] . x :=400 ; t [ 3 ] . y :=50 ;
t [ 4 ] . x :=300 ; t [ 4 ] . y :=150 ;
t [ 5 ] . x :=250 ; t [ 5 ] . y :=100 ;
Form1 . Canvas . Polygon ( t ) ;
Form1 . Canvas . TextOut ( 10 ,10 ,'это␣вывели␣текстст" ' );

end;
```

Procedure TextOut (X, Y : Integer ; const Text : String) ;

Эта функция пишет строку текста Text, начиная с позиции с координатами(X,Y). Текущая позицияPenPos переPen перемещается в конец выведенного текста. Надпись выводится в соответствии с текущими установками шрифтаFont, фон надписи определяется установками текущей кисти. Для выравнивания позиции текста на канве можно использовать методы, позволяющие определить высоту и длину текста в пикселях TextExtent,TextHeight иTextWidth. Рассмотрим эти функции.

Function TextExtent (const Text : String) : Tsize ;

Эта функция возвращает структуру типа Tsize, содержащую длину и высоту в пикселях текстаText, который предполагается написать на канве текущим шрифтом.

type

Tsize = record cx : Longint ; cx : Longint ;

end ;

Function TextHeight (const Text : String) : Integer ;

Функция возвращает высоту в пикселях текста Text, который предполагается написать на канве текущим шрифтом.

Function TextWidth (const Text : String) : Integer ;

Функция возвращает длину в пикселях текста Text, который предполагается написать на канве текущим шрифтом. Это позволяет перед выводом текста на канву определить размер надписи и расположить её и другие элементы изображения наилучшим образом.

Если цвет кисти в момент вывода текста отличается от того, которым закрашена канва, то текст будет выведен в цветной прямоугольной рамке, но её размеры будут точно равны размерам надписи.

Мы рассмотрели основные функции рисования. Теперь можно перейти непосредственно к рисованию. Но перед этим следует заметить, что если вы свернёте окно с графикой, а затем его восстановите, то картинка на форме исчезнет. Изменение размеров окна также может испортить графическое изображение в нём. Для решения этой проблемы существуют процедуры обработки событий

Объект.FormPaint и Объект.FormResize. Процедура Объект.FormPaintвыполняется после появления формы на экране, а процедура Объект.FormResizeпосле изменения размера формы. Следовательно, все операторы рисования нужно помещать внутрь Объект.FormPaint и дублировать в процедуреОбъект.FormResize.

.

Класс TCanvas — сердцевина графической подсистемы Delphi (Lazarus). Он объединяет в себе и "холст" (контекст конкретного устройства GDI), и "рабочие инструменты" (перо, кисть, шрифт) и даже "подмастерьев" (набор функций по рисованию типовых геометрических фигур).

Канва не является компонентом, но она входит в качестве свойства во многие другие компоненты, которые должны уметь нарисовать себя и отобразить какую-либо информацию.

Для рисования канва включает в себя шрифт, перо и кисть:

(pb) **property** Font: TFont ; {TFont: Charset, Color, Style, Size}

(Pt) **property** Pen: TPen; {TPen: Color, Mode, Style, Width}

(Pb) **property** Brush: TBrush; *{TBrush: Bitmap, Color, Style}*

Кроме того, можно рисовать и поточечно, получив доступ к каждому пикселу. Значение свойства

property Pixels[X, Y: **Integer**]: TColor;

соответствует цвету точки с координатами (X,Y).

Класс TCanvas

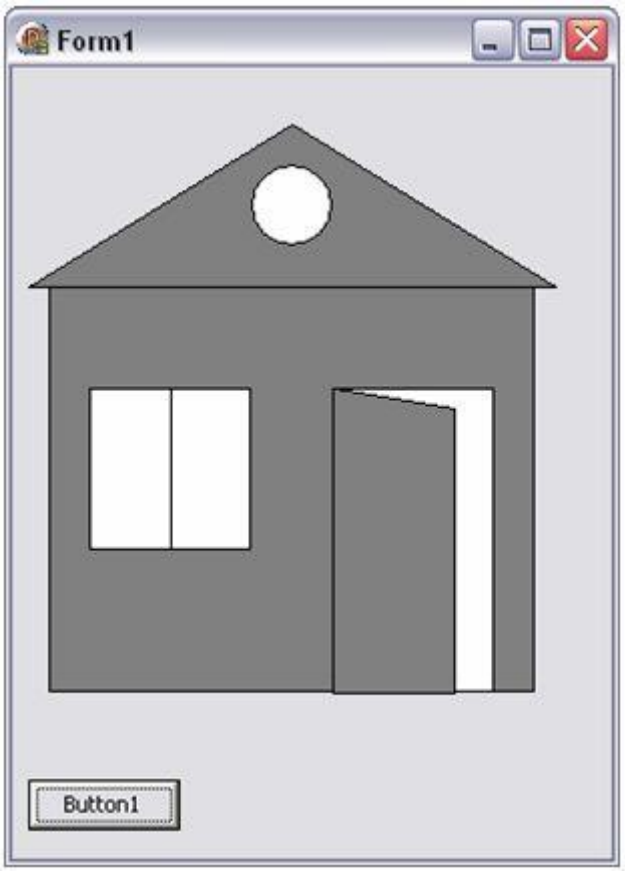
procedure Arc (X1, Y1, X2, Y2, X3,Y3, X4, Y4: **Integer**);	Метод рисует сегмент эллипса. Эллипс определяется описывающим прямоугольником (X1,Y1) — (X2,Y2). Начальная точка сегмента лежит на пересечении эллипса и луча, проведенного из его центра через точку (X3,Y3). Конечная точка сегмента лежит на пересечении эллипса и луча, проведенного из его центра через точку (X4,Y4). Сегмент рисуется против часовой стрелки.
procedure Chord(X1, Y1, X2, Y2, X3,Y3, X4, Y4: **Integer**);	Рисует хорду и заливает отсекаемую ею часть эллипса. Эллипс, начальная и конечная точки определяются, как в методе Arc.
procedure Ellipse(X1, Y1, X2, Y2:**Integer**);	Рисует и закрашивает эллипс, вписанный в прямоугольник (X1,Y1) — (X2,Y2).
procedure MoveTo(X, Y: **Integer**);	Перемещает текущее положение пера (свойство PenPos) в точку (X,Y).
procedure LineTo(X, Y: **Integer**);	Проводит линию текущим пером из текущей точки в (X,Y).
procedure BrushCopy(**const** Dest:TRect; Bitmap: TBitmap; **const**Source: TRect; Color: TColor);	Производит специальное копирование. Прямоугольник Source из битовой карты Bitmap копируется в прямоугольник Dest на канве; при этом цвет Color заменяется на цвет текущей кисти (Brush.Color).
procedure CopyRect(**const** Dest:TRect; Canvas: TCanvas; **const**Source: TRect);	Производит копирование прямоугольника Source из канвы

	Canvas в прямоугольник Dest в области самого объекта.
procedure FillRect(**const** Rect:TRect);	Производит заливку прямоугольника (текущей кистью).
procedure Draw(X, Y: **Integer**; Graphic: TGraphic);	Осуществляет рисование графического объекта Graphic
procedure StretchDraw(**const** Rect:TRect; Graphic: TGraphic);	Осуществляет рисование объекта Graphic в заданном прямоугольнике Rect. Если размеры их не совпадают, Graphic масштабируется.
procedure FloodFill(X, Y: **Integer**; Color: TColor; FillStyle:TFillStyle); TFillStyle =(fsSurface, fsBorder);	Производит заливку области текущей кистью. Процесс начинается с точки (X,Y). Если режим FillStyle равен fsSurface, то он продолжается до тех пор, пока есть соседние точки с цветом Color. В режиме fsBorder закрашивание, наоборот, прекращается при выходе на границу с цветом Color.
procedure Pie (X1, Y1, X2, Y2, X3,Y3, X4, Y4: **Integer**);	Рисует сектор эллипса, описываемого прямоугольником (X1,Y1) — (X2,Y2). Стороны сектора лежат на лучах, проходящих из центра эллипса через точки (X3.Y3) и (X4,Y4).
procedure Polygon(**const** Points:**array of** TPoint);	Строит многоугольник, используя массив координат точек Points. При этом последняя точка соединяется с первой и внутренняя область закрашивается. Polygon ([Point(10,10), Point(30,30),Point(20,40)])
procedure Polyline(**const** Points:**array of** TPoint);	Строит ломаную линию, используя массив координат точек Points.
procedure Rectangle(X1, Y1, X2, Y2 :**Integer**);	Рисует прямоугольник с диагональю заданной координатами (X1,Y1) и (X2,Y2).
procedure RoundRect (X1, Yl, X2, Y2,XW, YH: **Integer**);	Рисует прямоугольник с закругленными углами. Координаты вершин — те же, что и в методе Rectangle. Закругления рисуются как сегменты эллипса с размерами осей по горизонтали и вертикали XW и YH.
function TextHeight(const Text:string): Integer;	Возвращает высоту строки Text в пикселах.
function TextWidth(const Text:string): Integer;	Возвращает ширину строки Text в пикселах.
procedure TextOut(X, Y: Integer;const Text: string);	Производит вывод строки Text. Левый

	верхний угол помещается в точку канвы (X,Y).
procedure TextRect(Rect: TRect; X,Y: **Integer**; **const** Text: **string**);	Производит вывод текста с отсечением. Как и в TextOut, строка Text выводится с позиции (X,Y); при этом часть текста, лежащая вне пределов прямоугольника Rect, отсекается и не будет видна.
property PenPos: TPoint;	Содержит текущую позицию пера канвы (изменяется посредством метода MoveTo).

Пример

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  with Form1.Canvas do begin
    Brush.Color := clGray;
    Rectangle(10,100,250,300);  // корпус
    Polygon([Point(0,100),Point(130,20),Point(260,100)]); // крыша
    Brush.Color := clWhite;
    Ellipse(110,40,150,80); // чердак
    Rectangle(30,150,110,230);   // окно
    MoveTo(70,150);
    LineTo(70,230);
    Rectangle(150,300,230,150);  // дверь
    Brush.Color := clGray;

Polygon([Point(150,300),Point(150,150),Point(210,160),Point(210,300)]
);

  end;
end;
```



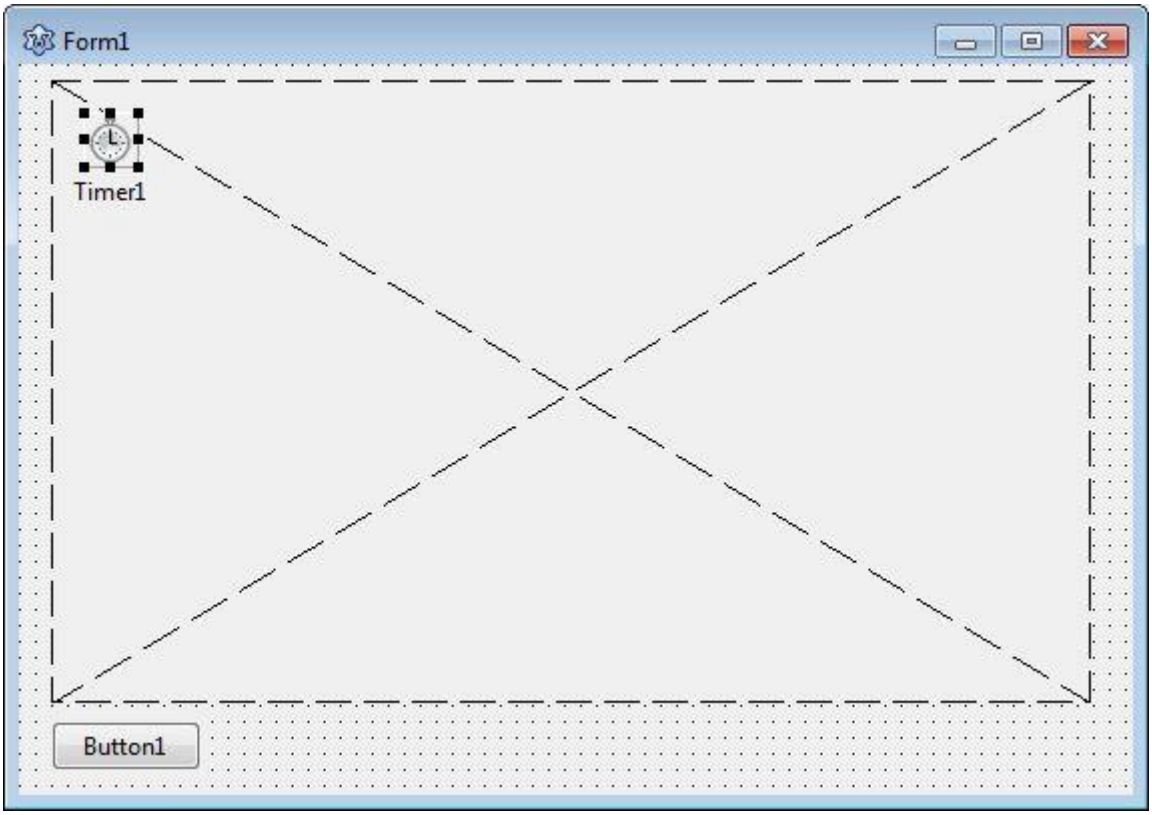
Для очистки формы надо установить на форму кнопку, обработчик событий которой вызывает функцию Form1.Refresh.

