ИНФОРМАТИКА

А.Г. Паутова

Visual Basic

ТВОРЧЕСКОЕ ПРОЕКТИРОВАНИЕ В ШКОЛЕ И ДОМА

Часть 3

КЛАССИКС СТИЛЬ

УДК 519.6
ББК 32.973-018
П 21

Паутова А.Г.

П 21 Visual Basic. Творческое проектрование в школе и дома. В 3 ч. Ч. 3. — М.: Классикс Стиль, 2003. — 136 с. ISBN 5-94603-061-2.

Книга предназначена для обучения детей и подростков среднего и старшего школьного возраста началам объектно-ориентированного программирования на языке Visual Basic. Ее можно использовать на уроках информатики, факультативных занятиях, а также для самостоятельного изучения основ программирования.

Книга состоит из трех частей. В первую часть включены 7 проектов, во вторую и третью по 5. От проекта к проекту задания усложняются и становятся интереснее.

Третья часть книги является логическим продолжением первых двух частей. Она содержит описание проектов повышенного уровня сложности и предназначена старшеклассникам, увлекающимся программированием, и учителям информатики. Все проекты, описанные в книге, имеют практическое значение.

УДК 519.6
ББК 32.973-018

Учебное издание

ПАУТОВА Альбина Геннадьевна

VISUAL BASIC

Творческое проектрование в школе и дома

(в трех частях)

Часть 3

Редактор *Н.В. Даниленков*
Корректор *Г.А. Уранова*
Компьютерная графика *Н.В. Мантузова*
Компьютерная верстка *В.Г. Чернышев*

Изд. лиц. ИД № 04691 от 28.04.2001 г.

Издательство «Классикс Стиль»
127018, Москва, Сущевский вал, д. 49, стр. 1, п/я 48.

Подписано в печать 03.03.2003. Формат 70×90/16. Гарнитура «Таймс».
Печать офсетная. Бумага офсетная. Усл. печ. л. 9,95. Тираж 5000 экз. Зак. № 2888.

ОАО «Ивановская областная типография».
153008, г. Иваново, ул. Типографская, 6.
E-mail: 091-018@adminet.ivanovo.ru

ISBN 5-94603-061-2 (ч. 3)
ISBN 5-94603-057-4

Оглавление

Посвящается
Сереже, Володе, Катюше,
Лизе, Ванечке, Саше, Валере
и другим моим ученикам,
которых помню и люблю

Предисловие

Третья часть книги включает пять проектов высокого уровня сложности. В проектах «Графический редактор "Зеркало"» и «Сказки для малышей» подробно раскрыты различные варианты работы с файлами:

- запись в графический файл с расширением *.bmp изображений, созданных в период выполнения программы, написанной на Visual Basic;
- запись и чтение информации из файлов последовательного доступа (текстовых файлов);
- чтение информации из файлов однотабличных баз данных, созданных в программе MS Access (файлы с расширением *.mdb).

Проекты «Калейдоскоп», «Найди клад» и «Тренажер памяти и логического мышления» вносят разнообразие в организацию взаимодействия пользователя с созданными программами. Работая над этими проектами, вы научитесь:

- с помощью события DragDrop программировать перетаскивание объектов по экрану с помощью мышки;
- обрабатывать в программе нажатие управляющих клавиш клавиатуры;
- создавать рисунки с помощью мышки.

Кроме того, вы научитесь проигрывать звуковые файлы с расширением *.wav, выводить на принтер графическую информацию, использовать в своей программе стандартные диалоговые окна (выбора цвета, сохранение файла, открытие файла, вывод на принтер), создавать пользовательское меню.

Создание проектов требует определенной алгоритмической культуры. В программном коде использованы различные циклические конструкции, конструкция множественного выбора, общие процедуры-подпрограммы, процедуры-функции, стандартные функции обработки строковых данных.

В результате работы над проектами вы создадите программы, которые можно будет подарить младшим братьям и сестрам, использовать на уроках информатики и информационных технологий в начальной школе. Программа «Графический редактор "Зеркало"» позволяет создавать, записывать на диск и печатать рисунки, имеющие центральную симметрию. Программа «Сказки для малышей» внесет элементы игры в уроки чтения. Программа «Тренажер памяти и логического мышления» является инструментом для тренировки зрительной памяти и внимания.

Инструментарий программиста, представленный в данной книге, позволит вам приступить к созданию собственных оригинальных творческих проектов, имеющих практическое значение.

Проект 1
«Калейдоскоп»

Работая над проектом, вы научитесь создавать меню команд в проекте, выводить форму на принтер, а также создавать программу, которая позволяет рисовать на экране с помощью мышки.

Постановка задачи

Этот проект предназначен для создания узоров, симметричных относительно четырех осей симметрии. Рисунок возникает, если двигать по форме мышкой с прижатой левой кнопкой. В том месте, где находится указатель курсора, рисуется круг с центром в точке с координатами (x, y). Для этого используется метод **Circle**. После этого программа создает еще семь кругов так, чтобы на форме получился узор, симметричный относительно четырех осей. На рис. 1 показаны оси симметрии, точка A с координатами (x, y), координаты семи точек, образующих симметричный узор, а также пример узора. Координаты точек, указанные на рисунке, справедливы только в том случае, если система координат, связанная с формой, имеет начало в середине формы.

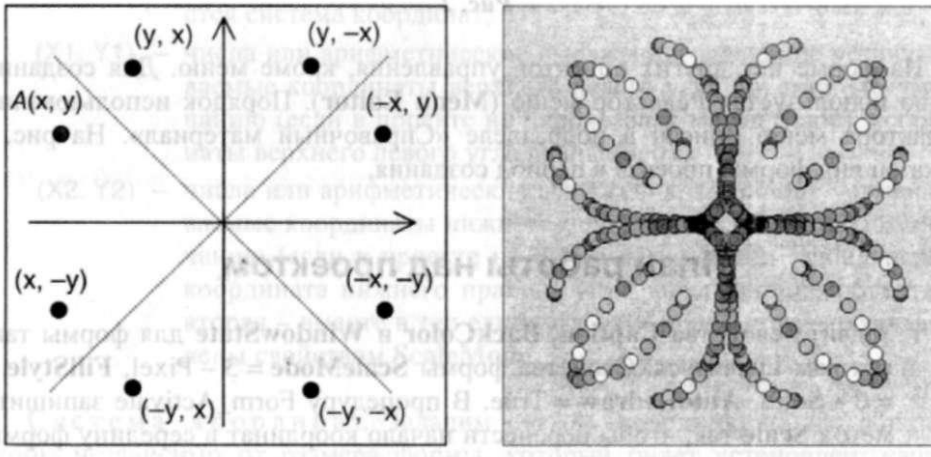


Рис. 1

Наверху формы имеется меню команд. Команды этого меню позволяют:

- стирать рисунок;
- устанавливать начало системы координат в середину формы после изменения ее размера в процессе работы программы;
- выводить полученный рисунок на принтер, установленный на компьютере по умолчанию;
- выходить из программы.

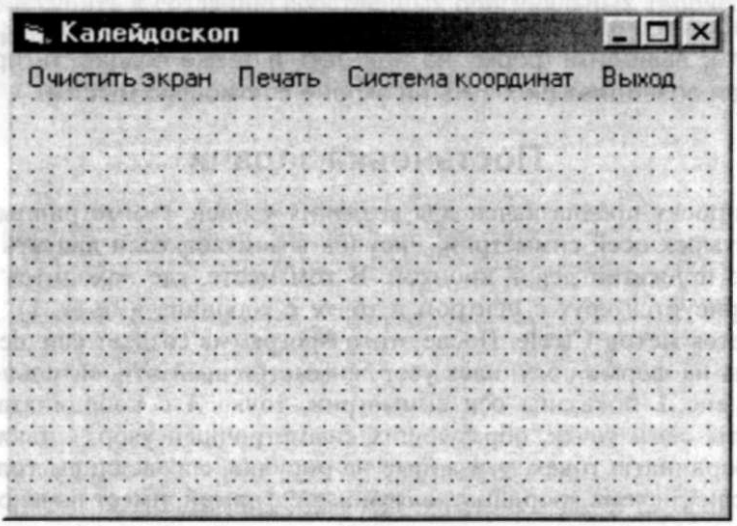


Рис. 2

На форме нет других объектов управления, кроме меню. Для создания меню используется Редактор меню (**Menu Editor**). Порядок использования редактора меню описан в подразделе «Справочный материал». На рис. 2 показан вид формы проекта в период создания.

План работы над проектом

1. Задайте свойства **Caption, BackColor** и **WindowState** для формы так, как вам нравиться. Свойства формы **ScaleMode** = 3 – Pixel, **FillStyle** = = 0 – Solid, **AutoRedraw** = True. В процедуру Form_Activate запишите метод **Scale** так, чтобы перенести начало координат в середину формы и направить ось *OX* вправо, а ось *OY* вверх *(см. п.1, пр.1)*.

2. В процедуру **Form_MouseMove** запишите программный код, который рисует круги вслед за курсором мышки, если мышка перемещается с нажатой левой кнопкой *(см. п.2, пр.1)*.
3. Запишите команды рисования семи симметричных кругов *(см. п.3, пр.1)*.
4. Создайте пользовательское меню *(см. п.4, пр.1)*.
5. Запишите проект на диск, проверьте правильность работы. Если надо, внесите исправления и еще раз сохраните проект.

Справочный материал

1. Метод Scale

С методом **Scale** мы уже встречались в первой части книги (проект 7 «Мини-мультфильм "Утро в лесу"»). Давайте рассмотрим этот метод поподробнее.

Назначение. Метод устанавливает систему координат, связанную с формой или объектом **PictureBox** в соответствии с желанием программиста.

Формат:

<Объект> . **Scale** (X1, Y1) – (X2, Y2)

где Объект – имя формы или объекта **PictureBox** для которого определяется система координат;

(X1, Y1) – числа или арифметические выражения, задающие устанавливаемые координаты верхнего левого угла объекта. По умолчанию (если в проекте не использован метод **Scale**), координаты верхнего левого угла равны (0, 0);

(X2, Y2) – числа или арифметические выражения, задающие устанавливаемые координаты нижнего правого угла объекта. По умолчанию (если в проекте не использован метод **Scale**), первая координата нижнего правого угла равна ширине объекта, а вторая – высоте в тех единицах измерения, которые установлены свойством **ScaleMode**.

Система координат. Зададим систету координат таким образом, чтобы независимо от размера формы, который будет установлен, начало

координат всегда располагалось в середине формы. Рассмотрим рис. 3. В системе координат, задаваемой по умолчанию, координата по оси *OX* точки *A* была равна 0, а точки *W* – ширине объекта (в нашем проекте **Form1.ScaleWidth**). Ось *OY*1 сдвинулась по отношению к оси *OY* на половину ширины объекта. При этом координаты по оси *OX* всех точек, лежащих на объекте, уменьшились на эту же величину. Следовательно, первая координата точки *A* в новой системе координат будет равна:

0 – Form1.ScaleWidth / 2 = – Form1.ScaleWidth / 2.

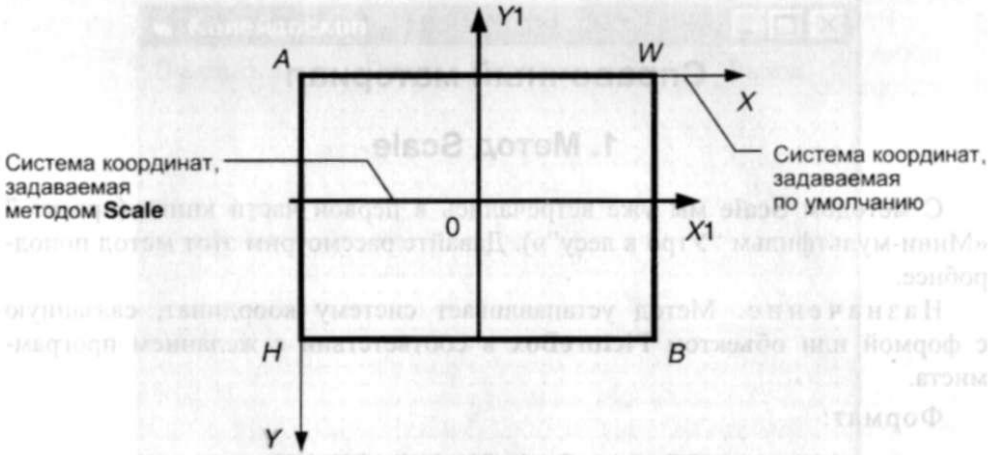


Рис. 3

Аналогично определим изменение второй координаты точки *A*, вызванное перемещением оси *OX* на половину высоты объекта и изменением направления оси *OY*:

– (0 – Form 1.ScaleHeight / 2) = Form1.ScaleHeight / 2

Таким образом, координаты верхней левой точки формы в новой системе координат будут равны:

A(– Form1.ScaleWidth / 2, Form1.ScaleHeight / 2)

Координаты точки *B* соответственно равны:

B(Form1.ScaleWidth / 2, – Form1.ScaleHeight / 2)

Запишем программный код, который установит новую систему координат непосредственно после старта программы. Независимо от размера

формы, заданного в период создания проекта, начало координат будет расположено в середине формы.

```
Private Sub Form_Activate()
Form1.Scale (–Form1.ScaleWidth / 2, Form1.ScaleHeight / 2) _
    –(Form1.ScaleWidth / 2, –Form1.ScaleHeight / 2)
End Sub
```

2. Использование графических методов в процедуре обработки события MouseMove

Вспомните все, что вы знаете о событии **MouseMove** (часть вторая, проект 4 «Раскрась картинку») и о графических методах (там же, проект 5 «Графические методы»), и запишите самостоятельно в процедуру **Form_MouseMove** программный код, реализующий алгоритм, представленный на блок-схеме (рис. 4).

Блок-схема процедуры **Form_MouseMove**

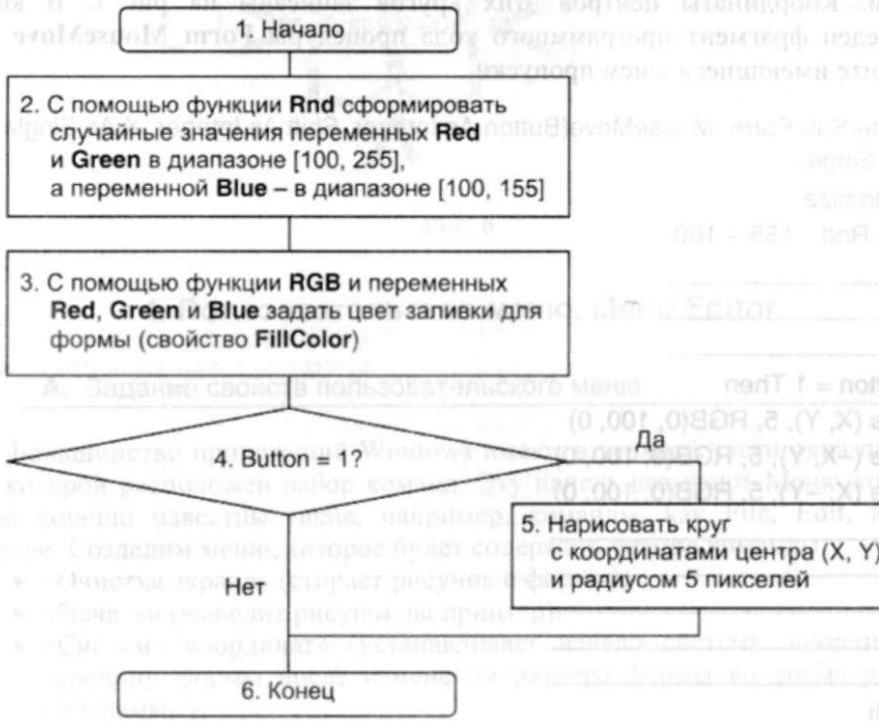


Рис. 4

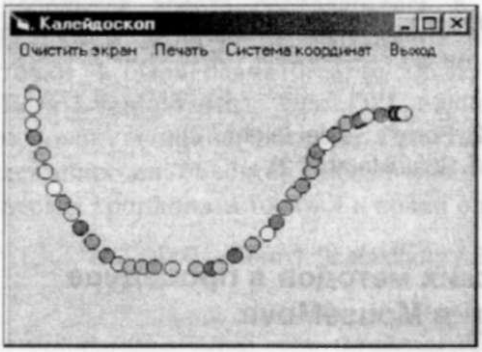


Рис. 5

Запишите программу на диск и проверьте ее работоспособность. Если вслед за мышкой, которую вы перемещаете по форме с прижатой левой кнопкой, тянется цепочка цветных кружочков (рис. 5), вы можете переходить к следующему пункту плана. Если же нет, найдите и исправьте ошибки.

3. Координаты симметричных точек

В программном коде записан метод **Circle**, который рисует окружность с центром в точке с координатами (X, Y). В следующей за ним строке добавьте еще семь методов **Circle**, которые будут рисовать симметричные круги. Координаты центров этих кругов записаны на рис. 1. В книге приведен фрагмент программного кода процедуры **Form_MouseMove**. Заполните имеющиеся в нем пропуски.

```
Private Sub Form_MouseMove(Button As Integer, Shift As Integer, X As Single,
Y As Single)
Randomize
red = Rnd * 155 + 100
  _____________
  _____________
  _____________
If Button = 1 Then
Circle (X, Y), 5, RGB(0, 100, 0)
Circle (–X, Y), 5, RGB(0, 100, 0)
Circle (X, –Y), 5, RGB(0, 100, 0)
  _____________
  _____________
  _____________
  _____________
  _____________
End If
End Sub
```

Проверьте, как работает программа. Если все команды записаны правильно, вы сможете нарисовать множество симметричных узоров, похожих на те, которые приведены на рис. 6.

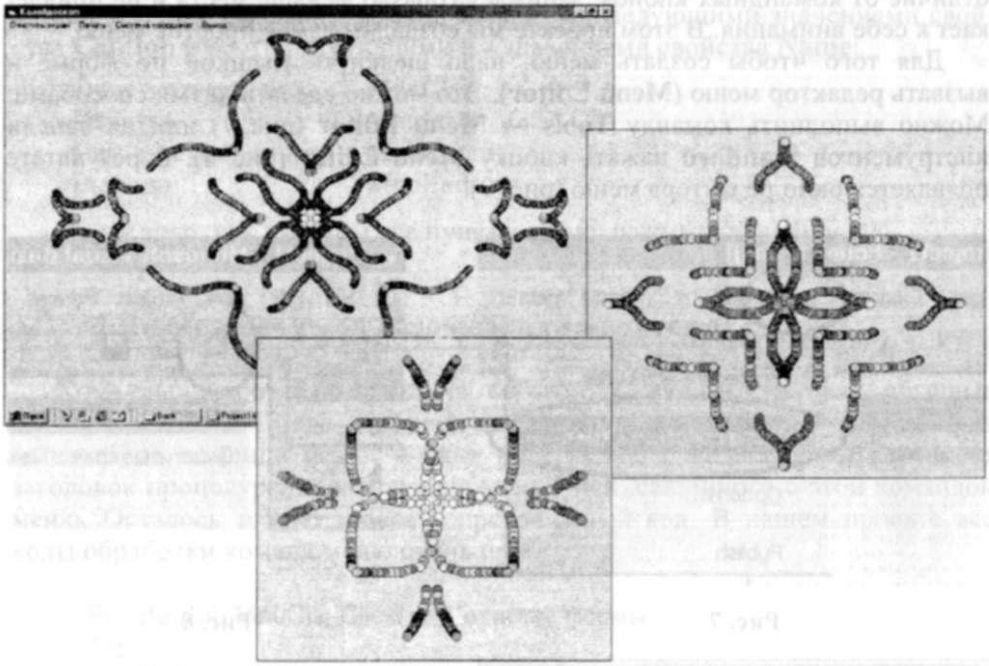


Рис. 6

4. Пользовательское меню. Menu Editor

А. Задание свойств пользовательского меню

Большинство приложений Windows имеют в верхней части окна панель, на которой расположен набор команд. Эту панель называют Меню команд. Нам хорошо известны такие, например, команды, как File, Edit, Run и другие. Создадим меню, которое будет содержать четыре команды:

- «Очистка экрана» (стирает рисунок с формы);
- «Печать» (выводит рисунок на принтер);
- «Система координат» (устанавливает начало системы координат в середину формы после изменения размера формы во время работы программы);
- «Выход» (выход из программы).

Все эти действия можно было бы реализовать при помощи командных кнопок. Но в данном проекте нам особенно важно сохранить свободной всю площадь формы, так как форма предназначена для рисования. Меню, в отличие от командных кнопок, занимает гораздо меньше места и не привлекает к себе внимания. В этом проекте мы создадим самое простое меню.

Для того чтобы создать меню, надо щелкнуть мышкой по форме и вызвать редактор меню (**Menu Editor**). Это можно сделать двумя способами. Можно выполнить команду **Tools → Menu Editor** (рис. 7) или на панели инструментов **Standard** нажать кнопку **Menu Editor** (рис. 8). В результате появляется окно редактора меню (рис. 9).

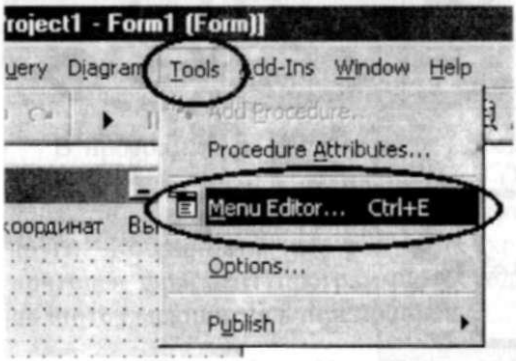


Рис. 7

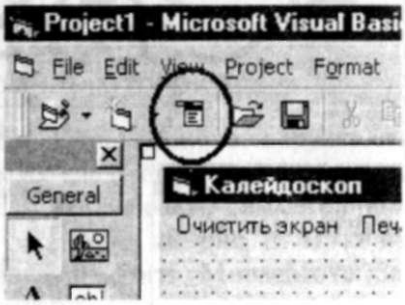


Рис. 8

1. В окне **Caption** запишите название команды, которое вы хотите видеть в меню.
2. В окне **Name** запишите имя команды, которое вы будете использовать в программном коде.
3. Нажмите кнопку **Next**, чтобы создать следующий пункт меню.

Если хотите изменить порядок следования пунктов меню, используйте клавиши со стрелками вверх и вниз.

Рис. 9

Действия 1, 2 и 3 (см. рис. 9) повторяют до тех пор, пока не будут набраны все пункты меню. Договоримся, что имена **Name** пунктов меню будут начинаться с букв **Mnu** (сокращение от слова menu). Вторая половина имени должна иметь смысловую нагрузку и напоминать о назначении команды меню. Создадим команды меню со следующими значениями свойства **Caption** и соответствующими им значениями свойства **Name**:

«Очистка экрана» – MnuCls;
«Печать» – MnuPrint;
«Система координат» – MnuCoord;
«Выход» – MnuEnd.

После того, как набраны все пункты меню, нажмите клавишу OK.

Б. Программные коды пользовательского меню

Команды меню реагируют на событие **Click** так же, как обычные командные кнопки. Если во время создания программы один раз щелкнуть мышкой по команде меню, в окне программного кода формы возникнет заголовок процедуры обработки события **Click**, связанного с этой командой меню. Осталось только записать программный код. В нашем проекте все коды обработки команд меню очень просты:

```
Private Sub MnuCls_Click()   ''очистка формы
Cls
End Sub

Private Sub MnuCoord_Click()   ''задание систеты координат
Form1.Scale (–Form1.ScaleWidth / 2, Form1.ScaleHeight / 2 ) _
     – (Form1.ScaleWidth / 2, –Form1.ScaleHeight / 2)
End Sub

Private Sub MnuEnd_Click()        ''выход из программы
End
End Sub

Private Sub MnuPrint_Click()      ''вывод рисунка на печать
Form1.PrintForm
End Sub
```

Обратите внимание на то, что программный код в процедурах **MnuCoord_Click** и **Form_Activate** абсолютно одинаковый. Событие **Activate** происходит только в начале работы программы. Если во время

работы программы были изменены размеры формы, надо до начала рисования заново задать систему координат с началом в середине формы. Для этих целей мы и будет щелкать по пункту меню «Система координат».

B. Метод **PrintForm**

Метод **PrintForm** выводит на принтер изображение формы со всеми объектами, видимыми в момент выполнения метода.

ВНИМАНИЕ! Изображение, созданное во время работы программы при помощи графических методов **Pset, Line, Circle**, только в том случае выводится на принтер, если свойство **Autoredraw** формы имеет значение **True**.

Если к вашему компьютеру подключено более одного принтера, будет использоваться тот, который задан операционной системой по умолчанию.

Проект 2

«Графический редактор “Зеркало”»

Работая над проектом, вы познакомитесь с объектами **ProgressBar**, **CommonDialog**, **Printer** и графическими методами **Point** и **PaintPicture**, научитесь создавать диалоговые окна выбора цвета, сохранения и открытия файла, сохранять на диске и выводить на принтер изображения, созданные в программе Visual Basic.

Постановка задачи

Описание проекта

Проект представляет собой графический редактор с круглой областью рисования. Рисунок может быть отражен относительно четырех осей симметрии по выбору пользователя. Симметричное отражение рисунка производится после нажатия на командную кнопку, на которой изображена ось симметрии (рис. 10).

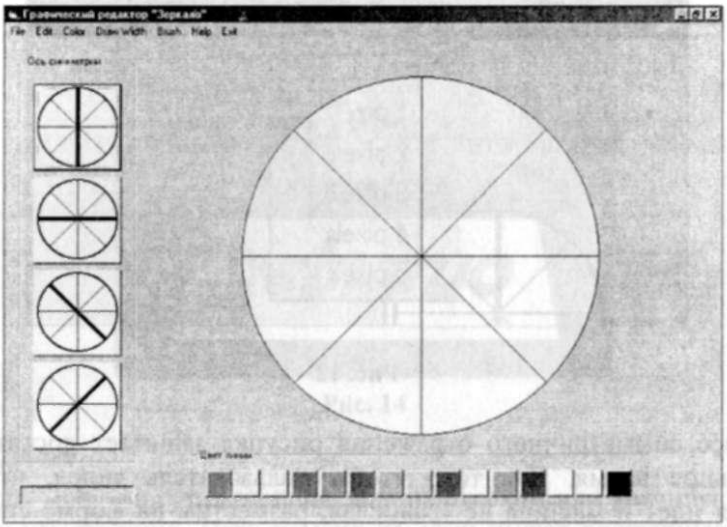


Рис. 10

Рисовать на рабочем поле можно мышкой с нажатой левой кнопкой. Пользователь может выбрать одну из трех имеющихся кистей – точка, окружность, еловая лапа. Образцы кистей вы видите на рис. 11. Пользователь также может изменить цвет линии и цвет фона.

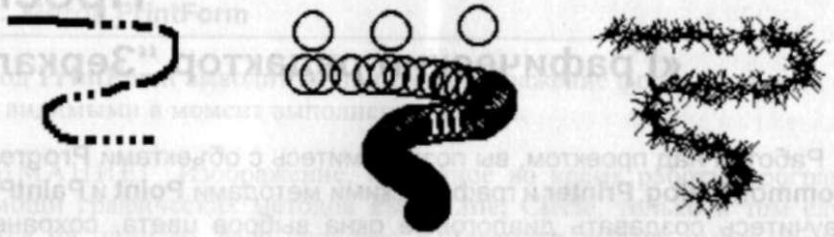


Рис. 11

Цвет фона можно выбрать при помощи диалогового окна, которое вызывается командой меню. Цвет линии также можно выбрать в диалоговом окне или при помощи группы объектов **OptionButton**. В меню также можно выбрать толщину линий рисования. При этом в меню галочкой отмечается выбранное значение (рис. 12). Созданный рисунок можно записать на диск в файл с расширением ***.bmp** и распечатать на принтере. С диска можно прочитать файл с рисунком и отредактировать его.

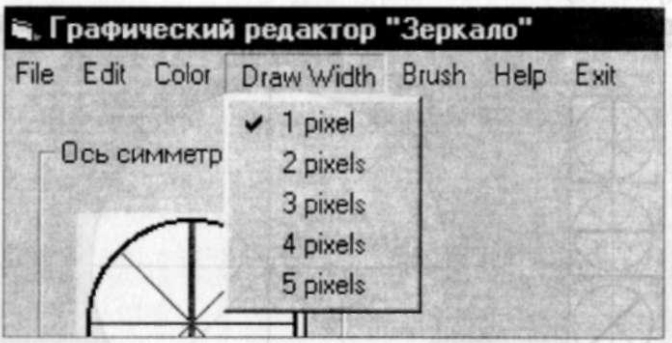


Рис. 12

Процесс симметричного отражения рисунка занимает достаточно продолжительное время. Для того чтобы пользователь видел, что процесс отражения идет и машина не «зависла», разместим на форме специальный объект – **ProgressBar**.

На рис. 13 показана последовательность создания изображения с помощью графического редактора «Зеркало».

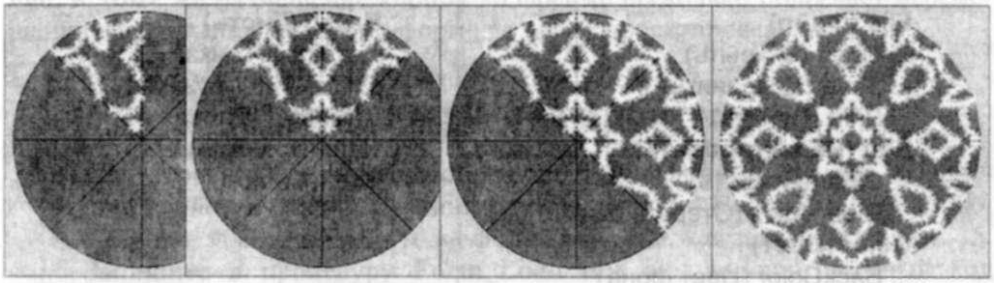


Рис. 13

Набор форм проекта

Проект состоит из трех форм. Форма с именем **FrmTitle** (рис. 14) представляет собой заголовок программы. Здесь расположено ее название, имя, фамилия автора и руководителя проекта (если он есть), год создания проекта. Через 2 секунды после открытия эта форма закрывается и вызывает главную форму **FrmMain** (см. рис. 10). При нажатии на одну из команд меню можно вызвать форму **FrmHelp**, которая содержит описание программы и примеры ее использования. Подробно мы рассмотрим создание основной формы **FrmMain**. Создание остальных форм труда не представляет.

Рис. 14

Структура меню

Команды меню могут быть написаны на русском или английском языках по вашему усмотрению. Пять команд из семи имеют вложенные команды, которые появляются при щелчке мышкой по основной команде. Список

команд меню (выделены жирным шрифтом) и списки вложенных команд (выпадающих при нажатии на кнопки с командами меню):

File (Файл)
... Open (Открыть)
... Save (Сохранить)
... Print (Печать)
Edit (Редактировать)
... Clear (Стереть)
... Clear Axes (Стереть оси)
Color (Цвет)
... BackColor (Цвет фона)
... ForeColor (Цвет линии)
Draw Width (Толщина линии)
... 1 pixel (1 пиксель)
... 2 pixels (2 пикселя)
... 3 pixels (3 пикселя)
... 4 pixels (4 пикселя)
... 5 pixels (5 пикселей)
Brush (Кисть)
... Point (Точка)
... Circle (Окружность)
... Fir tree (Еловая ветка)
Help (Помощь)
Exit (Выход)

*Основные объекты формы **FrmMain***

Создание проекта начнем с формы **FrmMain**. Сначала разместим на форме объекты в соответствии со схемой, показанной на рис. 15. Слева на форме размещен объект **Frame**, а в нем четыре командные кнопки, образующие

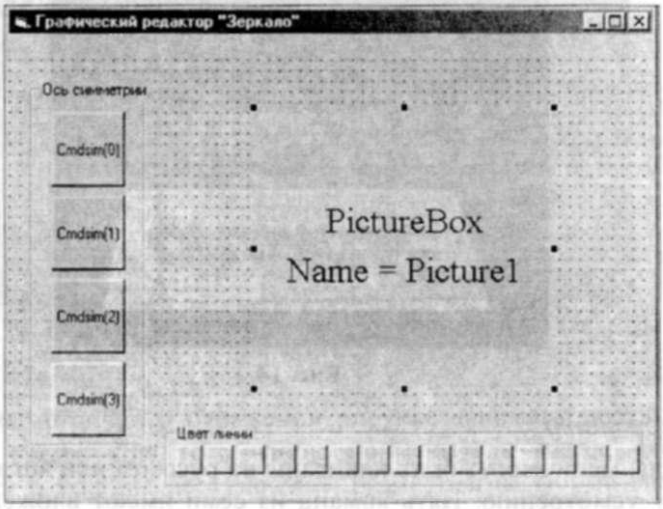


Рис. 15

массив объектов с именем **Cmdsim**. Внизу формы – во фрейме 15 радиокнопок (объекты **OptionButton**). Фреймы имеют чисто декоративное значение. К тому же размещение кнопок во фрейме несколько облегчает перемещение всей группы кнопок по форме во время проектирования.

