VISUAL BASIC 6

Глава 1. ТЕХНОЛОГИЯ ПРОГРАММИРОВАНИЯ НА VISUAL BASIC 4 В СРЕДЕ WINDOWS

1. 1. ЭКРАННЫЕ ЭЛЕМЕНТЫ

После запуска Visual Basic на экране появляются следующие окна (рис 1 1)'

- основное окно сверху экрана Miciosoft Visual Basic Project 1 (design), содержащее главное меню и панель инструментов (Toolbar);
- окно формы в центре с заголовком Form;
- окно шаблонов (palett windows) или инструментов (Toolbox) слева от формы;
- окно свойств (Properties) сверху справа;
- окно проекта (Project 1) снизу справа

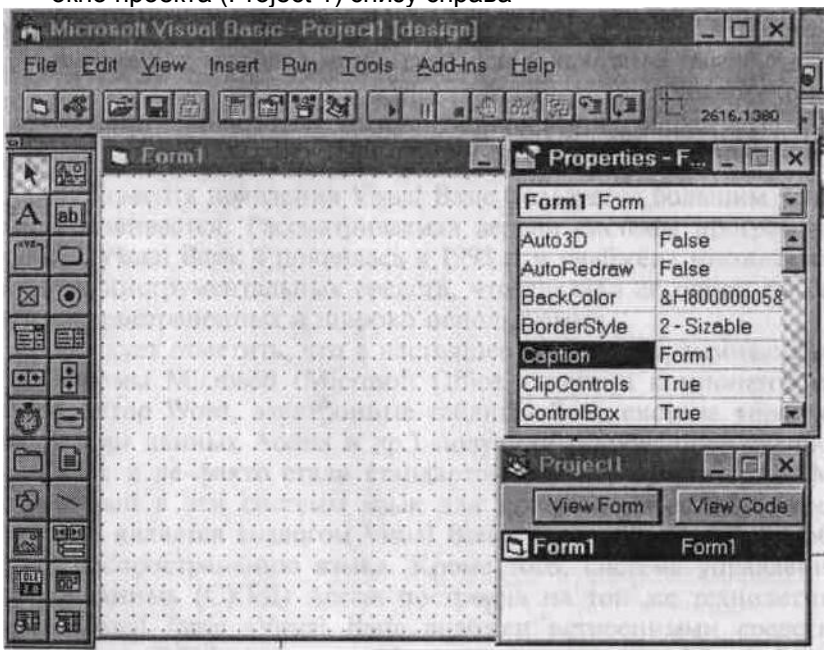


Рис 1 1

1. 1. 1. Основное окно

Главное меню основного окна содержит стандартные для Windows меню Файл (File), Правка (Edit), Вид (View) и меню собственно Visual Basic Вставка (Insert), Выполнить (Run), Tools (Средства), Add-in (Дополнения), (Help)

Файл (File) - команды для открытия, сохранения, печати и компиляции проекта Visual Basic
Правка (Edit) - команды редактирования.

Вид (View) - команды просмотра компонентов Visual Basic
Вставка (Insert) - команды для добавления в проект новых
форм и модулей

Выполнить (Run) - команды для выполнения и компиляции проекта

Tools (Средства) - команды для конфигурирования среды
программирования Visual Basic
Add-in (Дополнения) - дополнительные средства для расширения возможностей Visual Basic
(Help) - доступ к справочному руководству Панель инструментов (Toolbar) основного окна содержит кнопки-пиктограммы для быстрого вызова часто используемых команд (рис 1 2)

Рис. 1.2.

Функции пиктограмм (слева направо) следующие-создать форму (New Form);
создать модуль (New Module);
открыть проект (Open Project);
сохранить проект (Save Project),
блокировка элементов управления на форме (Lock
contro s),
редактор меню (Menu Editor), свойства (Properties Windows), просмотр объектов (Object Browser), проект (Project);

старт (Start),
прервать выполнение (Break);

поставить/убрать точку прерывания (BieakPoint), • немедленный просмотр (Instant Watch);

- вызовы (Calls);
- пошаговое выполнение (Single Step);
- выполнение по процедурам (Procedure Step).

Кнопки создания формы (New Form), создания модуля (New Module), открытия проекта (Open Project), сохранения проекта (Save Project) позволяют создать новый или открыть существующий проект (единый программный комплекс), ввести в него новые диалоговые окна (формы) и сохранить проект на диске с внесенными изменениями.

Кнопка блокировки элементов управления на форме (Lock controls) позволяет зафиксировать положение элементов управления на форме.

Кнопка редактора меню (Menu Editor) используется для создания пользовательского меню проекта и определения его свойств.