2. Линейное движение векторного объекта в Lazarus

Линейное движение по однородному фону является довольно простым в плане программной реализации. Достаточно закрашивать объект цветом фона, изменять его координаты и прорисовывать в новом месте, повторяя эти действия через определенный интервал времени.

Для реализации анимации, помимо двух уже известных компонентов TPaintBox (поле для рисования) и TButton (кнопка запуска), понадобится компонент TTimer со вкладки System. Компонент Timer имеет единственное событие OnTimer, которое выполняется пока Timer включен с интервалом по времени, установленным в свойстве Interval.

Расположите компонент Timer1 на форме. Установите его свойства Timer1.Interval := 100 и Timer1.Enabled := false



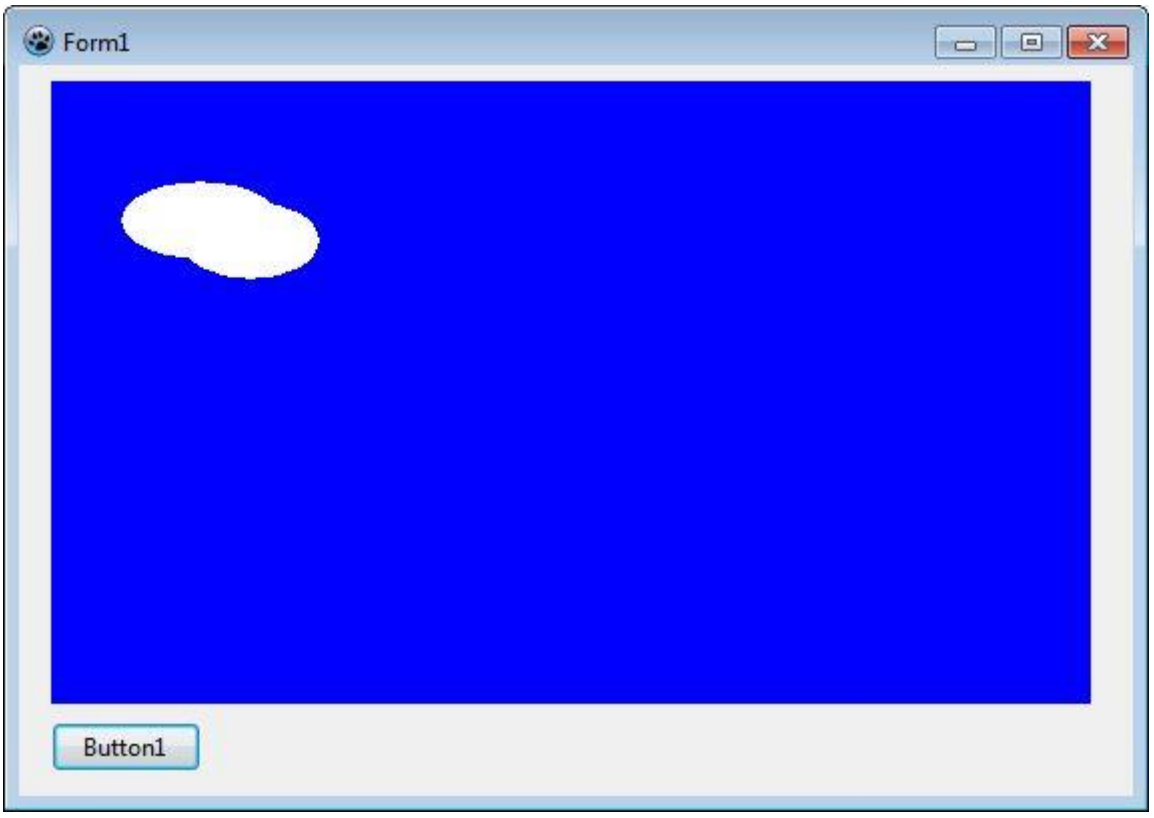
В коде программы необходимо прописать три процедуры. Процедуру отрисовки объекта procedure TForm1.Cloud, процедуру, отрабатывающую на событие OnTimer, - procedure TForm1.Timer1Timer и процедуру запуска анимации, срабатывающую на нажатие кнопки, procedure TForm1.Button1Click.

```
unit Unit1;

{$mode objfpc}{$H+}

interface

uses
  Classes, SysUtils, FileUtil, Forms, Controls, Graphics, Dialogs,
Buttons,
  ExtCtrls, StdCtrls;

type

  { TForm1 }

  TForm1 = class(TForm)
    Button1: TButton;
    PaintBox1: TPaintBox;
    Timer1: TTimer;
    procedure Button1Click(Sender: TObject);
    procedure Timer1Timer(Sender: TObject);
```

```
  private
    { private declarations }
     // координаты прорисовки объекта. Доступны всем процедурам
класса TForm1
     x1, y1 : Integer;
  public
    { public declarations }
    // процедура прорисовки облака
    procedure Cloud (x, y: Integer; ColorCloud: TColor);
  end;

var
  Form1: TForm1;


implementation

{$R *.lfm}

{ TForm1 }

procedure TForm1.Cloud(x, y: Integer; ColorCloud: TColor);
begin
  // прорисовка облака из двух эллипсов
  with PaintBox1.Canvas do begin
   Pen.Style := psClear;
   Brush.Color := ColorCloud;
   Ellipse(x,y,x+80,y+40);
   Ellipse(x+30,y+10,x+100,y+50);
  end;
end;


procedure TForm1.Button1Click(Sender: TObject);
begin
  // установка начальных значений
  x1:=0;
  y1:=50;
  Timer1.Interval:=100;
  // прорисовка картинки по которой двигается объект
  PaintBox1.Canvas.Brush.Color := clBlue;
  PaintBox1.Canvas.Rectangle(0,0, PaintBox1.Width, PaintBox1.Height);
  // Включение таймера - запуск анимации
  Timer1.Enabled := true;
end;

procedure TForm1.Timer1Timer(Sender: TObject);
begin
  // Закраска объекта цветом фона
  Cloud(x1,y1,clBlue);
  // Изменение координат прорисовки
```

```
  x1:=x1+1;
  // Прорисовка объекта в новом месте
  Cloud(x1,y1,clWhite);
end;

end.
```

end.



Индивидуальные задания, объекты для прорисовки

№ варианта (по журналу)	Описание объекта
1	Авто Миксер - бетономешалка
2	Акула
3	Антенна параболическая
4	Бармалей
5	Бегемот
6	Ведьма
7	Вертолёт
8	Вилка столовая
9	Грузовой автомобиль
10	Дельфин
11	Жираф
12	Комбайн зерноуборочный
13	Крокодил
14	Кружка
15	Легковой автомобиль
16	Маяк
17	Мишка олимпийский
18	Мотоцикл
19	Муха
20	Ниндзя
21	Пароход
22	Парусник
23	Планета Сатурн
24	Ракета
25	Самолет
26	Слон
27	Спутник связи
28	Стрекоза
29	Стул
30	Танк
31	Телега
32	Трактор
33	Цветок розы
34	Цветок ромашки
35	Цистерна
36	Чайник
37	Чебурашка
38	Чемодан с ручкой и наклейкой
39	Яблоко надкусанное
40	Ягоды – вишни 2 шт.

ЛАБОРАТОРНАЯ РАБОТА №3. Практическое освоение растровых алгоритмов для отрезков

Цель работы: освоить алгоритмы формирования растровых представлений произвольного отрезка прямой и окружности.

1.1. Постановка задачи

Сгенерировать двухмерное изображение, содержащее заданное количество отрезков различной длины и заданное количество окружностей различного радиуса с помощью самостоятельно разработанных функций. Для программной реализации функций разложения прямой и окружности в растр разрешается использовать только целочисленные алгоритмы.

1.2. Алгоритм разложения отрезка в растр

Рассмотрим уравнение идеальной прямой, проходящей через две точки,

$y = y_1 + (x-x_1) *(y_2-y_1)/(x_2-x_1) = y_1 + (x-x_1)*\otimes y/\otimes x,$

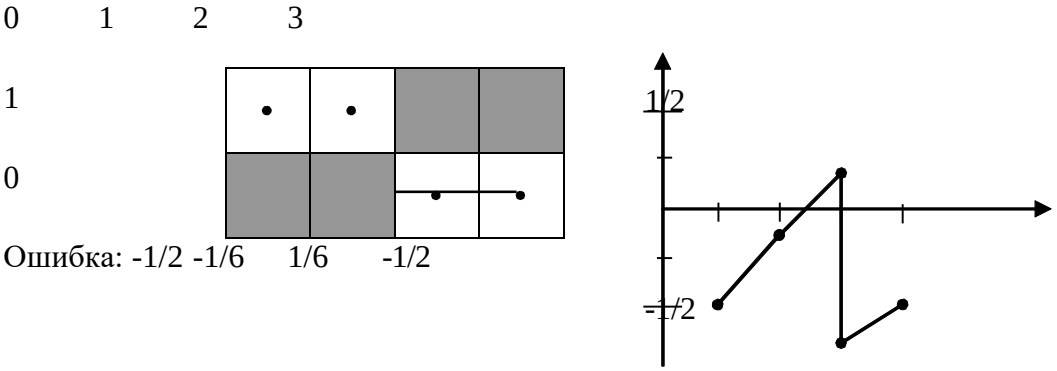
где $\otimes y$ – разность *y*-координат концов отрезка,

$\otimes x$ – разность *x*-координат.

Если *x* и *y* изменяются вдоль прямой дискретно на δ_x и δ_y, тогда $y_i+1=y_i+\delta_y$ $=y_i + \delta_x*\otimes y/\otimes x$. Пусть δ_x и/или δ_y равняется величине пикселя, т.е., 1.

Растровое представление отрезка прямой – это связное множество пиксе- лей, имеющих наименьшее отклонение от идеальной прямой. При этом может использоваться 4- и 8-связность. Точное растровое представление (4- и 8- связное) можно построить только для вертикального и горизонтального отрез- ка. Для них $\delta x = 1$, $\delta y = 0$, либо $\delta x = 0$, $\delta y = 1$. Для отрезка, расположенного под углом 45° (135°) к горизонтальной оси, можно построить точное 8-связное рас- тровое представление. Для него $\delta x = \delta y = 1$. Растровое представление всех ос- тальных отрезков выглядит в виде ступенчатой последовательности пикселей (рис. 1.1).

В 1965 г. Дж. Брезенхем (J. Bresenham) опубликовал целочисленный алгоритм формирования растрового представления произвольного отрезка прямой. Пусть для растрового представления отрезка выбран некоторый пиксель, на- пример имеющий координаты (x_1, y_1). Как выбрать следующий соседний пик- сель? Алгоритм использует понятие функции, оценивающей величину отклоне- ния выбираемого пикселя от идеальной прямой. Пусть $0<\otimes y<\otimes x$, тогда фикси- руем $\delta x = 1$ (т. е. $x_i = x_i+1$) и оцениваем, пиксель с какой у-координатой ($y_i+1=y_1$ или $y_i+1=y_1+1$) расположен ближе к идеальной прямой. Выбирается тот пиксель, который имеет меньшее (из двух) значение отклонения, т.е. оце- ночной функции. При этом отклонение ближайшего пикселя не превышает 1/2.