План работы над проектом

1. Подготовьте форму **FrmMain** к проектированию – задайте следующие свойства: **Name, Caption, ScaleMode** = 3 – Pixel, **Width** = 12000, **Height** = 9000. Разместите на форме объекты, как показано на рис. 15. Запишите в процедуру **Form_Load** и в модуль программный код создания круглой области рисования с осями симметрии и задания начальных характеристик режима рисования *(см. пп. 1.А и 1.Б, пр.2)*.
2. Запишите программный код, раскрашивающий **OptionButton** *(см. п.1.В, пр.2)*
3. Создайте меню, содержащее команду выбора кисти рисования и запишите программный код, реализующий выполнение этой команды *(см. п.2, пр.2)*.
4. Запишите в процедуру **Picture1_MouseMove** программный код рисования выбранной кистью *(см. п.3, пр.2)*.
5. Запишите программный код, позволяющий выбирать цвет рисования с помощью радиокнопок *(см. п.4, пр.2)*.
6. Оформите командные кнопки **Cmdsim**. Создайте рисунки в графическом редакторе, сохраните их в файле и поместите на кнопки при помощи свойства **Picture** (Style = Graphical). При создании рисунков помните, что размер рисунка должен быть на два пикселя меньше, чем ширина и высота кнопки. Надписей на кнопках нет.
7. Запишите в процедуру **Private Sub Cmdsim_Click** код, выполняющий симметричное отражение рисунка (*см. п.5, пр.2*).
8. Добавьте объект **ProgressBar** и запишите программный код, обеспечивающий действие этого объекта *(см. п.6, пр.2)*.
9. Добавьте в меню команду выбора толщины линии рисования и запишите программный код, реализующий выполнение этой команды *(см. п.7, пр. 2)*.
10. Добавьте в меню команду **Edit** и запишите программный код, реализующий выполнение этой команды *(см. п.8. пр.2)*.
11. Добавьте в меню команду **Color** и запишите программный код, реализующий выполнение этой команды *(см. пп. 9.А и 9.Б, пр. 2)*.
12. Добавьте в меню команду **File** и запишите программный код, реализующий сохранение и открытие графических файлов *(см. пп. 9.А, 9.В и 9.Г, пр. 2)*.

13. Запишите программный код, реализующий вывод рисунка на принтер *(см. пп. 9.А и 9.Д, пр.2)*.
14. Добавьте в меню команду **Exit** и запишите команду **End**.
15. Добавьте в проект форму **FrmHelp**, содержащую описание программы. Добавьте в меню команду **Help** и запишите код, открывающий эту форму (*см. п.10, пр.2*).
16. Создайте форму-заголовок **FrmTitle** и измените свойство проекта **Startup Object** *(см. п.11, пр.2)*.

Справочный материал

1. Область и режим рисования

А. Характеристики режима рисования

Результат рисования в нашем проекте зависит от следующих характеристик:

- цвет линии;
- толщина линии;
- кисть;
- цвет фона.

В процессе рисования пользователь программы сможет сам менять эти характеристики, а в начале работы сразу после запуска программы значение характеристик режима рисования должно быть задано программным путем. Объявите в разделе **General** три переменные для хранения номера кисти, толщины линии и цвета фона. Цвет линии будем задавать, непосредственно меняя свойство **ForeColor** объекта **Picture1**. Список переменных приведен в таблице.

Таблица

Переменные режима рисования

Имя переменной	Что хранит переменная	Принимаемые значения	Тип переменной
cback	Цвет фона	Целые числа от 0 до 16 777 216*	Long
dw	Толщина линии	1, 2, 3, 4, 5	Byte
nbrush	Номер кисти	1, 2, 3	Byte

* Количество цветов в цветовой модели RGB.

Напомню, что целочисленные значения могут принимать переменные типов **Byte** (число от 0 до 255), **Integer** (числа от –32768 до +32767) и **Long** (числа от –2,147,483,648 до 2,147,483,647). Исходя из ожидаемых значений

переменных для них был выбран подходящий тип. Даже если вы захотите создать дополнительные кисти, вряд ли их будет больше чем 256. Следовательно, тип **Byte** позволяет сохранить все возможные значения переменной **nbrush**.

Пусть после старта программы цвет фона круглой области рисования будет белым, толщина линии равна 2 пикселям, номер кисти равен 1. В процедуре **Form_Load** присвойте значения переменным **cback, dw, nbrush**:

```
Private Sub Form_Load ()
cback = vbWhite
dw = 2
nbrush = 1
End Sub
```

Напомню, что этот код имеет смысл только в том случае, если переменные объявлены с помощью оператора **Dim** в области **General** (в самом начале программного кода).

✓ Б. Создание круглой области рисования. Общие (General) процедуры

Область рисования

Область рисования представляет собой круг с нанесенными четырьмя осями симметрии. Круг нарисован в объекте **Picture1** при помощи метода **Circle**. Центр круга находится в середине **Picture1**, радиус подобран так, чтобы круг был вписан в **PictureBox**. Это возможно только в том случае, когда длина объекта **Picture1** равна его ширине.

Цвет заливки соответствует заданному переменной **cback**. Цвет осей – черный.

Рисовать круг с осями потребуется не только сразу после старта программы, но и тогда, когда пользователь захочет стереть созданный рисунок или изменить цвет фона. Поэтому оформим создание области рисования как общую процедуру с именем **Clear**.

*Создание подпрограмм. Оператор **Sub***

Весь программный код проекта, созданного на Visual Basic, находится внутри каких-либо процедур. Часто мы используем процедуры обработки событий, заголовок которых определен заранее и формируется автоматически. Однако программист может создавать и собственные процедуры, которые принято называть общими (General). Общие процедуры могут быть разных типов, например, функции (Function) и подпрограммы (Sub). Под-

программы создают тогда, когда одна и та же группа операторов выполняется в разных местах программного кода с разными исходными данными.

Подпрограмма имеет следующую структуру:

```
[Private | Public] Sub <имя подпрограммы> ([список аргументов])
<оператор 1>
...
[<оператор n>]
[Exit Sub]
[<оператор n + 1>]
...
[<оператор n + k>]
End Sub
```

где Private – необязательное ключевое слово, которое означает, что данная подпрограмма может быть вызвана только из того модуля или формы, в которых она описана;

| – знак, означающий союз «или»;

Public – необязательное ключевое слово, которое означает, что данная подпрограмма может быть вызвана из любого модуля или формы данного проекта. **При этом сама процедура должна быть описана в модуле**;

имя подпрограммы – строка символов, подчиняющаяся тем же правилам, что и имя переменной (состоит из латинских букв и цифр, на первом месте стоит буква, не является зарезервированным словом);

список аргументов – список переменных, которые содержат исходные данные для данной процедуры. Переменные перечисляются через запятую. Список берется в круглые скобки;

оператор 1, ..., оператор n+k – любые операторы языка Visual Basic. Операторы образуют тело процедуры;

Exit Sub – оператор, вызывающий немедленное прекращение выполнения процедуры. Используется, как правило, в сочетании с оператором **If**.

Список аргументов может иметь следующие части:

```
([ByVal | ByRef] <имя переменной 1> as <тип>, [ByVal | ByRef]
<имя переменной 2> as <тип>, ...)
```

где ByVal – ключевое слово, означающее, что в подпрограмму передается значение переменной – аргумента. В таком случае в подпрограмме нельзя изменить значение этой переменной;

ByRef – ключевое слово, означающее, что в подпрограмму передается адрес переменной – аргумента. Значение такой переменной можно изменять в подпрограмме.

Для того чтобы вызвать подпрограмму, используется оператор **Call**:

[**Call**] <имя подпрограммы>([список аргументов])

Ключевое слово **Call** можно не писать. Если опускается ключевое слово **Call**, список аргументов пишется без скобок. Приведем несколько примеров описания и вызова подпрограмм.

Пример 1 (в примерах 1 и 2 жирным шрифтом выделены заголовки и вызов общих процедур).

```
Private Sub Command1_Click()
Dim X As Integer
For X = 0 To Form1.ScaleWidth Step 50
Call Leaf(X, 0, 10)
Next
End Sub
Private Sub Command2_Click()
Dim p As Integer
Dim q As Integer
p = Rnd * Form1.ScaleWidth
q = Rnd * Form1.ScaleHeight
Call Leaf(p, q, 20)
End Sub

Private Sub Leaf(X As Integer, Y As Integer, k As Byte)
PSet (X, Y)
Form1.DrawWidth = 6
Line Step(0, 0)–Step(k, k)
Line –Step(k, –k), RGB(0, 150, 0)
Line –Step(k, 0), RGB(0, 150, 0)
Line –Step(0, k), RGB(0, 150, 0)
Line –Step(–k, 0), RGB(0, 150, 0)
Line –Step(k, k), RGB(0, 150, 0)
Line –Step(0, k), RGB(0, 150, 0)
Line –Step(–k, 0), RGB(0, 150, 0)
Line –Step(–k, –k), RGB(0, 150, 0)
Line –Step(0, k), RGB(0, 150, 0)
Line –Step(–k, 0), RGB(0, 150, 0)
Line –Step(0, –k), RGB(0, 150, 0)
Line –Step(k, –k), RGB(0, 150, 0)
End Sub
```

Подпрограмма с именем **Leaf** рисует лист. Аргументами подпрограммы являются переменные X и Y – координаты основания листа и переменная k – определяющая размер листа. Подпрограмма вызывается из двух разных процедур при помощи операторов **Call**. Сравните список аргументов в описании процедуры **Leaf** при вызове этой пороцедуры:

```
Private Sub Leaf  (X As Integer, Y As Integer, k As Byte)
Call Leaf         (X,            0,           10)
Call Leaf         (p,            q,           20)
```

При вызове подпрограммы в списке аргументов можно записать константу или переменную, передающую в подпрограмму свое значение. При этом имена переменных в операторах **Sub** и **Call** не обязательно должны совпадать, но должны совпадать их типы. Переменные, которые используются в списке аргументов в операторе **Call**, обязательно должны быть описаны при помощи оператора **Dim**.

В этом примере и подпрограмма и операторы, вызывающие ее, записаны на листе программного кода формы. Скопируйте текст примера в область кода формы и поэкспериментируйте с ним. На форме должны быть расположены две командные кнопки – **Command1** и **Command2**.

Пример 2

В этом примере (рис. 16) две подпрограммы описаны в модуле. Подпрограмма **Square** рисует прямоугольник, а подпрограммы **Triang** – треугольник.

Подпрограммы многократно вызываются из формы **Form1** из процедуры обработки события **Command1_Click()**. Кнопка **Command1** находится на форме **Form1**.

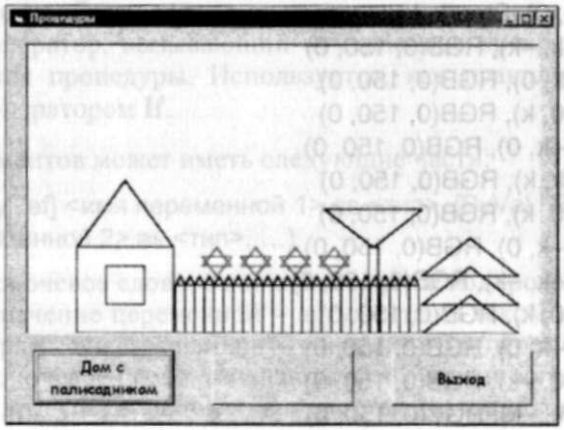


Рис. 16

Скопируйте текст подпрограмм **Square** и **Triang** в модуль, а процедуру **Private Sub Command1_Click()** – в область кодов формы. Разберитесь, как работают подпрограммы, и с их помощью создайте свой собственный рисунок, состоящий из прямоугольников и треугольников.

```
Public Sub square _
   (X0 As Single, Y0 As Single, a As Single, b As Single, col As Long)
Form1.Picture1.Line (X0, Y0) – (X0 + a, Y0 – b), col, B
End Sub
Public Sub triang _
   (x0 As Single, y0 As Single, a As Single, h As Single, _col As Long)
Form1.Picture1.Line (X0, Y0) – (X0 + a, Y0), col
Form1.Picture1.Line –Step(–a / 2, –h), col
Form1.Picture1.Line –Step(–a / 2, h), col
End Sub
Dim X As Single
Dim Y As Single
Private Sub Command1_Click()
Picture1.Cls
Call square(50, 300, 100, 90, RGB(110, 70, 25))
Call square(80, 270, 40, 40, RGB(110, 70, 25))
Call triang(50, 210, 100, 70, RGB(110, 70, 25))
For X = 152 To 400 Step 10
Call square(X, 300, 10, 50, RGB(0, 120, 0))
Call triang(X, 250, 10, 10, RGB(0, 120, 0))
Next
Call square(350, 300, 10, 90, RGB(110, 70, 25))
Call triang(305, 170, 100, –40, RGB(0, 120, 0))
For y = 300 To 240 Step –30
Call triang(405, Y, 100, 40, RGB(0, 120, 0))
Next
For X = 180 To 300 Step 40
Call triang(X, 230, 30, 20, RGB(210, 100, 0))
Call triang(X, 220, 30, –20, RGB(210, 100, 0))
Next
End Sub
```

ВНИМАНИЕ! Текст подпрограммы записан в модуле **Module1**, а вызывается из формы **Form1**. Поэтому:

- подпрограммы описаны с ключевым словом **Public**;
- в подпрограмме указывается не только имя объекта **PictureBox**, в котором производится рисование, но и имя формы, на которой находится этот объект:

```
Form1.Picture1.Line –Step(–a / 2, –h), col
```

*Создание подпрограммы **Clear** в проекте «Зеркало»*

Добавьте к проекту модуль и запишите там подпрограмму с именем **Clear**. Эта подпрограмма будет иметь один аргумент **cf** типа **Long**, задающий цвет заливки круглой области рисования. Блок-схема и код подпрограммы **Clear** приведены на рис. 17.

Блок-схема и код подпрограммы Clear

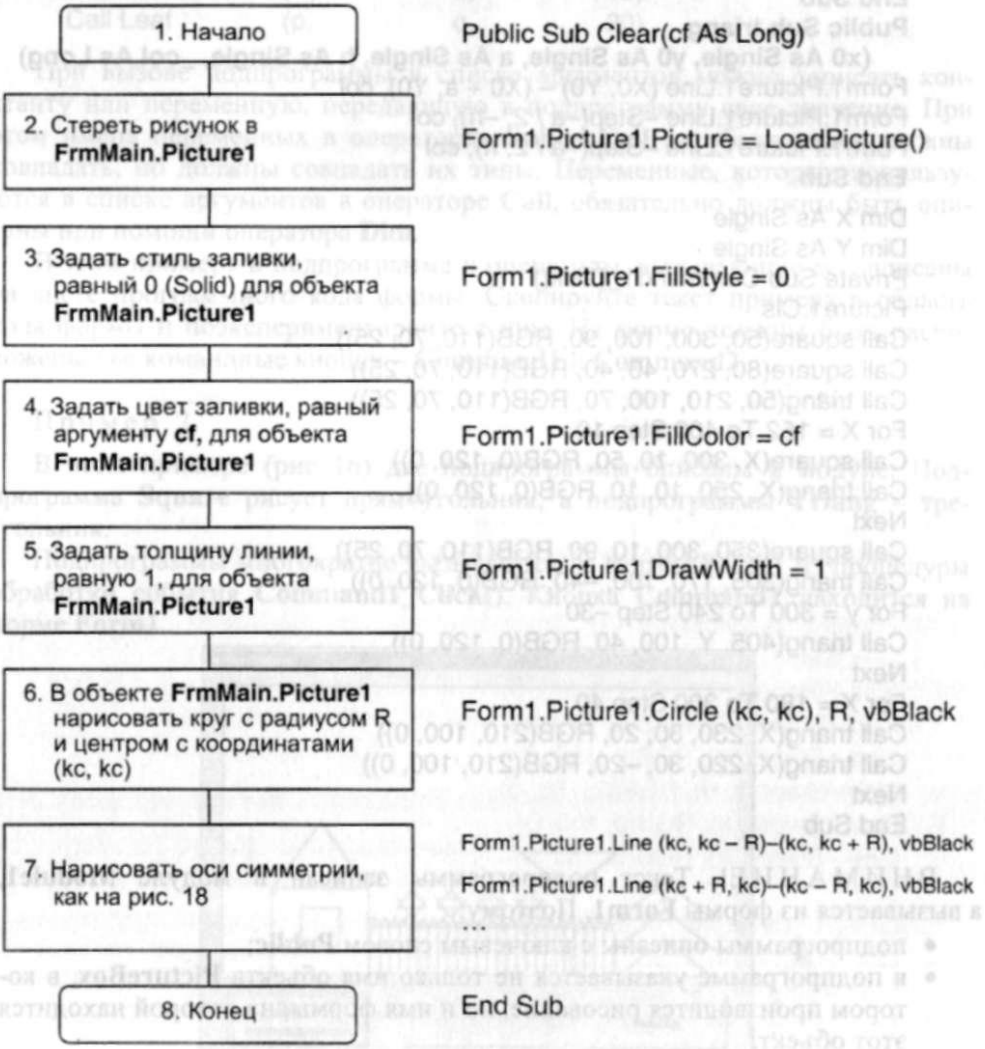
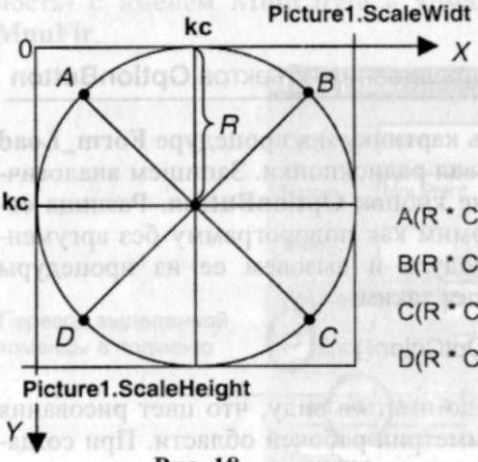


Рис. 17

На рис. 18 представлен чертеж зоны рисования в системе координат, связанной с объектом **Picture1**. Координаты точек *A*, *B*, *C* и *D* помогут вам записать программный код рисования осей симметрии.



A(R * Cos (5 * 3.14 / 4) + kc, R * Sin (5 * 3.14 / 4) + kc)

B(R * Cos (7 * 3.14 / 4) + kc, R * Sin (7 * 3.14 / 4) + kc)

C(R * Cos (3.14 / 4) + kc, R * Sin (3.14 / 4) + kc)

D(R * Cos (3 * 3.14 / 4) + kc, R * Sin (3 * 3.14 / 4) + kc)

Рис. 18

Чтобы провести отрезок, соединяющий точки *A* и *C*, запишите метод **Line**:

```
Form1.Picture1.Line (R * Cos(3.14 / 4) + kc, R * Sin(3.14 / 4) + kc) _
  –(R * Cos(5 * 3.14 / 4) + kc, R * Sin(5 * 3.14 / 4) + kc), vbBlack
```

Переменные kc и R опишите в модуле перед процедурой **Clear** при помощи оператора **Public**. Тогда они будут доступны и в модуле, и в форме:

```
Public R As Integer
Public kc As Integer
```

Осталось вычислить значение переменных kc и R в процедуре **Form_Load()** и вызвать оттуда подпрограмму **Clear**. Для этого добавим в процедуру **Form_Load** четыре оператора:

```
Private Sub Form_Load()
cback = vbWhite
dw = 2
nbrush = 1
kc = Picture1.ScaleWidth / 2
R = Picture1.ScaleWidth / 2
Call Clear(cback)
Picture1.DrawWidth = dw
End Sub
```

Запишите программу на диск и проверьте ее работоспособность. Переходите к следующему этапу только тогда, когда при запуске программы в объекте **Picture1** будет нарисован белый круг с четырьмя черными осями симметрии.

В. Подпрограмма OptColor раскрашивания объектов OptionButton

Во второй части (проект 4 «Раскрась картинку») в процедуре **Form_Load** мы записывали код, который раскрашивал радиокнопки. Запишем аналогичные команды для имеющихся на форме кнопок **OptionButton**. Разница состоит в том, что эти команды мы оформим как подпрограмму без аргументов, запишем эту подпрограмму в модуль и вызовем ее из процедуры **Form_Load**(). Заголовок процедуры будет таким:

```
Public Sub OptColor ()
```

При подборе цвета радиокнопок надо иметь в виду, что цвет рисования не должен совпадать с цветом осей симметрии рабочей области. При создании осей использовалась цветовая константа VBBlack. Она соответствует цвету, задаваемому функцией RGB(0, 0, 0). Если вы хотите, чтобы была возможность рисовать черным цветом, используйте цвет RGB(1, 1, 1). Добавьте в процедуру **Form_Load** оператор **Call OptColor** ().

2. Меню с выпадающими списками команд

В нашем графическом редакторе три кисти. Создадим команду меню, которая позволит выбирать кисть. Напротив выбранной кисти будет ставиться галочка (рис. 19). После старта программы переменная nbrush, которая содержит номер выбранной кисти, получит значение 1. Поэтому в начале работы галочка должна стоять напротив названия первой кисти.

Рис. 19

Вызовите меню и создайте команду меню с надписью **Brush** (Кисть) и с именем **MnuBrush**. Если вы забыли, как это сделать, посмотрите в проекте 1 «Калейдоскоп».

Теперь создадим выпадающий список из трех команд. Каждая команда этого списка является названием кисти. Нажмем кнопку **Next,** и в очистившихся окнах **Caption** и **Name** наберем соответственно **Point** (Точка) и **MnuPoint**. В окошке рядом со словом **Checked** поставим галочку. Для того чтобы эта команда стала членом выпадающего списка, сдвинем ее вправо

при помощи кнопки со стрелкой (рис. 20 показывает создание выпадающего меню). Теперь можно еще раз нажать кнопку **Next** и перейти к созданию следующей команды. Таким образом создайте команду **Circle** (Окружность) с именем **MnuCircle** и команду **Fir tree** (Еловая ветка) с именем **MnuFir**.

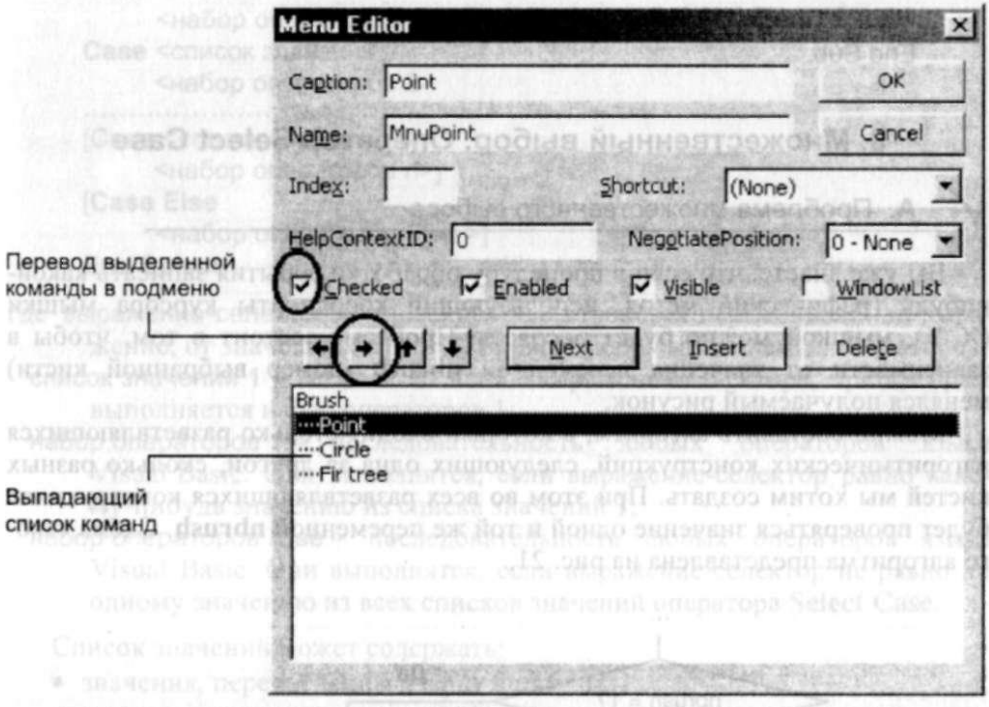


Рис. 20

На форме один раз нажмите мышкой по команде меню **Point**, и в области программного кода возникнет заголовок процедуры, обрабатывающей событие **Click** для данной команды. Программный код этой процедуры присваивает соответствующее значение переменной **nbrush** и устанавливает значение свойства **Checked**. Если свойство равно **True**, напротив данного пункта меню стоит галочка, если **False** – галочки нет. Приведем программный код для двух процедур, а для третьей запишите код самостоятельно.

```
Private Sub MnuPoint_Click()
nbrush = 1
MnuPoint.Checked = True
MnuCircle.Checked = False
MnuFir.Checked = False
End Sub
```

```
Private Sub MnuCircle_Click()
nbrush = 2
Form1.Picture1.FillStyle = 1    ''кисть рисует прозрачные круги
MnuPoint.Checked = False
MnuCircle.Checked = True
MnuFir.Checked = False
End Sub
```

3. Множественный выбор. Оператор Select Case

А. Проблема множественного выбора

Вы уже знаете, что если в процедуру обработки события записать какой-нибудь графический метод, использующий координаты курсора мышки (X, Y), мышкой можно будет рисовать. Проблема состоит в том, чтобы в зависимости от значения переменной **nbrush** (номер выбранной кисти) менялся получаемый рисунок.

Можно составить алгоритм, в который входит столько разветвляющихся алгоритмических конструкций, следующих одна за другой, сколько разных кистей мы хотим создать. При этом во всех разветвляющихся конструкциях будет проверяться значение одной и той же переменной **nbrush**. Схема такого алгоритма представлена на рис. 21.

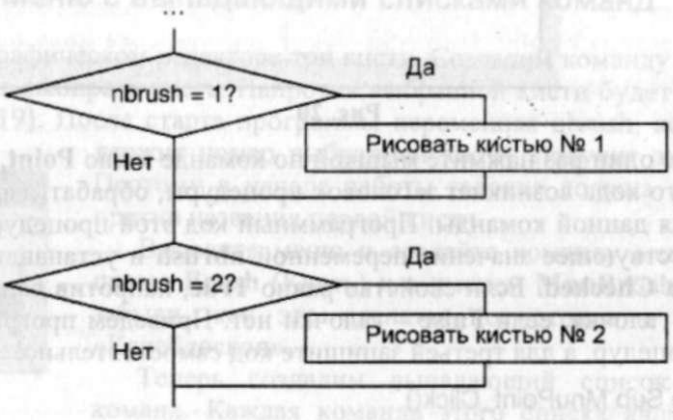


Рис. 21

В таких случаях удобно использовать оператор **Select Case**, который позволяет сократить программный код и сделать его более наглядным.

Б. Оператор **Select Case**

Оператор **Select Case** имеет следующий формат:

```
Select Case <выражение-селектор>
Case <список значений 1>
      <набор операторов 1>
Case <список значений 2>
      <набор операторов 2>
.....
[Case <список значений n>
      <набор операторов n>]
[Case Else
      <набор операторов Else >]
End Select
```

где выражение-селектор – численная или строковая переменная, или выражение, от значения которого зависит выбор выполняемых операторов;

список значений 1 – список значений выражения-селектора, при которых выполняется набор операторов 1;

набор операторов 1 – последовательность любых операторов языка Visual Basic. Они выполнятся, если выражение-селектор равно какому-нибудь значению из списка значений 1;

набор операторов **Else** – последовательность любых операторов языка Visual Basic. Они выполнятся, если выражение-селектор не равно ни одному значению из всех списков значений оператора **Select Case**.

Список значений может содержать:

- значения, перечисленные через запятую;
- выражения вида **<значение> to <значение>**, задающие диапазон значений;
- выражение, содержащее знак сравнения, вида **is > <значение>**.

Приведем несколько примеров записи оператора **Select Case**.

Пример 3

```
D = b^2 – 4*a*c
Select Case D
Case 0
      Label1.Caption = "Уравнение имеет два равных корня"
Case is > 0
      Label1.Caption = "Уравнение имеет два разных корня"
Case Else
      Label1.Caption = "Уравнение не имеет корней"
End Select
```

Пример 4

```
Select Case Number
Case 2 to 5
    Picture1.Line(0, 0) – (Number * 50, Number * 40)
Case 6, 7
    Picture1.Line(0, 0) – (Number * 30, Number * 20), , B
Case 8, 15 to 20
    Picture1.Line(0, 0) – (Number * 20, Number * 10), , BF
End Select
```

В блок–схеме (рис. 22) конструкция множественного выбора выглядит следующим образом.

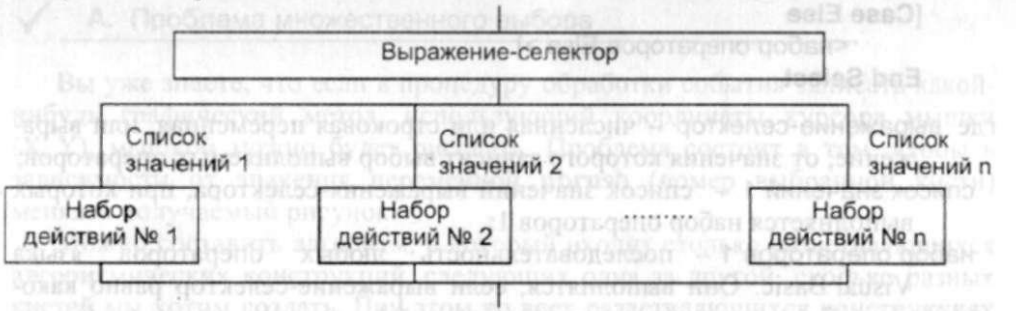


Рис. 22

В. Процедура **Sub Picture1_MouseMove**

На рис. 24 представлена блок-схема алгоритма процедуры **Picture1_MouseMove**. Рисование происходит в том случае, если нажата левая кнопка мышки (Button = 1) и точка находится внутри круга. Для того чтобы проверить, лежит ли точка с координатами (X, Y) внутри круга (рис. 23), вспомним уравнение окружности из школьного курса алгебры:

$$(X - a)^2 + (Y - b)^2 = R^2,$$

где (a, b) – центр окружности, R – радиус.

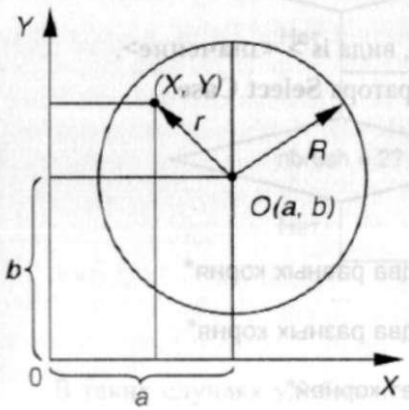


Рис. 23

Если равенство выполняется, точка (X, Y) лежит на окружности. Если квадрат радиуса больше выражения, записанного слева ($r < R$), точка лежит внутри окружности.

Блок-схема алгоритма процедуры **Picture1_MouseMove**

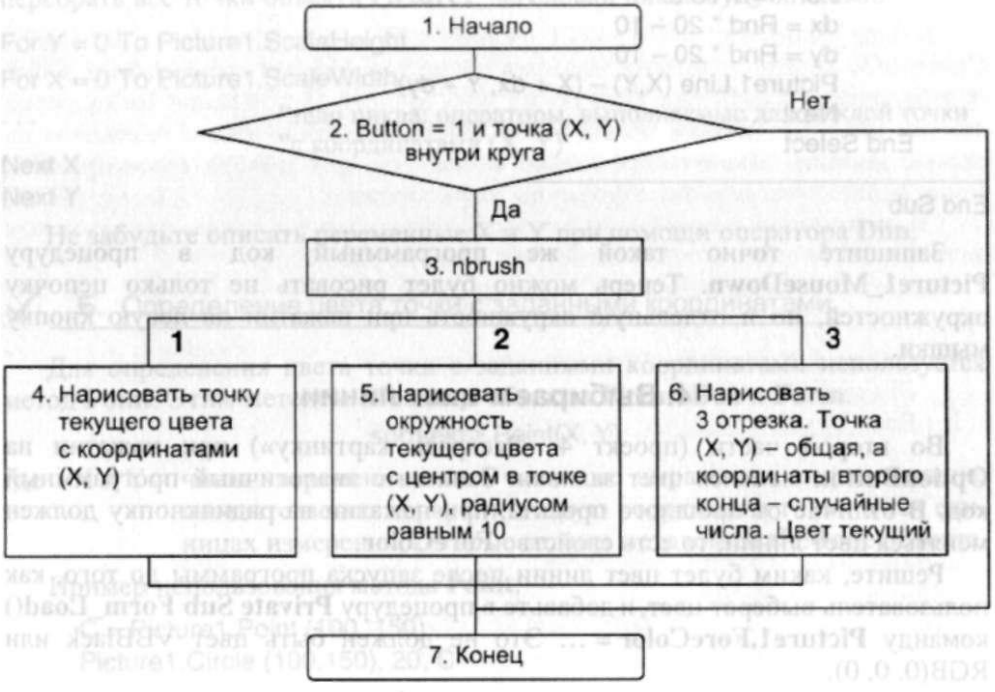


Рис. 24

Далее приведен программный код, реализующий алгоритм. В коде есть пропуски. Заполните пропущенную часть кода, руководствуясь блок-схемой. Запишите код в проект на компьютере и сохраните проект. Запустите программу и проверьте каждую кисть, работает ли она. Если необходимо, исправьте ошибки. При желании придумайте какую-нибудь свою кисть и дополните программный код.

```
Private Sub Picture1_MouseMove _
    (Button As Integer, Shift As Integer, X As Single, Y As Single)
Dim dx as Integer
Dim dy as Integer
If Button = 1 And (X – kc) ^ 2 + (Y – kc) ^ 2____R ^ 2 Then
      Select Case _____________
      Case 1
            Picture1.PSet (X, Y)
      Case 2
      _____________
```

```
    Case 3
        For k = 1 To 3
        dx = Rnd * 20 – 10
        dy = Rnd * 20 – 10
        Picture1.Line (X,Y) – (X + dx, Y + dy)
        Next
    End Select
End Sub
```

Запишите точно такой же программный код в процедуру **Picture1_MouseDown**. Теперь можно будет рисовать не только цепочку окружностей, но и отдельную окружность при нажатии на левую кнопку мышки.