а б

Рис.1.1. Отрезок с вершинами (0,0) и (3,1).

а) – его растровое представление.

б) – график функции «ошибки» растрового представления отрезка по отношению к его идеальному положению (при инициализации нулевого значения ошибки значением $f_0 = -1/2$).

Основа алгоритма – это движение вдоль основной оси на один пиксель и поддержание текущего отклонения от идеальной прямой в заданных пределах. Если текущее отклонение превышает норму, то шаг делается по неосновной оси, соответственно отклонение уменьшается на единицу. Идея алгоритма основывается на использовании понятия «ошибки». В каноническом случае, т.е. при построении отрезка в первом октанте с началом в центре системы координат, процесс рисования 8-связной произвольной прямой линии можно закодировать последовательностью следующего вида: sdssd…, где s – горизонтальное смещение, а d – диагональное смещение от пикселя к пикселю. «Ошибкой» в таком случае будет называться расстояние между значением ординаты (при фиксированном значении *x*) идеального отрезка, который алгоритм стремится изобразить, и значением ординаты «центра» ближайшего пикселя (с тем же фиксированным значением *x*), который и будет в результате закрашен. Алгоритм реализован таким образом, что в случае построения 8-связной линии, одна координата (в каноническом случае – *x*) изменяется на единицу на каждом шаге, а другая либо остаётся без изменений, либо также изменяется на единицу. Пример подобного построения приведён на рисунке 1.1 а), перемеще- ние вдоль растрового построения можно закодировать как – «sds» .

Так как значение «ошибки» на каждом шаге не может превышать 1/2 расстояния между центрами соседних пикселей, которые претендуют на право быть закрашенными на текущем шаге, то можно изначально «сместить» значе- ние «ошибки» на −1/2. Это позволит принимать решение – оставлять вторую координату без изменений либо инкрементировать её – на основе проверки зна- ка текущего значения «ошибки», вместо сравнения её значения с 1/2.

С целью увеличения быстродействия алгоритма можно использовать целочисленную арифметику и исключить операции деления. Для этого значение ошибки, вычисляемой по формуле

$f = \otimes y/\otimes x - 0{,}5$, (1.1)

надо умножить на $2\otimes x$, тогда

$2\otimes x*f = 2\otimes y - \otimes x$. (1.2)

Обозначив $F = 2\otimes x*f$, получим

$F = 2\otimes y - \otimes x$. (1.3)

С помощью полученного выражения можно вычислить начальное значение ошибки. В цикле прорисовки отрезка в первом октанте её значение будет вычисляться в соответствии с выражением 1.4

$F_{i+1} = F_i + 2\otimes y$. (1.4)

Целочисленный алгоритм Брезенхема для канонического случая построе- ния отрезка можно сформулировать в виде последовательностей операций:

Инициализация нулевого значения ошибки в соответствии с (1.3).

1. Цикл по значению *x*:
2. Отображение пикселя с текущими координатами.
3. Модификация значения ошибки в соответствии с (1.4), инкре ментация *x*.
4. В случае, если $F \geq 0$, коррекция $F = F - 2\otimes x$ и инкрементация *y*.
5. Если x равен значению конца отрезка – x2, то конец алгоритма, отрезок отображён, если иначе – повторить шаг 2.

Следует отметить, что в общем случае не все отрезки, изображаемые на экране, начинаются в центре системы координат и лежат целиком в первом октанте. Соответственно, приведённый алгоритм не является универсальным. Для построения отрезка с любыми конечными координатами можно воспользоваться геометрическими преобразованиями на плоскости – переносом и отражением, либо разработать общий алгоритм, который будет учитывать положение отрезка в заданной системе координат. Общий алгоритм должен выбирать основную ось (т.е. ту, координаты которой будут изменяться на каждом шаге в зависимости от величины углового коэффициента), а также учитывать направление движения, инкрементируя либо декрементируя координаты по соответствующим осям. Общий вариант целочисленного алгоритма Брезенхема предлагается реализовать в программном виде самостоятельно на основе материалов лекций и приведенной литературы.

1.3. Задание к лабораторной работе

В рамках лабораторной работы выполнить следующую последовательность

операций:

1. Программно реализовать целочисленные алгоритмы Брезенхема для растеризации отрезка (в общем случае).
2. Используя разработанные функции, сгенерировать и вывести на экран осмысленную двухмерную сцену, содержащую как минимум 30 – 40 отрезков и 10 – 15 окружностей различного диаметра.

1.4. Контрольные вопросы

1. В чем состоит идея алгоритма Брезенхема для построения растрового представления отрезка?
2. Каковы достоинства алгоритма Брезенхема? Как можно улучшить алгоритм?
3. Всегда ли совпадают растровые представления отрезков, заданных координатами $(x_1, y_1) – (x_2, y_2)$ и $(x_2, y_2) – (x_1, y_1)$?
4. Будут ли отличаться растровые представления отрезка $(x_1, y_1) – (x_2, y_2)$ и отрезков
   а) $(x_1+e, y_1) – (x_2+e, y_2)$;
   б) $(x_1+e, y_1+e) – (x_2+e, y_2+e)$;
   в) $(x_1, y_1-e)- (x_2, y_2-e)$,
   где e – константа $0<e<1$?
5. Накапливается ли ошибка в процессе выполнения алгоритма Брезенхема?

Лабораторная работа 4. Практическое освоение растровых алгоритмов для окружностей.

Данный алгоритм позволяет строить эллипсы на сновании координат центра фигуры и длин большей и меньшей полуосей.

Алгоритм является модификацией алгоритма для генерации окружностей.

Эллипс описывается каноническим уравнением $X^2/a^2+Y^2/b^2=1$ и для любого эллипса можно найти такую декартову систему координат, что это уравнение будет его описывать. При этом центр эллипса лежит в начале координат, а оси совпадают с координатными осями. При $a=b$ данный алгоритм строит окружность, так как окружность действительно является частным случаем эллипса.

Как и в оригинальном алгоритме Брезенхема, выбор ближайшей точки основан на анализе знаков управляющих переменных, для вычисления которых используется исключительно целочисленная арифметика, что значительно повышает скорость выполнения алгоритма на любой ЭВМ.

Рассмотрим непосредственно алгоритм:

Для построения растровой развертки эллипса с осями, параллельными осям координат, и радиусами a, b воспользуемся каноническим уравнением $X^2/a^2+Y^2/b^2=1$, которое перепишем в виде $f(x, y) \equiv b^2x^2 + a^2y^2 – a^2b^2 = 0$.

В отличие от окружности, для которой было достаточно построить одну восьмую ее часть, а затем воспользоваться свойствами симметрии, эллипс имеет только две оси симметрии, поэтому придется строить одну четверть всей фигуры. За основу возьмем дугу, лежащую между точками $(0, b)$ и $(a, 0)$ в первом квадранте координатной плоскости.

В каждой точке (x, y) эллипса существует вектор нормали, задаваемый градиентом функции f. Дугу разобьём на две части: первая – с углом между нормалью и горизонтальной осью больше 45° (тангенс больше 1) и вторая – с углом, меньшим 45° (Рис.1).

Движение вдоль дуги будем осуществлять по часовой стрелке, начиная с точки $(0, b)$.

Направление нормали соответствует вектору $grad(x, y) = (\partial f/\partial x, \partial f/\partial y) = (2b^2x, 2a^2y)$.

Отсюда находим тангенс угла наклона вектора нормали: $t = a^2y/b^2x$. Приравнивая его единице, получаем что координаты точки деления дуги на вышеуказанные части удовлетворяют равенству $b^2x = a^2y$. Поэтому критерием того, что мы переходим ко второй области будет соотношение $a^2(y-1/2) \leq b^2(x+1)$ (координаты дополнительной точки $(x+1, y-1/2)$ (Рис.2)), или , переходя к целочисленным операциям, $a^2(2y-1) \leq 2b^2(x+1)$.

Вдоль всей дуги координата Y является монотонно убывающей, но в

первой части дуги она убывает медленнее, чем растёт *X*, а во второй быстрее. Поэтому в первой части дуги будем сначала увеличивать значение *X*, а во второй сначала уменьшать значение *Y*.

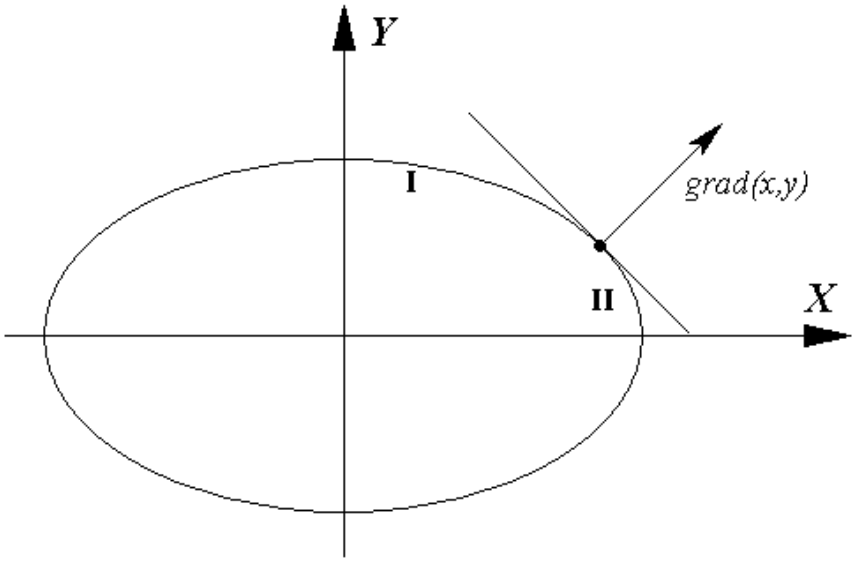
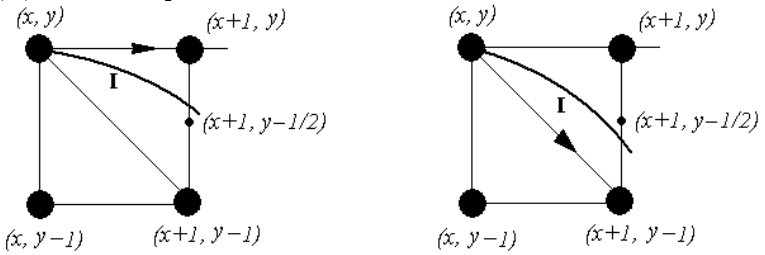
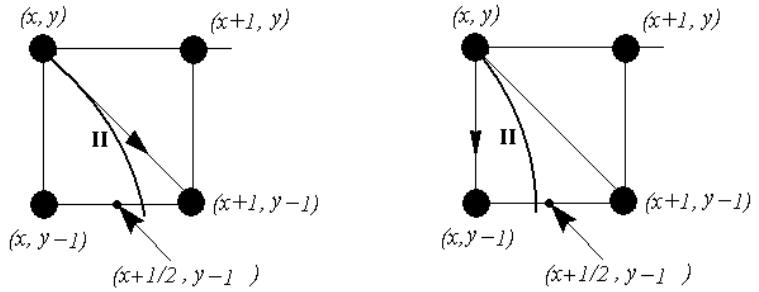


Рис.1 Деление дуги на 2 части



а)



б)

Рис.2 Способы поставить следующий пиксел

При перемещении вдоль первого участка дуги мы каждый раз переходим либо по горизонтали, либо по диагонали, критерий перехода напоминает тот, который используется при построении окружности. Находясь в точке *(x, y)*, будем увеличивать значение X на единицу, а затем вычислять значение $\Delta = f(x+1, y-1/2)$. Если это значение меньше нуля, то дополнительная точка *(x+1, y-1/2)* находится внутри эллипса, следовательно, ближайшая точка растра есть *(x+1, y)*, в противном случае это точка *(x+1, y-1)* (Рис2. а)). На втором участке дуги возможен переход либо по вертикали, либо по

диагонали, поэтому здесь сначала уменьшаем значение *Y*, затем вычисляем $\Delta = f(x+1/2, y-1)$, и направление перехода выбирается аналогично предыдущему случаю (Рис2. б)).
Остаётся, чтобы перейти к целочисленной арифметике, оптимизировать вычисление параметра Δ, умножив его на *4* и представив в виде функции координат точки, тогда для первой части дуги получим:

$\Delta (x, y) = 4b^2(x+1)^2+a^2(2y-1)^2-4a^2b^2$
$\Delta (x+1, y) = \Delta (x, y)+4b^2(2x+3)$
$\Delta (x+1, y-1) = \Delta (x, y)+4b^2(2x+3)-8a^2(y-1)$

Для второй половины дуги получим:

$\Delta (x, y) = b^2(2x+1)^2+4a^2(y+1)^2-4a^2b^2$
$\Delta (x, y-1) = \Delta (x, y)+4a^2(2y+3)$
$\Delta (x+1, y-1) = \Delta (x, y)+4a^2(2y+3)-8b^2(x+1)$

Все оставшиеся дуги эллипса строятся параллельно: после получения очередной точки *(x, y)*, можно инициализировать ещё три точки с координатами *(-x, y), (x, -y), (-x, -y)*.