4. Выбираем цвет линии

Во второй части (проект 4 «Раскрась картинку») при нажатии на **OptionButton** менялся цвет заливки. Запишите аналогичный программный код. В отличие от прошлого проекта, при нажатии на радиокнопку должен меняться цвет линии, то есть свойство **ForeColor**.

Решите, каким будет цвет линии после запуска программы до того, как пользователь выберет цвет, и добавьте в процедуру **Private Sub Form_Load()** команду **Picture1.ForeColor = …** Это не должен быть цвет VBBlack или RGB(0, 0, 0).

5. Симметричное отражение рисунка. Метод Point

В проекте 1 «Калейдоскоп» одновременно с рисованием основной точки мы рисовали симметричные ей точки, используя тот же самый цвет рисования. В этом проекте пользователь может выбрать ось симметрии и выполнить построение симметричного изображения после того, как рисование основного изображения окончено. В связи с этим возникает задача перебрать все точки области рисования, определить их цвет и нарисовать точку того же цвета симметрично выбранной оси симметрии.

А. Организация перебора точек

Для того чтобы перебрать все точки прямоугольной области, надо организовать два вложенных цикла. Внешний цикл будет менять значение координаты Y от 0 до величины, равной высоте прямоугольной области. Внутренний цикл изменяет значение координаты X от 0 до величины, равной ширине области. Точки будут просматриваться по строкам слева направо, начиная с нулевой строки. Переход к следующей строке происходит только

после того, как все точки текущей строки просмотрены. Если требуется перебрать все точки объекта **Picture1**, заголовки цикла будут такими:

```
For Y = 0 To Picture1.ScaleHeight
For X = 0 To Picture1.ScaleWidth
. . .                    'тело цикла: операторы, выполняемые для каждой точки
                         'с координатами (X, Y)
Next X
Next Y
```

Не забудьте описать переменные X и Y при помощи оператора **Dim**.

Б. Определение цвета точки с заданными координатами

Для определения цвета точки с заданными координатами используется метод **Point**. Этим методом обладают объекты **PictureBox** и **Form**.

<объект>.Point(X, Y)

где X и Y – числа, переменные или числовые выражения, определяющие соответственно первую и вторую координаты точки в тех единицах измерения, которые заданы методом **ScaleMode** объекта.

Пример использования метода **Point**:

```
C = Picture1.Point (100, 150)
Picture1.Circle (100,150), 20, C
```

В. Координаты точек, симметричных данной относительно прямой

На рис. 25 показано, как определяются координаты точки, симметричной точке с координатами (X, Y), относительно прямой *b*. Здесь (kc, kc) – координаты круглой области рисования. Аналогично можно вывести формулы, связывающие координаты точки (X, Y) с координатами точек, симметричных ей, относительно прямых *a*, *c*, *d*.

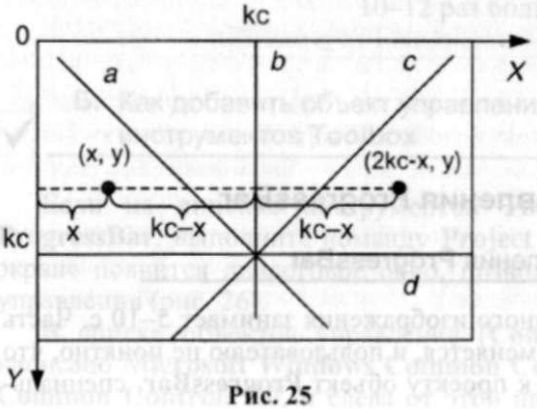


Рис. 25

Ось симметрии	Координаты симметричной точки
a	(y, x)
b	(2 * kc – x, y)
c	(2 * kc – y, 2 * kc – x)
d	(x, 2 * kc – y)

Г. Процедура Cmdsim_Click

Четыре командные кнопки **Cmdsim(0)**, **Cmdsim(1)**, **Cmdsim(2)**, **Cmdsim(3)** образуют массив, поэтому процедура **Private Sub Cmdsim_Click** обрабатывает нажатие на каждую из этих кнопок. Переменная **Index** равна номеру нажатой кнопки. Различие в действиях, связанных с нажатием на разные кнопки, заключается только в том, что для вывода симметричной точки используются разные формулы для координат.

Симметричные точки будем рисовать только для тех точек, цвет которых не равен цвету фона и осей симметрии области рисования.

Разберите написанный код процедуры и заполните имеющиеся в нем пропуски:

```
Private Sub Cmdsim_Click(Index As Integer)
Picture1.DrawWidth = 1
For _____________________
For _____________________
c = Picture1._____________(X, Y)
    If c <> cback And c <> VBBlack Then
        Select Case Index
        Case 0
            Picture1.PSet (2 * kc – X, Y), c
        Case 1
            _____________________
        Case 2
            Picture1.PSet (Y, X), c
        Case 3
            Picture1.PSet (2 * kc – Y, 2 * kc – X), c
        _____________
    End If
Next X
Next Y
Picture1.DrawWidth = dw
End Sub
```

6. Объект управления ProgressBar

А. Описание объекта управления ProgressBar

Процесс построения симметричного изображения занимает 5–10 с. Часть времени изображение на экране не меняется, и пользователю не понятно, что происходит в программе. Добавим к проекту объект **ProgressBar**, специаль-

но предназначенный для того, чтобы наглядно отражать ход выполнения различных операций.

Назначение: отражает выполнение операций, занимающих продолжительное время, путем заполнения прямоугольника синими штрихами. Заполнение производится слева направо.

Пиктограмма: 

Свойства: помимо общих с другими объектами управления свойств: Name, Top, Left, Width, Height, BorderStyle, Visible, Enabled, объект **ProgressBar** имеет свойства, которые отражают его индивидуальность:

Min – число, соответствующее незаполненному прямоугольнику (начало отражаемого процесса);

Max – число, соответствующее полностью заполненному прямоугольнику (окончание отражаемого процесса), Max> Min;

Value – число в диапазоне [Min, Max]. Чем больше значение свойства, тем большая часть прямоугольника закрашена. Свойство не доступно в процессе создания проекта;

Orientation – если свойство равно 0, ProgressBar расположена горизонтально, если 1 – вертикально;

Scrolling – изменение свойства меняет плавность заполнения прямоугольника (1 – более плавное заполнение, 0 – скачкообразное заполнение). Если Scrolling = 0, ширина объекта должна быть в 10–12 раз больше высоты.

Б. Как добавить объект управления ProgressBar на линейку инструментов Toolbox

Если на линейке инструментов **Toolbox** нет пиктограммы объекта **ProgressBar**, выполните команду **Project → Components...**. После этого на экране появится диалоговое окно, позволяющее добавлять новые объекты управления (рис. 26).

В списке объектов управления (**Controls**) найдите пункт, в котором написано **Microsoft Windows Common Controls 6.0** или **Microsoft Windows Common Controls 5.0**, и слева от этой надписи щелчком мышки поставьте

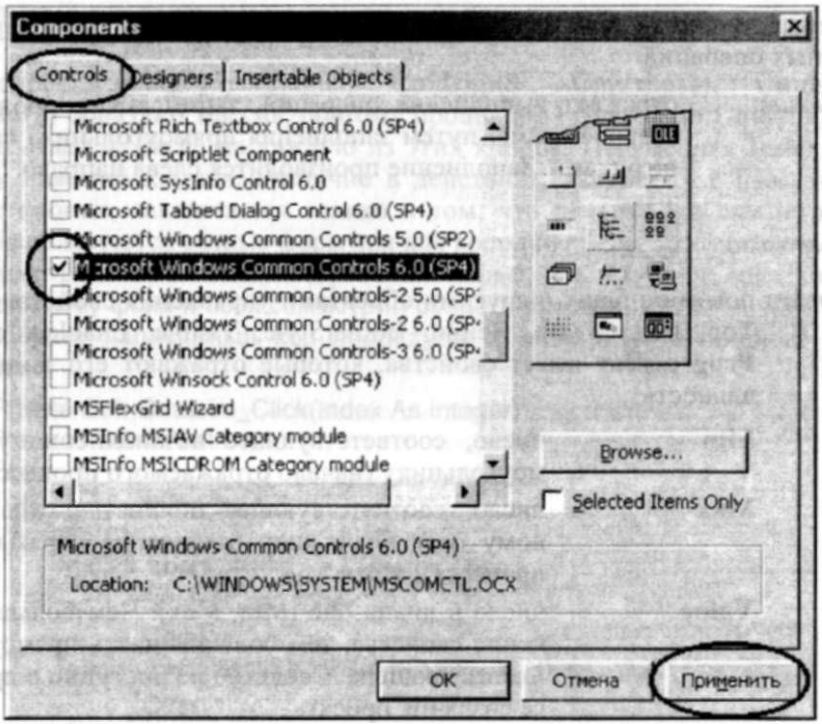


Рис. 26

галочку. После этого нажмите на кнопку **Apply** (Применить). На линейке **Toolbox** появится несколько новых пиктограмм, в том числе и принадлежащая объекту **ProgressBar**.

B. Использование объекта управления ProgressBar

Разместите объект **ProgressBar** в левом нижнем углу формы под командными кнопками **CmdSim**. В меню свойств **Properties** задайте свойству **Scrolling** значение 1. Как определить, каким должно быть значение свойств **Min** и **Max**?

Мы хотим, чтобы **ProgressBar** отражал течение процесса создания симметричного изображения. Начинается этот процесс, когда в процедуре **Cmdsim_Click** начинает выполняться цикл с параметром Y и заканчивается после того, как этот цикл выполнен полностью. Рассмотрим заголовок этого цикла:

```
For Y = 0 To Picture1.ScaleHeight
```

По мере выполнения цикла переменная Y меняет свое значение от 0 до величины **Picture1.ScaleHeight**. Свяжите свойства **Min**, **Max** и **Value** объекта **ProgressBar** со значением переменной Y. Для этого задайте значения свойств:

- ProgressBar1.Min = 0;
- ProgressBar1.Max = Picture1.ScaleHeight.

Задать эти значения можно в меню свойств в процессе создания проекта или записать соответствующие операторы в процедуру **Form_Load**.

В процессе выполнения цикла свойству **Value** надо присваивать значение переменной Y, а после окончания цикла присвоить ему значение свойства **Min** для того, чтобы убрать закраску прямоугольника после окончания отображаемого процесса:

- ProgressBar1. Value = y
- ProgressBar1. Value = Min.

Найдите самостоятельно в какое место программного кода надо добавить эти операторы. Имейте в виду, что в тексте программного кода эти операторы идут не подряд.

7. Выбор толщины линии рисования

Откройте **Menu Editor** и создайте команду для выбора толщины линии, как показано на рис. 27. Действуйте так же, как при создании команды **Brush**. Для того чтобы команда **DrawWidth** расположилась в меню левее уже существующей команды **Brush**, в списке команд окна **Menu Editor** выделите команду **Brush** и нажмите на кнопку **Insert**. Над словом **Brush** появится пустая строка, выделите ее и запишите в окнах **Caption** и **Name** информацию о команде **DrawWidth**.

Рис.27

Для каждой команды выпадающего меню создайте процедуру, которая выполняет следующий алгоритм:

Начало.

1. Переменной **dw** присвоить выбранное значение (1, 2, 3, 4 или 5).
2. Свойству **DrawWidth** объекта **Picture1** присвоить значение **dw**.
3. Вызвать подпрограмму **MnuPixel**, которая снимает галочки со всех команд выпадающего меню (Form1.MnuPixel.Checked = False, …).
4. Свойству **Checked** выбранной команды выпадающего меню присвоить значение **True** (поставить галочку).

Конец.

Подпрограмму **MnuPixel** создайте в модуле. Она не имеет аргументов. В ней всего пять операторов, которые присваивают значение **False** свойству **Checked**.

8. Создание команды Edit пользовательского меню

Создайте в пользовательском меню команды, как показано на рис. 28. Команда **Clear** стирает рисунок из поля рисования и закрашивает круг в выбранный пользователем цвет. Для этого из процедуры **MnuClear_Click** надо вызвать из модуля подпрограмму **Clear** так же, как она вызывалась из процедуры **Form_Load**. Аргументом при вызове подпрограммы будет переменная **cback**. После этого надо восстановить значение свойства **DrawWidth** объекта **Picture1**, которое было изменено в подпрограмме **Clear**.

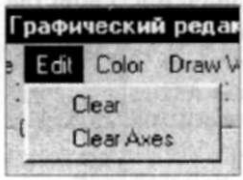


Рис. 28

Команда **Clear Axes** стирает на рисунке оси симметрии. Те точки, которые имели цвет осей (VbBlack), перекрашиваются в цвет фона (cback). Для этого используется программный код, очень похожий на код процедуры **Cmdsim_Click** (точно так же организован цикл перебора всех точек **Picture1**, так же определяется цвет точки, так же изменяется свойство **Value** объекта **ProgressBar**). Отличается код процедуры, реализующей команду **Clear Axes**, только тем, что вместо оператора **Select Case** должет быть записан следующий оператор **If**:

```
If c = vbBlack Then Picture1.PSet (X, Y), cback.
```

9. Сохранение, открытие и вывод на принтер рисунков

A. Объект CommonDialog

Назначение: объект **CommonDialog** обеспечивает вывод на экран набора стандартных окон для выполнения операций открытия и сохранения файлов, выбора цвета и характеристик шрифта. **CommonDialog** является средством связи между программами, написанными на языке Visual Basic, и процедурами из динамической библиотеки **Commdlg.dll**. Файл **Commdlg.dll** находится в системной директории Microsoft Windows.

Пиктограмма: 

Методы: **ShowColor** – выводит на экран стандартное диалоговое окно выбора цвета;
ShowOpen – выводит на экран стандартное диалоговое окно открытия файла;
ShowSave – выводит на экран стандартное диалоговое окно сохранения файла;
ShowPrinter – выводит на экран стандартное диалоговое окно вывода на принтер;
ShowFont – выводит на экран стандартное диалоговое окно выбора характеристик шрифта.

Методы выполняются только во время работы программы. Во время создания проекта объект **CommonDialog**, размещенный на форме, выглядит как пиктограмма. Свойства объекта управления рассмотрим отдельно для каждого диалогового окна.

Прежде чем использовать объект **CommonDialog**, его надо добавить к проекту, выполнив команду **Project → Components...**. После этого в появившемся на экране диалоговом окне **Components** выберите вкладку **Controls** и в списке объектов управления найдите строку **Microsoft Common Dialog Control**, слева от этой надписи щелчком мышки поставьте галочку и нажмите на кнопку **Apply** (Применить). На линейке **Toolbox** появится пиктограмма, принадлежащая объекту **CommonDialog**.

Б. Диалоговое окно выбора цвета

Для того чтобы использовать стандартное диалоговое окно выбора цвета (рис. 29), надо сделать следующее:

- разместить на форме объект **Common Dialog**;
- задать в программном коде свойства объекта, относящиеся к окну выбора цвета;
- применить метод **ShowColor**;
- использовать свойство **Color** для получения выбранного цвета.

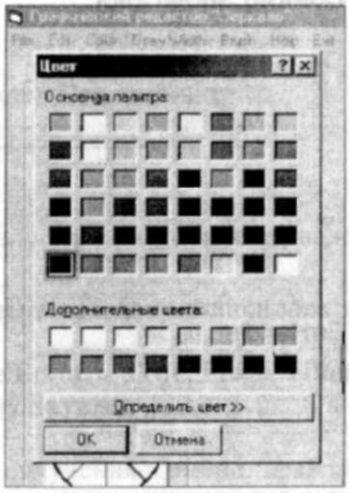


Рис. 29

Разместим на этой форме объект **CommonDialog**. Автоматически он получил имя **CommonDialog1**. Изменим имя и присвоим свойству **Name** значение **ComDlg**.

Прежде чем открыть диалоговое окно, присвоим значения двум свойствам объекта **ComDlg** – **CancelError** и **Flags**. Если свойство **CancelError** имеет значение **True,** появляется возможность в программном коде реагировать на нажатие пользователем клавиши **Cancel** (Отмена). Значение свойства **Flags** влияет на внешний вид и поведение диалогового окна. Далее представлена общая схема использования диалогового окна выбора цвета. Этот программный код можно связать с нажатием командной кнопки, радиокнопки, команды пользовательского меню или какими-нибудь другими событиями, которые приведут к открытию диалогового окна.

Общая схема использования диалогового окна выбора цвета:

```
ComDlg.CancelError = True      "включаем формирование ошибки при нажатии
                               "на кнопку Cancel
On Error GoTo ErrHandler       "если возникла ошибка (нажата кнопка Cancel),
                               "управление переходит к строке с меткой
                               "ErrHandler и выполняется следующий за ней
                               "оператор
ComDlg.Flags = cdlCCRGBInit    "благодаря этому значению свойства при
                               "открытии диалогового окна на нем будет
                               "выделен тот цвет, который пользователь выбрал
                               "в предыдущий раз. Если это свойство не
                               "указывать совсем, при каждом обращении будет
                               "выделен черный цвет
ComDlg.ShowColor               "этот метод выводит на экран диалоговое окно
                               "«Color» (Цвет)
...                            "эти операторы выполняются только тогда,
                               "когда нажата кнопка OK. Если нажата кнопка
                               "Cancel, они пропускаются. Состав команд
                               "зависит от целей конкретного проекта
...                            "вместо многоточия надо записать операторы,
                               "использующие свойство ComDlg.Color, равное
                               "выбранному цвету
ErrHandler:
...                            "вместо многоточия надо записать операторы,
                               "которые выполняются, если нажата кнопка
                               "Cancel
Exit Sub                       "вызывает немедленный выход из процедуры
```

Рис. 30

Создадим в нашем проекте команды пользовательского меню, как показано на рис. 30. Команда **BackColor** меняет цвет фона, а команда **ForeColor** – цвет линии рисования. Теперь цвет линии можно будет выбрать двумя способами – используя радиокнопки или команду пользовательского меню. Применив общую схему использования диалогового окна выбора цвета, составим программный код, реализующий команду **BackColor**:

```
Private Sub MnuBackColor_Click()
ComDlg.CancelError = True
On Error GoTo ErrHandler
ComDlg.Flags = cdlCCRGBInit
ComDlg.ShowColor
cback = ComDlg.Color  'использовать свойства ComDlg.Color – выбор цвета
Call Clear(cback)
Picture1.DrawWidth = dw
ErrHandler:
    Exit Sub
End Sub
```

Код, реализующий команду **ForeColor**, запишите самостоятельно, опираясь на общую схему использования диалогового окна **Color**. Он будет отличаться от кода процедуры **MnuBackColor_Click** только оператором, использующим выбранный цвет (свойство **ComDlg.Color**).

ВНИМАНИЕ! Если диалоговое окно оказывается сверху созданного рисунка, рисунок стирается. Чтобы этого не происходило, задайте значение **True** свойству **Autoredraw** объекта **Picture1**.

В. Диалоговое окно **Save as** (Сохранить как). Сохранение графического файла

*Общая схема использования окна **Save as***

Для того чтобы использовать стандартное диалоговое окно **Save as**, надо сделать следующее:

- разместить на форме объект **CommonDialog** или использовать размещенный ранее;
- задать в программном коде свойства **CancelError = True, Flags**, Filter объекта **CommonDialog**, относящиеся к окну **Save as**;
- применить метод **ShowSave**;
- использовать свойство **FileName** для получения имени файла, набранного пользователем.

Свойство **Flags** может иметь множество значений. Полезными для нас являются два из них: &H2 и &H4 (их описание приведено в таблице).

Таблица

Имя константы	Значение	Описание
CdlOFNOverwritePrompt	**&H2**	Если файл с набранным именем уже существует, появляется окно сообщения с предупреждением
CdlOFNHideReadOnly	**&H4**	Из диалогового окна **Save as** исчезает **CheckBox "Read Only"** (Только для чтения)

Свойство **Flags** может иметь несколько значений. Все значения записываются в одном операторе присваивания и отделяются друг от друга логическим действием **Or**. Не забудьте отделить знак действия **Or** пробелами справа и слева. Для задания значений свойства **Flags** можно использовать как константы, так и значения. В качестве примера приведены два оператора присваивания, которые действуют абсолютно одинаково. Выберите тот, который вам кажется более удобным:

```
ComDlg.Flags = cdlOFNOverwritePrompt Or cdlOFNHideReadOnly
```

или

```
ComDlg.Flags = &H2 Or &H4
```

Свойство **Filter** задает типы файлов, которые будут показаны в списке файлов диалогового окна **Save as** (Сохранить как) или **Open** (Открыть). Структура значения свойства **Filter**:

```
<Объект>.Filter = "<описание 1> | <фильтр 1> | <описание 2> | <фильтр 2> ..."
```

где Объект – имя объекта **CommonDialog**;

описание 1 – описание фильтра № 1, этот текст появится в списке типов файлов в окне **Save as type**;

фильтр 1 – строка символов, определяющая расширение имени файла (*.bmp – файлы с расширением bmp, *.doc – файлы с расширением doc и т. д.).

Описание фильтров и фильтры разделяются символом |, который расположен рядом с клавишей **BackSpace** (←) на латинском регистре. Справа и слева от этого символа не должно быть пробелов. Приведем несколько примеров.

Пример 5

```
ComDlg.Filter = "Web Page | *.htm | Word Documents | *.doc | Text Files | *.txt"
ComDlg.FilterIndex = 2
```

Описание фильтра № 2 (может быть на русском языке) — Фильтр № 2

В примере описано три фильтра. Свойство **FilterIndex** показывает, что фильтр № 2 будет использоваться по умолчанию, то есть в текстовом окне **Save as a type** будет видно описание фильтра № 2. Описания двух других фильтров будут спрятаны в выпадающем списке.

Пример 6

```
ComDlg.Filter = "Bitmap pictures (*.bmp) | *.bmp"
```

Описание фильтра — Фильтр

Далее приведена общая схема использования диалогового окна **Save as**. Команды сохранения файлов зависят от типа файла, который мы хотим записать на диск. В этом проекте мы рассмотрим, как сохранить графический файл с расширением *.bmp. Сохранение текстовых файлов мы изучим в следующих проектах.

Общая схема использования диалогового окна сохранения файла:

```
ComDlg.CancelError = True      "используется так же, как при вызове
On Error GoTo ErrHandler       "диалогового окна
                               "выбора цвета

ComDlg.Flags = cdlOFNOverwritePrompt Or cdlOFNHideReadOnly
ComDlg.Filter = "Bitmap pictures (*.bmp) | *.bmp"
ComDlg.ShowSave                "этот метод выводит на экран диалоговое окно
                               "Save as (Сохранить как)
...                            "вместо многоточия записать операторы,
                               "использующие свойство ComDlg.FileName –
                               "набранное пользователем имя файла. Операторы
                               "выполняются только тогда, когда нажата
                               "кнопка OK. Состав команд зависит от типа файла
ErrHandler:
...                            "вместо многоточия записать операторы, которые
                               "выполняются, если нажата кнопка Cancel
Exit Sub                       "вызывает немедленный выход из процедуры
```

Оператор SavePicture

Оператор **SavePicture** сохраняет графическое изображение из свойства **Picture** или **Image** объекта управления в виде файла. Рассмотрим два варианта использования этого оператора.

Первый вариант:

SavePicture <объект>.**Picture**, <имя файла>

где <объект> – имя любого объекта, имеющего свойство **Picture** (PictureBox, Form, Image, CommandButton, ...);
<имя файла> – строка символов в кавычках или строковое выражение, содержащее имя создаваемого файла.

Если рисунок был загружен во время проектирования или во время работы программы в свойства **Picture** из файлов формата **bitmap**, **icon**, **metafile** или **enhanced metafile**, их следует сохранять в том же формате. Если рисунок загружен из GIF или JPEG файлов, его следует сохранять с расширением *.bmp.

Если рисунок не был загружен, возникает состояние ошибки и работа программы прерывается.

Изображение, созданное во время работы программы при помощи графических методов (Pset, Line, Circle, PaintPicture), не записывается.

Второй вариант:

SavePicture <объект>.**Image**, <имя файла>

где <объект> – имя объекта **PictureBox** или **Form**;
<имя файла> – строка символов в кавычках или строковое выражение, содержащее имя создаваемого файла с расширением ***.bmp**.

Изображение сохраняется независимо от того, каким способом оно получено – загружено из файла в свойство **Picture** или создано во время работы программы графическими методами. При этом свойству **AutoRedraw** объекта должно быть присвоено значение **True** до начала создания изображения.

Если рисунок в объекте отсутствует, сохраняется файл, содержащий прямоугольник цвета фона.

Сохранение графических файлов

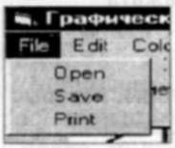


Рис. 31

Создайте в пользовательском меню команду **File**, как показано на рис.31, и запишите в процедуру **MnuSave_Click** программный код открытия диалогового окна **Save as** и сохранения рисунка из объекта **Picture1** (для сохранения рисунка используйте второй вариант записи оператора **SavePicture**). На основе общей схемы использования диало-

гового окна сохранения файла составлен программный код. Заполните имеющиеся пропуски в программном коде и используйте в вашем проекте.

```
Private Sub MnuSave_Click()
ComDlg.CancelError = _____
_____ GoTo ErrHandler
ComDlg.Flags = cdlOFNOverwritePrompt Or cdlOFNHideReadOnly
ComDlg.Filter = "Bitmap pictures (*.bmp)|_____
ComDlg._____
SavePicture Picture1. Image, ComDlg. FileName
ErrHandler:
Exit Sub
End Sub
```

Г. Диалоговое окно **Open** (Открыть файл). Открытие графического файла

Использование стандартного диалогового окна **Open** отличается от использования окна **Save as** только значением свойства **Flags** (флаги) и оператором, использующим значение свойства **FileName**.

Присвоим свойству **Flags** два значения: **&H1000** и **&H4** (их описание дано в таблице).

Таблица

Имя константы	Значение	Описание
CdlOFNFileMustExist	**&H1000**	Если файл с набранным именем не существует, появляется окно сообщения с предупреждением и действие отменяется
CdlOFNHideReadOnly	**&H4**	Из диалогового окна **Open** исчезает **CheckBox "Read Only"** (Только для чтения)

Свойство **Flags** при открытии окна **Open** в проекте «Зеркало» будет иметь следующее значение:

```
ComDlg.Flags = cdlOFNFileMustExist Or cdlOFNHideReadOnly
```

или

```
ComDlg.Flags = &H1000 Or &H4
```

Для открытия графических файлов используется функция **LoadPicture**, знакомая вам по проектам «Проверка интелекта» и «Раскрась картинку». Имя открываемого файла, которое надо записать в скобках после имени функции **LoadPicture**, содержится в свойстве **ComDlg.FileName**.

Запишите самостоятельно программный код процедуры **MnuOpen_Click**, который будет открывать окно диалога **Open** и загружать выбранный пользователем графический файл в объект **Picture1**.

✓ Д. Диалоговое окно Print. Вывод на принтер графического файла

Общая схема использования окна Print

Диалоговое окно **Print** позволяет выбрать настройки принтера (ориентацию бумаги, количество копий, размер бумаги и др.) и сам принтер, если к вашему компьютеру подключено несколько принтеров. Существует множество флагов и свойств объекта **CommonDialog**, предназначенных для работы с диалоговым окном **Print**. Однако работа многих из них зависит от драйвера каждого конкретного принтера. Вполне возможно, что с вашим принтером флаги будут работать не так, как сказано в описании объекта **CommonDialog**, или не будут действовать вообще. Поэтому обойдемся без них.

Вызов диалогового окна **Printer** в нашем проекте осуществляется следующим образом:

```
ComDlg.CancelError = True
On Error GoTo ErrHandler
ComDlg.ShowPrinter
...
ErrHandler:
Exit Sub
```

Если вы хотите запретить пользователю выбирать принтер, запишите перед строкой **ComDlg.ShowPrinter** оператор присваивания **ComDl.PrinterDefault = False**.

Объект Printer

Однажды в проекте «Калейдоском» мы выводили на принтер всю форму при помощи метода **PrintForm**. Сейчас попробуем вывести на печать не всю форму, а только рисунок, находящийся в свойстве **Image** объекта **Picture1**. Используем для этого специальный обект **Printer**.

Обект **Printer** позволяет выводить текст и графические изображения на принтер, установленный по умолчанию. Если на компьютере доступно

несколько принтеров, тот принтер, который пользователь выбрал в диалоговом окне, становится установленным по умолчанию.

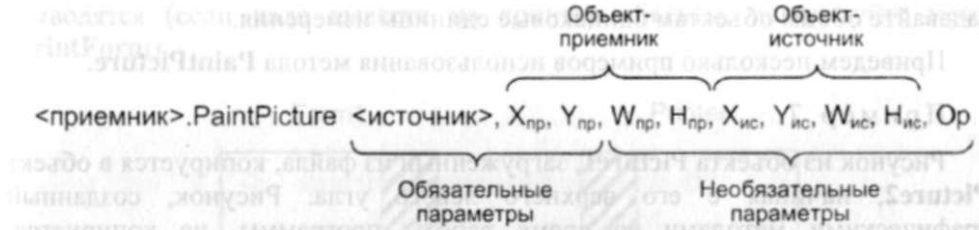
Объект **Printer** обладает многими свойствами и методами, которые имеются у объектов **Form** или **PictureBox**. Для вывода графического изображения на печать его надо создать на объекте **Printer**, применяя графические методы. Графические методы **Pset**, **Line** и **Circle** можно использовать с объектом **Printer** так же, как с объектами **Form** или **PictureBox**. Однако гораздо меньше труда и времени вы затратите, если для переноса изображения, созданного в **PictureBox** или на форме, будете использовать метод **PainPicture**, который мы рассмотрим в следующем подразделе.

После того, как изображение на объекте **Printer** создано, его надо отправить на печать при помощи метода **Printer.EndDoc**. Если этот метод не выполнен, после закрытия программы-приложения он выполняется автоматически. Приведем код вывода на печать окружности радиусом 5 см:

```
Printer.ScaleMode = 7    "единицы измерения сантиметры
Printer.Circle (10,10), 5
Printer.EndDoc
```

Метод ***PaintPicture***

Метод **PaintPicture** копирует графическое изображение из объекта-источника в объект-приемник. Объектом-источником могут быть объекты **Form**, **PictureBox**, **Image**, а объектом-приемником – объекты **Form**, **PictureBox**, **Printer**. Формат оператора:



где <приемник> – объект, в который копируется изображение; это может быть **Printer** или объекты **Form**, **PictureBox**;

<источник> – свойство **Picture** или **Image** объектов **Form**, **PictureBox**, **Image**. Например, Picture1.Picture или Form1.Image;

$X_{пр}$, $Y_{пр}$ – числа или численные выражения (тип Single), задающие координаты левой верхней вершины прямоугольника на объекте-приемнике, в который копируется изображение;

$W_{пр}$, $H_{пр}$ – числа или численные выражения (тип Single), задающие ширину (Width) или высоту (Height) прямоугольника на объекте-приемнике, в который копируется изображение.

Если они не равны размерам рисунка-источника, рисунок растягивается или сжимается. Если параметры пропущены, сохраняется размер рисунка-источника. Если один из параметров или оба отрицательные, изображение переворачивается относительно вертикальной, горизонтальной или обеих осей;

$X_{ис}$, $Y_{ис}$ – числа или численные выражения (тип Single), задающие координаты левой верхней вершины прямоугольника на объекте-источнике, из которого берется изображение; если параметры пропущены, берется точка с координатами (0,0);

$W_{ис}$, $H_{ис}$ – числа или численные выражения (тип Single), задающие ширину (Width) или высоту (Height) прямоугольника на объекте-источнике, из которого берется изображение. Если они меньше размеров рисунка-источника, копируется только его часть. Если параметры пропущены, копируется весь рисунок-источник;

Op – необязательный параметр, определяющий логическую операцию (AND, XOR и другие), которая выполняется при копировании изображения. В текущем проекте не рассматривается.

Все размеры и координаты задаются в тех единицах издмерения, которые заданы свойством **ScaleMode** для каждого из объектов. Чтобы не ошибиться, задавайте обоим объектам одинаковые единицы измерения.

Приведем несколько примеров использования метода **PaintPicture**.

Пример 7

Рисунок из объекта **Picture1**, загруженный из файла, копируется в объект **Picture2**, начиная с его верхнего левого угла. Рисунок, созданный графическими методами во время работы программы, не копируется. Размеры рисунка в **Picture2** в два раза больше по сравнению с источником. Рисунок перевернут относительно горизонтальной оси.

```
W = Picture1.ScaleWidth
H = Picture1.ScaleHeight
Picture2.PaintPicture Picture1.Picture, 0, 0, 2*W, -2*H
```

Пример 8

Маленький рисунок из объекта **Picf**, загруженный из файла с расширением *.ico, копируется многократно вдоль верхней границы формы, образуя

бордюр (рис. 32). Для того чтобы этот программный код можно было поместить в процедуру **Form_Load()**, свойство **AutuRedraw** формы должно быть равно **True**.

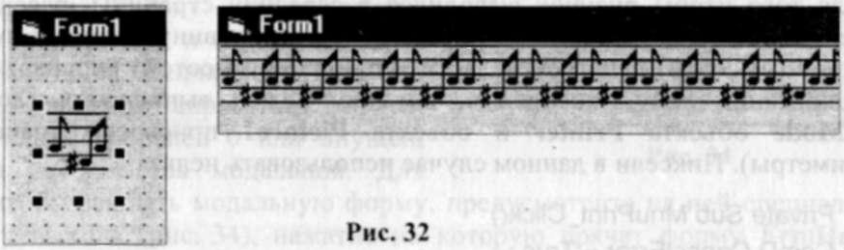


Рис. 32

```
Private Sub Form_Load()
For X = 0 To Form1.ScaleWidth Step 30
Form1.PaintPicture Picf.Picture, X, 0
Next
End Sub
```

Пример 9

Рисунок из правой четверти формы (рис. 33) выводится на принтер, начиная с верхнего левого угла листа. На печать выводится как изображение, загруженное из файла в свойство **Picture** формы, так и созданное графическими методами. Объекты, размещенные на форме, на печать не выводятся (если надо вывести на принтер объекты, используйте метод **PrintForm**).

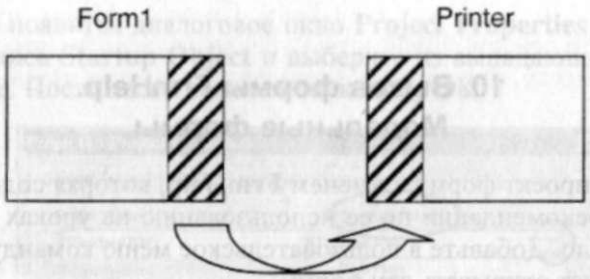


Рис. 33

```
W = Form1.ScaleWidth / 4
X = Form1.ScaleWidth – W
Printer.PaintPicture Form1.Image, 0, 0, , ,X, 0, W
Printer.EndDoc
```

Вывод рисунка на печать в проекте «Зеркало»

Запишем в процедуру **MnuPrint_Click** программный код вывода на печать рисунка из свойства **Image** объекта **Picture1**.

Для того чтобы рисунок выводился в середину страницы, рассчитаем размер поля слева – **cx**, и сверху – **cy**, как половину разности между шириной (высотой) печатной области и шириной (высотой) рисунка. Обратите внимание на то, что прежде, чем производить вычисления, свойству **ScaleMode** объекта **Printer** и объекта **Picture1** присвоено значение 7 (сантиметры). Пиксели в данном случае использовать нельзя.

```
Private Sub MnuPrint_Click()
ComD.CancelError = True
On Error GoTo ErrHandler
ComD.PrinterDefault = False
ComD.ShowPrinter
Printer.ScaleMode = 7
Picture1.ScaleMode = 7
cx = (Printer.ScaleWidth – Picture1.ScaleWidth) / 2
cy = (Printer.ScaleHeight – Picture1.ScaleHeight) / 2
Printer.PaintPicture Picture1.Image, cx, cy
Printer.EndDoc
Picture1.ScaleMode = 3
ErrHandler:
Exit Sub
End Sub
```

10. Вызов формы FrmHelp. Модальные формы

Добавьте в проект форму с именем **FrmHelp**, которая содержит описание программы и рекомендации по ее использованию на уроках информатики в начальной школе. Добавьте в пользовательское меню команду **Help**, нажатие на которую будет открывать эту форму.

Когда читаешь описание программы, удобно видеть ее основную форму. Поэтому откроем форму **FrmHelp**, не закрывая форму **FrmMain**:

```
Private Sub MnuHelp_Click()
Frmhelp.Show 1
End Sub
```

В методе **Show** использован параметр *модальности* формы. Если он равен 1, форма является модальной. Это значит, что она всегда будет расположена сверху других открытых форм проекта и не сможет из-за неаккуратного щелчка мышкой по форме **FrmMain** спрятаться за нее. Если параметр равен 0 или опущен, форма не является модальной. Для того чтобы закрыть модальную форму, предусмотрите на ней специальную кнопку выхода (рис. 34), нажатие на которую прячет форму **FrmHelp**.

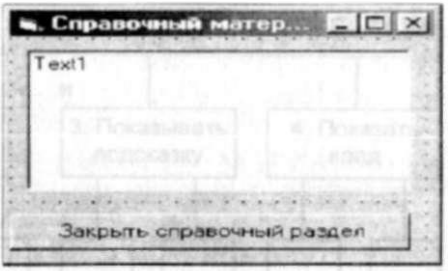


Рис. 34

11. Форма FrmTitle. Изменение свойств проекта

Создайте еще одну форму, которая будет титульным листом вашего проекта. Оформите ее так, как вам нравится, не забудьте указать свою фамилию и год создания программы.

Разместите на этой форме объект **Timer** и запишите в него программный код закрытия формы **FrmTitle**, открытия формы **FrmMain** и отключения таймера (свойство **Enabled** равно **False**). Задайте свойству **Interval** объекта **Timer** значение 2000 (т. е. 2 секунды).

Запустите проект. Вы увидите, что работа проекта начинается не с формы **FrmTitle**, как нам хотелось бы, а с формы **FrmMain**. Чтобы исправить положение, выполните команду:

Project → Project Properties

После этого появится диалоговое окно **Project Properties** (рис. 35). Найдите в нем надпись **Startup Object** и выберите из выпадающего списка имя формы **FrmTitle**. После этого нажмите на кнопку OK.

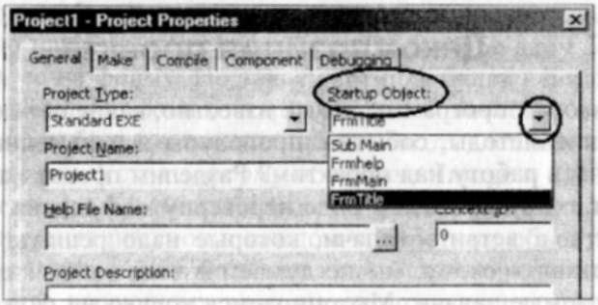


Рис. 35

Проект 3
«Найди клад»

Работая над проектом, вы познакомитесь с событиями **KeyDown** (прижать клавишу на клавиатуре) и **KeyUp** (отпустить нажатую клавишу), научитесь проигрывать звуковые файлы с расширением ***.wav**, создавать процедуры–функции, объявлять константы, а также приобретете опыт работы с методами **PaintPicture** и **Move**, оператором **Select Case**.

Постановка задачи

Давайте сделаем игрушку для малышей: «Где-то в лесу зарыт клад. Его надо отыскать. Охотник за кладом может двигаться вправо, влево, вверх и вниз. Управлять охотником будем, нажимая клавиши со стрелками. Охотник может ходить куда угодно, не наступая на деревья. Когда охотник находит клад, раздается музыка и, постепенно увеличиваясь, появляется ларец с кладом. Играющий имеет подсказку. Он видит, приближается охотник к кладу или удаляется от него. Можно выполнить команду «Новая игра» и начать все сначала. Лес при каждой новой игре будет выглядеть по другому, клад будет зарыт в новом месте, охотник начинает поиски клада с новой позиции».

Декомпозиция проекта

Известен сюжет программы, но не известно, какие объекты расположены на форме, какие методы, события, процедуры и переменные использовать. С чего же начать работу над проектом? Разделим проект на отдельные задачи. Нарисуем схему проекта в виде перевернутого дерева, где корень – название проекта, а ветви – задачи, которые надо решить, создавая проект (рис. 36). Составляя схему, мы не думаем о том, какими средствами будем решать выделяемые задачи. Мы опираемся только на сюжет проекта. Процесс разбивки проекта на задачи называется *декомпозицией*.

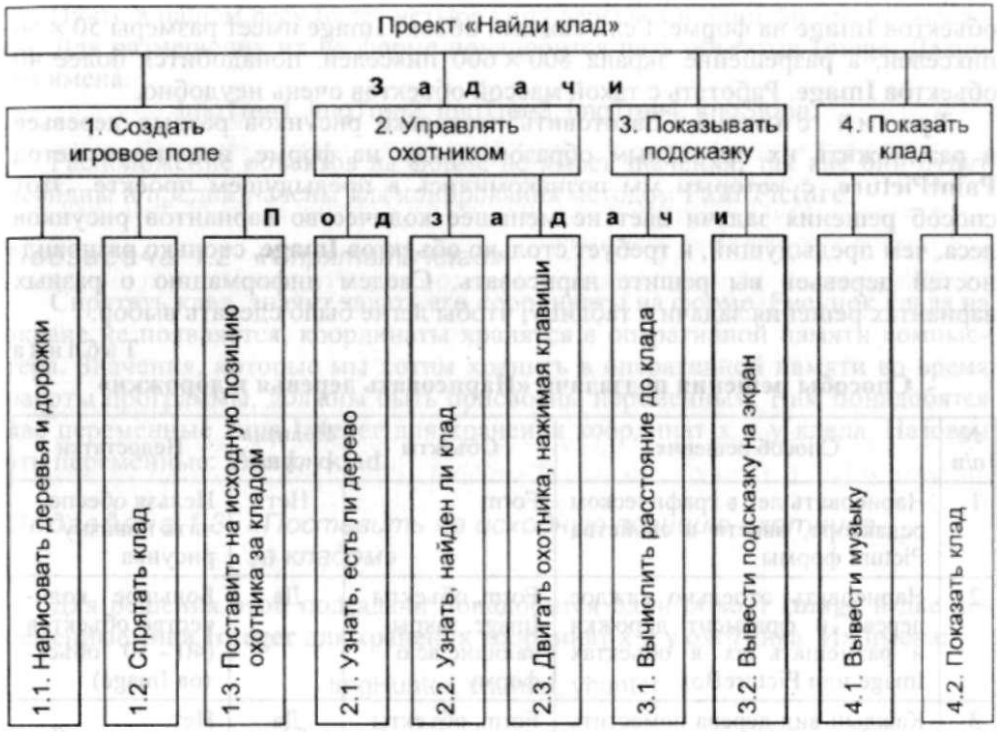


Рис. 36

Проект разбит на четыре задачи, а каждая из задач – на более мелкие подзадачи. Используя знания, полученные при создании предыдущих проектов, мы можем решить все подзадачи, кроме двух (вывод музыки и движение охотника при помощи клавиш со стрелками). Начнем с деревьев и дорожек.

Подзадача 1.1 «Нарисовать деревья и дорожки»

Первый способ. Деревья и дорожки можно нарисовать разными способами. Можно нарисовать в графическом редакторе одну большую картинку и положить ее на форму. Но такая картинка будет неизменной. Можно, конечно, нарисовать несколько картинок, но количество новых игр все равно может оказаться больше, чем количество заготовленных картинок.

Второй способ. Устлать форму объектами **Image** и в каждый поместить рисунок дерева или фрагмента дорожки. Заготовив несколько рисунков разных деревьев и меняя рисунок в каждом конкретном объекте случайным образом, можно получить очень большое количество вариантов нашего сказочного леса, так что играющему вряд ли удастся увидеть два одинаковых рисунка. Недостатком этого способа является очень большое количество

объектов **Image** на форме. Если каждый объект **Image** имеет размеры 50 × 50 пикселей, а разрешение экрана 800 × 600 пикселей, понадобится более 40 объектов **Image**. Работать с такой массой объектов очень неудобно.

Третий способ. Заготовить несколько рисунков разных деревьев и размножить их случайным образом прямо на форме, используя метод **PaintPicture**, с которым мы познакомились в предыдущем проекте. Этот способ решения задачи дает не меньшее количество вариантов рисунков леса, чем предыдущий, и требует столько объектов **Image**, сколько разновидностей деревьев вы решите нарисовать. Сведем информацию о разных вариантах решения задачи в таблицу, чтобы легче было сделать выбор.

Таблица

Способы решения подзадачи «Нарисовать деревья и дорожки»

№ п/п	Способ решения	Объекты	Новизна рисунка	Недостатки
1	Нарисовать лес в графическом редакторе, ввести в свойство Picture формы	Form	Нет	Нельзя обеспечить новизну рисунка
2	Нарисовать отдельно каждое дерево и фрагмент дорожки и размещать их в объектах Image или PictureBox	Form, объекты Image, закрывающие всю форму	Да	Большое количество объектов (40 – 50 объектов Image)
3	Каждый вид дерева поместить в Image и размножить методом PaintPicture на форме	Form, объекты Image, по одному на каждый вид дерева	Да	Нет

Проанализировав таблицу, выбираем третий вариант решения, связанный с использованием метода **PaintPicture**.

Пусть размер каждого дерева и фрагмента дорожки будет равен 50 × 50 пикселей. Цвет фона у всех рисунков должен быть одинаковым.

Когда будете рисовать в графическом редакторе, обязательно запишите RGB код цвета фона и цвета середины дорожки, который вы выберете. Можно использовать рисунки деревьев, которые имеются в MS Office в библиотеке Clip Art (см. часть вторую, проект 1 «Составь слово»), и скопировать их прямо в **Image** через буфер обмена, выполняя команды **Edit → Copy** и **Edit → Paste**. Примеры изображений деревьев и дорожки представлены на рис. 37.

Рис. 37

Пусть в нашем лесу будет четыре типа деревьев и один фрагмент дорожки. Для размещения их на форме понадобится пять объектов **Image**. Дадим им имена:

ImgTree1, ImgTree2, ImgTree3, ImgTree4, ImgStone.

Расположение объектов на форме не имеет значения, так как они будут невидны и предназначены для копирования методом **PaintPicture**.

Подзадача 1.2 «Спрятать клад»

Спрятать клад, значит задать его координаты на форме. Рисунок клада на экране не появляется, координаты хранятся в оперативной памяти компьютера. Значения, которые мы хотим хранить в оперативной памяти во время работы программы, должны быть присвоены переменным. Нам понадобятся две переменные типа **Integer** для хранения координат x и y клада. Назовем эти переменные: **xklad**, **yklad**.

Подзадача 1.3 «Поставить на исходную позицию охотника за кладом»

Для решения этой подзадачи понадобится один объект **Image** и две переменные типа **Integer** для хранения координат x и y охотника. Их имена:

ImgHunter, xhunter, yhunter.

Решение подзадач 1.1, 1.2, и 1.3, связанных с созданием игрового поля, оформим как подпрограмму, которую назовем **Forest**.

Задача 2 «Управлять охотником»

Вы пока не знаете, как использовать в проекте клавиши клавиатуры, но очевидно, что должна быть создана процедура, реагирующая на нажатие клавиш.

Рисунок охотника мы решили разместить в объекте **ImgHunter**. Для того чтобы двигать объект, надо знать его координаты: **xhunter**, **yhunter**, а чтобы выяснить, найден ли клад, еще нужны координаты клада: **xklad**, **yklad**. Следовательно, эти переменные должны быть доступны и в процедуре **Forest**, и в процедуре обработки нажатия клавиши.

Задача 3 «Показывать подсказку»

Чтобы сформировать подсказку, надо вычислить расстояние между охотником и кладом – подзадача 3.1. Для этого надо знать координаты охотника и клада – переменные **xhunter, yhunter, xklad, yklad**.

Вычисления оформим как процедуру с именем **Distance**.

Вывести на экран подсказку можно разными способами:

- при каждом шаге охотника выводить в объект **Label** числа, равные расстоянию до клада;
- использовать тот же объект **Label** и раскрашивать его в разные цвета: чем ближе к кладу, тем ближе цвет метки к красному, чем дальше – тем ближе к синему (тепло – холодно);
- использовать **ProgressBar** – чем ближе охотник к кладу, тем меньшая часть **ProgressBar** закрашена.

Я предлагаю использовать в проекте объект **ProgressBar**. Дадим этому объекту имя **PrgBar**.

Задача 4 «Показать клад»

Для того чтобы показать клад, нужен объект, в котором находится рисунок клада – **ImgKlad**. Поскольку рисунок увеличивается постепенно, понадобится объект **Timer** с именем **TmrKlad**. Для вывода музыки будем использовать API функцию **PlaySound**, с которой вы познакомитесь в этом проекте.

Подведем итоги. При создании проекта нам понадобятся:

1. Объекты:
   - 7 объектов **Image** с именами **ImgTree1**, **ImgTree2**, **ImgTree3**, **ImgTree4**, **ImgStone**, **ImgHunter**, **ImgKlad**;
   - объект **ProgressBar** с именем **PrgBar**;
   - объект **Timer** с именем **TmrKlad**.
2. Переменные:
   - координаты охотника – **xhunter, yhunter** (тип Integer);
   - координаты клада – **xklad, yklad** (тип Integer).
3. Пользовательские процедуры:
   - подпрограмма **Forest** (создание игрового поля);
   - функция **Distance** (вычисление расстояния между охотником и кладом).
4. Процедуры обработки события:
   - **TmrKlad_Timer** (постепенное увеличение рисунка клада);
   - процедура обработки нажатия клавиши.

План работы над проектом

1. Создайте графические файлы с рисунками четырех разных деревьев, фрагмента дорожки, клада и охотника. Все рисунки имеют размер 50×50 пикселей и одинаковый цвет фона. RGB код цвета фона и середины фрагмента дорожки запишите в тетради.

2. Создайте форму **FrmMain** и разместите на ней объекты, измените свойство **name** этих объектов.

Задайте указанные в таблице значения свойств объектов.

Объект	Имя свойства	Значение свойства
Form	Autoredraw	True
	WindowState	2
	ScaleMode	3 – Pixel
ProgressBar	Align	2 – vbAlignBottom
Все объекты Image	Visible	False

3. Создайте подпрограмму **Forest** и вызовите ее из процедуры **Form_Load** *(см. п. 1, пр. 3)*.
4. Создайте пользовательское меню с командами «Новая игра» (вызывает подпрограмму **Forest**) и «Выход».
5. Запишите программный код функции **Distance** *(см. п. 2, пр. 3)*.
6. Запишите программный код процедуры **Form_KeyDown** *(см. п. 3, пр. 3)*.
7. Запишите программный код процедуры **TmrKlad_Timer** *(см. п.4, пр.3)*.
8. Запишите в модуль объявление API функции **PlaySound** и добавьте в процедуру **Form_KeyDown** вызов этой функции *(см. п. 5, пр.3)*.
9. Сохраните проект на диске и создайте выполнимый файл (файл с расширением *.exe).

Справочный материал

1. Создание подпрограммы Forest

Нарисуем мысленно на форме **FrmMain** сетку с размером клеток 50×50 пикселей. В каждую такую клетку скопируем при помощи метода **PaintPicture** рисунок из одного из перечисленных объектов: **ImgTree1, ImgTree2, ImgTree3, ImgTree4, ImgStone**. В одну из клеток с рисунком фрагмента дорожки поместим рисунок охотника и присвоим координатам охотника значение координат этой клетки. Координаты другой клетки с рисунком фрагмента дорожки присвоим координатам клада.

Два вложенных цикла с параметром организуют формирование координат левого верхнего угла клетки. Параметр внешнего цикла – координата клетки по оси *OY*, а параметр внутреннего – координата по оси *OX*. Какой

именно рисунок будет скопирован на форму, зависит от значения случайного числа **n**. Формируется целое случайное число в диапазоне от 1 до 30:

```
n = Int(Rnd * 30) + 1
```

Как выбран диапазон для числа n? Всего на форме 7 объектов **Image**, которые мы собираемся копировать на форму. Если число n находится в диапазоне от 1 до 7, на форме будет появляться слишком много деревьев. Они могут образовывать замкнутые области. Если в такую область попадет охотник, он не сможет добраться до клада. Чем шире диапазон случайного числа, тем реже появляются числа 1, 2, 3 и 4 и тем меньше деревьев будет появляться. Это снижает вероятность появления замкнутых областей. Диапазон от 1 до 30 подобран опытным путем. Вы можете поэкспериментировать и выбрать другой диапазон. Блок-схема подпрограммы **Forest** представлена рис. 38.

Блок-схема подпрограммы **Forest**

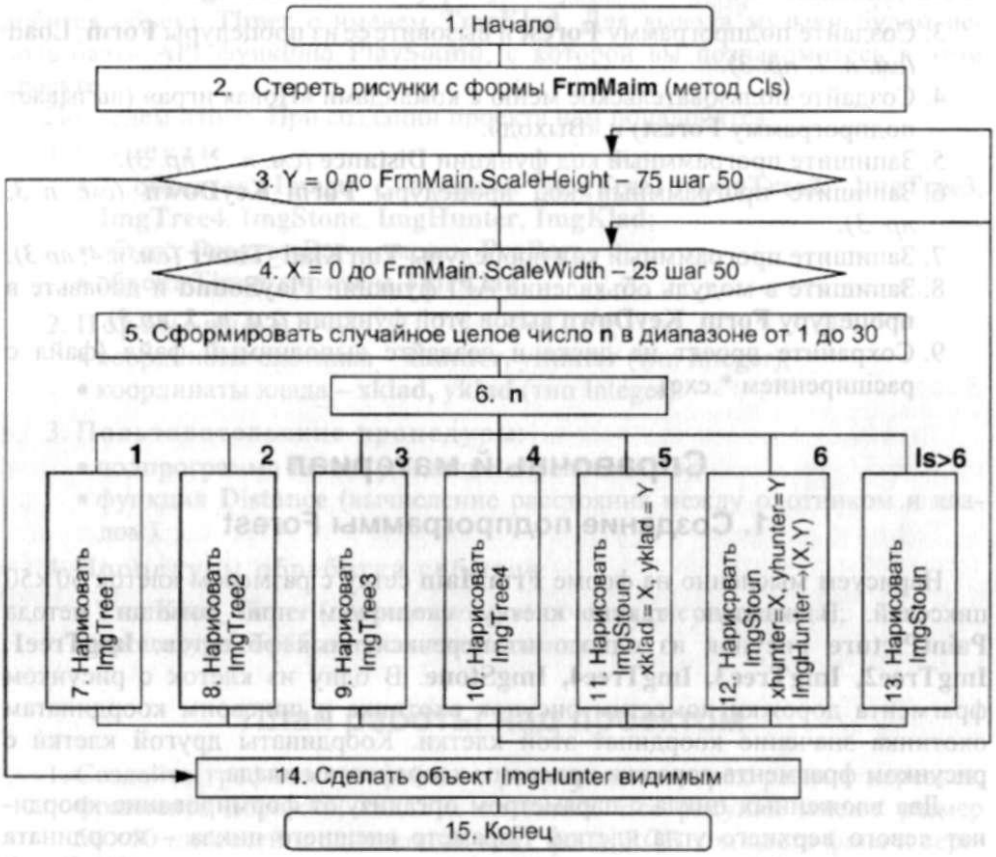


Рис. 38

Создайте подпрограмму **Forest** в модуле и вызовите ее из процедуры **Form_Load** с помощью оператора **Call**. Переменные **xhunter**, **yhunter**, **xklad**, **yklad** используются и в процедурах, записаных в модуле, и в записаных на листе кода формы. Поэтому они должны быть описаны в модуле с помощью оператора **Public**.

Заполните пропуски в программном коде и опробуйте его на компьютере. Создайте пользовательское меню с командой «Новая игра», которая вызывает подпрограмму **Forest**.

```
Public xklad As Integer
Public yklad As Integer
Public xhunter As Integer
Public yhunter As Integer
Public Sub Forest()
Dim X As Integer
Dim Y As Integer
Dim n As Byte
Randomize
FrmMain.Cls
For Y = 0 To _____
For X _____
n = Int(Rnd*30) + 1
Select Case n
Case 1
        FrmMain.PaintPicture FrmMain.ImgTree1.Picture, X, Y
_____
        _____
_____
        _____
_____
        _____
Case 5
_____
_____
FrmMain.PaintPicture FrmMain.ImgStone.Picture, X, Y
```

```
    Case 6:
        FrmMain.ImgHunter.Left = X
        FrmMain.ImgHunter.Top = Y
        xhunter = x
        yhunter = Y
        FrmMain.PaintPicture FrmMain.ImgStone.Picture, X,Y
    Case Is>6
    End
        FrmMain.ImgHunter.Visible = True
    Next X
    End Sub
```

Если вы все сделали правильно, при запуске программы на форме будет возникать изображение, похожее на рис. 39. При каждом нажатии на надпись «Новая игра» изображение будет меняться.

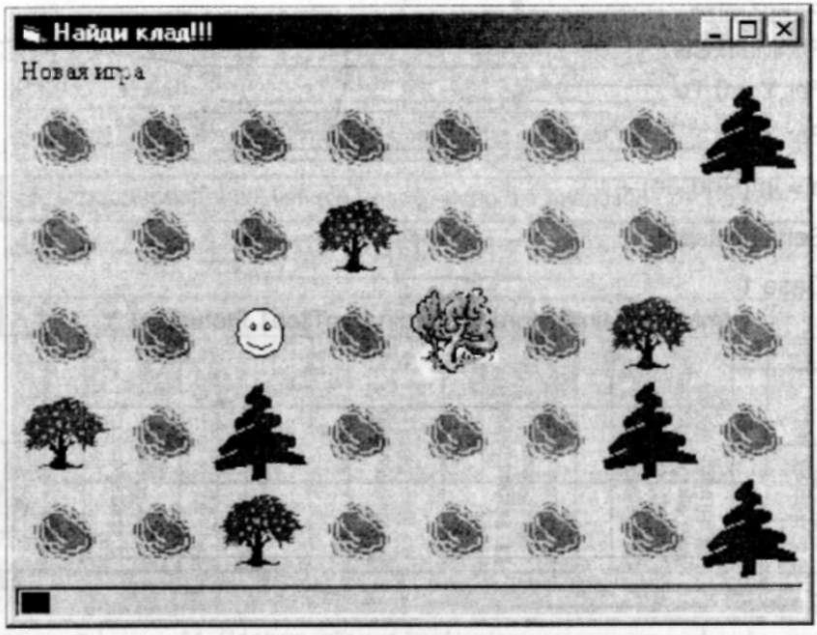


Рис. 39

Запишите проект на диск в отдельную папку. Проверьте работоспособность программы и, если все в порядке, переходите к следующему этапу.

2. Создание процедур-функций. Функция Distance

А. Процедура-функция

Функция (Function) – один из видов процедур, который в отличие от подпрограммы (Sub) можно использовать в правой части оператора присваивания. Функции создают тогда, когда результатом выполнения процедуры является значение какой-нибудь величины. Величина может быть численного, строкового или логического типа.

Функция имеет следующую структуру:

```
[Private|Public] Function <имя функции> ([список аргументов]) As <type>
<оператор 1>
...
<имя функции> = <выражение>
...
End Function
```

где
- Private – необязательное ключевое слово, которое означает, что данная функция может быть вызвана только из того модуля или формы, в которых она описана;
- Public – необязательное ключевое слово, которое означает, что данная функция может быть вызвана из любого модуля или формы данного проекта. При этом сама функция должна быть описана в модуле;
- имя функции – строка символов, подчиняющаяся тем же правилам, что и имя переменной (состоит из латинских букв и цифр, на первом месте стоит буква, не является зарезервированным словом);
- список аргументов – список переменных, которые содержат исходные данные для данной процедуры. Переменные перечисляются через запятую. Список берется в круглые скобки;
- type – тип значения, возвращаемого функцией, то есть тип величины, которая является результатом работы функции;
- выражение – арифметическое, строковое или логическое выражение, которое вычисляет возвращаемое функцией значение.

Список аргументов имеет такую же структуру, что и в подпрограмме.

Для того чтобы вызвать подпрограмму-функцию, надо записать имя функции в какое-либо выражение в том месте, где должно появиться значение вычисляемой функцией величины.

Пример 1

В примере рисуется 10 кругов со случайными радиусами, вычисляется сумма площадей кругов и площадь самого большого круга. Процедура-функция **SqCircle** имеет один аргумент **radius**. Функция возвращает значение площади (тип Single). Обращение к функции записано в операторе присваивания и в условном выражении в операторе **If**.

```
Dim sum As Single
Dim max As Single
Private Function SqCircle (BYVal radius As Single) As Single
SqCircle = 3.14 * radius ^ 2
End Function

Private Sub Command1_Click()
Dim R As Single
Dim X As Single
Dim Y As Single
Dim msg As String
Form1.Cls
Randomize
sum = 0
max = 0
For k = 1 To 10
R = Rnd * 101
X = Rnd * Form1.ScaleWidth
Y = Rnd * Form1.ScaleHeight
Circle (X, Y), R
If SqCircle(r) > max Then
max = SqCircle(r)
End If
sum = sum + SqCircle(r)
Next
msg = "Сумма площадей 10 кругов равна" + Chr(13) + Str(sum) + " кв. ед."
MsgBox msg
msg = "Площадь наибольщего круга равна" + Chr(13) + Str(max) + " кв. ед."
MsgBox msg
End Sub
```

Обратите внимание на функцию **MsgBox**. Она выводит на экран окошко с сообщением.

Б. Процедура-функция Distance

Создадим в модуле функцию **Distance**, вычисляющую расстояние между охотником и кладом. Поскольку охотник передвигается только по вертикали и горизонтали, расстояние будет равно:

$$d = |xhunter - xklad| + |yhunter - yklad|.$$

Для вычисления абсолютной величины числа в языке Visual Basic есть стандартная функция **Abs**. Запишите код функции в модуль.

```
Public Function Distance () As Integer
Distance = Abs (xhunter – xklad) + Abs (yhunter – yklad)
End Function
```

3. Обработка нажатия клавиши, события KeyDown и KeyUp. Процедура Form_KeyDown

А. События KeyDown и KeyUp

События **KeyDown** и **KeyUp** возникают, когда пользователь нажимает клавишу (KeyDown) или отпускает ее (KeyUp). Эти события поддерживаются объектами управления **Form**, **CommandButton**, **HscrollBar**, **VscrollBar**, **OptionButton**, **PictureBox**, **TextBox** и другими.

Мы уже знаем, что все события мышки (Click, MouseDown, MouseMove и т.д.) связаны с теми объектами, над которыми находится курсор в момент наступления события. Как же узнать, с каким объектом связано событие, возникшее при нажатии на клавишу? Если на форме есть объекты управления, имеющие события **KeyDown** и **KeyUp**, эти события связываются то с одним, то с другим объектом. В этом случае говорят, что объект получил событие **KeyDown** или **KeyUp**.

Форма получает события клавиатуры только в том случае, если на ней нет таких объектов или если одно из свойств **Visible** или **Enabled** этих объектов равно **False**. Между тем часто обработку нажатия клавиш удобно связывать именно с формой.

Для того чтобы форма получила событие **KeyDown** или **KeyUp**, в процедуре **Form_Load** свойству формы **KeyPreview** надо присвоить значение **True**:

```
<имя формы>. KeyPreview = True
```

Однако несмотря на значение свойства **KeyPreview** форма не получает события клавиатуры при нажатии на клавиши Enter, Tab и клавиши со стрелками, если на форме есть объекты **CommandButton** или **ListBox**. Поэтому, если вы решите использовать процедуру **Form_KeyDown**, придется обойтись без этих объектов.

Рассмотрим заголовок процедуры обработки события **KeyDown**:

```
Private Sub Form_KeyDown (KeyCode As Integer, Shift As Integer)
```

При нажатии на любую из клавиш возникает событие **KeyDown** и аргументы процедуры обработки события получают следующие значения:

- **KeyCode** – код нажатой клавиши;
- **Shift** – 1 (при нажатии удерживалась клавиша Shift), 2 (клавиша Ctrl), 4 (клавиша Alt).

В Visual Basic есть стандартные константы, значения которых равны кодам клавиш. Ими пользоваться удобнее, чем самими кодами. Приведем некоторые константы и соответствующие им клавиши:

VbKeyLeft	–	клавиша со стрелкой влево;
VbKeyRight	–	клавиша со стрелкой вправо;
VbKeyUp	–	клавиша со стрелкой вверх;
VbKeyDown	–	клавиша со стрелкой вниз;
VbKeyBack	–	клавиша BACKSPACE;
VbKeyReturn	–	клавиша ENTER;
VbKeyShift	–	клавиша SHIFT;
VbKeyControl	–	клавиша CTRL;
VbKeyEscape	–	клавиша ESC;
VbKeySpace	–	клавиша «пробел».