Достоинства алгоритма: высокая скорость работы, минимальное потребление памяти; значительная схожесть построенного эллипса с эллипс, нарисованным на непрерывной поверхности; использование исключительно целочисленной арифметики.

Недостатки алгоритма: необходима достаточная разрядность переменных при достаточно больших значениях *a* и *b*.

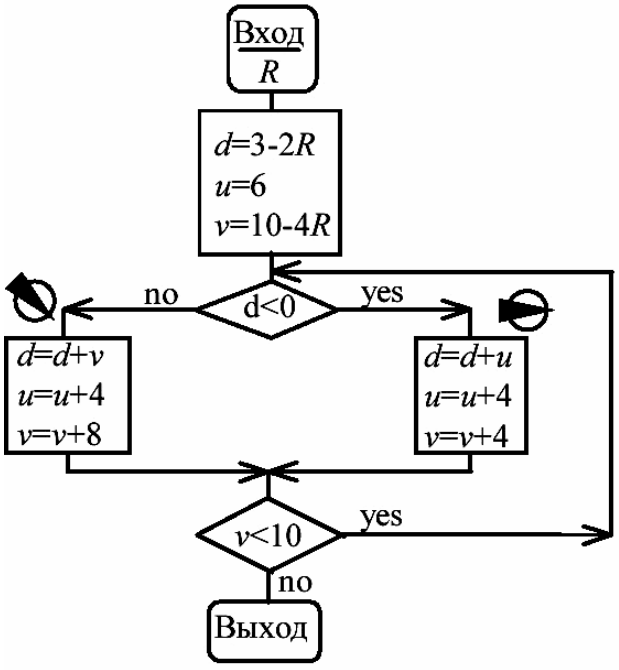
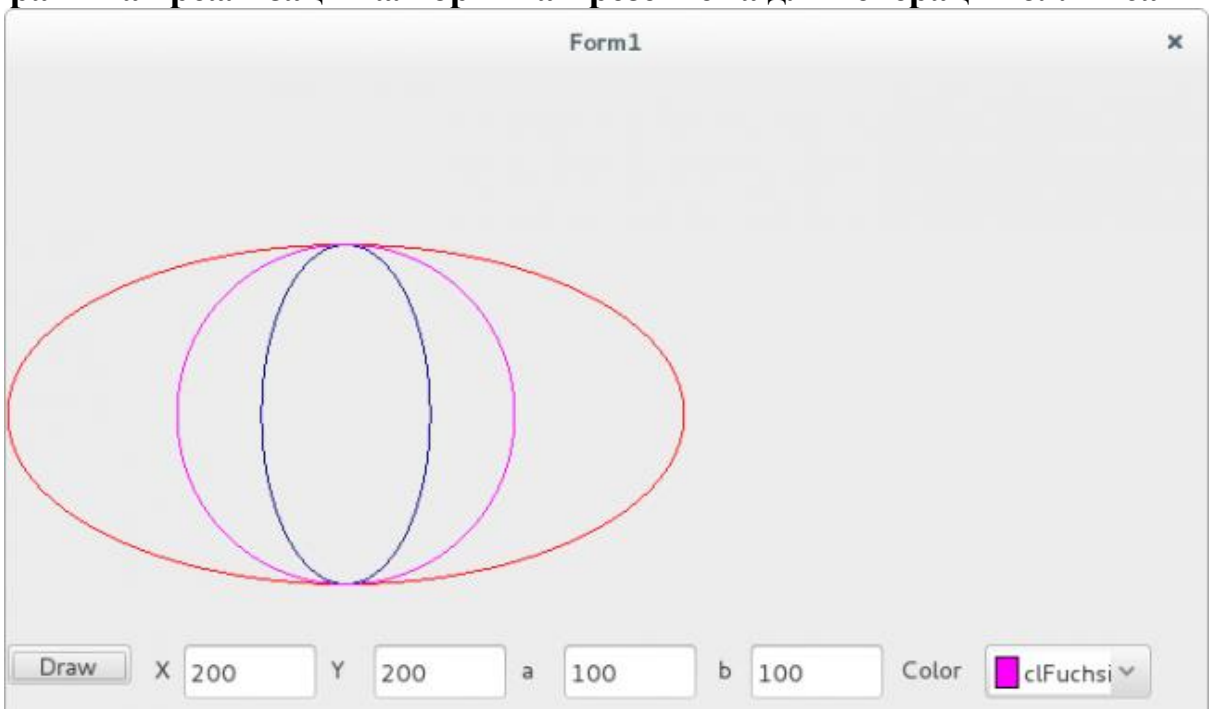


Рис. 1.5. Блок-схема алгоритма построения 1/8 части окружности

2. Программная реализация алгоритма Брезенхема для генерации эллипса



Задача : построить (растеризовать) эллипс, зная координаты его центра и длины меньшей и большей полуосей a и b соответственно. Суть алгоритма : использование модифицированного алгоритма Брезенхема для построение окружности . Как и в оригинальном алгоритме Брезенхема, выбор ближайшей точки основан на анализе знаков управляющих В поля "X" и "Y" вводятся координаты центра окружности, в поля "a" и "b" — длины соответствующих полуосей.

Код программы:

```
unit Unit1;

{$mode objfpc}{$H+}

interface

uses
  Classes, SysUtils, FileUtil, Forms, Controls, Graphics,
Dialogs, ExtCtrls,
  StdCtrls, ColorBox;

type

  { TForm1 }

  TForm1 = class(TForm)
    Button1: TButton;
    ColorBox1: TColorBox;
    Edit1: TEdit;
    Edit2: TEdit;
    Edit3: TEdit;
```

```
    Edit4: TEdit;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    PaintBox1: TPaintBox;
    procedure Button1Click(Sender: TObject);
  private
    { private declarations }
  public
    { public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.lfm}

{ TForm1 }

procedure DrawEllipse (x, y, a, b : Integer; color: TColor);
var
col,  row : Integer;  //Column - "Столбец", row - "ряд,строка"
// Нет необходимости создавать переменные ниже
// Но я пользуюсь правилом " Используется более 1 раза -- создай
переменную "
a_sqr, b_sqr, two_a_sqr, two_b_sqr, four_a_sqr, four_b_sqr, d:
Longint; // Чтобы избежать переполнения int при больших
значениях a или b
begin
a_sqr:=a*a;   //a^2
b_sqr:=b*b;   //b^2
row:=b;
col:=0;
two_a_sqr:=a_sqr<<1;     //2*a^2
two_b_sqr:=b_sqr<<1;     //2*b^2
four_a_sqr:=a_sqr<<2;    //4*a^2
four_b_sqr:=b_sqr<<2;    //4*b^2
d:=two_a_sqr*((row-1)*(row))+a_sqr+two_b_sqr*(1-a_sqr);
while (a_sqr*row > b_sqr*col) do
      begin
           Form1.PaintBox1.Canvas.Pixels[x+col, y+row]:=color;
           Form1.PaintBox1.Canvas.Pixels[x+col, y-row]:=color;
           Form1.PaintBox1.Canvas.Pixels[x-col, y+row]:=color;
           Form1.PaintBox1.Canvas.Pixels[x-col, y-row]:=color;
           if d >= 0 then
           begin
                row:=row-1;
```

```
                d:=d-four_a_sqr*row;
            end;
            d:=d+two_b_sqr*(3+col*2);
            col:=col+1;
      end;
      d:=two_b_sqr*(col+1)*col+two_a_sqr*(row*(row-2)+1)+(1-
two_a_sqr)*b_sqr;
      while ((row)+1)<>0 do
            begin
                  Form1.PaintBox1.Canvas.Pixels[x+col,
y+row]:=color;
                  Form1.PaintBox1.Canvas.Pixels[x+col, y-
row]:=color;
                  Form1.PaintBox1.Canvas.Pixels[x-col,
y+row]:=color;
                  Form1.PaintBox1.Canvas.Pixels[x-col, y-
row]:=color;
                  if d<=0 then
                     begin
                          col:=col+1;
                          d:=d+four_b_sqr*col;
                     end;
                  row:=row-1;
                  d:=d+two_a_sqr*(3-(row << 1 ));
            end;

end;
procedure TForm1.Button1Click(Sender: TObject);     // Обработка
события нажатия кнопки Draw
begin
  DrawEllipse (StrToInt(Edit1.Text),
StrToInt(Edit2.Text),StrToInt(Edit3.Text), StrToInt(Edit4.Text),
ColorBox1.Selected);        // 4 координаты и цвет, который
исрользуется для рисования
end;

end.
```

1.5. Задание к лабораторной работе

В рамках лабораторной работы выполнить следующую последовательность операций:

- Программно реализовать целочисленные алгоритмы Брезенхема для растеризации отрезка (в общем случае) и окружности.
- Используя разработанные функции, сгенерировать и вывести на экран осмысленную двухмерную сцену, содержащую как минимум 30 – 40 отрезков и 10 – 15 окружностей различного диаметра.

1.6. Контрольные вопросы

1. В чем состоит идея алгоритма Брезенхема для построения растрового представления отрезка/окружности?
2. Каковы достоинства алгоритма Брезенхема? Как можно улучшить ал- горитм?
3. Всегда ли совпадают растровые представления отрезков, заданных ко- ординатами $(x_1, y_1) – (x_2, y_2)$ и $(x_2, y_2) – (x_1, y_1)$?
4. Будут ли отличаться растровые представления отрезка $(x_1, y_1) – (x_2, y_2)$ и отрезков

а) $(x_1+e, y_1) – (x_2+e, y_2)$;

б) $(x_1+e, y_1+e) – (x_2+e, y_2+e)$;

в) $(x_1, y_1-e)- (x_2, y_2-e)$,

где e – константа $0<e<1$?

5. Накапливается ли ошибка в процессе выполнения алгоритма Брезенхема?
6. С какой точностью можно восстановить уравнение непрерывной прямой по растровому представлению отрезка?
7. Всегда ли совпадают растровые представления окружностей, заданных координатами $(x_1, y_1) – (x_2, y_2)$ и $(x_2, y_2) – (x_1, y_1)$?
8. Что проще в вычислительном плане: сформировать растровое представление окружности с помощью а) алгоритма Брезенхема для вычерчивания отрезка путем отображения правильного N-угольника, аппроксимирующего окружность с нужной точностью, или б) алгоритма Брезенхема для построения растрового представления окружности?
9. Будут ли различаться растровые представления окружностей, если: а) их радиусы отличаются на e;,

б) координаты их центров отличаются на e,

где e – константа $0<e<1$?

Литература по теме работы

1. Роджерс, Д. Алгоритмы машинной графики/ Д. Роджерс. – М.: Мир, 1989, §§2.1 – 2.6.
2. Фоли, Дж. Основы интерактивной машинной графики/ Дж. Фоли, А. вэн Дэм. – М.: Мир, 1985, т. 2, §§11.2, 11.4.
3. Херн, Д. Микрокомпьютерная графика и стандарт OpenGL/ Д. Херн, М.Бейкер. – М.: “Вильямс”, 2005, §§3.4, 3.5, 3.9.
4. Лабораторный практикум по курсу «Машинная графика»/ Р.Х. Садыхов [и др.]. – Минск: БГУИР, 2002.