Полный список констант можно получить, если, находясь в среде Visual Basic, выполнить команду **View→Object Browser**, в появившемся текстовом окне набрать **Keycode** и нажать на клавишу **Search** (рис. 40).

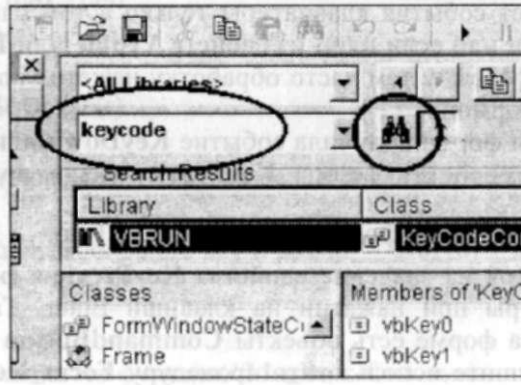


Рис. 40

Пример 2

На форме есть один объект **Shape1**. При нажатии на клавиши со стрелками он перемещается вправо, влево, вверх или вниз в зависимости от нажатой клавиши. За пределы формы объект не выходит. Такое поведение объекта **Shape1** обусловлено следующим кодом:

```
Dim xmax As Integer    "максимальное допустимое значение X
Dim ymax As Integer    "максимальное допустимое значение Y
Dim X As Integer       "текущая координата X
Dim Y As Integer       "текущая координата Y
Dim step As Byte       "длина шага
Private Sub Form_Load ()
Form1.KeyPreview = True
X = Shape1.Left
Y = Shape1.Top
step = 5
End Sub
Private Sub Form_KeyDown(KeyCode As Integer, Shift As Integer)
xmax = Form1.ScaleWidth – Shape1.Width
ymax = Form1.ScaleHeight – Shape1.Height
Select Case KeyCode
    Case vbKeyLeft
        If X – step >= 0 Then X = X – step
    Case vbKeyRight
        If X + step <= xmax Then X = X + step
    Case vbKeyDown
        If Y + step <= ymax Then Y = Y + step
    Case vbKeyUp
        If Y – step >= 0 Then Y = Y – step
End Select
Shape1.Move X,Y
End Sub
```

Обратите внимание на метод **Move**. Выражение **Shape1.Move X, Y** заменяет два оператора присваивания **Shape1.Left = X** и **Shape1.Top = Y**.

✓ Б. Процедура Form_KeyDown в проекте «Найди клад»

Если бы не надо было проверять нашел ли охотник клад и нет ли на пути дерева, процедура **Form_KeyDown** в проекте «Найди клад» мало чем отличалась бы от приведенного примера 2. Поскольку в процедуре кроме

собственно движения охотника выполняются дополнительные операции, приведем укрупненную блок-схему алгоритма этой процедуры (рис. 41).

Укрупненная блок-схема процедуры **Form_KeyDown**

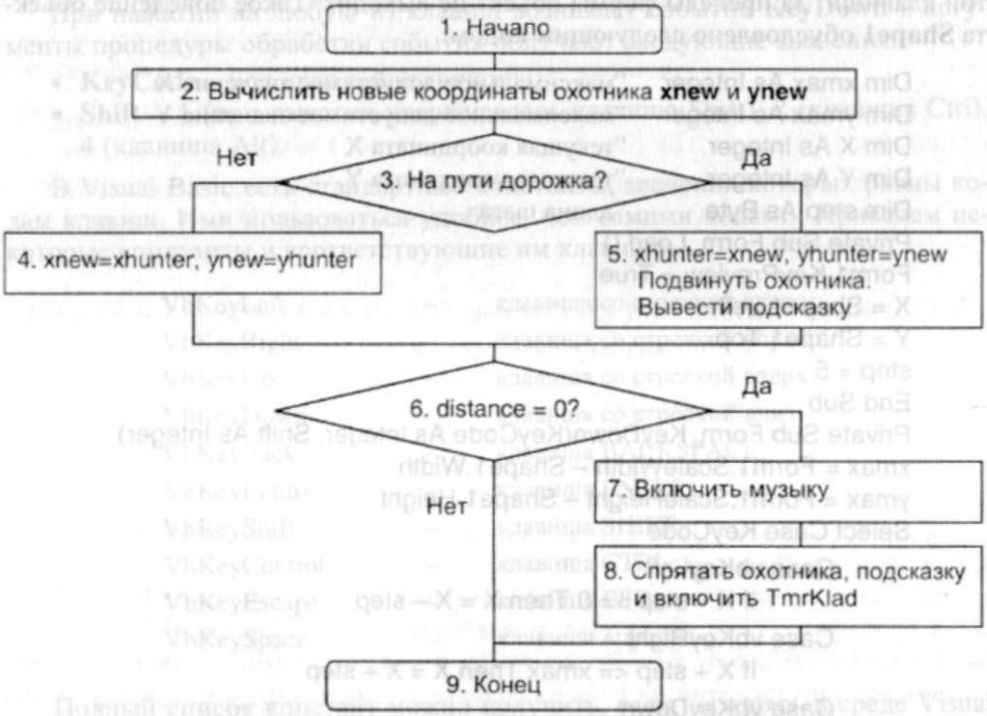


Рис. 41

Каким образом выяснить, что на пути – дорожка или дерево? Когда мы обсуждали, как создать рисунки деревьев и фрагментов дорожки, мы договорились запомнить код цвета середины дорожки. Сейчас он нам пригодится. Выясним, каков цвет точки в середине квадрата, на который собирается ступить охотник. Если он равен цвету дорожки, значит можно идти:

```
If FrmMain.Point(xnew + 25, ynew + 25) = RGB(130, 100, 0) Then
xhunter = xnew
yhunter = ynew
ImgHunter.Move xhunter, yhunter
PrgBar.Value = Distance
Else
xnew = xhunter
ynew = yhunter
End If
```

код цвета точки дорожки в нашем примере

Этот программный код реализует блоки 3, 4 и 5 укрупненной блок-схемы. Эта проверка не только не дает охотнику пойти на то место, где нарисовано дерево, но одновременно не дает ему выйти за границы формы. Поэтому код, реализующий блок 2, будет совсем простым. Возьмем за основу пример 1 (проект 3). В операторе **Select Case** не нужна проверка условия (оператор If отсутствует). Надо только записать операторы вычисления новых координат. Начало оператора **Select Case** выглядит так:

```
Select Case KeyCode
    Case vbKeyLeft
            xnew = xhunter – 50
...
```

Переменные **xnew** и **ynew** должны быть описаны в разделе **General** листа кода формы.

Блок 7 пока пропустите. Подключение звуковых файлов обсудим позже. Сейчас добейтесь, чтобы программа правильно реагировала на нажатие клавиш со стрелками. Для того чтобы объект управления **ProgressBar** работал как подсказка, его надо настроить в процедуре **Form_ Load**.

```
Private Sub Form_Load()
FrmMain.KeyPreview = True
PrgBar.Min = 0
PrgBar.Max = FrmMain.ScaleWidth – 25 + FrmMain.ScaleHeight – 75
PrgBar.Visible = True
Call Forest
PrgBar.Value = Distance
xnew = xhunter
ynew = yhunter.
```

Такой же код должен быть записан в процедуре, реализующей команду «Новая игра».

4. Процедура обработки события TmrKlad_Timer

Процедура **TmrKlad_Timer** копирует изображения ларца с кладом на форму в квадрат с координатами верхнего левого угла (xklad, yklad). Размер квадрата, заданный переменной **size** меняется от 1 до 50. Перед каждым выполнением метода **PaintPicture** переменная **size** увеличивается на 1. Когда переменная **size** становится равной 51, **Timer** выключается (свойство Enabled), а переменной присваивается значение 0. Копирование рисунка осуществляется следующим программным кодом:

```
FrmMain.PaintPicture FrmMain.ImgKlad.Picture, xklad, yklad, size, size
```

Не забудьте объявить в разделе **General** переменную **size** и присвоить значение отличное от 0 свойству таймера **Interval**.

5. Подключение к проекту звуковых файлов с расширением *.wav

Чтобы прослушать звуковой файл с расширением *.wav, используйте API процедуру **PlaySound**. В модуле в разделе **General** запишите оператор **Declare**. Списывайте текст очень аккуратно, чтобы не сделать ошибки.

```
Public Declare Sub PlaySound Lib "winmm.dll" Alias "PlaySoundA" _
    (ByVal lpszName As String, ByVal hModule As Long, ByVal dwFlags As Long)
```

Когда клад найден, включите проигрывание звукового файла. Для этого в процедуру **Form_KeyDown** (см. рис. 41, блок 7) запишите код вызова процедуры **PlaySound**. Процедура работает по-разному в зависимости от значения аргументов. Не обсуждая все существующие возможности, приведем несколько вариантов вызова процедуры, которые могут быть полезны:

а) PlaySound "имя файла", hmodel, SND_ASYNC

Звуковой файл с расширением *.wav проигрывается и параллельно с ним выполняются другие операторы программы. Действие идет на фоне музыки;

б) PlaySound "имя файла", hmodel, SND_SYNC

После выполнения процедуры PlaySound проигрывается звуковой файл. При этом никакие другие операторы программы не выполняются. Программа продолжает свою работу только после того, как звучание закончится;

в) PlaySound "имя файла", hmodel, SND_ASYNC Or SND_LOOP

Программа работает на фоне звука. Как только звуковой файл заканчивается, он начинает звучать сначала. Для того чтобы прервать звучание, надо выполнить вызов процедуры **PlaySound**, записанный в следующем пункте;

г) PlaySound 0, hmodel, SND_ASYNC

Прекращается звучание всех **wav** файлов.

Так же, как графические файлы, звуковой файл следует хранить в папке проекта и использовать **App.Path** для определения пути к звуковому файлу. Пример вызова процедуры, которая служит фоном для выполняемых операторов программы:

```
PlaySound "Blip.wav", hmodel, SND_ASYNC
```

SND_ASYNC, SND_SYNC, SND_LOOP – константы. Их значения в Visual Basic заранее не известны, их надо объявить. В модуле сразу после оператора **Declare** запишите следующий код:

```
Public Const SND_ASYNC = &H1
Public Const SND_SYNC = &H0
Public Const SND_LOOP = &H8
```

Файлы с расширением ***.wav** можно найти в папках Microsoft Office вашего компьютера, в Интернете, на лазерных дисках, а также записать самому. Это не обязательно должна быть музыка, можно записать фразу-поздравление.

В окончательном виде код процедуры **Form_KeyDown** будет таким:

```
Private Sub Form_KeyDown (KeyCode As Integer, Shift as Integer)
Select Case KeyCode
        Case vbKeyLeft: xnew = xhunter – 50
        Case vbKeyRigh: xnew = xhunter + 50
        Case vbKeyDown: ynew = yhunter + 50
        Case vbKeyUp: ynew = yhunter – 50
End Select
If Color (xnew, ynew) = cvet Then
        xhunter = xnew
        yhunter = ynew
        ImgHunter.Move xhunter, yhunter
        PrgBar.Value = Distance
Else
        xnew = xhunter
        ynew = yhunter
End If
If Distance = 0 Then
PrgBar.Visible = False
ImgHunter.Visible = False
Timer1.Enabled = True
PlaySound App.Path & "/Blip.wav", hmodel, SND_ASYNC
End If
End Sub
```

Проект 4

«Тренажер памяти и логического мышления»

Работая над проектом, вы научитесь использовать событие **DragDrop**, узнаете, что такое цикл с послеусловием, познакомитесь со свойствами **DragMode, DragIcon**, **Drawstyle** и переменными типа **Control**.

Постановка задачи

Этот проект задуман для того, чтобы познакомить вас с событием **DragDrop**, которое позволяет захватывать объект управления мышкой и перетаскивать его по форме. Одновременно мы сделаем симпатичный тренажер зрительной памяти.

Проект состоит из четырех форм:

- **FrmLog** – «Тренировка логического мышления»;
- **FrmMemory** – «Тренировка зрительной памяти»;
- **FrmHelpLog** – «Справка к форме FrmLog»;
- **FrmHelpMemory** – «Справка к форме FrmMemory».

*Форма **FrmLog** «Тренировка логического мышления»*

Играющий должен разложить геометрические фигуры в подходящие мешки. В первый мешок можно складывать фигуры желтого цвета, не являющиеся треугольниками, во второй мешок – только желтые треугольники, в третий – все треугольники, кроме желтых. Мышкой можно захватывать геометрическую фигуру, перемещать ее по форме и отпускать. Если перемещаемый объект отпущен над соответствующим мешком, объект остается в

том месте, где был отпущен. Если мешок выбран неправильно, захваченный объект остается на месте. На рис. 42 показано, как выглядит форма до и после выполнения задания.

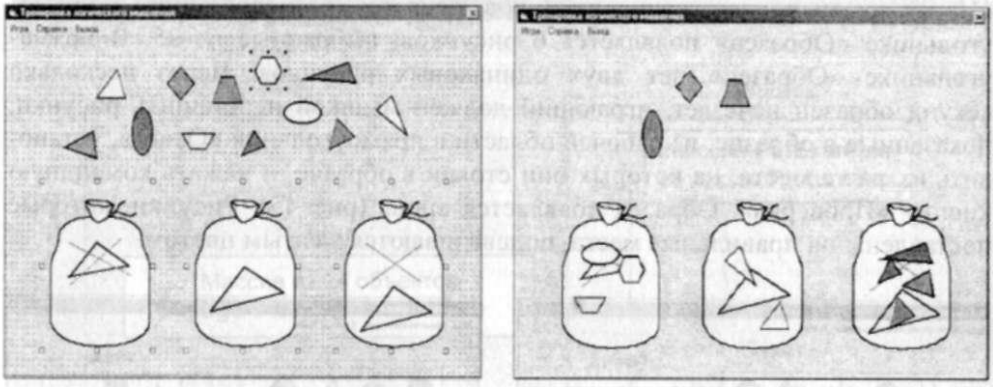


Рис. 42

Пользовательское меню формы содержит команды: «Игра», «Справка», «Выход».

Нажатие на команду «Тренировка памяти» вызывает открытие формы **FrmMemory**.

При нажатии на команду «Справка» открывается форма **FrmHelpLog** (рис. 43).

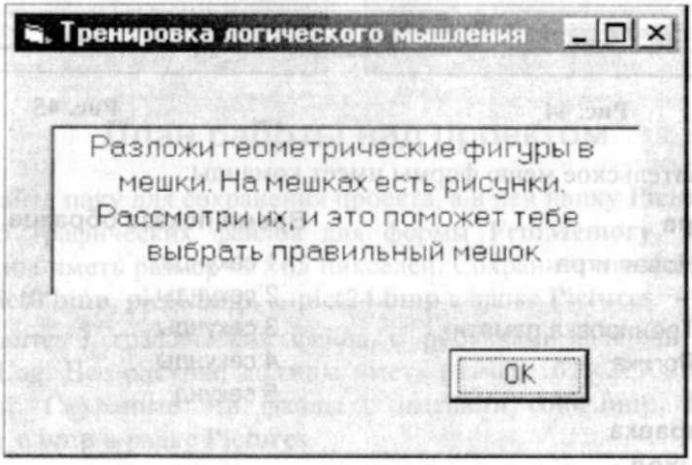


Рис. 43

ВНИМАНИЕ! Форма **FrmHelpLog** является модульной.

*Форма **FrmMemory** «Тренировка зрительной памяти»*

В квадрате находятся 25 рисунков, взаимное расположение которых выбирается случайным образом (рис. 44). При нажатии на команду меню «Новая игра» расположение рисунков меняется. Одновременно в прямоугольнике «Образец» появляется 6 рисунков, выбранные из 25. В прямоугольнике «Образец» нет двух одинаковых рисунков. Через несколько секунд образец исчезает, играющий должен мышкой перетащить рисунки, показанные в образце, из рабочей области в прямоугольник «Ответ», установить их на те места, на которых они стояли в образце, и нажать командную кнопку «Проверка». Образец появляется вновь (рис. 45). Рисунки, которые поставлены на правильные места, подсвечиваются желтым цветом.

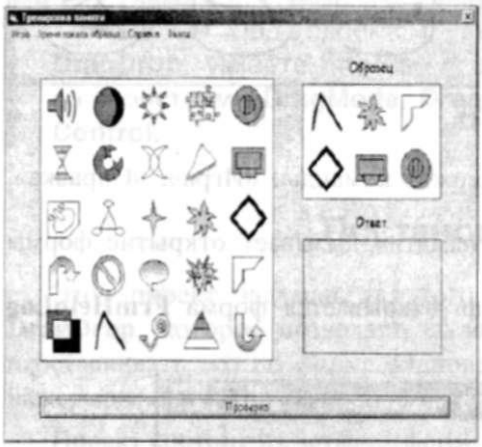


Рис. 44

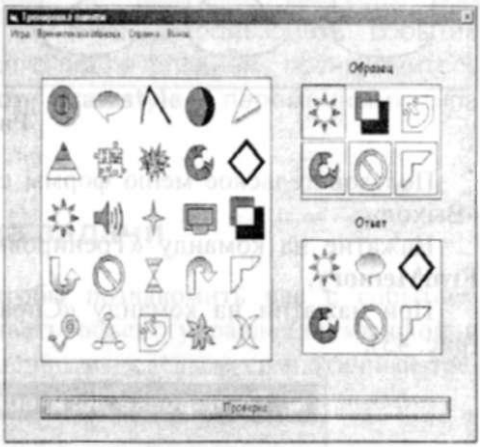


Рис. 45

Пользовательское меню формы имеет команды:

Игра
...Новая игра
... –
...Тренировка памяти
...Логика

Время показа образца
...1 секунда
...2 секунды
...3 секунды
...4 секунды
...5 секунд

Справка
Выход

Нажатие на команду «Логика» вызывает открытие формы **FrmLog**. При нажатии на команду «Справка» открывается форма **FrmHelpMemory**. Она

выглядит так же, как форма **FrmHelpLog**. Форма **FrmHelpMemory** является модульной. На рис. 46 показана схема расположения объектов формы **FrmMemory**.

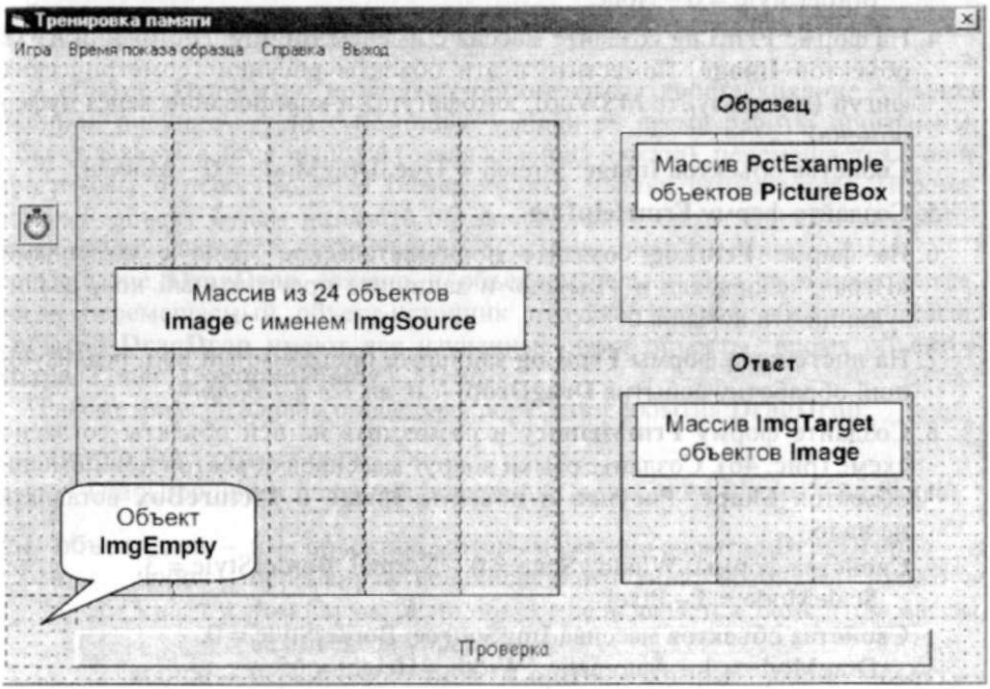


Рис. 46

План работы над проектом

1. Создайте паку для сохранения проекта, а в ней папку **Pictures**. Создайте 25 графических файлов для формы **FrmMemory**. Все рисунки должны иметь размер 65×65 пикселей. Сохраните эти файлы с именами **pict0.bmp, pict1.bmp, …pict24.bmp** в папке **Pictures**.
2. Создайте 3 графических файла с рисунками мешков для формы **FrmLog**. Все рисунки должны иметь размер 161×217 пикселей, фон серый. Сохраните эти файлы с именами **color.bmp, form.bmp** и **form_c.bmp** в папке **Pictures**.
3. Создайте форму **FrmLog**, разместите на ней 3 объекта **PictureBox** с именами **PctColor, PctFC, PctForm** и загрузите туда рисунки мешков.

Свойства формы: WindowState = 0 – Normal. BorderStyle = 3 (последнее свойство не дает пользователю менять размер формы).

Свойства объектов **PictureBox**: ScaleMode = 1 – Twip, BorderStyle = 0 – None.

4. На форме **FrmLog** создайте массив с именем **ImgFig**, состоящий из 12 объектов **Image**. Поместите в эти объекты рисунки геометрических фигур (используйте MSWord, автофигуры и копирование через буфер обмена).

   Свойства объектов **Image**: Stretch = True, DragMode = 0 – Manual.

5. Создайте форму **FrmHelpLog**.
6. На форме **FrmLog** создайте пользовательское меню с командами «Игра», «Справка» и «Выход» и запишите программный код, реализующий эти команды.
7. На листе кода формы **FrmLog** запишите программный код, реализующий обработку события **DragDrop**. *(см. пп.1.А и 1.Б, пр.4).*
8. Создайте форму **FrmMemory** и разместите на ней объекты согласно схеме (рис. 46). Создайте рамки вокруг массивов объектов при помощи объектов **Shape**. Рисунки в объекты **Image** и **PictureBox** вставлять не надо.

   Свойства формы: WindowState = 0 – Normal. BorderStyle = 3, ScaleMode = 3 – Pixel.

   Свойства объектов массива **ImgSource**: BorderStyle = 0, DragMode = 1 – Automatic, Width = Height = 65.

   Свойства объектов массива **ImgTarget**: BorderStyle = 0, DragMode = 0 – Manual, Width = Height = 65.

   Свойства объектов массива **PctExample**: BorderStyle = 0, DragMode =0 – Manual, Width = Height = 65, DrawMode =9 – MaskPen, AutoRedraw = True.

   Свойства объекта **Timer**: Enabled = False, Interval = 5000.

9. Создайте форму **FrmHelpMemory**.
10. На форме **FrmMemory** создайте пользовательское меню и запишите программный код, реализующий все команды меню, кроме команды «Новая игра» *(см. п.2, пр.4).*
11. Запишите программный код команды «Новая игра» *(см. п.3, пр.4).*
12. Запишите программный код, реализующий обработку события **DragDrop** для объектов **ImgTarget** *(см. п.1.В, пр.4).*
13. Запишите программный код, реализующий работу командной кнопки «Проверка» *(см. п.4, пр.4).*

Справочный материал

1. Событие DragDrop

А. Назначение и примеры использования события DragDrop

Событие **DragDrop** позволяет организовать перетаскивание объекта (кнопки, рисунка и т.д.) с помощью мышки во время работы программы. Объект захватывается мышкой (левая клавиша нажата), перемещается в окне программы и освобождается (левая кнопка мышки отпускается). Перемещаемый объект будем называть *объектом-источником*. Объект, над которым освобождается объект-источник, будем называть *объектом-целью*.

Событие **DragDrop** связано с объектом-целью. Оно наступает тогда, когда перемещаемый объект-источник отпускается над объектом-целью. Событие **DragDrop** имеют все изученные нами объекты, кроме объектов **Shape**, **Timer**, **CommonDialog** и **Menu**.

Рассмотрим заголовок процедуры обработки события **DragDrop**:

```
Private Sub <объект-цель>_
    DragDrop([index As Integer,]source As Control, X As Single, Y As Single)
```

где объект-цель – имя объекта, с которым связано событие **DragDrop**;

index – переменная целочисленного типа; возникает только в том случае, если объект-цель является элементом массива объектов управления;

source – переменная типа **Control**, которая равна объекту-источнику, захваченному мышкой;

X и Y – координаты курсора мышки в момент наступления события **DragDrop** относительно верхнего угла объекта-цели. При этом, если объектом-целью является форма или **PictureBox**, координаты даются в тех единицах, которые заданы свойством **ScaleMode** объекта-цели. Для всех других объектов координаты всегда даются в единицах **Twip**.

Поведение объекта-источника зависит от его свойств – **DragMode** и **Enabled**. Для того чтобы объект можно было захватить мышкой, его свойство **Enabled** должно быть равно **True**.

Свойство **DragMode** может принимать следующие значения:

- 0 – Manual (для того чтобы захватить объект мышкой, нужно применить к этому объекту метод **Drag**);
- 1 – Automatic (для того чтобы захватить объект мышкой, не нужен никакой программный код).

Рассмотрим несколько примеров организации обработки события **DragDrop**, которые можно будет использовать как заготовки программного кода.

Пример 1

Передача значения свойств объекта-источника объекту-цели.

На форме четыре объекта **Image** с именами ImgSource1, ImgSource2, ImgSource3, ImgTarget. В объекты ImgSource1, ImgSource2 и ImgSource3 вставлены рисунки.

Свойство **DragMode** объектов ImgSource1, ImgSource2 и ImgSource3 равно 1 – Automatic.

Когда один из объектов **Image**, содержащих рисунки, захватывается мышкой и освобождается над объектом **ImgTarget**, объект **ImgTarget** получает рисунок объекта-источника. Объект-источник остается на прежнем месте.

Программный код:

```
Private Sub ImgTarget_DragDrop (Source As Control, X As Single, Y As Single)
ImgTarget.Picture = Source.Picture
End Sub
```

Переменная **Source** автоматически становится равна тому объекту, который захвачен в данный момент.

Пример 2

Объект-источник переносится на новое место.

На форме один объект **CommandButton** с именем **CmdSource** и объект **PictureBox** с именем **PctSource**. Оба объекта можно перенести на новое место с помощью мышки.

а) У обоих объектов свойство **DragMode = 1** (Automatic).

```
Private Sub Form_DragDrop (Source As Control, X As Single, Y As Single)
Source.Move X, Y
End Sub
```

Недостаток этого метода состоит в том, что независимо от того, где находился курсор в момент захвата объекта-источника, при освобождении верхний левый угол объекта перемещается в точку с координатами курсора мышки. Объект как бы отскакивает.

б) У обоих объектов свойство **DragMode = 1** (Automatic). В свойство **DragIcon** обоих объектов загружается рисунок из графического файла с расширением ***.ico**. Если на компьютере установлен Visual Basic 6.0, коллек-

ция таких файлов обычно находится в директории C:\Program Files\Microsoft Visual Studio\ Common\ Graphics\Icons.

Программный код остается таким же, как в пункте а). Недостаток метода не устраняется, но маскируется, так как в момент движения мы видим не контуры передвигаемого объекта, а маленькую иконку.

в) Свойства DragMode = 0 (Manual), DragIcon = (None)

```
Dim dragx As Integer
Dim dragy As Integer

Private Sub CmdSource _MouseDown (Button As Integer, Shift As Integer, _
   X As Single, Y As Single)
dragx = X
dragy = Y
CmdSource.Drag 1
End Sub

Private Sub PctSource_MouseDown (Button As Integer, Shift As Integer, _
   X As Single, Y As Single)
dragx = X
dragy = Y
PctSource.Drag 1
End Sub

Private Sub Form_DragDrop(Source As Control, X As Single, Y As Single)
Source.Move (X – dragx), (Y – dragy)
End Sub
```

Событие **MouseDown** позволяет для каждого захваченного объекта вычислить поправку к координатам, с помощью которой можно устранить «отскакивание» объекта в момент освобождения. Но объект, чье свойство **DragMode = 1** (Automatic), не получает ни одного события мышки, кроме **DragDrop**. Поэтому свойству присвоено значение 0 (Manual). Теперь, чтобы объект начал двигаться, к нему надо применить метод **Drag** с аргументом 1. Например, **PctSource.Drag 1**.

ВНИМАНИЕ! Очень важно, чтобы у объекта-источника и объекта-цели были установлены одинаковые единицы измерения.

В нашем примере у объекта **CommandButton** единицы измерения координат всегда равны **Twip**, поэтому у объектов **Form** и **PictureBox** тоже установим единицы измерения **Twip**.

✓ Б. Реализация события **DragDrop** на форме **FrmLog**

Воспользуемся методом, описанным в пункте в). Запишем одну процедуру **MouseDown**, так как объекты **ImgFig** образуют массив, и три процедуры обработки события **DragDrop**, связанные с объектами **PictureBox**.

```
Dim dragx As Integer
Dim dragy As Integer
Private Sub Imgfig_MouseDown(Index As Integer, Button As Integer, _
   Shift As Integer, X As Single, Y As Single)
dragx = X
dragy = Y
Imgfig(Index).Drag 1
End Sub
Private Sub PctColor_DragDrop(Source As Control, X As Single, Y As Single)
Set Source.Container = PctColor
Source.Move (X – dragx), (Y – dragy)
Source.Enabled = False
End Sub
Private Sub PctFC_DragDrop(Source As Control, X As Single, Y As Single)
Set Source.Container = PctFC
...
```

Продолжите код самостоятельно. Обратите внимание на то, что свойство **Enabled** источника приравнивается **False**. Это значит, что рисунок нельзя будет вынуть из мешка, на зато несколько рисунков можно будет класть один на другой, они не будут мешать друг другу.

В этом коде встречается незнакомое нам свойство **Container**. Если его не применить, объект **Image** после перемещения окажется под объектом **PictureBox**, а не в нем. Свойство **Container** используют тогда, когда перемещают объект из одного контейнера в другой. Контейнерами являются объекты **Form**, **PictureBox** и **Frame**. Для того чтобы поместить объект 1 в объект 2 как в контейнер, надо записать следующий код:

<объект 1>. Container = <объект 2>

где <объект 1>– имя объекта, который вводится в контейнер (любой из изученных нами объектов, кроме **Timer** и **CommonDialog**);

<объект 2>– имя объекта, в который как в контейнер вводится объект 1 (**Form, PictureBox, Frame**).

Если мы хотим осуществить проверку и «не пускать» геометрическую фигуру в неправильно выбранный мешок, надо в процедуры обработки события добавить оператор **If** и объявить массив переменных типа **Control**,

которые можно было бы сравнивать с переменной **Source** того же типа. В процедуре **Form_Load** эти переменные получат значение. В окончательном виде программный код, реализующий события **DragDrop** на формы **FrmLog**, будет таким:

```
Dim dragx As Integer
Dim dragy As Integer
Dim img(11) As Control

Private Sub Form_Load()
For k = 0 To 11
Set img(k) = Imgfig(k)    'переменная img(0) получает значение объекта
                          'Imgfig(0), переменная img(1) – значение объекта
                          'Imgfig(1) и т. д.
Next
End Sub

Private Sub Imgfig_MouseDown(Index As Integer, Button As Integer, _
        Shift As Integer, X As Single, Y As Single)
Imgfig(Index).Drag 1
    dragx = X
    dragy = Y
End Sub
Private Sub PctColor_DragDrop(Source As Control, X As Single, Y As Single)

If Source = img(1) Or Source = img(2) Or Source = img(4) Then
                          'геометрические фигуры желтого цвета и не треугольники
                          'находятся в объектах ImgFig(1), ImgFig(2) и ImgFig(4).
                          'Поэтому только если переменная Source равна одному из
                          'этих объектов, объект-источник переносится в PictureBox
                          'PctColor (мешок с желтым пятном и зачеркнутым
                          'треугольником
    Set Source.Container = PctColor
    Source.Move (X – dragx), (Y – dragy)
    Source.Enabled = False
End If
End Sub

Private Sub PctFC_DragDrop(Source As Control, X As Single, Y As Single)
If Source = img(0) Or Source = img(3) Then
    Set Source.Container = PctFC
    Source.Move (X – dragx), (Y – dragy)
    Source.Enabled = False
End If
End Sub
```

```
Private Sub PctForm_DragDrop (Source As Control, X As Single, Y As Single)
If Source=img(5) Or Source=img(6) Or Source=img(7) Or Source=img(8) Then
    Set Source.Container = PctForm
    Source.Move (X – dragx), (Y – dragy)
    Source.Enabled = False
End If
End Sub
```

✓ В. Реализация события DragDrop на форме FrmMemory

Для реализации события **DragDrop** на форме **FrmMemory** возьмем за основу пример 1. Наш реальный код будет отличаться от примера только тем, что в нашем проекте **ImgTarget** массив и надо не забыть указать номер элемента массива:

```
Private Sub ImgTarget_DragDrop(Index As Integer, Source As Control, _
        X As Single, Y As Single)
ImgTarget(Index).Picture = Source.Picture
End Sub
```

2. Совершенствование пользовательского меню

✓ А. Разделитель в выпадающем меню

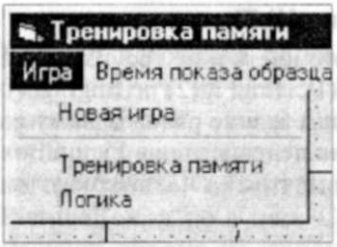


Рис. 47

В выпадающем меню можно разделить связанные по смыслу команды на группы горизонтальной чертой (рис. 47). Для этого после того, как опишите в окне **Menu Editor** команду «Новая игра», в окне **Caption** наберите знак «минус» (–). В окне **Name** наберите любое имя, например **MnuDivider**, и сдвиньте этот пункт меню вправо при помощи кнопки со стрелкой в окне редактора меню.