ЛАБОРАТОРНАЯ РАБОТА №5. Проецирование трёхмерных объектов

Цель работы: освоить функции инициализации, построения примитивов, визуализации программного интерфейса OpenGL.

2.1. Постановка задачи

Сгенерировать двухмерное изображение осмысленной сцены заданной сложности с помощью функций программного интерфейса OpenGL.

2.2. Базовые функции OpenGL

Основные функции интерфейса OpenGL изначально были представлены в виде основной платформонезависимой библиотеки GL (от Graphics Library – графическая библиотека) и набора библиотек AGL, GLX, WGL, отвечающих за интерфейс с различными оконными системами – Apple, X-Window, Windows соответственно, на которых производилось выполнение итоговых программ. Впоследствии к интерфейсу OpenGL добавилась библиотека GLU – OpenGL Utility Library, расширяющая графические возможности базовой библиотеки, а набор специализированных под упомянутые системы библиотек был трансформирован в единый переносимый оконный интерфейс – GLUT (OpenGL Utility Toolkit). Следует отметить, что функции основных библиотек OpenGL поддерживаются Windows на системном уровне и устанавливать библиотеки при написании Windows программ, использующих интерфейс OpenGL, нет необходимости (справедливо для функций стандарта OpenGL v1.1). В таком случае, однако, текст программы на C/C++/C# должен начинаться с подключения соответствующих заголовочных файлов:

```
#include <GL/gl.h>
#include <GL/glu.h>
```

В случае, если разрабатывается переносимый вариант графической программы, то для визуализации необходимо использовать функции библиотеки GLUT (её можно без затруднений скачать из Интернета, например, отсюда –

[см. 2.4 п.3]). Программа, соответственно, должна начинаться с подключения заголовочного файла библиотеки:

```
#include <GL/glut.h>.
```

Заголовочные файлы «gl.h» и «glu.h» явно подключать необязательно – при использовании «glut.h» они будут подключены автоматически.

Типы данных, используемые на разных системах, могут существенно различаться. Для обеспечения переносимости программ в OpenGL определены собственные перенумерованные типы, которые будут оставаться неизменяемыми при выполнении программы на каждой из требуемых систем:

```
GLbyte – 8-битовое целое, соответствует типу signed char в C/C++,
GLshort – 16-битовое целое, соответствует типу short в C/C++,
GLint, GLsizei – 32-битовое целое – long в C/C++,
GLfloat, GLclampf – 32-битовое вещественное – float в C/C++,
GLdouble, GLclampd – 64-битовое вещественное – double в C/C++,
GLubyte, GLboolean – 8-битовое целое без знака – unsigned char,
GLushort – 16-битовое целое, – unsigned short в C/C++,
GLuint, GLenum – 32-битовое целое без знака – unsigned long,
GLbitfield – поля двоичных разрядов.
```

Все типы данных, используемые в программе, должны начинаться с заглавных символов GL. Типы GLsizei и GLclamp(f/d) введены для более удобной «читаемости» исходных кодов программ, использующих библиотеки OpenGL. GLsizei принято использовать для переменных, содержащих целочисленные значения размеров или глубины. Имена GLclampf/GLclampd «подсказывают», что соответствующие переменные будут ограничены согласно допустимому диапазону 0,0 – 1,0 (clamped – «зажато», англ.).

Массивы и указатели обозначаются аналогично нотации C/C++:

```
GLint  ndigits[10]; // Массив из 10 переменных типа GLint;
GLdouble *doubles[10]; // Массив из 10 указателей на переменные
типа GLdouble.
```

Названия функций OpenGL представляют собой составные конструкции, которые начинаются с прописных символов имени библиотеки, к которой они относятся, – *gl, glu, glut, glt, aux и т.д.*, и имеют следующий формат: *<Префикс библиотеки> <Имя команды> <Необязательный счётчик аргументов> <Необязательный тип аргументов>* (рис. 2.1).

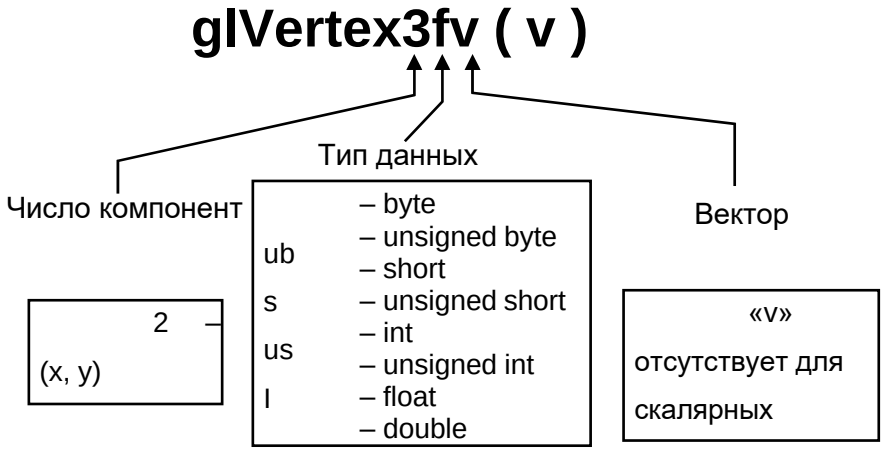


Рис. 2.1. Пример расшифровки названия функции OpenGL

Все отображаемые с помощью OpenGL сцены состоят из ограниченного набора примитивов: GL_POINTS, GL_LINE_STRIP, GL_LINE_LOOP, GL_LINES, GL_TRIANGLES, GL_TRIANGLE_STRIP, GL_TRIANGLE_FAN, GL_QUADS, GL_QUAD_STRIP, GL_POLYGON (рис. 2.2).

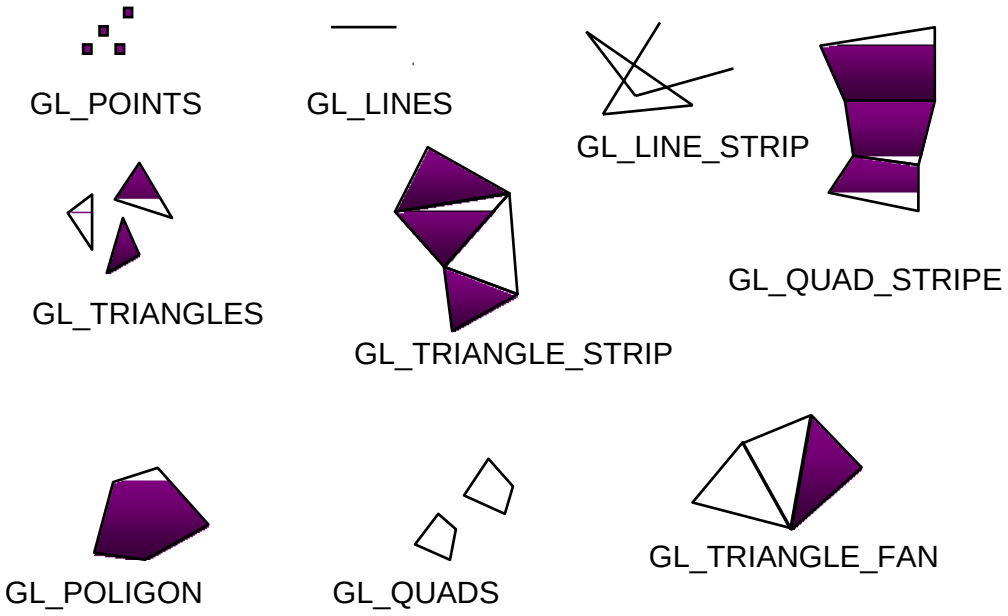


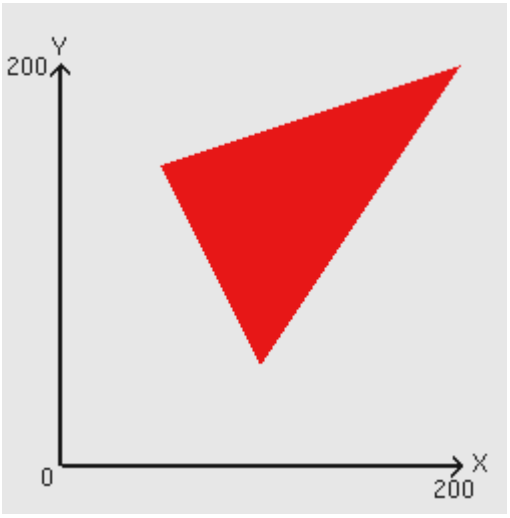
Рис. 2.2. Примеры наиболее часто используемых примитивов OpenGL

Для задания примитивов используется конструкция

```
glBegin( PRIMITIVE_TYPE );
// список вершин
glEnd();
```

где *PRIMITIVE_TYPE* – константа, определяющая последовательность вершин.

```
glBegin(GL_TRIANGLES);
  glColor3f( 1.0, 0.0, 0.0);
  glVertex2f(150.0f, 50.0f);
  glVertex2f(50.0f, 150.0f);
  glVertex2f(200.0f, 200.0f);
glEnd();
```



а б

Рис. 2.3. Пример рисования треугольника.

а) – часть кода программы, б) – отображение треугольника

Если рассматривать минимальный состав консольной программы ОС Windows, использующей функции библиотеки OpenGL, то код функции main на языке C/C++ будет иметь вид

```
void main (void)
{   glutInitDisplayMode(GLUT_SINGLE, GLUT_RGB);
    glutCreateWindow("Пример");
    glutDisplayFunc(MyDrawFunction);
    MyInitFunction();
    glutMainLoop();
}
```

Из приведённых выше функций библиотеки GLUT первая отвечает за инициализацию режима отображения – в данном случае с использованием одного пиксельного буфера (GLUT_SINGLE) и цветовой системы координат RGB (GLUT_RGB). Функция *glutCreateWindow("Пример")* создаёт окно с заголовком «Пример», в котором будет производиться вывод сцены.

Функция *glutDisplayFunc()* устанавливает с помощью механизма обратного вызова указатель на авторскую функцию *MyDrawFunction()* для её вызова и выполнения при каждой необходимости отрисовки окна. Именно в функции *MyDrawFunction()* производится задание элементов сцены, которые будут отображены. Дополнительной авторской функцией является *MyInitFunction()*,

вызываемая однократно для инициализации состояния OpenGL. В принципе, команды инициализации OpenGL, содержащиеся в этой функции, не обязательно оформлять в виде отдельного функционального модуля, однако с точки зрения хорошего стиля программирования лучше их всё же обособить.

Последней командой должна являться *glutMainLoop()*, вызываемая однократно и выполняемая до завершения программы. Она запускает все механизмы OpenGL, обрабатывает все сообщения ОС, нажатий клавиш и т.п.

В свою очередь, функция инициализации может содержать минимум команд:

```
void MyInitFunction(void)
{
  glClearColor(0.0f, 0.0f, 1.0f, 1.0f);
  gluOrtho2D (0.0, 400.0, 0.0, 300.0);
}
```

В данном случае с помощью *glClearColor()* при инициализации задаётся фоновый цвет экрана – так называемый «цвет очистки экрана». Функция *gluOrtho2D()* задаёт план проекции и диапазоны значений X и Y в этой проекции – без функций подобного назначения сцена будет оставаться «непривязанной» к окну отображения, со всеми вытекающими последствиями.

Тело функции *MyDrawFunction()* должно содержать как минимум вызовы двух функций OpenGL – *glClear()* и *glFlush()*, между которыми производится задание объектов сцены:

```
void MyDrawFunction(void)
{
  glClear(GL_COLOR_BUFFER_BIT); //Очистка – т.е., заливка
              // пиксельного буфера цветом очистки экрана
  glBegin(GL_TRIANGLES);
    glColor3f( 1.0, 0.0, 0.0);
    glVertex2f(150.0f, 50.0f);
    glVertex2f(50.0f, 150.0f);
    glVertex2f(200.0f, 200.0f);
  glEnd();
  glFlush();  // Указание выполнения всех невыполненных конвейером
              // OpenGL команд
}
```

Если собрать приведённые выше части программы в единый файл, скомпилировать её и запустить на выполнение, то в окне программы будет выведен треугольник, отображённый на рис. 2.3, б.