✓ Б. Массив объектов в пользовательском меню

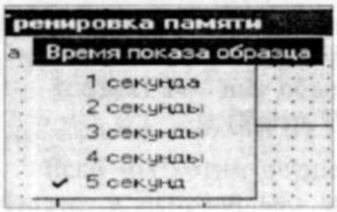


Рис. 48

Пункт меню «Время показа образца» очень похож на команду выбора толщины линии в проекте «Зеркало». Его можно запрограммировать по аналогии. Но можно сократить код в 5 раз, если объявить команды «1 секунда», «2 секунды» и так далее элементами массива (рис. 48). Для этого в окне редактора меню надо всем этим командам в окне **Name** приписать

одно и то же имя (например, MnuSec), а в окне **Index** записать целое число. У команды «1 секунда» **Index** = 1, у команды «2 секунды» – 2,... у команды «5 секунд» – 5. Тогда код, реализующий эти команды, будет таким:

```
Private Sub Mnusec_Click(Index As Integer)
For k = 1 To 5
Mnusec(k).Checked = False
Next
Mnusec(Index).Checked = True
Timer1.Interval = Index * 1000     'время Timer измеряет в миллисекундах
End Sub
```

3. Программный код команды «Новая игра»

А. Укрупненный алгоритм

Рассмотрим укрупненный алгоритм процедуры **MnuNew_Click**, реализующей команду «Новая игра».

Начало.

1. Для всех элементов массива **ImgTarget** стереть рисунки, а для элементов массива **PctExample** свойству **Visible** присвоить значение **True**.
2. Для всех элементов массива **ImgSource** стереть рисунки и свойству **Enabled** присвоить значение **False** (чтобы нельзя было перетаскивать объекты пока виден образец).
3. Заполнить рисунками массив объектов **ImgSource**, располагая рисунки случайным образом.
4. Заполнить рисунками массив объектов **PctExample**, выбирая их случайным образом из массива **ImgSource**.
5. Включить **Timer**.

Конец.

Приведем полностью программный код, реализующий пункт 1 алгоритма.

```
For k = 0 To 5
PctExample (k).Visible = True
ImgTarget (k).Picture = LoadPicture ()
Next
```

Программный код пункта 2 запишите самостоятельно.

Алгоритм реализации пункта 3 рассмотрим подробнее. Идея алгоритма такова: берем первый файл с рисунком, формируем случайным образом номер элемента массива **ImgSource**, в который собираемся ввести этот рисунок, и проверяем, не был ли в этот элемент массива введен рисунок раньше.

Для сравнения используем объект управления **ImgEmpty**, в котором рисунка заведомо нет. Если рисунок не был введен, вводим его. Если был – не вводим, а формируем новый случайный номер элемента. Когда рисунок из первого файла введен, берем следующий файл и продолжаем работу аналогично, пока не прочтем все графические файлы и не заполним все элементы массива **ImgSource**. Проблема заключается в том, что заранее не известно, сколько раз придется формировать случайное число. То есть мы имеем дело с циклом, число повторений которого заранее не известно.

Б. Циклы с предусловием и послеусловием

Существует два вида циклов с неизвестным заранее числом повторений – цикл *с предусловием* и цикл *с послеусловием*. В цикле с послеусловием условие окончания цикла проверяется каждый раз после того, как выполнены повторяющиеся в цикле действия. Этот вид цикла является одной из базовых алгоритмических конструкций. Общая блок-схема цикла с послеусловием приведена на рис. 49.

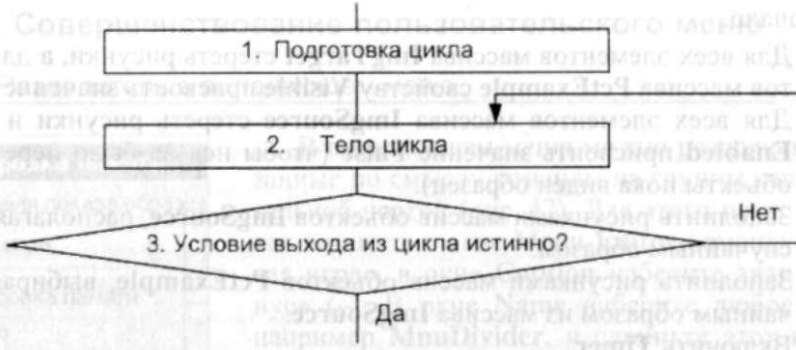


Рис. 49

Такой цикл реализует специальная конструкция **Do...Loop** языка Visual Basic, которая имеет четыре разновидности:

1. Цикл с предусловием, повторяется до тех пор, пока условие истинно:

```
Do While <условие>
<операторы тела цикла>
Loop
```

2. Цикл с предусловием, повторяется до тех пор, пока условие ложно:

```
Do Until <условие>
<операторы тела цикла >
Loop
```

3. Цикл с послеусловием, повторяется до тех пор, пока условие истинно:

```
Do
<операторы тела цикла >
Loop While <условие>
```

4. Цикл с послеусловием, повторяется до тех пор, пока условие ложно:

```
Do
<операторы тела цикла >
Loop Until <условие>
```

Для заполнения массива **ImgSource** рисунками воспользуемся последним (четвертым) вариантом конструкции **Do...Loop**:

```
Randomize
n = 0
Do
r = Int(Rnd * 25)
If ImgSource(r).Picture = ImgEmpty.Picture Then
ImgSource(r).Picture =LoadPicture(App.Path & "\Pictures\pict" & Trim(Str(n)) & ".bmp")
n = n + 1
End If
Loop Until n = 25
```

Реализация выборки шести рисунков и копирования их в массив **PctExample** осуществляется более сложной комбинацией цикла с послеусловием и **For –Next** цикла:

```
n = 0                    "номер элемента массива PctExample
Do
r = Int(Rnd * 25)        "номер элемента массива ImgSource
Copy = True              "логическая переменная Copy получает значение True
For k = n To 0 Step –1   "рисунок из ImgSource(r) сравнивается со всеми
                         "элементами массива PctExample, номер которых
                         "меньше n (рисунки в них вводились раньше). Если
                         "такой рисунок уже был введен, переменная Copy
                         "получает значение False
If ImgSource(r).Picture = PctExample(k).Picture Then
Copy = False
End If
Next
If Copy Then
PctExample(n) = ImgSource(r).Picture
n = n + 1
End If
Loop Until n = 6
```

Запишите программный код в проект и проверьте его работоспособность. Если два массива заполняются рисунками и нет ни пропусков, ни повторений, можете двигаться дальше. Для того чтобы рисунок–образец исчезал через заданный промежуток времени, запишите код процедуры **Timer1_Timer**:

```
Private Sub Timer1_Timer()
For k = 0 To 5
PctExample(k).Visible = False
Next
For k = 0 To 24
ImgSource(k).Enabled = True
Next
Timer1.Enabled = False
End Sub
```

4. Проверка правильности заполнения области «Ответ». Свойство DrawMode

Проверка правильности очень проста. Интересным моментом в этой части программы является использование свойства **DrawMode** объекта **PictureBox**.

В элементы массива **PctExample** введены рисунки из файлов. Если сейчас нарисовать в этих объектах заполненный прямоугольник с помощью метода **Line**, результат будет зависеть от значения свойства **DrawMode**. Обычно это свойство имеет значение 13 (Copy Pen). В этом случае прямоугольник полностью загораживает рисунок. Если же свойству **DrawMode** объекта **PictureBox** присвоить значение 9 (Mask Pen), через прямоугольник будет просвечивать рисунок. Этот прием будем использовать для того, чтобы выделить рисунки, которые поставлены правильно.

Значения свойству **DrawMode**, вообще говоря, можно присвоить в меню **Properties**. Однако это не всегда удается. Поэтому добавим в процедуру **Form_Load** программный код, задающий значение этому свойству:

```
Private Sub Form_Load()
For k = 0 To 5
PctExample(k).DrawMode = 9
Next
Randomize
End Sub
```

Программный код кнопки «Проверка»:

```
Private Sub CmdCheck_Click()
For k = 0 To 5
PctExample(k).Visible = True
If ImgTarget(k).Picture = PctExample(k).Picture Then
PctExample(k).Line (0, 0)–(900, 900), , B
End If
Next
End Sub
```

Для того чтобы прямоугольник был закрашенным, всем элементам массива **PctExample** установите значение свойства **FillStyle** равным 0 (Solid), а **FillColor** – желтым. Не забудьте, что свойство **AutoRedraw** у этих объектов должно быть равно **True**.

Если программа работает так, как ожидалось, не останавливайтесь на достигнутом. Придумайте свои собственные задания для развития логического мышления и связанные с выполнением события **DragDrop**, добавьте в проект дополнительные формы. Желаю успеха!

Проект 5
«Сказки для малышей»

Работая над проектом, вы научитесь создавать текстовые файлы, читать информацию из текстовых файлов, размещать текст, прочитанный из файла, в объектах **TextBox**, **PictureBox**, **Form**, познакомитесь с объектами **Data** и **ListBox**, функцией **MsgBox** и функциями, обрабатывающими текстовые строки (**Len**, **Mid**, **Right**, **Left**).

Постановка задачи

Проект «Сказки для малышей» сделает занимательным обучение чтению детей 5–7 лет. В сказках пропущены некоторые слова. Задача пользователя состоит в том, чтобы прочитать сказку и на место пропущенных слов перетащить с помощью мышки подходящий рисунок (см. форму **FrmRead** на рис. 50).

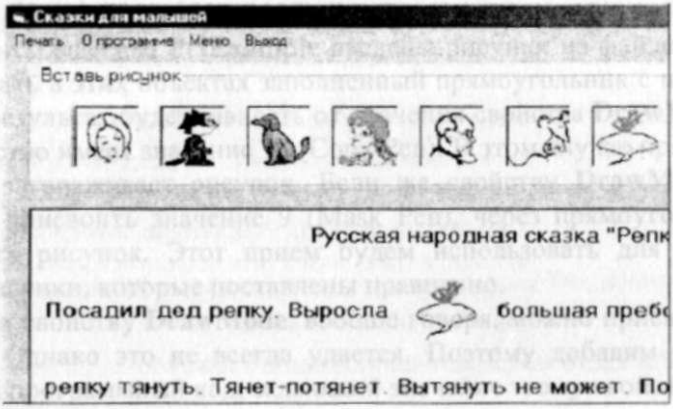


Рис. 50

Этот проект отличается от всех предыдущих тем, что программа, которую мы создадим, будет брать исходную информацию (название и текст сказок, название файлов, содержащих рисунки, и сами рисунки) из внешних по отношению к ней файлов. Это значит, что после создания программы в процессе ее эксплуатации можно будет расширять ее возможности, создавая

все новые и новые сказки. Текст каждой сказки будет хранится в отдельном текстовом файле (файл с расширением *.txt). Такие файлы можно создавать разными способами, например при помощи программы Notepad (Блокнот) или прямо из программы, написанной на языке Visual Basic. Файл с расширением *.mdb, созданный с помощью программы Microsoft Access, будет содержать название сказок, имена файлов с текстом сказок и имена графических файлов с рисунками.

Программа будет считывать сказку из текстового файла и размещать в объекте **PictureBox** с помощью метода **Print**, оставляя пустые места для вставки рисунков. После того, как пользователь вставит рисунки, сказку можно будет вывести на принтер.

Форма **FrmWrite** (рис. 51) предназначена для создания текстовых файлов. Использование этой формы не дает никаких преимуществ по сравнению с программой Notepad. Ее создание носит чисто учебный характер.

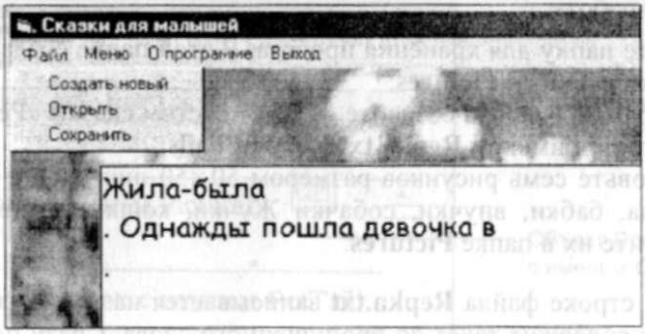


Рис. 51

Формы **FrmMenu** и **FrmChose** позволяют выбрать режим работы и сказку для чтения (рис. 52).

Рис. 52

Работу над проектом разобьем на три этапа. Каждый этап создания программы будет связан с изучением большого объема нового материала и заканчиваться набором тренировочных упражнений.

План работы над проектом

Первый этап. Вывод текста на экран методом Print. Ввод данных из текстового файла

На этом этапе создадим форму **FrmRead** в упрощенном варианте. Она будет работать с одной единственной сказкой, вводимой из текстового файла **Repka.txt**. Схема расположения объектов управления на форме **FrmRead** показана на рис. 53.

Прежде чем приступить к работе в среде Visual Basic, выполните подготовительную работу:

- создайте папку для хранения проекта. В этой папке создайте две папки с именами **Tails** и **Pictures**;
- в программе Notepad создайте файл с текстом сказки «Репка» и сохраните его под именем **Repka.txt** в папке **Tails**;
- подготовьте семь рисунков размером 50×50 пикселей с изображениями деда, бабки, внучки, собачки Жучки, кошки, мышки и репки и поместите их в папке **Pictures**.

В первой строке файла **Repka.txt** записывается название сказки. Каждая строка файла содержит текст до пропущенного слова. Сразу после пропуска слова текст переносится на новую строку.

Текст файла **Repka.txt**:

```
Русская народная сказка «Репка»
Посадил дед репку. Выросла
большая-пребольшая. Стал
репку тянуть. Тянет-потянет. Вытянуть не может. Позвал дед бабку.
за дедку.
за репку. Тянут-потянут. Вытянуть не могут. Позвала бабка внучку.
за бабку.
за дедку.
за репку. Тянут-потянут. Вытянуть не могут. Позвала внучка Жучку.
за внучку.
за бабку.
за дедку.
за репку. Тянут-потянут. Вытянуть не могут. Позвала Жучка кошку.
за Жучку.
```

за внучку.
за бабку.
за дедку.
за репку. Тянут-потянут. Вытянуть не могут. Позвала кошка мышку.
за кошку.
за Жучку.
за внучку.
за бабку.
за дедку.
за репку. Тянут-потянут и вытянули.

Расположите на форме FrmRead объекты в соответствии со схемой (рис. 53) и установите такие значения свойств этих объектов, которые указаны в табл. 1.

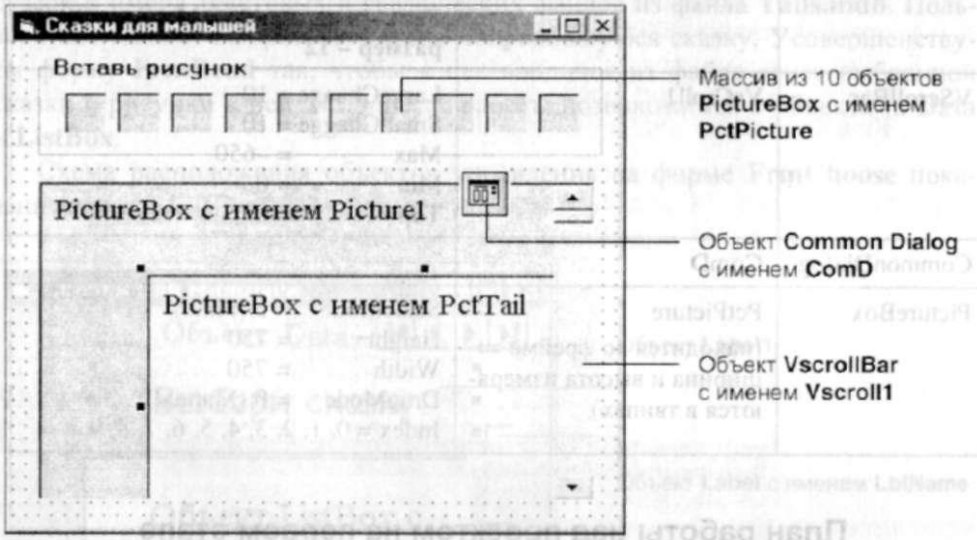


Рис. 53

Таблица 1

Свойства объектов управления формы FrmRead

Тип объекта	Имя объекта	Свойства объектов
Форма	FrmRead	ScaleMode = 3 (Pixel) Height = 7965 Width = 11055 StartUpPosition= 2 Caption = «Сказки для малышей»

Окончание табл. 1

Тип объекта	Имя объекта	Свойства объектов
PictureBox	Picture1	ScaleMode = 3 (Pixel) Height = 350 Width = 700 Left = 16 Top = 120
PictureBox	PctTail	**AutoRedraw = True** ScaleMode = 3 (Pixel) Height = 1000 Width = 700 Left = –2 Top = –2 BackColor – белый **Font – имя шрифта MS Sans Serif, размер – 12**
VScrollBar	VsCroll1	LargeChange = 10 SmallChagge = 10 Max = –650 Min = 0 Top = 0
CommonDialog	ComD	
PictureBox	PctPicture (находится во фрейме ⇒ ширина и высота измеряются в твипах)	ScaleMode = 3 (Pixel) Height = 750 Width = 750 DragMode = 0 (Manual) Index = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

План работы над проектом на первом этапе

1. Создайте форму **FrmRead** и расположите на ней объекты управления как показано на рис. 53. Объекты **PctTail** и **Vscroll1** находятся внутри объекта **Picture1**.
2. Используя меню свойств Properties, введите в объекты **PctPicture(0)**, **PctPicture(1)**, …, **PctPicture(6)** рисунки к сказке «Репка». Запишите проект на диск в подготовленную папку.
3. В процедуру **Form_Activate** запишите программный код для чтения из текстового файла **Repka.txt** названия сказки *(см. п.1, пр.5)*.
4. Добавьте в процедуру **Form_Activate** программный код для ввода из текстового файла **Repka.txt** остального текста сказки *(см. п.1, пр.5)*.

5. Запишите программный код прокрутки текста в объекте **PctTail** *(см. п.2, пр. 5)*.
6. Организуйте обработку события **DragDrop** для копирования рисунков из объектов **PctPicture** в выбранное пользователем место объекта **PctTail** *(см. пп. 1.А и 1.Б, пр.4)*.
7. Создайте пользовательское меню и запишите программный код, реализующий команды меню *(см. п.3, пр. 5)*.
8. Сохраните проект, проверьте его работоспособность. Если надо, исправьте ошибки.

Второй этап. Использование объектов Data и ListBox

На этом этапе создадим форму **FrmChoose**, которая будет читать список сказок и имена текстовых и графических файлов из файла **Tails.mdb**. Пользователь сможет выбрать из списка понравившуюся сказку. Усовершенствуем форму **FrmRead** так, чтобы в нее вводился из файла текст выбранной сказки и рисунки к ней. В процессе работы познакомимся с объектами **Data** и **ListBox**.

Схема расположения объектов управления на форме **FrmChoose** показана на рис. 54.

Рис. 54

План работы над проектом на втором этапе

1. С помощью программы MS Access создайте файл **Tails.mdb**, содержащий имена файлов с рисунками и текстом сказок *(см. п.4, пр.5)*.
2. Создайте форму **FrmChoose** и разместите на ней все объекты управления **Label** в соответствии со схемой *(см. рис. 54)*.
3. Свяжите форму **FrmChoose** с базой данных **Tail.mdb** *(см. п.5, пр. 5)*.
4. Сделайте невидимым объект **Data** (свойство Visible равно False). Создайте объект **Label1** и запишите в него слова «Выбери сказку». Разместите на форме объект **ListBox** и запишите в процедуру **Form_Load** код, выводящий на экран список сказок *(см. п.6, пр.5)*.
5. Запишите программный код процедуры **LstTails_Click**, читающий из базы данных названия файлов, содержащих выбранную пользователем сказку *(см. п.7, пр.5)*.
6. Внесите изменения в процедуру **Form_Activate** формы **FrmRead**, чтобы читалась выбранная сказка *(см. п.8, пр.5)*.
7. Запишите код процедуры для командной кнопки ОК. При нажатии на кнопку форма **FrmChoose** прячется с помощью метода **Hide**, а форма **FrmRead** показывается (метод **Show**).

Третий этап. Создание текстовых файлов из программы VisualBasic

На третьем этапе создадим форму **FrmWrite**, которая позволит создавать текстовые файлы из программы Visual Basic, и форму **FrmMenu**. Соединим формы проекта в единое целое. Схема расположения объектов на форме **FrmWrite** показана на рис. 55.

Рис. 55

План работы над проектом на третьем этапе

1. Добавьте форму **FrmWrite** и расположите на ней объекты управления в соответствии со схемой, показанной на рис. 55. Текстовое окно имеет вертикальную полосу прокрутки.
2. Создайте пользовательское меню (см. рис. 51).
3. Создайте форму **FrmMenu** (см. рис. 52), содержащую командные кнопки «Прочитать сказку», «Написать сказку» и «Выход». Свяжите все формы воедино. Измените свойство **Start** *(см. п.9, пр.5)*.
4. Запишите программный код, реализующий выполнение команды «Файл → Сохранить» пользовательского меню формы **FrmWrite** *(см. п.10, пр. 5)*.
5. Запишите программный код, реализующий выполнение команд «Создать новый», «Меню», «Выход» и «Открыть» *(см. п.11. пр.5)*.
6. Создайте справочную форму с именем **FrmHelp** и организуйте открытие этой формы при нажатии на команду «О программе».

Справочный материал

1. Запись и чтение из последовательного файла

А. Понятие последовательного файла и порядок работы с ним

Язык Visual Basic содержит много функций и операторов, позволяющих читать информацию из дисковых файлов и записывать информацию в виде дискового файла. Выбор операторов зависит от того, какой тип файла мы хотим создать. Различают файлы трех типов:

- файлы последовательного доступа;
- файлы произвольного доступа;
- двоичные файлы.

Мы с вами рассмотрим только файлы последовательного доступа. Они состоят из записей переменной длины. Такие файлы называются последовательными потому, что могут читаться только с самого начала. Если нам требуется третья от начала файла запись, надо сначала прочитать первую, затем вторую и только потом третью. Нельзя прочитать запись сразу из середины файла. Каждая запись является строкой текста, которая заканчивается специальными символами-разделителями. Поэтому файлы последовательного доступа называют иногда текстовыми файлами.

Работа с файлом, имеющим любой тип доступа, состоит из трех этапов:

- открытие файла;
- чтение или запись информации в файл;
- закрытие файла.

Для открытия файла используется оператор **Open**, имеющий следующий формат:

Open <имя файла> **For** <режим доступа> **As #**<номер файла>

где **имя файла** – имя открываемого файла; если путь к файлу не указан, файл открывается в текущей папке. Имя файла может быть задано как строковая константа (заключается в кавычки), переменная типа String или строковое выражение;

режим доступа – определяет способ доступа к файлу; для последовательных файлов (файлов последовательного доступа) может иметь следующие значения:

- **Input** – файл открывается для чтения информации из него. Если файл с указанным именем на диске не существует, возникает ошибка;
- **Output** – последовательный файл открывается для записи в него информации. Если файл с указанным именем не существовал на диске, он создается. Если файл уже существует, он удаляется с диска (существовавшая раньше информация пропадает) и создается заново;
- **Append** – последовательный файл открывается для записи в него информации. Если файл с указанным именем уже существует на диске, запись производится в конец файла (существовавшая раньше информация сохраняется). Если файл не существовал на диске, он создается;

номер файла – целое число от 1 до 511. После открытия файла операторы и функции, работающие с файлом, обращаются к нему не по имени, а по номеру.

Закрытие файла осуществляется с помощью оператора **Close:**

Close [#<номер файла>][, #<номер файла>][, #<номер файла>] ...

Оператор закрывает файлы, номера которых указаны в списке. Если номера файлов не указаны, закрываются все файлы, открытые в текущий момент. Файлы необходимо закрывать сразу после того, как работа с ними окончена.

Б. Чтение данных из последовательного файла

Для чтения информации из текстового файла можно использовать операторы **Line Input #** и **Input #**:

Line Input # <номер файла>, <имя переменной типа string>

Оператор **Line Input #** читает из файла строку текста и присваивает прочитанное значение строковой переменной. Признаком конца строки служит символ возврата каретки или комбинация символов возврата каретки и конца строки. Когда мы просматриваем текстовый файл с помощью программы Notepad, эти символы не видны на экране. Они вызывают перенос текста на следующую строку. Если файл создавался в программе Notepad, символ возврата каретки формируется при нажатии на клавишу **Enter**. Если файл создавался из программы, написанной на Visual Basic, этот символ добавляется в файл операторами **Write #** или **Print #**, отвечающими за запись информации в открытый файл. Приведем пример чтения информации из текстового файла.

Пример 1

Файл с именем text1.txt содержит следующий текст:

Что означает фразеологизм "Троянский конь" ?
1) вещь, поразившая воображение;
2) желанный подарок;
3) подарок с подвохом.

В процедуру обработки события **Command1_Click** записан программный код, который читает текст из файла и помещает его в объекты **Label:**

```
Private Sub Command2_Click()
Dim a As String
Dim b As String
Open "text1.txt" For Input As #1
Line Input #1, a
Line Input #1, b
Close #1
Label1.Caption = a
Label2.Caption = b
End Sub
```

На рис. 56 показана форма после нажатия на кнопку **Command1**:

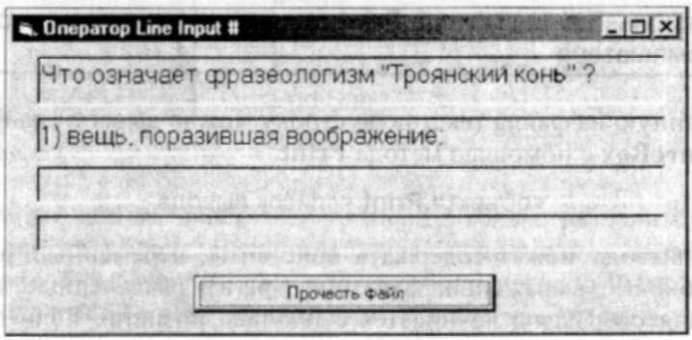


Рис. 56

В отличие от **Line Input #** оператор **Input #** может читать не всю текстовую строку, а только ее часть, называемую полем. Поля отделяются друг от друга запятой. Если текст поля записан в кавычках, эти кавычки при чтении из файла отбрасываются. Отбрасывается также запятая, разделяющая поля. Если в примере 1 все операторы **Line Input #** заменить операторами **Input #**, то форма после нажатия на кнопку «Прочесть файл» будет выглядеть как на рис. 57. Формат оператора **Input #**:

Input # <номер файла>, <имя переменной>, <имя переменной>,...

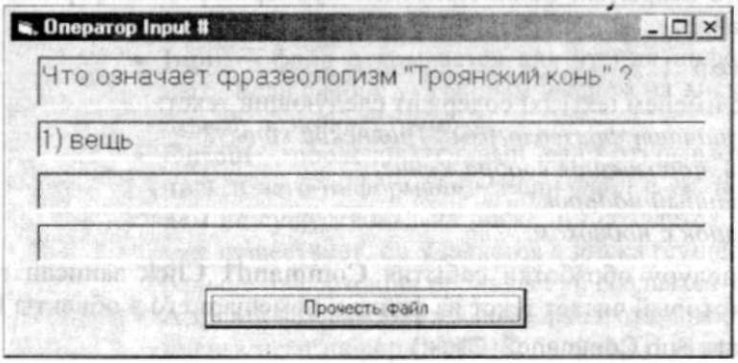


Рис. 57

Переменные, используемые в операторе, могут быть как строкового, так и численного типа. Если оператор **Input #** читает данные из файла, не присваивает значение числовой переменной, признаками конца поля могут служить не только запятая, закрывающие кавычки или символ возврата каретки, но и пробел. Оператор **Input #** обычно используют для чтения файлов, созданных с помощью оператора **Write #**.

В. Использование метода Print для вывода текста на экран компьютера

Прочитанную из файла текстовую строку можно вывести на форму или в объект **PictureBox** с помощью метода **Print**:

<объект>.**Print** <список вывода>

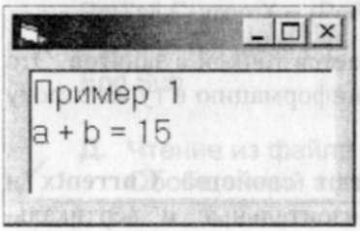
Список вывода может содержать константы, переменные, арифметические и строковые выражения, функцию **Spc(n)**, разделенные запятой или точкой с запятой. Вывод начинается с текущей позиции. Если до выполне-

ния оператора **Print** к форме или объекту **PictureBox** не применялись графические методы или другой оператор **Print**, текущая позиция находится в верхнем левом углу объекта.

Функция **Spc(n)** выводит на экран n пробелов. Вместо n надо записать целое число или целочисленную переменную.

Чтобы понять, как работает метод **Print,** внимательно изучите примеры 2–5.

Пример 2

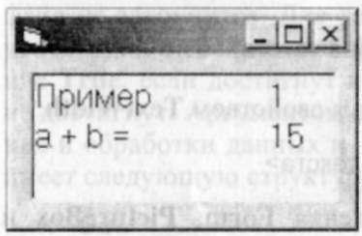



```
Private Sub Form_Activate()
a = 2
b = 13
Picture1.Print "Пример "; 1
Picture1.Print "a + b ="; a + b
End Sub
```

Рис. 58

В списке вывода первого метода **Print** два элемента – строковая константа «Пример» (строковые константы всегда заключаются в кавычки) и численная константа 1. Разделителем между элементами списка вывода служит точка с запятой (форма показана на рис. 58). Второй метод **Print** также имеет список вывода, состоящий из двух элементов – строковая константа "a+b=" и арифметическое выражение a+b. Константы выведены в объект **Picture1** так, как они были записаны в списке вывода. Вместо арифметического выражения a+b на экран выведено его значение 15.

Пример 3

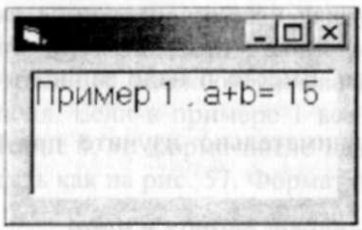



```
Private Sub Form_Activate()
a = 2
b = 13
Picture1.Print "Пример ", 1
Picture1.Print "a + b =", a + b
End Sub
```

Рис. 59

Программный код отличается от примера 2 только разделителем между элементами списка вывода. В данном случае используется запятая (форма показана на рис. 59). Сравните результаты работы методов **Print** в примерах 2 и 3 и сделайте вывод о том, какую роль играют разделители.

Пример 4




```
Private Sub Form_Activate()
a = 2
b = 13
Picture1.Print "Пример"; 1;
Picture1.Print ", a+b="; a + b
End Sub
```

Рис. 60

Список вывода первого метода **Print** оканчивается точкой с запятой. Это приводит к тому, что следующий метод выводит информацию в ту же строку (рис. 60).

Объекты **Form**, **PictureBox** и **Printer** имеют свойства **Currentx** и **Currenty**, которые позволяют установить горизонтальные и вертикальные координаты точки, в которой начинается печать текста методом **Print**. Координаты заданы в единицах измерения, заданных свойством **ScaleMode**.

Г. Чтение из файла и вывод в объект PctTail названия сказки

В первой строке текстового файла **Repka.txt** записано название сказки. Прочтем его из файла и напечатаем с помощью метода **Print** в объекте **PctTail**. Пусть название сказки будет напечатано в середине первой строки. Для этого надо определить ту координату по горизонтали, с которой начинается печать. Вычислим координату по формуле:

X = (ширина объекта PctTail – ширина текста) / 2.

Для определения ширины текста воспользуемся свойством **TextWidth**:

<объект>.**TextWidth** <строка текста>

Этим свойством обладают объекты управления **Form**, **PictureBox** и **Printer**. Оно равно ширине строки текста в единицах измерения, заданных свойством ScaleMode. При этом учитывается размер шрифта (свойства **FontName** и **FontSize** данного объекта). Строка текста может быть задана константой или переменной типа **String**. Представленный код читает из файла **Repka.txt** первую строку, помещает прочитанный текст в переменную **tailstring** и печатает его в середину первой строки объекта **PctTail**.