Метод обратного вызова часто применяется в OpenGL, в частности:

– *glutReshapeFunc(MyChangeSizeFunction)* позволяет вызвать в случае изменения размеров окна авторскую функцию *MyChangeSizeFunction()*;

– *glutKeyboardFunc(MyKeyProcessingFunction)* вызовет в случае нажатия ASCII-клавиши клавиатуры обработчик события – *MyKeyProcessingFunction();*

– *glutMouseFunc(MyMouseProcessingFunction)* в случае поступления сообщения от «мыши» вызовет функцию *MyMouseProcessingFunction();*

– *glutSpecialFunc(MySpecialKeyProcessingFunction)* при нажатии функциональной клавиши вызовет функцию *MySpecialKeyProcessingFunction()*;

– *glutTimerFunc(50, MyTimerProcessingFunction, 1)* регистрирует функцию MyTimerProcessingFunction(), которая будет вызвана по прошествии 50 ms.

В OpenGL возможны два способа добавления динамики в сцену: 1) изменение координат точки (и/или других параметров) наблюдения и 2) анимация – изменение взаимного расположения объектов сцены. В простейшем случае подобные изменения можно производить с помощью динамического пересчёта координат отображаемого объекта и вызова функции перерисовки изображения по прошествии заданного интервала времени.

Принудительное обновление текущего окна осуществляется с помощью вызова функции *glutPostRedisplay()*. Для многократного вызова функции, пересчитывающей координаты отображаемых объектов, необходимо использовать рекурсивный вызов функции *glutTimerFunc()*, так как таймер в OpenGL срабатывает при вызове однократно. Пример подобной рекурсии:

```
void   MyTimerProcessingFunction (int value)
{
    x1 ++; y1 ++;  // Инкрементация координат вершин треугольника
    x2 ++; y2 ++;
    glutPostRedisplay();
    glutTimerFunc(50, MyTimerProcessingFunction,1);
}
```

Соответственно, первый вызов *glutTimerFunc(50, MyTimerProcessingFunction,1)* должен находиться в основной функции программы – *main()*.

Для более плавного отображения анимации динамические сцены рекомендуется отображать с использованием двойной буферизации, для чего константу GLUT_SINGLE следует заменить на GLUT_DOUBLE, а функцию *glFlush()* – на *glutSwapBuffers()*.

2.3. Примеры типовых заданий к лабораторной работе

1. Разработать программу, использующую функции библиотек OpenGL, отображения в заданном окне движущихся по произвольным траекториям различных фигур – треугольников, квадратов, ломаных линий, лент из треугольников и четырёхугольников.
2. Используя функции библиотек OpenGL, разработать программу, аналогичную стандартному скринсейверу Windows, в котором множество ломаных линий (замкнутых и/или не замкнутых) двигается в границах окна по единой траектории с заданным шагом/задержкой, отражаясь от границ окна.
3. Разработать программу отображения произвольной осмысленной двухмерной сцены, содержащей не менее 50 разных примитивов OpenGL, часть которых должна изменять своё положение с течением времени.

2.4. Литература по теме работы

1. Херн, Д. Микрокомпьютерная графика и стандарт OpenGL/ Д. Херн, М.Бейкер. – М.: “Вильямс”, 2005, §§3.2 - 3.4.
2. Райт-мл., Р.С. OpenGL суперкнига/ Р.С. Райт-мл., Б. Липчак. – М.: “Вильямс”, 2005, §§2-3.
3. The Industry's Foundation for High Performance Graphics. OpenGL. [Электронный ресурс]. – 1992–2009. – Режим доступа: http://www.opengl.org/resources/libraries/glut/

4.

ЛАБОРАТОРНАЯ РАБОТА №6. WEB – интерфейс на основе html тегов.

«HTML Формы»

1. Общие сведения о формах

Некоторые WWW browser позволяют пользователю, заполнив специальную форму, возвращающую полученное значение, выполнять некоторые действия на вашем WWW-сервере. Когда форма интерпретируется WEB-броузером, создается специальные экранные элементы, такие, как поля ввода, checkboxes, radiobuttons, выпадающие меню, скроллируемые списки, кнопки и т.д. Когда пользователь заполняет форму и нажимает кнопку "Подтверждение" (SUBMIT - специальный тип кнопки, который задается при описании документа), информация, введенна пользователем в форму, посылается HTTP-серверу для обработки и передаче другим программам, работающим под сервером. Все формы начинаются тэгом <FORM> и звершаются тэгом </FORM>.

Синтаксис: <FORM METHOD="get | post" ACTION="URL">
Элементы формы и другие элементы HTML
</FORM>

Атрибуты:

METHOD	Метод посылки сообщения с данными из формы. В зависимости от используемого метода вы можете посылать результаты ввода данных в форму двумя путями: • GET: Информация из формы добавляется в конец URL, который был указан в описании заголовка формы. • POST: Данный метод передает всю информацию о форме немедленно после обращения к указанному URL в теле запроса.
ACTION	ACTION описывает URL, который будет вызываться для обработки формы. Данный URL почти всегда указывает на программу, обрабатывающую данную форму.

2. Тэги Формы

2.1. TEXTAREA

Тэг <TEXTAREA> используется для того, чтобы позволить пользователю вводить более одной строки информации (свободный текст). Если вы хотите, чтобы в поле ввода по умолчанию выдавался какой-либо текст, то необходимо вставить его внутри тэгов <TEXTAREA> и </TEXTAREA>.

Синтаксис: <TEXTAREA NAME="" ROWS= COLS= > </TEXTAREA>

Атрибуты:

NAME	Имя поля ввода
ROWS	Высота поля ввода в символах
COLS	Ширина поля ввода в символах

Пример1: Example1.html

```
<HTML>
<HEAD> <meta charset="utf-8"><TITLE>Пример 1</TITLE></HEAD>
<BODY>
    <FORM METHOD="post" ACTION="Example2.html">
      <TEXTAREA NAME="address" ROWS=10 COLS=50> Москва, Дмитровкое шоссе, д.9Б, офис
      </TEXTAREA>
      <INPUT TYPE= "SUBMIT"  VALUE=" Готово!"><br>
    </FORM>
</BODY></HTML>
```

Когда вы описываете форму, каждый элемент ввода данных имеет тэг <INPUT>. Когда пользователь помещает данные в элемент формы, инфоромация размещается в разделе VALUE данного элемента.

2.2. INPUT

Тэг <INPUT> используется для ввода одной строки текста или одного слова.

Атрибуты:

TYPE	Определяет тип поля ввода. По умолчанию это простое поле ввода для одной строки текста. Остальные типы должны быть явно указаны:
TEXT	Данный тип поля ввода описывает однострочное поле ввода. Используйте атрибуты MAXLENGTH и SIZE для определения максимальной длинны вводимого значения в символах и размера отображаемого поля ввода на экране (по умолчанию принимается 20 символов).
PASSWORD	То же самое, что и атрибут TEXT, но вводимое пользователем значение не отображается броузером на экране.
RADIO	Данный атрибут позволяет вводить одно значение из нескольких альтернатив. Для создания набора альтернатив вам необходимо создать несколько полей ввода с атрибутом TYPE="RADIO" с разными значениями атрибута VALUE, но с одинаковыми значениями атрибута NAME. В CGI-программу будет передано значение типа NAME=VALUE, причем VALUE примет значение атрибута VALUE того поля ввода, которое в данный момент будет выбрано (будет активным). При выборе одного из полей ввода типа RADIO все остальные поля данного типа с тем же именем (атрибут NAME) автоматически станут невыбранными на экране.
CHECKBOX	Используется для простых логических (BOOLEAN) значений. Значение, ассоциированное с именем данного поля, которое будет передаваться в вызываемую программу, может принимать значение ON или OFF.

IMAGE	Данный тип поля ввода позволяет вам связывать графический рисунок с именем поля. При нажатии мышью на какую-либо часть рисунка будет немедленно вызвана ассоциированная форме программа. Значения, присвоенные переменной NAME будут выглядеть так - создается две новых переменных: первая имеет имя, обозначенное в поле NAME с добавлением .x в конце имени. В эту переменную будет помещена X-координата точки в пикселах (считая началом координат левый верхний угол рисунка), на которую указывал курсор мыши в момент нажатия, а переменная с именем, содержащимся в NAME и добавленным .y, будет содержать Y-координату. Все значения атрибута VALUE игнорируются. Само описание картинки осуществляется через атрибут SRC и по синтаксису совпадает с тэгом <IMG>.
HIDDEN	Поля данного типа не отображаются броузером и не дают пользователю изменять присвоенные данному полю по умолчанию значение. Это поле используетс для передачи в программу статической информации, как то ID прользователя, пароля или другой информации.
SUBMIT	Данный тип обозначает кнопку, при нажатии которой будет вызвана CGI-программа (или URL), описанная в заголовке формы. Атрибут VALUE может содержать строку, которая будет высвечена на кнопке.
RESET	Данный тип обозначает кнопку, при нажатии которой все поля формы примут значения, описанные для них по умолчанию.
NAME	Имя поля ввода. Данное имя используется как уникальный идентификатор поля, по которому, впоследствии, вы сможете получить данные, помещенные пользователем в это поле.
SIZE	Определяет визуальный размер поля ввода на экране в символах.
MAXLENGTH	Определяет количество символов, которое пользователи могут ввести в поле ввода. При превышении количества допустимых символов броузер реагирует на попытку ввода нового символа звуковым сигналом и не дает его ввести. Не путать с атрибутом SIZE. Если MAXLENGTH больше чем SIZE, то в поле осуществляется скроллинг. По умолчанию значение MAXLENGTH не ограничено.
CHECKED	Означает, что CHECKBOX или RADIOBUTTON будет выбран.
SRC	URL,. указывающий на картинку (используется совместно с атрибутом IMAGE).

VALUE	Присваивает полю значение по умолчанию или значение, которое будет выбрано при использовании типа RADIO (для типа RADIO данный атрибут обязателен)

Пример 2: Example2.html

```
<HTML>
<HEAD>
<meta charset="utf-8">
<TITLE>Пример 2</TITLE></HEAD>
<BODY>
    < FORM METHOD="post" ACTION="Example1.html">
      <INPUT TYPE= "TEXT" NAME="N1" SIZE="20" MAXLENGTH=30" VALUE=" "><br>
      <INPUT TYPE= "RADIO" NAME="N2"  VALUE=" " CHECKED> 1<br>
      <INPUT TYPE= "RADIO" NAME="N2"  VALUE=" "> 2<br>
      <INPUT TYPE= "SUBMIT"  VALUE=" Submit"><br>
    </FOFM>
<p>
 <form action="Example2.html">
  <p><b>Как по вашему мнению расшифровывается аббревиатура &amp;quot;ОС&amp;quot;?</b></p>
  <p><input type="radio" name="answer" value="a1">Офицерский состав<Br>
  <input type="radio" name="answer" value="a2">Операционная система<Br>
  <input type="radio" name="answer" value="a3">Большой полосатый мух</p>
  <p><input type="submit"></p>
 </form>
</p>
 </body>
</html>
```

2.3. Меню выбора в формах

Под меню выбора в формах понимают такой элемент интерфейса, как LISTBOX. Существует три типа тэгов меню выбора для форм:

2.3.1. SELECT

Тэг SELECT позволяет пользователю выбрать значение из фиксированного списка значений. Обычно это представлено выпадающим меню. Тэг SELECT имеет один или более параметр пежду стартовым тэгом <SELECT> и завершающим </SELECT>. По умолчанию, первый элемент отображается в строке выбора.

```
<FORM>
<SELECT NAME= >
      <OPTION >
      <OPTION>
</SELECT>
</FORM>
```

2.3.2. SELECT SINGLE

Тэг SELECT SINGLE - это то же самое, что и SELECT, но на экране пользователь видит одновременно несколько элементов выбора (три по умолчанию). Если их больше, то предоставляется автоматический вертикальный скроллинг. Количество одновременно отображаемых элементов определяется атрибутом SIZE.

```
<FORM>
<SELECT SINGLE NAME= SIZE= >
        <OPTION>
        <OPTION>
        <OPTIONS>
</SELECT>
</FORM>
```

2.3.3. SELECT MULTIPLE

Тэг SELECT MULTIPLE похож на тэг SELECT SINGLE, но пользователь может одновременно выбрать более чем один элемент списка. Атрибут SIZE определяет количество одновременно видимых на экране элементов, атрибут MULTIPLE - максимальное количество одновременно выбранных элементов. Если выбрано одновременно несколько значений, то серверу передаютс соответствующее выбранному количество параметров NAME=VALUE с одинаковыми значениями NAME, но разными VALUE.