Код выполняется сразу после того, как форма стала активной, то есть начала работать.

```
Dim tailstring As String
Private Sub Form_Activate ()
PctTail.Cls
PctTail.CurrentY = 10      "число 10 подобрано опытным путем
Open App.Path + "\tails\" + "Repka.txt" For Input As #1
Line Input #1, tailstring
PctTail.CurrentX = (PctTail.ScaleWidth – PctTail.TextWidth (tailstring)) / 2
PctTail.Print tailstring
End Sub
```

Д. Чтение из файла и вывод в объект PctTail текста сказки. Обобщенный алгоритм

Чтение текста из файла состоит из стандартного набора шагов и не представляет трудности. Проблема состоит в том, что при выводе текста с помощью метода **Print** автоматически не производится перенос на новую строку, если ширина текста больше, чем ширина объекта **Form** или **PictureBox**. Перенос текста на новую строку должен организовать программист. Кроме того, надо оставить место для вставки в текст рисунка пользователем нашей программы. Программный код чтения текста из файла и вывода его в объект **PctTail** запишем в процедуру **Form_Activate**. На рис. 61 представлен обобщенный алгоритм этой процедуры.

Блоки алгоритма с четвертого по одиннадцатый образуют цикл, который реализуется с помощью операторов **Do – Loop**. В этом цикле из файла читается текстовая строка и обрабатывается. Цикл повторяется до тех пор, пока файл не закончится. Для того чтобы выяснить прочитан ли файл до конца, используется специальная функция – **EOF**. Эта функция возвращает значение **True**, если достигнут конец файла, и значение **False**, если конец файла не достигнут. Аргументом является номер открытого файла. Реализация чтения и обработки данных из файла последовательного доступа в Visual Basic имеет следующую структуру:

```
Do
...
<Чтение строки из файла>
<Обработка прочитанной информации>
...
Loop Until EOF(<номер открытого файла>)
Close #<номер открытого файла>
```

Блок-схема процедуры **Form_Activate**

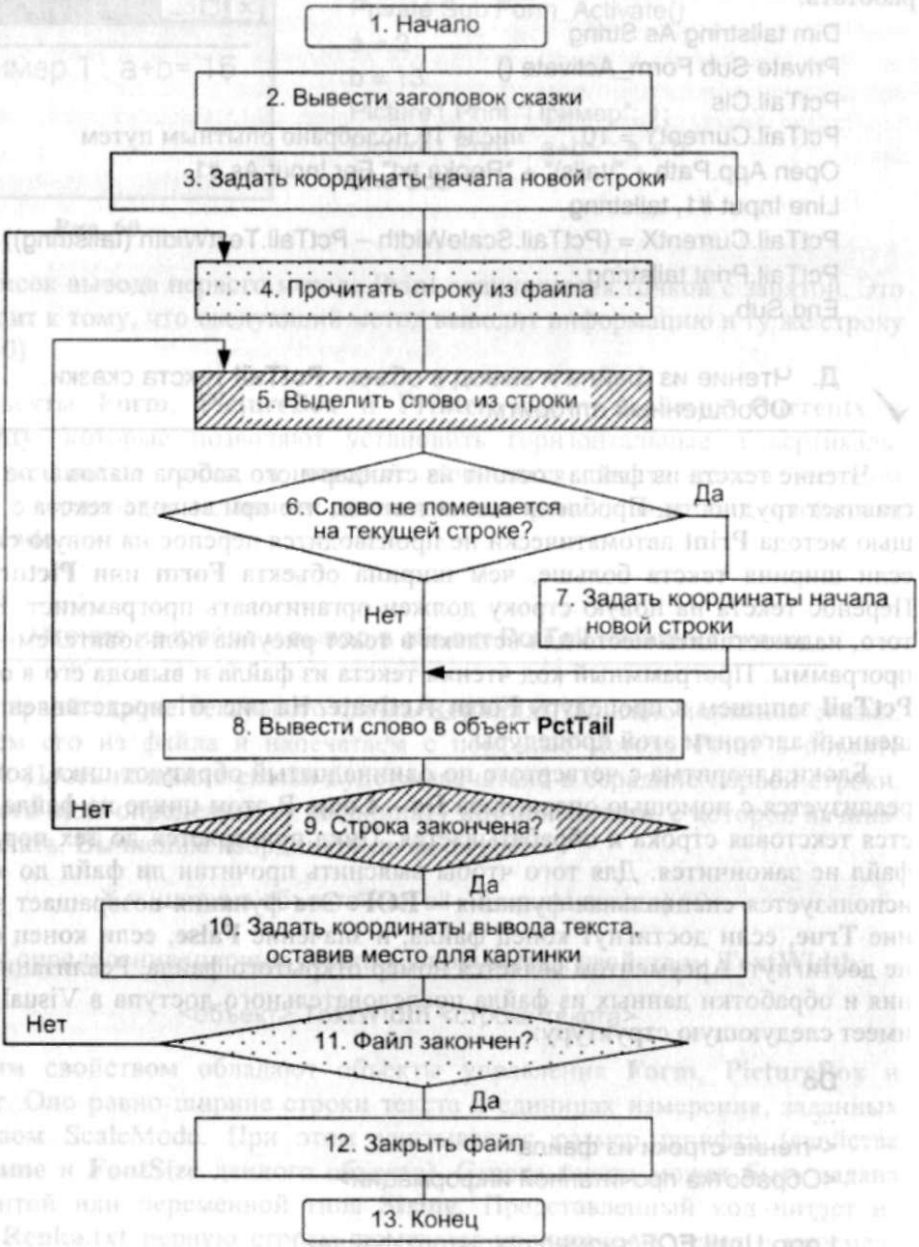


Рис. 61

Блоки с пятого по девятый организуют разбивку прочитанной из файла строки текста на слова и вывод их с помощью оператора **Print** в объект **PctTail** с переносом текста на следующую строку, когда это необходимо. Этот цикл может быть реализован с помощью операторов **For – Next**. В нем используются функции обработки текста, которые мы разберем в следующем подразделе.

Функции ***Len, Left, Right, Mid***

Функции **Len**, **Left**, **Right Mid** предназначены для обработки строк текста. Функция **Len** возвращает число, равное количеству символов в строке текста. Аргументом функции может быть строковая константа, переменная или выражение. Формат функции **Len**:

Len (<строка текста>)

Пример 5

```
N = Len ("Весна идет!")
Print N
```

На форме напечатается число 11.

Пример 6

```
S = "А и Б"
N = Len (S)
Print N
```

На форме напечатается число 5 (в строке три буквы и два пробела).

Функция **Left** возвращает первые n символов строки–аргумента. Строка символов может быть задана как строковая константа, переменная или выражение. При этом вместо n записывается целочисленная константа, переменная или выражение. Формат функции **Left**:

Left (<строка текста>, n)

Пример 7

```
S1 = "Весна идет!"
S2 = Left (S1, 5)
Print S2
```

На форме напечатается слово «Весна» (функция Left выделила первые пять символов строки).

Функция **Right** возвращает последние n символов строки-аргумента. Строка символов может быть задана как строковая константа, переменная

или выражение. При этом вместо n записывается целочисленная константа, переменная или выражение. Формат функции **Rignt**:

Right (<строка текста>, n)

Пример 8

```
S1 = "Весна идет!"
S2 = Right (S1, 5)
Print S2
```

На форме напечатается текст «идет!» (функция Right выделила последние пять символов строки).

Функция **Mid** возвращает n символов строки-аргумента, начиная с k-го. Строка символов может быть задана как строковая константа, переменная или выражение. k и n – целочисленные константы, переменные или выражения. Формат функции **Mid**:

Mid (<строка текста>, k, [n])

Пример 9

```
S1 = "Весна идет!"
S2 = Mid (S1, 4, 2)
Print S2
```

На форме напечатается слово «на».

Пример 10

```
S1 = "235 12 127 4563"    'группы цифр разделены пробелами
L = Len (S1)              'переменная L получает значение 15
S2 = ""                   'между кавычками нет пробела. Значение
                          'переменной S2 – пустая строка
For j = 1 to L
Letter = Mid (S1, j, 1)
S2 = S2 & Letter
If Letter = "" then
Print S2
S2 = ""
Next
```

В На форме напечатается столбик из четырех чисел:

235
12
127
4563

Программный код процедуры ***Form_Activate***

Программный код, реализующий алгоритм чтения из файла текста сказки и вывода его в объект **PctTail**, использует следующие переменные:

- **tailstring** – переменная типа **string**, содержит строку текста, прочитанную из текстового файла оператором **Line Input #**;
- **letter** – переменная типа **string**, содержит один символ, выделенный из строки **tailstring**;
- **word** – переменная типа **string**, содержит одно слово, выделенное из строки **tailstring**.

Текст переносится на новую строку целыми словами. При выделении слов из строки учитывается, что слова в тексте разделены пробелами. Текстовый файл составлен так, что после вывода на экран каждой строки текста надо оставить пустое место для вставки рисунка.

```
Dim letter As String
Dim word As String
Dim tailstring As String
Private Sub Form_Activate ()
PctTail.Cls
word = ""
PctTail.CurrentY = 10
Open App.Path + "\tails\Repka.txt" For Input As #1
Line Input #1, tailstring    'чтение заголовка сказки
PctTail.CurrentX = (PctTail.ScaleWidth – PctTail.TextWidth (tailstring)) / 2
PctTail.Print tailstring
PctTail.CurrentY = PctTail.CurrentY + 40
PctTail.CurrentX = 10
Do                           'начало цикла чтения и вывода на экран текста сказки
Line Input #1, tailstring
n = Len(tailstring)
For k = 1 To n               'начало цикла обработки прочитанной строки текста
   letter = Mid (tailstring, k, 1)
   word = word & letter
   If letter = "" Or k = n Then    'если выделенный символ равен пробелу или
                                   'является последним в строке, слово надо
                                   'вывести на экран
      If PctTail.CurrentX + PctTail.TextWidth (word) > _
            PctTail.ScaleWidth – VScroll1.Width Then
      'если слово не помещается в текущей строке, задаются
      'координаты следующей строки. Расстояние между
      ' строками (60 пикселей) подобрано опытным путем
```

```
            PctTail.CurrentY = PctTail.CurrentY + 60
            PctTail.CurrentX = 10
        End If
        PctTail.Print word;
        word = ""
    End If
Next
If PctTail.CurrentX + 80 > PctTail.ScaleWidth – VScroll1.Width Then
   PctTail.CurrentY = PctTail.CurrentY + 60
   PctTail.CurrentX = 10
End If
PctTail.CurrentX = PctTail.CurrentX + 80    "оставили 80 пикселей для вставки
                                            "рисунка
Loop Until EOF (1)      "если достигнут конец файла № 1 переход к следующей
                        "строке. Если нет – к началу цикла Do
Close #1
End Sub
```

* Оператор **If** проверяет достаточно ли места на текущей строке для картинки? Если нет – переходит на новую строку.

2. Организация прокрутки текста в объекте PictureBox

Текст сказки может не поместиться на экране целиком. Поэтому надо организовать прокрутку текста. Объект **PictureBox**, в который выводится текст, не способен прокручивать текст автоматически. Возможность прокручивать длинный текст обеспечивается за счет того, что объект **PctTail** имеет высоту больше чем высота экрана и может вместить весь текст. Этот объект, в свою очередь, расположен внутри объекта **Picture1**, который имеет меньшую высоту. То есть больший объект находится внутри меньшего. Мы как бы смотрим на объект **PctTail** сквозь маленькое окошко и видим только его часть. Так как объект **PctTail** находится внутри контейнера **Picture1**, его свойство **Top** задает координату верхнего левого угла объекта относительно системы координат, связанной с контейнером. Если **PctTail.Top = 0**, мы видим верхнюю границу **PctTail** и начало сказки (рис. 62). Если **PctTail.Top < 0**, начала сказки не видно, но в поле зрения появилось продолжение сказки, которое раньше было скрыто (рис. 63).

Таким образом, меняя значение свойства **Top** объекта **PctTail**, можно организовать прокрутка текста и рисунков. Изменение свойства **Top** связано с объектом **Vscroll1**, который тоже находится внутри контейнера **Picture1**.

Пусть верхнему положению движка полосы прокрутки (свойство **Min**) соответствует число **0** – самое большое значение свойства **PctTail.Top**,

а нижнему положению (свойство **Max**) число 650 – самое маленькое значение свойства **PctTail.Top**. Как определить самое маленькое и самое большое значение свойства **PctTail** показано на рис. 62–64.

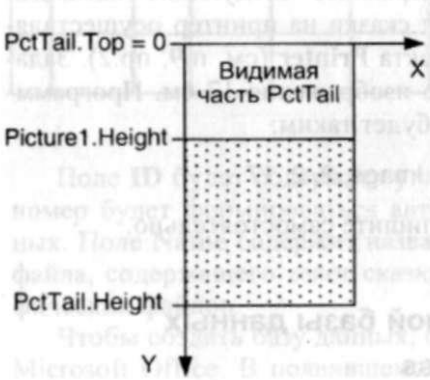


Рис. 62

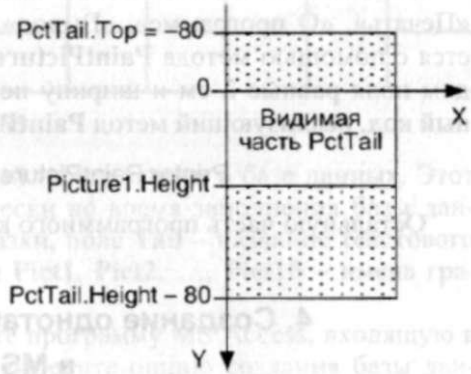


Рис. 63

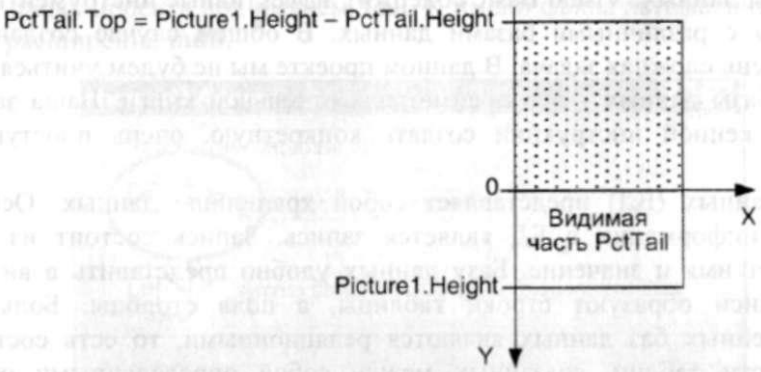


Рис. 64

Программный код, реализующий прокрутку текста и рисунков, очень прост. Он состоит из двух процедур обработки событий, связанных с объектом **VscrollBar**:

```
Private Sub VScroll1_Change ()
PctTail.Top = VScroll1.Value
End Sub

Private Sub VScroll1_Scroll ()
PctTail.Top = VScroll1.Value
End Sub
```

3. Пользовательское меню формы FrmRead. Печать сказки

Создайте пользовательское меню, содержащее следующие команды: «Печать», «О программе», «Выход». Вывод сказки на принтер осуществляется с помощью метода **PaintPicture** и объекта **Printer** (см. п.9, пр.2). Зададим поля равные 2 см и ширину печатного изображения 17 см. Программный код, реализующий метод **PaintPicture**, будет таким:

```
Printer.PaintPicture PctTail.Image, 2, 2, 17
```

Остальную часть программного кода напишите самостоятельно.

4. Создание однотабличной базы данных в MS Access

Внешние по отношению к программе данные могут храниться на диске в виде базы данных. Visual Basic содержит эффективные инструменты связи программы с различными базами данных. В общем случае создание баз данных очень сложная задача. В данном проекте мы не будем учиться проектировать базы данных – это предмет для отдельной книги. Ваша задача – по предложенной инструкции создать конкретную, очень простую базу данных.

База данных (БД) представляет собой хранилище данных. Основной единицей информации в БД является запись. Запись состоит из полей. Поле имеет имя и значение. Базу данных удобно представить в виде таблицы. Записи образуют строки таблицы, а поля столбцы. Большинство современных баз данных являются реляционными, то есть состоят из совокупности таблиц, связанных между собой определенными отношениями. Мы же создадим однотабличную базу данных, что существенно упростит нашу задачу. Однотабличную базу данных можно создать с помощью программы MS Excel или MS Access. Прежде чем приступать к созданию базы данных, решим какую информацию будет содержать каждая запись.

Запись в нашей базе данных будет содержать информацию, необходимую для чтения сказки из внешних файлов программой, написанной на Visual Basic. Это название сказки, имя текстового файла, содержащего текст сказки, и имена графических файлов с рисунками для данной сказки. Таблица базы данных будет иметь следующую структуру (рис. 65):

ID	Name	Tail	Pict1	Pict2	Pict3	Pict4	Pict5	Pict6	Pict7	Pict8	Pict9	Pict10

Рис. 65

Поле **ID** будет содержать уникальный номер записи в базе данных. Этот номер будет формироваться автоматически во время заполнения базы данных. Поле **Name** содержит название сказки, поле **Tail** – название текстового файла, содержащего текст сказки, поля **Pict1**, **Pict2**, ..., **Pict10** – имена графических файлов.

Чтобы создать базу данных, откройте программу MS Access, входящую в Microsoft Office. В появившемся окне выберите опцию создания базы данных с помощью пустого бланка (рис. 66) и нажмите на кнопку OK. После этого откроется диалоговое окно сохранения файла базы данных. Сохраните файл в папке проекта под именем **Tail**. К имени файла автоматически добавляется расширение **mdb**.

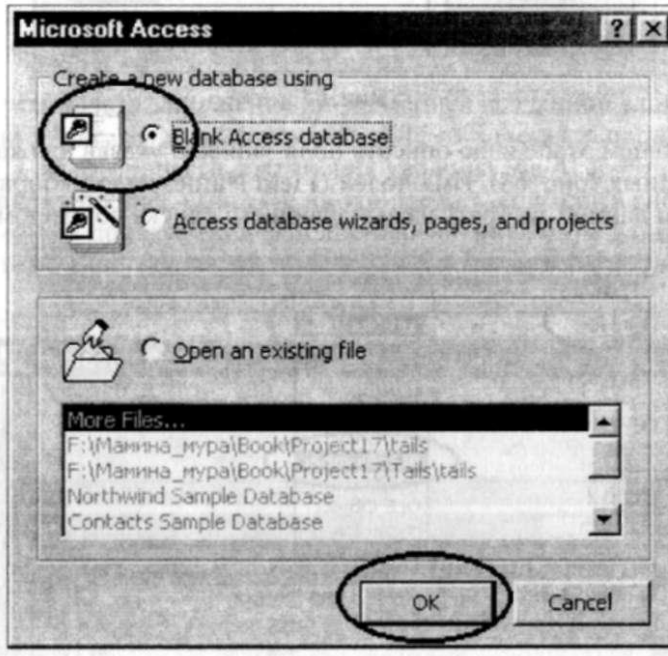


Рис. 66

Следующим шагом является выбор способа создания таблицы базы данных. Предлагается создать таблицу одним из трех способов:

- в режиме конструктора;
- с использованием мастера;
- путем ввода данных в таблицу.

Выберем первый способ – создание в режиме конструктора. Для этого дважды щелкните по соответствующей строке (рис. 67).

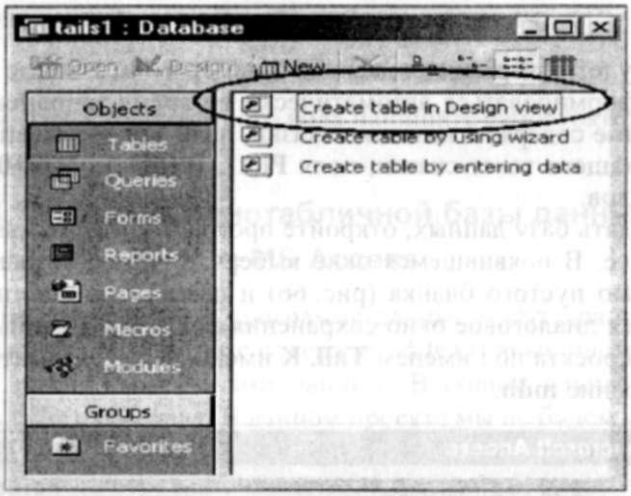


Рис. 67

На следующем этапе надо описать поля записей, указав для каждого поля имя и тип данных (рис. 68). Имя полей (**Field Name**) надо набирать в первом столбце, а тип данных выбрать из выпадающего списка во втором столбце.

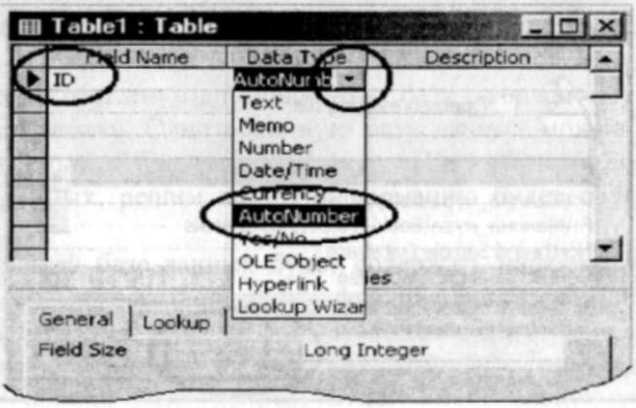


Рис. 68

Данные в поле ID имеют тип **AutoNumber**, данные во всех остальных полях – тип **Text**. После ввода информации о всех полях окно конструктора форм будет выглядеть как на рис. 69.

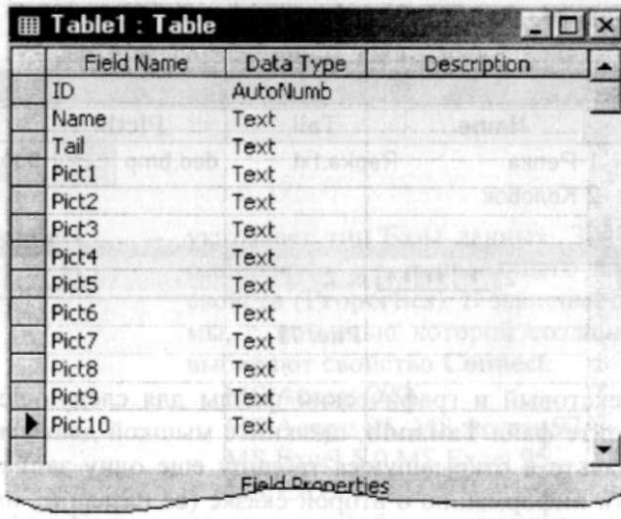


Рис. 69

Прежде чем начать заполнение информацией созданной таблицы, ее надо сохранить на диске, выполнив команду **File → Save as**. Сохраните таблицу под именем **Tail**. В процессе сохранения таблицы Ms Access предложит вам создать первичный ключ. Ответьте на это предложение согласием.

Теперь можно приступать к заполнению таблицы базы данных информацией. Для этого надо нажать на кнопку **View** (Вид) на панели инструментов (рис. 70).

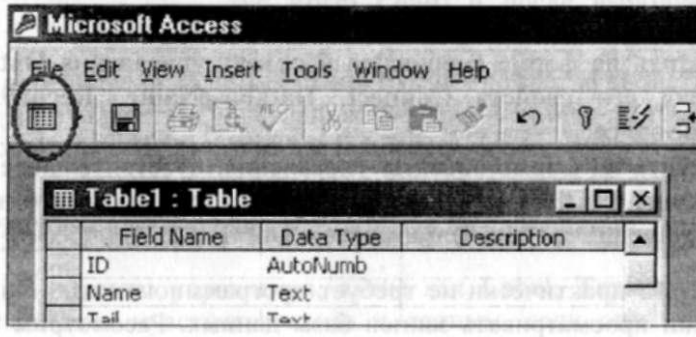


Рис. 70

Заполните первую строку таблицы информацией о сказке «Репка».

На рис. 71 показан фрагмент заполненной таблицы базы данных. База данных, состоящая из одной записи, готова. Программу MS Access можно закрыть. При закрытии программы введенная информация автоматически сохраняется на диске.

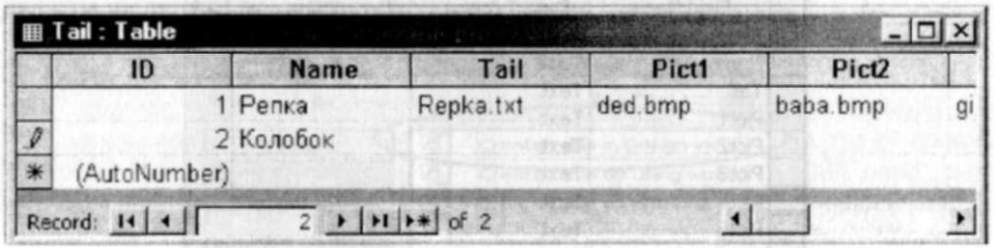


Рис. 71

Создайте текстовый и графический файлы для следующей сказки или рассказа. Откройте файл **Tail.mdb**, щелкните мышкой дважды по названию таблицы и добавьте в открывшуюся таблицу еще одну запись. Эта запись будет содержать информацию о второй сказке (ее название, имя текстового файла, имена графических файлов). Закройте программу MS Access и приступайте к созданию связи проекта Visual Basic с базой данных. Этот процесс описан в следующем подразделе.

5. Связывание проекта с базой данных. Объект управления Data

Для того чтобы программа, написанная на языке Visual Basic, могла получить информацию из внешней базы данных, можно поступить следующим образом:

- разместить на форме специальный объект управления **Data** и задать значения его свойств **Connect**, **DatabaseName**, **RecordsetType** и **RecordSource**;
- разместить на форме объекты управления, чувствительные к данным (например, TextBox или Label), и связать их с полями базы данных, задав значения свойств **DataSource** и **DataField**.

Объект **Data** практически не требует программного кода. Он позволяет автоматически просматривать записи базы данных. Рассмотрим более подробно объект **Data**, его свойства, события и методы.

А. Объект управления **Data**

Назначение: обеспечивает доступ к данным, хранящимся в базе данных. Позволяет, перемещаясь от одной записи базы данных к другой, просматривать и редактировать содержимое записей в информационно зависимых (чувствительных к данным) объектах управления.

Пиктограмма: 

Свойства: **Connect** – указывает тип базы данных. Значение свойства выбирается из выпадающего списка в меню свойств (**Properties**). В зависимости от программы, с помощью которой создана база данных, выбирают свойство **Connect**:

MS Access 2000	– Access 2000;
MS Access 95, MS Access 97	– Access;
MS Excel 5.0 MS Excel 95	– Excel 5.0;
MS Excel 97	– Excel 8.0;

DatabaseName – полное имя файла, содержащего базу данных;

RecordsetType – должно иметь значение 0 – Table для баз данных, созданных с помощью программы MS Excel, 1 – Dynaset для баз данных, созданных с помощью программы MS Access;

RecordSource – имя таблицы базы данных, с которой связывается объект **Data**;

RecordSet – позволяет использовать объект **RecordSet**. Этот объект создается объектом **Data** сразу после запуска программы (раньше, чем наступает событие **Form_Load**) и после выполнения метода **Refresh** объекта **Data**.

Кроме перечисленных, объект **Data** имеет множество свойств, известных нам по другим объектам управления: Name, BackColor, ForeColor, Visible, Caption, Font, Index, Height, Width и так далее.

Б. Информационно зависимые объекты управления

Обычные объекты управления, такие как **Label**, **TexBox** и некоторые другие, становятся чувствительными к данным, если они связываются с объектом **Data** и настраиваются на конкретное поле таблицы базы данных.

Для того чтобы связать чувствительный к данным объект управления с базой данных, надо задать следующие свойства:

- **DataSource** – значением является имя объекта Data, связанного с нужной базой данных;
- **DataField** – значением является имя поля базы данных, то есть имя колонки таблицы базы данных.

B. Связывание проекта «Сказки для малышей» с базой данных **Tail.mdb**

На форме **FrmChoose** создайте объект **Data** и в меню **Properties** задайте его свойствам значения, указанные в табл. 2.

Таблица 2

Имя свойства	Значение
Name	Data1
Caption	Выбери сказку
Connect	Access 2000 или Access (в зависимости от используемой версии программы MS Access)
DatabaseName	Полное имя файла базы данных, выбирается в диалоговом окне
RecordsetType	1 – Dynaset
RecordSource	Tail – имя таблицы базы данных, выбирается из выпадающего списка

На этой же форме создайте все объекты **Label**, кроме объекта **Label1** (см. рис. 54), и в меню **Properties** задайте свойствам этих объектов значения, указанные в табл. 3.

Таблица 3

Имя объекта Label или элемента массива LblPict	Значение свойства DataSource (выбирается из выпадающего списка)	Значение свойства DataField (выбирается из выпадающего списка)
LblName	Data1	Name
LblText	Data1	Tail
LblPict(0)	Data1	Pict1
LblPict(1)	Data1	Pict2
LblPict(2)	Data1	Pict3
LblPict(3)	Data1	Pict4
LblPict(4)	Data1	Pict5
LblPict(5)	Data1	Pict6
LblPict(6)	Data1	Pict7
LblPict(7)	Data1	Pict8
LblPict(8)	Data1	Pict9

Важно для каждого объекта сначала задавать свойство **DataSource** и только после этого свойство **DataField.**

Форма **FrmChoose** примет вид, показанный на рис. 72.

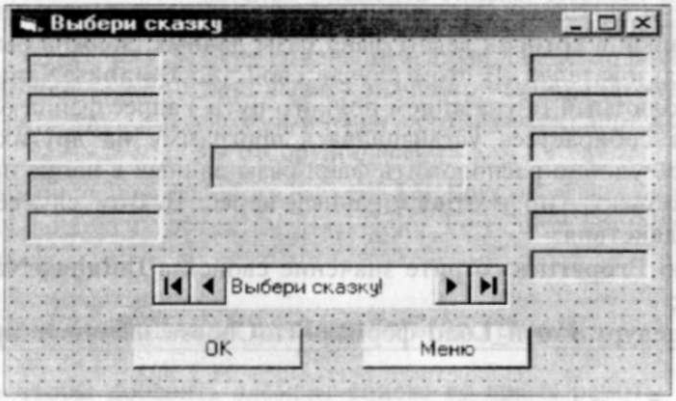


Рис. 72

После старта программы в объектах **Label** автоматически появится информация из соответствующих полей базы данных (рис. 73). Если нажимать на стрелки объекта **Data**, автоматически осуществится переход к следующей записи базы данных, то есть к следующей строке таблицы **Tail**, и информация в объектах **Label** поменяется. Осталось только сделать невидимыми (свойство **Visible** равно **False**) объекты **Label**, обведенные на рисунке овалами, и формой можно пользоваться.

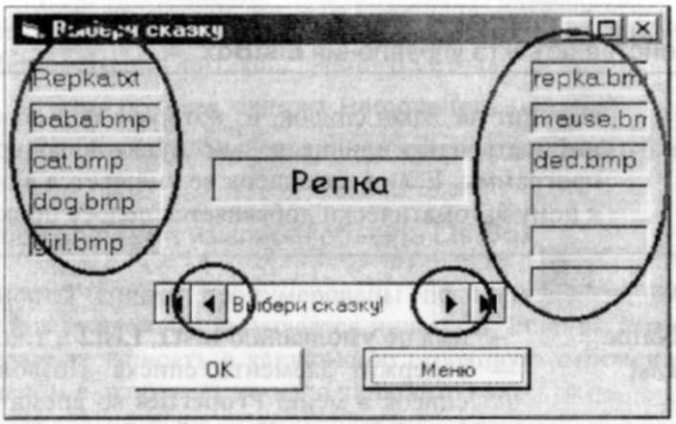


Рис. 73

Прежде чем запускать программу, не забудьте изменить свойство **Startup Object** проекта.

Г. Абсолютный и относительный адрес файла Tail.mdb

Если вы работаете в локальной сети и хотите, чтобы программа «Сказки для малышей» использовалась разными пользователями на разных рабочих станциях, папка, в которой лежит файл базы данных, должна быть доступна для всех пользователей. В этом случае свойство **DatabaseName** может содержать абсолютный (с указанием полного пути) адрес файла базы данных. Если же вы собираетесь устанавливать программу на других локальных компьютерах, удобно расположить файл базы данных в папке проекта и использовать объект **App** и относительный адрес. В этом случае выполните следующие действия:

- в меню **Properties** сотрите значение свойства **DatabaseName** объекта **Data1**;
- в процедуру **Form_Load** формы **FrmChoose** запишите программный код:

```
Private Sub Form_Load()
Data1.DatabaseName = App.Path + "\tails.mdb"
Data1.Refresh
End Sub
```

Первая строка кода присваивает значение свойству **DatabaseName**, а вторая «открывает» базу данных.

6. Объект управления ListBox и его использование

А. Свойства объекта управления ListBox

Назначение: выводит на экран список, из которого пользователь может выбирать один или несколько пунктов во время работы программы. Если весь список не умещается в окне **ListBox**, к нему автоматически добавляется полоса прокрутки.

Пиктограмма: 

Свойства: **Name** – имя по умолчанию **List1**, **List2** и т. д.;

List – содержит элементы списка. Позволяет набрать список в меню **Properties** во время конструирования формы. Каждый элемент списка надо набирать с новой строки. Переход на следующую строку осуществляется при нажатии комбинации клавиш Ctrl + Enter;

ListCount – целое число, равное количеству элементов в списке;

ListIndex – индекс (порядковый номер) выбранного пользователем элемента списка. Первый элемент списка имеет индекс равный 0. Если никакой элемент списка не выбран, свойство равно –1.