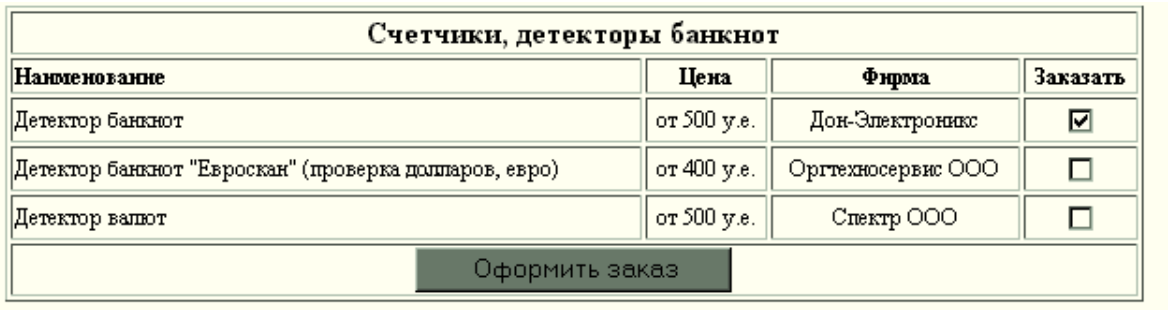
```
<FORM>
<SELECT MULTIPLE NAME= SIZE=
MULTIPLE= >
        <OPTION>
        <OPTION>
        <OPTIONS>
</SELECT>
</FORM>
```

Пример 3: Example3.html

```
<HTML>
<HEAD><TITLE>Пример 3</TITLE></HEAD>
<BODY>
    <FORM METHOD="post" ACTION="Example3.html">
      < b>Оставьте отзыв о посещенном сайте</b><p>
      <b> Фамилия: </b><br>
      <INPUT TYPE=”TEXT” NAME=”name” SIZE=”15” MAXLENGTH=”25” VALUE=” ”><br>
      <b> Имя: </b><br>
      <INPUT TYPE=”TEXT” NAME=”name” SIZE=”10” MAXLENGTH=”15” VALUE=” ”><br>
      <b> Отчество:</b><br>
      <INPUT TYRE=”TEXT” NAME=”name” SIZE=”15” MAXLENGTH=”25” VALUE=” ”><br>
      <b> Как часто Вы посещаете наш сайт:</b><br>
      < SELECT NAME= SIZE=>
          <OPTION VALUE=” ”> Частота посещения сайта
          <OPTION VALUE=”0 ”>Несколько раз в месяц
          <OPTION VALUE=”1 ”> Несколько раз в неделю
          <OPTION VALUE=”2 ”> Каждый день
          <OPTION VALUE=”3 ”> Это мой первый визит
          <OPTION VALUE=”4 ”> Не посещаю
      </SELECT><br>
      <b> Ваши замечания:</b><br>
      <TEXTAREA NAME="Comments" ROWS=3 COLS=40></TEXTAREA><br>
      <INPUT TYPE= "SUBMIT"  VALUE=" Submit"><br>
    </FORM></BODY></HTML>
```

3. ЗАДАНИЕ

1. Отладьте примеры 1 и 2 (там есть синтаксические ошибки). Убедитесь, что они могут работать совместно.
2. Отладьте пример 3.
3. Продолжите, расширьте анкету (пример 3). Используйте при создании формы все возможные атрибуты формы, максимальное число элементов.
4. Создайте страницу с формой «виртуальной торговой тележки».



Счетчики, детекторы банкнот			
Наименование	**Цена**	**Фирма**	**Заказать**
Детектор банкнот	от 500 у.е.	Дон-Электроникс	☑
Детектор банкнот "Евроскан" (проверка долларов, евро)	от 400 у.е.	Оргтехносервис ООО	☐
Детектор валют	от 500 у.е.	Спектр ООО	☐
Оформить заказ			

При нажатии кнопки «Оформить заказ» проверьте, выбран ли хотя бы один из предложенных заказов.

Если ни один из заказов не выбран, выдайте предупреждение : « Не выбран ни один заказ!».

Если хотя бы один заказ выбран, перейдите на страницу регистрации пользователя.

Создайте страницу содержащую сообщение «Необходима регистрация!» и форму для ввода полей “UserName” и “Password”, содержащие две кнопки: “OK”, “Cancel”.

При нажатии кнопки “Cancel” перейдите на страницу с выводом сообщения «Заказы оформляются только для зарегистрированных пользователей!».

При нажатии кнопки “OK” проверьте, введены ли поля “UserName” и “Password”, если нет – выдайте предупреждение, если да - перейдите на страницу с выводом сообщения «Ваш заказ оформлен».

Лабораторная работа 7. Использование каскадных таблицы стилей (CSS).

Цель работы:

а) ознакомление с каскадными таблицами стилей (CSS);

б) ознакомление с базовым синтаксисом, основными элементами CSS - документа;

в) приобретение навыков создания HTML – документов с использованием CSS.

Теоретические основы

Расшифровывается CSS (англ. *Cascading Style Sheets)* как каскадные таблицы стилей и является технологией оформления веб-страниц.

Основным понятием CSS является стиль – т. е. набор правил оформления и форматирования, который может быть применен к различным элементам документа. В стандартном HTML для присвоения какому-либо элементу определенных свойств (таких, как цвет, размер, положение на странице и т. п.) приходилось каждый раз описывать эти свойства, увеличивая размер файла и время загрузки на компьютер просматривающего ее пользователя.

CSS действует более удобным и экономичным способом. Для присвоения какому-либо элементу определенных характеристик необходимо один раз описать этот элемент и определить это описание как стиль, а в дальнейшем просто указывать, что элемент, который нужно оформить соответствующим образом, должен принять свойства указанного стиля.

Более того, можно сохранить описание стиля не в тексте кода документа, а в отдельном файле – это позволит использовать описание стиля на любом количестве Web-страниц, а также изменить оформление любого количества страниц, исправив лишь описание стиля в одном (отдельном) файле.

Кроме того, CSS позволяет работать со шрифтовым оформлением страниц на гораздо более высоком уровне, чем стандартный HTML, избегая излишнего утяжеления страниц графикой.

```
<html>
<head>
<style type="text/css">
.newfont{font-size:24px; color:#CC9933}
</style>
<title>Классы для создания тэгов.</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
```

```
</head>
<body>
<blockquote class=" newfont ">Заголовок</blockquote>
</body>
</html>
```

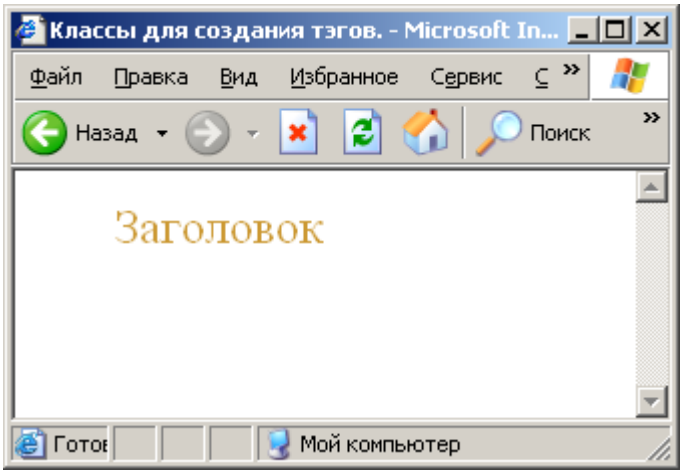


Рисунок 1

Данный пример иллюстрирует вариант объявления нового стиля в документе и потом его использования.

Синтаксис и элементы CSS

1 Добавление стилей CSS в HTML-документ

Существует несколько способов связывания документа и таблицы стилей:

- Связывание - позволяет использовать одну таблицу стилей для форматирования многих страниц HTML
- Внедрение - позволяет задавать все правила таблицы стилей непосредственно в самом документе
- Встраивание в теги документа - позволяет изменять форматирование конкретных элементов страницы
- Импортирование - позволяет встраивать в документ таблицу стилей, расположенную на сервере

Остановимся на каждом из этих способов более подробно.

Связывание. Напомним, что информация о стилях может располагаться либо в отдельном файле, либо непосредственно в коде документа. Расположение описания стилей в отдельном файле целесообразно при применении стилей при количестве страниц более 1. Для этого необходимо создать текстовый файл, описать необходимые стили и в коде документов, которые будут использовать эти стили необходимо создать ссылку на данный файл. Отметим, что данный файл может

располагаться где угодно, необходимым условием является только то, чтобы броузер клиента мог его загрузить на свою сторону. Осуществляется это с помощью тега LINK, располагающегося внутри тега HEAD документов:

<LINK REL=STYLESHEET TYPE="text/css" HREF="URL">

Первые два параметра этого тега являются зарезервированными именами, требующимися для того, чтобы сообщить броузеру, что на этой страничке будет использоваться CSS. Третий параметр – HREF= «URL» – указывает на файл, который содержит описания стилей. Этот параметр должен содержать либо относительный путь к файлу – в случае, если он находится на том же сервере, что и документ, из которого к нему обращаются – или полный URL («http://...») в случае, если файл стилей находится на другом сервере.

```
<head>
<title></title>
<meta http-equiv="content-type" content="text/html; charset=windows-1251">
<link rel="stylesheet" href="css/default.css">
</head>
```

*Внедрение.*Второй вариант, при котором описание стилей располагается в коде Web-странички, внутри тега HEAD, в теге <STYLE type="text/css">... </STYLE. В этом случае вы можете использовать эти стили для элементов, располагающихся в пределах странички. Параметр type="text/css" является обязательным и служит для указания броузеру использовать CSS.

```
<head>
<style type="text/css" >
        .el_cl_1{display:inline; z-index:1};
        .el_lst{display:list-item; margin:-1%; background:#ff0000 url("bc.jpg") no-
repeat};
</style>
<title></title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
</head>
```

*Встраивание в теги документа.*Данный, третий по счету, метод позволяет располагать описания стилей непосредственно внутри тега элемента, который описывает. Это осуществляется при помощи параметра STYLE, используемого при применении CSS с большинством стандартных тегов HTML. Данный метод нежелателен, т.к. приводит к потере одного из основных преимуществ CSS – возможности разделения информации и описания оформления информации.

<blockquote style="color:#CCFF66">Внимание!</blockquote>

Импортировании. В теге <STYLE> можно *импортировать* внешнюю

таблицу стилей с помощью свойства @import таблицы стилей:

@import: url(styles.css);

Его следует задавать в начале стилевого блока или связываемой таблицы стилей перед заданием остальных правил. Значение свойства @import является URL файла таблицы стилей.