Sorted – принимает значение **True** (элементы списка автоматически сортируются) или **False** (список не отсортирован).

Кроме перечисленных свойств, объект **ListBoix** обладает рядом свойств, знакомых нам по другим объектам: BackColor, ForeColor, FontName, FontSize и ряду других.

Б. Изменение списка во время работы программы

Для того чтобы добавить элемент списка во время работы программы, используется метод **AddItem**. Синтаксис (формат) данного метода:

<имя объекта ListBox>.**AddItem** <элемент списка>, [<индекс>]

где элемент списка – строковая константа, переменная или выражение, определяющие строку текста, добавляемого в список;

индекс – целое число, определяющее, в какое место списка будет добавлен новый элемент. Параметр не обязательный. Если он опущен, элемент добавляется в конец списка (свойство **Sorted** равно **False**).

Для удаления элемента из списка во время работы программы используется метод **RemoveItem**. Синтаксис данного метода:

<имя объекта ListBox>.**RemoveItem** <индекс>

где индекс – целое число, определяющее индекс удаляемого элемента.

В. Выбор элемента из списка объекта ListBox

Все элементы списка пронумерованы по порядку, начиная с нуля. Порядковый номер элемента называется индексом. Если вы хотите во время работы программы записать в какую-либо строковую переменную элемент списка с номером n, в программу надо включить код:

S = <имя объекта ListBox>.**List (n)**

Когда пользователь программы щелкает мышкой по элементу списка, свойство **ListIndex** принимает значение выбранного элемента.

Выражение <имя объекта ListBox>.List (<объект>.ListIndex) позволяет использовать в программе элемент списка, выбранный пользователем. Это выражение следует записать в процедуре обработки события **Click** объекта **ListBox**.

Приведем пример использования свойства **List**. На форме расположен объект **ListBox** с именем **List1** и два объекта **Label**. На рис. 74 показан вид формы во время проектирования, на рис. 75 – сразу после запуска программы, а на рис. 76 – после того, как пользователь щелкнул по элементу списка.

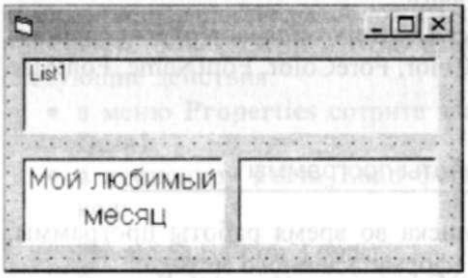


Рис. 74

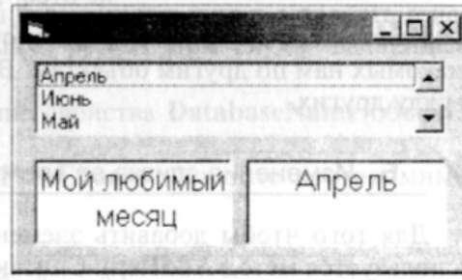


Рис. 75

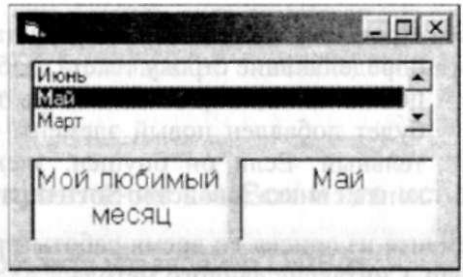


Рис. 76

Программный код примера состоит из двух процедур – **Form_Load** и **List1_Click**. В процедуре **Form_Load** в список добавляется шесть элементов. Свойство **Sorted** объекта **ListBox** равно **True**. Поэтому названия месяцев в списке (рис. 75) расположены по алфавиту. Порядок их в списке не соответствует порядку следования значений в методе **AddItem**.

Полоса прокрутки появилась в объекте **ListBox** автоматически, когда элементы списка перестали помещаться в окне объекта.

```
Private Sub Firm_Load ()
  List1.AddItem "Январь"
  List1.AddItem "Февраль"
  List1.AddItem "Март"
  List1.AddItem "Май"
  List1.AddItem "Июнь"
```

```
Label2.Caption = List1.List(0)    "первый элемент списка имеет индекс 0.
                                  "Поэтому в объекте Label2 появилось
                                  "слово «Апрель»
End Sub
Private Sub List1_Click ()
Label2.Caption = List1.List (List1.ListIndex)
End Sub
```

Г. Заполнение списка информацией из базы данных

Объект **RecordSet**

Мы уже знаем, что если на форме есть объект **Data**, доступ к записям базы данных осуществляется автоматически и переход от одной записи к другой не требует программного кода. Но если объект **Data** не виден (свойство **Visible** равно **False**), надо записать программный код, который позволит переходить от одной записи к другой. В этом коде будем использовать специальный объект **RecordSet**.

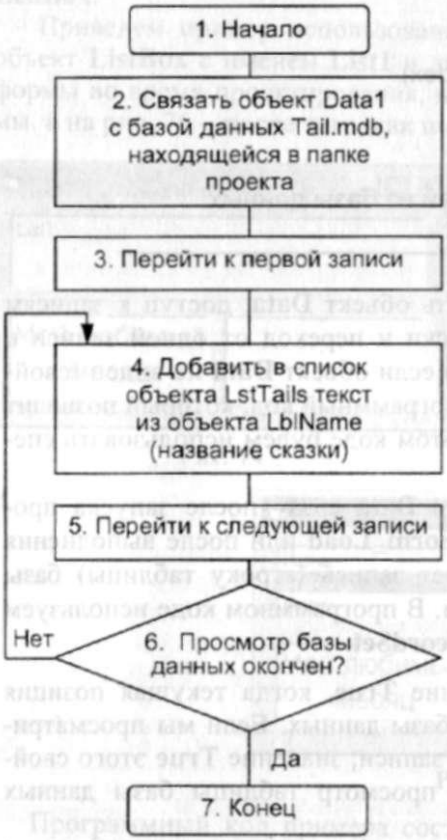
Объект **RecordSet** создается объектом **Data** сразу после запуска программы, еще до выполнения процедуры **Form_Load** или после выполнения метода **Refresh**. Этот объект представляет запись (строку таблицы) базы данных и позволяет оперировать данными. В программном коде используем следующие свойства и методы объекта **RecordSet**:

- свойство **EOF** – принимает значение **True**, когда текущая позиция находится после последней записи базы данных. Если мы просматривали базу данных начиная с первой записи, значение **True** этого свойства является сигналом того, что просмотр таблицы базы данных окончен;
- метод **MoveFirst** – делает текущей первую запись базы данных. В информационно зависимых объектах управления появляется информация из первой строки таблицы базы данных;
- метод **MoveNext** – делает текущей следующую по порядку запись базы данных.

*Процедура **Form_Load** формы **FrmChoose***

Запишем в виде блок-схемы алгоритм процедуры **Form_Load** формы **FrmChoose** (рис. 77). Эта процедура просматривает таблицу базы данных **Tail.mdb** и добавляет в список объекта **LstTails** названия сказок. Справа от блок-схемы приведен программный код, реализующий ее алгоритм.

Блок-схема и программный код процедуры **Form_Lorm** формы **FrmChoose**




```
Private Sub Form_Load ()
Data1.DatabaseName=App.Path+"\tail.mdb"
Data1.Refresh
Data1.Recordset.MoveFirst
Do
LstTails.AddItem LblName
Data1.Recordset.MoveNext
Loop Until Data1.Recordset.EOF
End Sub
```

Рис. 77

7. Процедура LstTails_Click ()

Программный код процедуры **LstTails_Click**() перебирает записи базы данных до тех пор, пока запись, содержащая выбранную сказку, не станет текущей. При этом информация о сказке появится в информационно зависимых объектах.

В программном коде используется логическая переменная с именем **Find**. Перед началом перебора записей базы данных эта переменная получает значение **False**. Когда найдена строка, соответствующая выбранной поль-

зователем сказке, эта переменная получает значение **True**. Цикл **Do – Loop** продолжается до тех пор, пока переменная **find** не станет равна **True**.

```
Private Sub LstTails_Click ()
Dim find as Boolean
Data1.Recordset.MoveFirst
find = False
Do
        If LstTails.List (LstTails.ListIndex) <> LblName Then
        Data1.Recordset.MoveNext
        Else
        find = True
        End If
Loop Until find
End Sub
```

Мы уверены, что в данной задаче в базе данных обязательно найдется сказка, выбранная пользователем. Если такой уверенности нет, в операторе **Loop** записывают сложное условие, которое обеспечивает выход из цикла просмотра базы данных в том случае, если найдена искомая запись или прочитана последняя строка таблицы базы банных. В этом случае оператор **Loop** будет выглядеть так:

```
Loop Until Data1.Recordset.EOF Or find
```

8. Совершенствование формы FrmRead

На первом этапе работы над проектом мы создали форму **FrmRead**, которая загружает из текстового файла всегда одну и ту же сказку. В объекты **PctPicture(0)**, **PctPicture(1)**, ..., **PctPicture(9)** во время проектирования были загружены рисунки к этой сказке. Усовершенствуем программный код так, чтобы в форму **FrmRead** загружалась сказка, выбранная пользователем в форме **FrmChoose**. Имена файлов, содержащих рисунки для выбранной сказки, прочитаны из базы данных и записаны в объекты к **LblPict(0)**, **LblPict(1)**, ..., **LblPict(9)** формы **FrmChoose**. Организуем цикл на десять повторений для чтения рисунков из файлов. Программный код этого цикла следует добавить в процедуру **Form_Activate** формы **FrmRead**.

```
For k = 0 To 9
If Form1.LblPict (k) <> "" Then
PctPicture (k). Picture =LoadPicture (App.Path +"\pictures\"+FrmChoose.LblPict (k))
Else
PctPicture (k). Picture = LoadPicture ()
End If
Next
```

Оператор **Open**, открывающий текстовый файл, изменим так, чтобы имя текстового файла выбиралось из объекта **LblText** формы **FrmChoose**. Перед этим оператором добавим оператор **If**, который проверяет, содержит ли свойство **Caption** объекта **LblText** текстовую строку. Если нет, работа процедуры прерывается с помощью оператора **Exit Sub**. Чтение текста из файла не производится.

```
PctTail.Cls
If Form1.LblText = "" Then Exit Sub
Open App.Path + "\tails\" + Form1.LblText For Input As #1
```

9. Взаимосвязь форм проекта

Возможности перехода из одной формы проекта в другую показаны на рис. 78. Переход программируется с помощью методов **Show** и **Hide**. Стартовой формой является форма – меню **FrmMenu**.

Запишите все необходимые команды, сделайте форму **FrmMenu** стартовой и, запустив программу, тщательно проверьте, все ли переходы работают. В случае сбоя найдите и исправьте ошибки. Не забывайте перед каждым запуском проекта сохранять изменения.

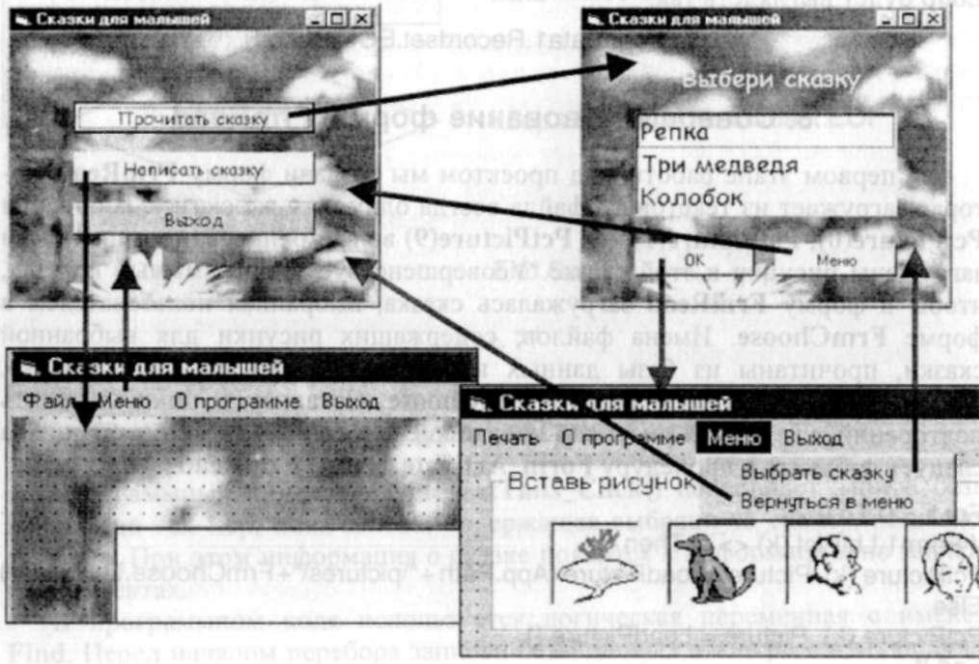


Рис. 78

10. Создание новой сказки. Запись информации в текстовый файл

В форме **FrmWrite** текст сказки набирается в текстовом окне **TxtTail**. Каждый раз, когда в сказке предполагается картинка, нажмите клавишу **Enter**, чтобы печатать продолжение сказки с новой строки. Запись в текстовый файл новой сказки осуществляется при нажатии команды меню «Файл→Сохранить».

Прежде чем записывать информацию в файл последовательного доступа, его надо открыть для чтения (режим доступа **OUTPUT** или **APPEND**). После этого следует использовать операторы **Print #** или **Write #.**

А. Оператор Write

Формат оператора:

Write #<номер файла>, [<список вывода>]

где <список вывода> – перечень строковых или численных выражений, значения которых записываются в файл. Выражения в списке разделяются запятой.

Оператор **Write #** после каждого элемента списка вставляет в файл запятую. Строка текста записывается в кавычках. После того, как записан последний элемент из списка вывода, в файл вставляется комбинация символов «конец строки» и «перевод каретки». Если список вывода опущен (запятая после номера файла при этом сохраняется), в файл будет записана пустая строка. Файл, созданный с помощью оператора **Write #**, читается обычно оператором **Input #**. Приведем пример использования оператора **Write #**. В результате выполнения приведенного программного кода в текущей папке будет создан файл с именем **write.txt**.

```
Dim a As String
Dim k As Integer
Private Sub Command1_Click ()
a = "–возводится в степень"
Open "write.txt" For Append As #1
Write #1, "Таблица квадратов и кубов"
For k = –3 To 3 Step 1.5
Write #1, k, a, k ^ 2, k ^ 3
Next
Close #1
End Sub
```

Первый оператор **Write#** имеет в списке вывода один элемент – строковую константу. Она записалась в файл в кавычках. Второй оператор **Write#** в списке имеет четыре элемента – численная переменная, строковая переменная и два арифметических выражения. В файл записаны значения переменных и арифметических выражений. Между ними записаны запятые. Строка текста записана в кавычках.

Текстовый файл write.txt:

```
"Таблица квадратов и кубов"
-3,"-возводится в степень",9,-27
-1.5,"-возводится в степень",2.25,-3.375
0,"-возводится в степень",0,0
1.5,"-возводится в степень",2.25,3.375
3,"-возводится в степень",9,27
```

Б. Оператор **Print #**

Формат оператора:

Print #<номер файла>, [<список вывода>]

где <список вывода> – перечень строковых или численных выражений, значения которых записываются в файл. Выражения в списке разделяются запятой, пробелами или точкой с запятой.

Оператор **Print #** после каждого элемента списка вставляет в файл один пробел (выражения в списке вывода разделены точкой с запятой) или несколько пробелов (выражения в списке вывода разделены запятой). После того, как записан последний элемент из списка вывода, в файл вставляется комбинация символов «конец строки» и «перевод каретки».

Если список вывода опущен (запятая после номера файла при этом сохраняется), в файл будет записана пустая строка.

Файл, созданный с помощью оператора **Print #**, читается обычно оператором **Line Input #**. Приведем пример использования оператора **Print #**. В результате выполнения приведенного программного кода в текущей папке будет создан файл с именем **print.txt**.

```
Dim a As String
Dim k As Integer
Private Sub Command1_Click()
a = "-возводится в степень"
Open "print.txt" For Append As #1
```

```
Print #1, "Таблица квадратов и кубов"
Print #1,                    "в файл вставляется пустая строка
For k = –3 To 3 Step 1.5
Print #1, k, a, k ^ 2, k ^ 3    "разделителем в списке служит запятая. В файл
                                "между записями вставляется по несколько пробелов
Next
Print #1,
For k = –3 To 3 Step 1.5
Print #1, k; a; k ^ 2; k ^ 3    "разделителем в списке служит точка с запятой.
                                "В файл между записями вставляется по одному
                                "пробелу
Next
Close #1
End Sub
```

Текстовый файл **print.txt**:

«Таблица квадратов и кубов»

```
 –3     –возводится в степень     9     –27
 –1     –возводится в степень     1     –1
  1     –возводится в степень     1      1
  3     –возводится в степень     9      27

 –3 –возводится в степень  9  –27
 –1 –возводится в степень  1  –1
  1 –возводится в степень  1   1
  3 –возводится в степень  9   27
```

Основное различие операторов **Print#** и **Write#** состоит в том, что они вставляют разные разделители между записями – пробелы или запятые. Файлы, созданные с помощью оператора **Write#**, следует читать оператором **Input#**. Файлы, созданные оператором **Print#**, читаются обычно оператором **Line Input#**.

В нашем проекте для записи текста сказки в файл использован оператор **Print#**. Один оператор записывает в файл сразу все содержимое текстового окна **TxtTail**.

✓ B. Создание файла в форме FrmWrite

Текст сказки набирается в текстовом окне **TxtTail**. Каждый раз, когда в сказке предполагается картинка, нажимайте клавишу **Enter**, чтобы печатать продолжение сказки с новой строки. Запись на диск новой сказки

осуществляется при нажатии команды меню «**Файл→Сохранить**». Для этого используется оператор **Print #**. Программный код команды меню **MnuSave**:

```
Private Sub MnuSave_Click()
ComD.CancelError = True
On Error GoTo ErrHandler
ComD.Flags = cdlOFNOverwritePrompt Or cdlOFNHideReadOnly
ComD.Filter = "Text files (*.txt)|*.txt"
ComD.ShowSave
Open ComD.FileName For Output As #1
Print #1, TxtTail
Close #1
ErrHandler:
End Sub
```

11. Программирование пользовательского меню. Функция MsgBox

Программный код, реализующий такие команды меню, как «Выход», «Файл → Создать новый», «Меню», сам по себе очень прост. Проблема возникает в том случае, если мы хотим обезопасить пользователя нашей программы от случайной утраты набранного в форме **FrmWrite** текста. Для этого используем окно диалога, которое выводится на экран с помощью функции **MsgBox**.

A. Функция MsgBox

Функция **MsgBox** выводит на экран диалоговое окно, ждет, когда пользователь нажмет кнопку на диалоговом окне, и возвращает число, значение которого зависит от того, какую кнопку нажал пользователь. Диалоговое окно (рис. 79) содержит заголовок окна, текст сообщения, иконку, обозначающую тип сообщения, и кнопки. Иконка и набор кнопок зависят от аргументов функции.

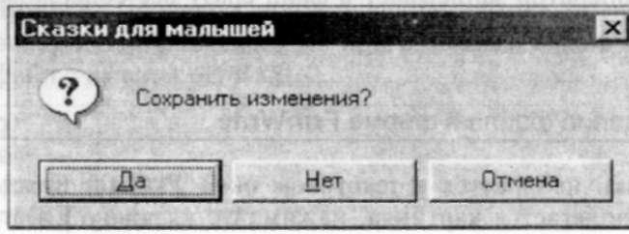


Рис. 79

Формат функции:

Msg = MsgBox(<prompt>[, <buttons>] [, <title>])

где <prompt> – текстовая константа или выражение, значение которого появляется в диалоговом окне; текстовая константа записывается в кавычках. Длина сообщения примерно равна 1024 символа. Если сообщение занимает несколько строк, в строковом выражении можно использовать символ возврата каретки (**Chr**(13)) для разделения строк;

<buttons> – численное выражение или константа, определяющие набор кнопок и икону в диалоговом окне. Число является суммой величин, которые выбираются из приведенных табл. 4 и 5. Параметр не является обязательным. Если он пропущен, диалоговое окно имеет одну кнопку **OK** и не имеет иконы;

<title> – строковое выражение, значение которого записывается в строке заголовка диалогового окна;

Msg – численная переменная, которая получает значение в результате выполнения функции MsgBox. Значение зависит от того, на какую кнопку нажал пользователь программы. Переменная должна быть объявлена как обычно.

Таблица 4

Значение слагаемого аргумента Buttons, определяющего набор кнопок в диалоговом окне

Имя константы	Значение слагаемого	Описание кнопок
vbOKOnly	0	Одна кнопка **OK**
vbOKCancel	1	Две кнопки: **OK** и **Cancel**
vbAbortRetryIgnore	2	Три кнопки: **Abort**, **Retry** и **Ignore**
vbYesNoCancel	3	Три кнопки: **Yes**, **No** и **Cancel**
vbYesNo	4	Две кнопки: **Yes** и **No**
vbRetryCancel	5	Две кнопки: **Retry** и **Cancel**

Таблица 5

Значение слагаемого аргумента Buttons, определяющего вид иконы

Имя константы	Значение слагаемого	Иконка
vbCritical	16	
vbQuestion	32	
vbExclamation	48	
vbInformation	64	

Если вы хотите вывести диалоговое окно с тремя кнопками **Yes** (Да), **No** (Нет) и **Cancel** (Отмена) и иконкой со знаком вопроса, значение аргумента **buttons** будет равно сумме 3 + 32 = 35. То есть вместо выражения **[, <buttons>]** в функции **MsgBox** надо будет написать **,35**. Приведем несколько примеров, демонстрирующих, как внешний вид диалогового окна зависит от значения аргументов функции **MsgBox**.

Пример 11

```
Msg = MsgBox ("Введенное значение аргумента" & Chr(13) & " недопустимо.", 16, "Пример 11").
```

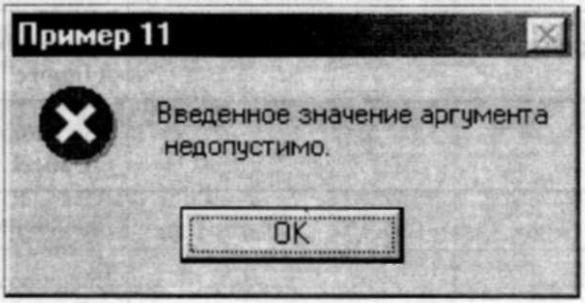


Рис. 80

Пример 12

```
Msg = MsgBox("Вы действительно хотите выйти из программы?", 33, "Пример 12")
```

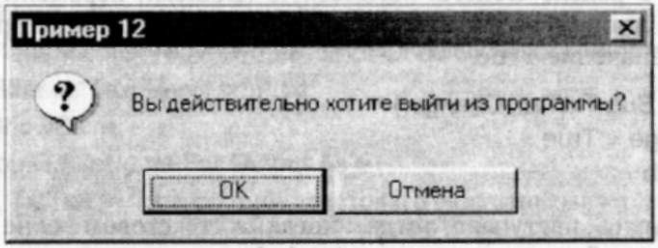


Рис. 81

Для того чтобы после нажатия на кнопку диалогового окна произошли какие–нибудь действия, надо написать программный код, который проверяет значение переменной **Msg** и в зависимости от этого значения производит те или иные действия. Значения, которые получает переменная **Msg**, зависят от того, какую кнопку диалогового окна нажал пользователь. Они записаны в табл. 6 в колонке «Значение функции». Для того чтобы не запоминать, какие значения принимает переменная, можно использовать встроенные константы. Имена констант приведены в первом столбце таблицы. Пример использования констант приведен в подразделе 11.Б.

Таблица 6

Значения, возвращаемые функцией MsgBox

Имя константы	Значение функции	Описание
vbOK	1	Нажата кнопка «OK»
vbCancel	2	Нажата кнопка «Cancel» (Отмена)
vbAbort	3	Нажата кнопка «Abort» (Стоп)
vbRetry	4	Нажата кнопка «Retry» (Повтор)
vbIgnore	5	Нажата кнопка «Ignore» (Пропустить)
vbYes	6	Нажата кнопка «Yes» (Да)
vbNo	7	Нажата кнопка «No» (Нет)

Для того чтобы после нажатия на кнопку диалогового окна произошли какие-нибудь действия, надо написать программный код, который будет проверять значение переменной **Msg** и в зависимости от этого значения выполнит те или иные действия.

Б. Программирование команды «Выход» пользовательского меню формы FrmWrite

Объявим в раделе **General** логическую (Boolean) переменную с именем **txtchange**. В процедуре обработки события **TxtTail_Change** присвоим этой переменной значение **True**.

```
Private Sub TxtTail_Change()
txtchange = True
End Sub
```

Это событие наступает тогда, когда в текстовом окне изменилась надпись. Таким образом, переменная **txtchahge** получает значение **True**, если пользователь редактировал содержимое текстового окна **TxtTail**.

В процедуре **MnuExit_Click**, реализующей команду пользовательского меню «Выход», запишем оператор **If**, который проверяет, чему равна переменная **txtchahge.** Если она равна **False**, программа закрывается, если **True** – на экран выводится диалоговое окно (рис. 82).

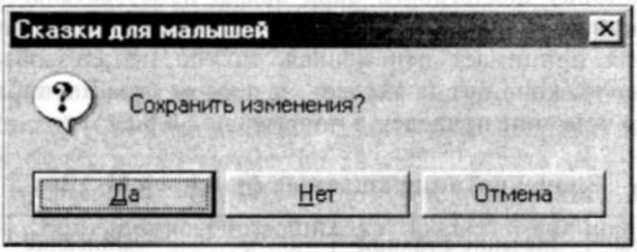


Рис. 82

Дальнейшее выполнение процедуры будет зависеть от того, какую кнопку нажмет пользователь программы. Если нажата кнопка **Cancle** (Отмена), ничего не происходит. Если нажата кнопка **No** (Нет), программа закрывается. Если же нажата кнопка **Yes** (Да), производится сохранение в текстовый файл значения свойства **Text** текстового окна **TxtTail**, а затем программа закрывается.

```
Private Sub MnuExit_Click()
Dim msg As Byte
If txtchange = True Then
        msg = MsgBox("Сохранить изменения?", 35, "Сказки для малышей")
Else
    End
End If
```

```
Select Case msg
Case 6      "переменная msg равна 6, следовательно нажата кнопка Yes (Да)
    ComD.CancelError = True
    On Error GoTo ErrHandler
    ComD.Flags = cdlOFNOverwritePrompt Or cdlOFNHideReadOnly
    ComD.Filter = "Text files(*.txt)|*.txt"
    ComD.ShowSave
    Open ComD.FileName For Output As #1
    Print #1, TxtTail     "содержимое текстового окна записывается в
                          "последовательный файл, открытый под номером 1
    Close #1
    End
Case 7          "переменная msg равна 7, следовательно нажата кнопка No (Нет)
    End         "работа программы прекращается
End Select
ErrHandler:
End Sub
```

Обратите внимание, переменная **msg** объявлена оператором **Dim** внутри процедуры. Благодаря этому значение переменной приравнивается 0 перед тем, как выполнять программный код процедуры. Эту переменную надо объявить в каждой из процедур, в которых она используется.

В. Программирование команды «Файл → Создать новый» пользовательского меню формы FrmWrite

Код процедуры **MnuNew_Click**, реализующий команду «Файл →Создать новый» пользовательского меню отличается от кода **MnuExit_Click** только тем, что вместо оператора **End**, закрывающего программу, используются два оператора присваивания, которые очищают текстовое окно **TxtTail** и присваивают значение **False** переменной **txtchahge**. Таким образом для создания процедуры **MnuNew_Click** скопируйте программный код процедуры **MnuExit_Click** и везде замените оператор **End** на два оператора присваивания:

```
TxtTail.Text = ""
txtchange = False
```

Г. Программирование команды «Меню» пользовательского меню формы FrmWrite

Команду «Меню» реализует процедура **MnuMenu_Click**(). Программный код этой процедуры запишите по аналогии, пользуясь блок-схемой (рис. 83).

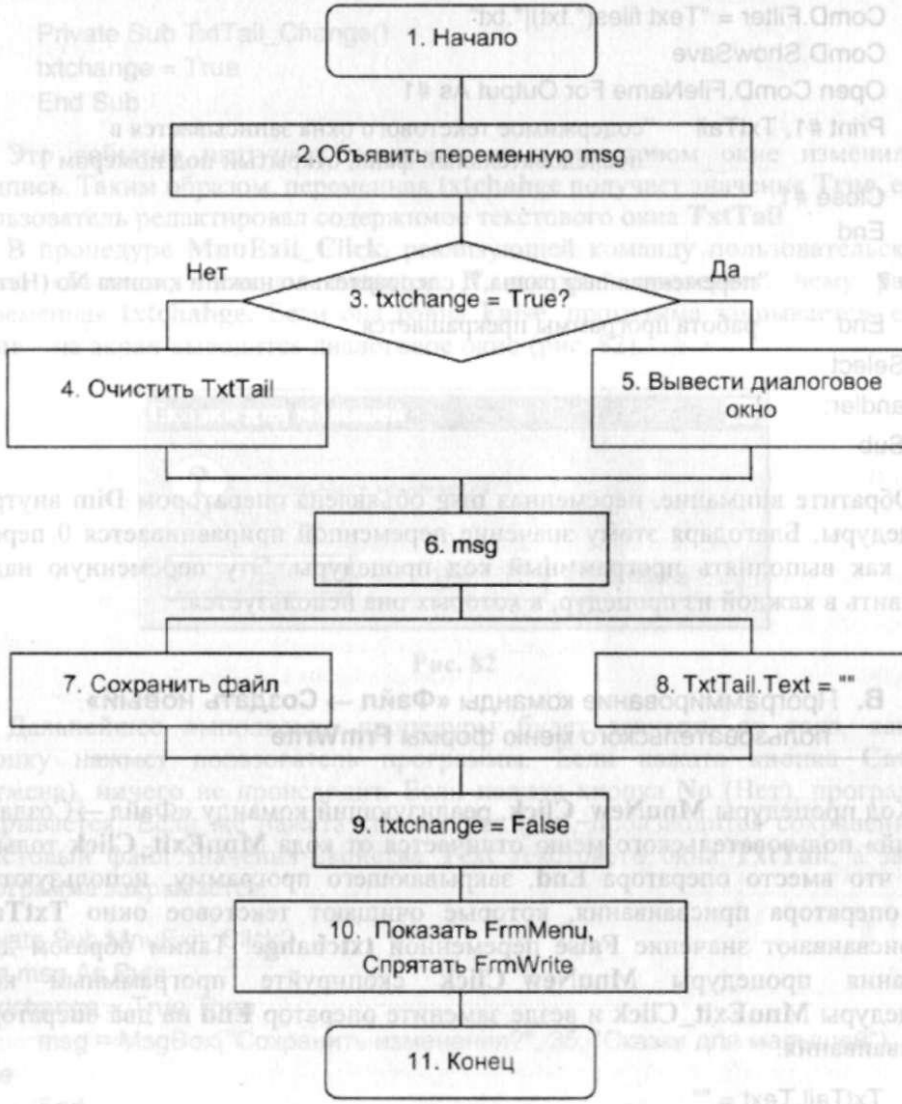


Рис. 83

Д. Программирование команды «Файл → Открыть» пользовательского меню формы FrmWrite

В приведенном программном коде процедуры **MnuOpen**, реализующей команду меню *«Файл → Открыть»*, заполните имеющиеся пропуски.

```
Private Sub MnuOpen_Click()
Dim msg As Byte
If _____________________ Then
msg = MsgBox("Сохранить изменения?", 35, "Сказки для малышей")
Else
msg = 7
End If

Select Case _____________
Case 6
   ComD.CancelError = True
   On Error GoTo ErrHandler
   ComD.Flags = &H2 Or &H4
   ComD.Filter = "Text files(*.txt)|*.txt"
   ComD.ShowSave
   Open ComD.FileName For Output As #1
   _____________________
   _____________________
   ComD.Flags = &H1000 Or &H4
   ComD.Filter = "Text files(*.txt)|*.txt"
   ComD.ShowOpen
   Open ComD.FileName For Input As #1
   TxtTail.Text = ""
   Do
      Line Input #1, tailstring
      TxtTail.Text = TxtTail.Text _
         + tailstring + Chr(13) + Chr(10)
      Loop Until EOF(1)
   Close #1
   txtchahge = False
Case 7
   ComD.CancelError = True
   On Error GoTo ErrHandler
   ComD.Flags = &H1000 Or &H4
   ComD.Filter = "Text files(*.txt)|*.txt"
   ComD.ShowOpen
   Open ComD.FileName For Input As #1
   TxtTail.Text = ""
   Do
      Line Input #1, tailstring
      TxtTail.Text = TxtTail.Text _
         + tailstring + Chr(13) + Chr(10)
      Loop Until EOF(1)
   Close #1
   txtchahge = False
End _____________
ErrHandler:
End Sub
```

Предметный указатель