Заметим, что импортирование от связывания отличается тем, что при импортировании можно не только поместить внешнюю таблицу стилей в документ, но и поместить одну внешнюю таблицу стилей в другую.

```
<head>
<title>Untitled </title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1251" />
<style type="text/css">
@import url('css/default.css');
</style>
</head>
```

Приведем пример использования свойства текста (рисунке 2.1)

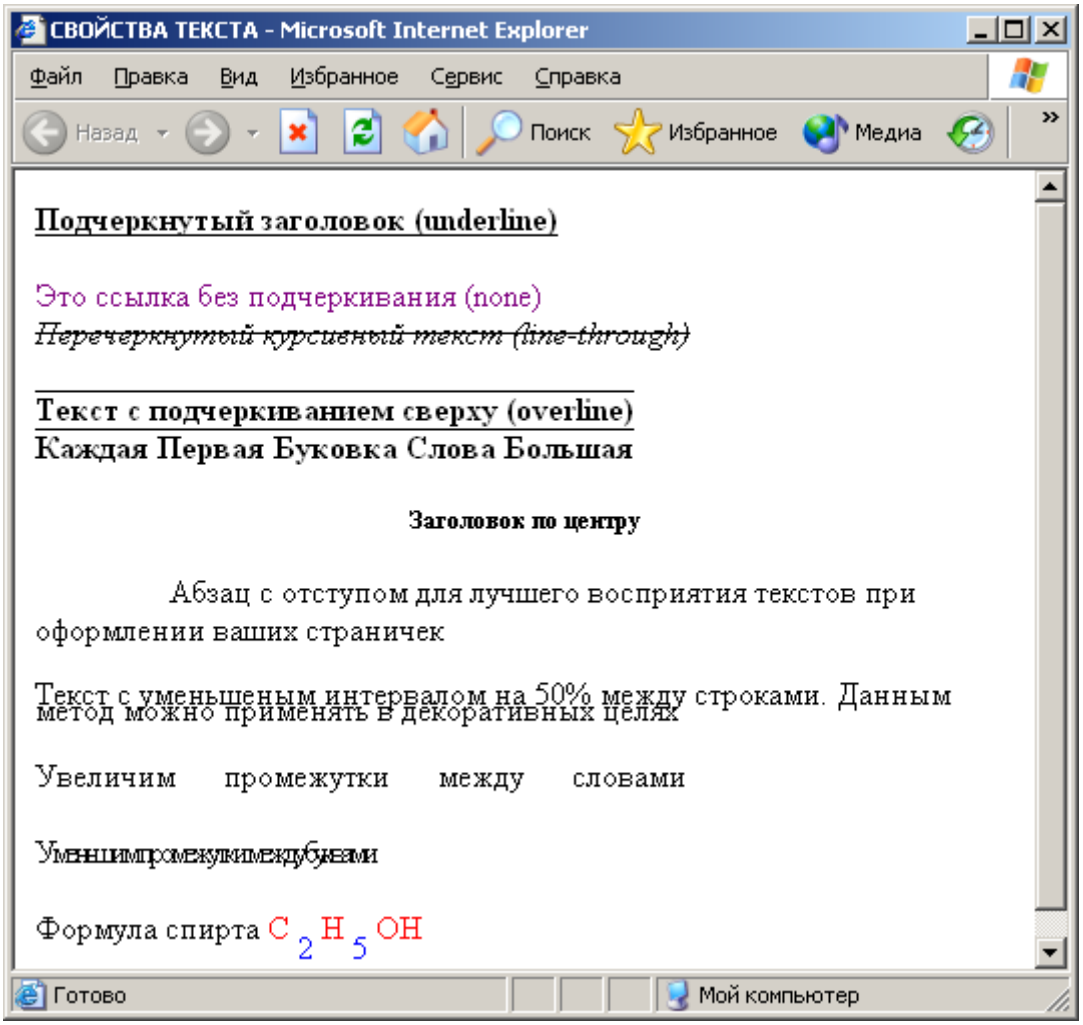


Рисунок 2.1

Ниже приведен код примера.

```
<STYLE type="text/css">
H4 {text-decoration: underline;}
A {text-decoration: none;}
i {text-decoration:line-through;}
b {text-decoration:overline;}
H5 {text-align: center}
b.cap {text-transform:capitalize;}
.otstup {text-indent: 50pt;}
.interval {line-height: 50 %}
</STYLE>
<h4>Подчеркнутый заголовок (underline)</h4>
<a href="/css/003/text.htm">Это ссылка без подчеркивания (none)</a><br>
<i>Перечеркнутый курсивный текст (line-through)</i><p>
<b>Текст с подчеркиванием сверху (overline)</b><br>
```

каждая первая буковка слова большая
<h5>Заголовок по центру</h5>
Абзац с отступом для лучшего восприятия текстов
при оформлении ваших страничек
Текст с уменьшеным интервалом на 50% между строками.
Данным метод можно применять в декоративных целях
Увеличим промежутки между
словами
Уменьшим промежутки между
буквами
Формула спирта
C
2
H
5
OH
4.

Задание:

Получите у преподавателя дизайн HTML-документя и оформите его при помощи CSS. Все стили необходимо вынести в отдельный файл.

Контрольные вопросы

1. Для чего используются каскадные таблицы стилей?
2. Какими способами таблицы стилей связываются с элементами документа?
3. Каковы основные отличия импортирования от связывания?
4. Каким образом сделать так, чтобы изменялся цвет ссылок только внутри тэга ?

Лабораторная работа 8. Использование языка сценариев JavaScript в графических интерфейсах.

Цель занятия: Ознакомить с основами языка JavaScript

Задания:

1. Ознакомьтесь с теоретическими аспектами темы.
2. Создайте простую веб-страницу с использованием JavaScript согласно методическим указаниям.
3. Создайте веб-страницу с формой и кнопкой на основе JavaScript согласно методическим указаниям.
4. Напишите скрипт, печатающий текст «Добро пожаловать на мою страницу! Это JavaScript» три раза подряд. Инструкция по выполнению задания представлена в методических указаниях.
5. Создайте веб-страницу с использованием функции calculation().

Необходимые приборы: ПК, текстовый редактор Блокнот, браузер

Методические рекомендации к выполнению лабораторной работы:

Методические рекомендации к выполнению задания 1

JavaScript - новый язык для составления скриптов, разработанный фирмой Netscape. С помощью JavaScript Вы можете легко создавать интерактивные Web-страницы.

Для запуска скриптов, написанных на языке JavaScript, нужен браузер, способный работать с JavaScript - например Netscape Navigator (начиная с версии 2.0) или Microsoft Internet Explorer (MSIE - начиная с версии 3.0). С тех пор, как оба этих браузера стали широко распространеными, множество людей получили возможность работать со скриптами, написанными на языке JavaScript. Код скрипта JavaScript размещется непосредственно на HTML-странице. Все, что стоит между тэгами <script> и </script>, интерпретируется как код на языке JavaScript. Инструкция*document.write()* - одна из наиболее важных команд, используемых при программировании на языке JavaScript. Команда *document.write()* используется, когда необходимо что-либо написать в текущем документе (в данном случае таком является наш HTML-документ).

События и обработчики событий являются очень важной частью для программирования на языке JavaScript. События, главным образом, инициируются теми или иными действиями пользователя. Если он щелкает по некоторой кнопке, происходит событие *"Click"*. Если указатель мыши пересекает какую-либо ссылку гипертекста - происходит событие *MouseOver*. Существует несколько различных типов событий. Мы можем заставить нашу JavaScript-программу реагировать на некоторые из них. И это может быть выполнено с помощью специальных программ обработки событий. Так, в результате щелчка по кнопке может создаваться

выпадающее окно. Это означает, что создание окна должно быть реакцией на событие щелка - *Click*. Программа - обработчик событий, которую мы должны использовать в данном случае, называется *onClick*. И она сообщает компьютеру, что нужно делать, если произойдет данное событие.

Вы можете использовать в скрипте множество различных типов функций обработки событий. В большинстве случаев функции представляют собой лишь способ связать вместе нескольких команд. Функции могут также использоваться в совместно с процедурами обработки событий.

Методические рекомендации к выполнению задания 2

1. Запустите блокнот
2. Введите текст

```
<html>
<body>
<br>
Это обычный HTML документ.
<br>
<scriptlanguage="JavaScript">
   document.write("Аэто JavaScript!")
</script>
<br>
Вновь документ HTML.
</body>
</html>
```

3. Сохраните документ в формате html
4. Запустите страницу в окне браузера.

Результат выполнения файла в случае, если используемый браузер поддерживает JavaScript:

Это обычный HTML документ.

Если браузер не поддерживает JavaScript, то он проигнорирует тег <script>. В этом случае измените исходный текст:

```
<html>
<body>
<br>
Это обычный HTML документ.
<br>
<script language="JavaScript">
<!-- hide from old browsers
   document.write("Аэто JavaScript!")
```

```
// -->
</script><br>
Вновь документ HTML.
</body>
</html>
```

В этом случае использован тег комментария из HTML - <!-- -->. В результате новый вариант нашего исходного кода будет выглядеть как:

Это обычный HTML документ.
Вновь документ HTML.

Методические рекомендации к выполнению задания 3

1. Создайте новый документ html.
2. Вставьте следующий код

```
<form>
<input type="button" value="Click me" onClick = "alert('Yo')">
</form>
```

3. Просмотрите результат.

В данном примере создается некая форма с кнопкой. Первая новая особенность - onClick="alert('Yo')" в тэге <input>. Таким образом, если имеет место событие *Click*, компьютер должен выполнить вызов alert('Yo'). Это и есть пример кода на языке JavaScript.

Функция *alert()* позволяет Вам создавать выпадающие окна. При ее вызове Вы должны в скобках задать некую строку. В нашем случае это *'Yo'*. И это как раз будет тот текст, что появится в выпадающем окне. Таким образом, когда читатель когда щелкает на кнопке, наш скрипт создает окно, содержащее текст *'Yo'*.

Еще одна особенность данного примера: в команде document.write() мы использовали двойные кавычки ("), а в конструкции alert() - только одинарные. В большинстве случаев Вы можете использовать оба типа кавычек. Однако в последнем примере мы написали onClick="alert('Yo')" - то есть мы использовали и двойные, и одинарные кавычки. Если бы мы написали onClick="alert("Yo")", то компьютер не смог бы разобраться в нашем скрипте, поскольку становится неясно, к которой из частей конструкции имеет отношение функция обработки событий onClick, а к которой - нет. Поэтому мы вынуждены использовать оба типа кавычек. Не имеет значения, в каком порядке Вы использовали кавычки - сначала двойные, а затем одинарные или наоборот. То есть Вы можете точно так же написать и onClick='alert("Yo")'.

Методические рекомендации к выполнению задания 4

1. Напишите скрипт, печатающий текст «Добро пожаловать на мою страницу! Это JavaScript» три раза подряд. Для начала рассмотрим простой подход:

```
<html>
<script language="JavaScript">
```

```
<!-- hide
document.write("Добро пожаловать на мою страницу!<br>");
document.write("Это JavaScript!<br>");
document.write("Добро пожаловать на мою страницу!<br>");
document.write("Это JavaScript!<br>");
document.write("Добро пожаловать на мою страницу!<br>");
document.write("Это JavaScript!<br>");
// -->
</script>
</html>
```

2. Если посмотреть на исходный код скрипта, то видно, что для получения необходимого результата определенная часть его кода была повторена три раза. Эту же задачу можно решить несколько иначе. Введите изменения:

```
<html>
<scriptlanguage="JavaScript">
<!-- hide
functionmyFunction() {
document.write("Добро пожаловать на мою страницу!<br>");
document.write("Это JavaScript!<br>");}
myFunction();
myFunction();
myFunction();
// -->
</script>
</html>
```

В этом скрипте мы определили некую функцию, состоящую из следующих строк:

```
function myFunction() {
  document.write("Добро пожаловать на мою страницу!<br>");
document.write("Это JavaScript!<br>");}
```

Все команды скрипта, что находятся внутри фигурных скобок - {} - принадлежат функции *myFunction()*. Это означает, что обе команды document.write() теперь связаны воедино и могут быть выполнены при вызове указанной функции. И действительно, в нашем примере есть три вызова этой функции В свою очередь, это означает, что содержимое этой функции (команды, указанные в фигурных скобках) было выполнено трижды.

Методические рекомендации к выполнению задания 5

Создайте новую веб-страничку:

```
<html>
<head>
<script language="JavaScript">
```

```
<!-- hide
function calculation() {
  var x= 12;
  var y= 5;
  var result= x + y;
  alert(result);}
// -->
</script>
</head>
<body>
<form>
<input type="button" value="Calculate" onClick="calculation()">
</form>
</body>
</html>
```

Здесь при нажатии на кнопку осуществляется вызов функции *calculation()*. Как можно заметить, эта функция выполняет некие вычисления, пользуясь переменными *x*, *y* и *result*. Переменную мы можем определить с помощью ключевого слова *var*. Переменные могут использоваться для хранения различных величин - чисел, строк текста и т.д. Так строка скрипта var result= x + y; сообщает браузеру о том, что необходимо создать переменную *result* и поместить туда результат выполнения арифметической операции *x* + *y* (т.е. 5 + 12). После этого в переменный *result* будет размещено число *17*. В данном случае команда alert(result) выполняет то же самое, что и alert(17). Иными словами, мы получаем выпадающее окно, в котором написано число 17.

Вопросы для самоконтроля:

1 Для чего предназначен язык JavaScript?
2 Что называют инструкциями?
3 В чем отличие процедур от событий?