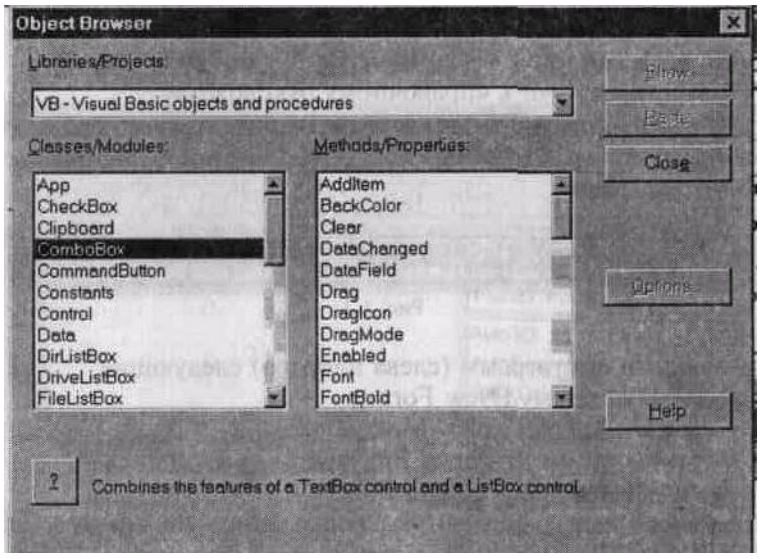


Рис 1 3

Кнопка свойств (Properties Windows) используется для вывода окна со списком свойств и их значений для формы и элементов управления.

Кнопка просмотра объектов (Object Browser) выводит окно, в котором можно для выбранного подмножества Visual Basic или

проекта получить список его компонентов или модулей и относящихся к каждому из них список методов и свойств (рис. 1.3). Для выбранных элементов списка выводится краткая аннотация и можно получить развернутую справку.

Кнопка проекта **(Project)** активизирует окно проекта. Кнопки «Старт» **(Start),** «Прервать выполнение» **(Break),** «Конец выполнения» **(End),** «Поставить/убрать точку прерывания» **(Breakpoint),** «Немедленный просмотр» **(Instant Watch),** «Вызовы» **(Calls),** «Пошаговое выполнение» **(Single Step),** «Выполнение по процедурам» **(Procedure Step)** используются при отладке программ.

Окно проекта (рис. 1.4) содержит список всех файлов, необходимых для выполнения создаваемой программы. Имя *Forml.frm* присваивается по умолчанию первой диалоговой форме проекта (это имя можно изменить или сохранить для дальнейшего использования). Этот файл содержит описание формы, элементов управления (кнопок, текстовых окон и др.) и текст программ, связанных с этими элементами.

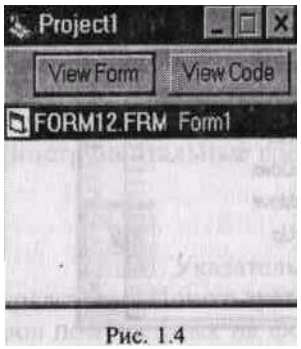


Рис. 1.4

Файл специальных средств управления имеет расширение .OCX или -BCX и содержит кнопки с мультипликацией, трехмерные средства управления и т.п. Обычно хранится в каталоге WINDOWS/SYSTEM, и его можно использовать в разных проектах.

Файл стандартных модулей имеет расширение .BAS и содержит текст программ.

Файл модулей классов имеет расширение **.CLS и** содержит определения классов.

Файл ресурсов имеет расширение .RES и содержит данные для проекта (текстовые строки, битовые образы и др.).

Файл проекта имеет расширение .VBP или .МАК и содержит пути назначения (диски и каталоги) и имена всех перечисленных файлов проекта.

Файл проекта может быть откомпилирован с целью получения исполняемого файла программы с расширением .EXE. В окне проекта имеются также две кнопки: **View Form** (Просмотр формы) **и View Code** (Просмотр кода). По умолчанию Visual Basic при выборе какого-либо файла проекта показывает форму (см. рис. 1.1с общим экраном Visual Basic).

При щелчке на кнопке **View Code** (Просмотр кода) открывается окно кода (текста программы), относящегося к рассматриваемой форме (заголовок *Forml.frm)* и задаваемым на ней диалоговым элементам. Окно (рис. 1.5) содержит раскрывающиеся окна **Object и Proc** и поле для записи программы. В первом окне находится список диалоговых элементов формы, включая саму форму **(Form).** Второе содержит список процедур обработки событий, относящихся к рассматриваемому диалоговому элементу формы (в данном случае к самой форме **- Form).**

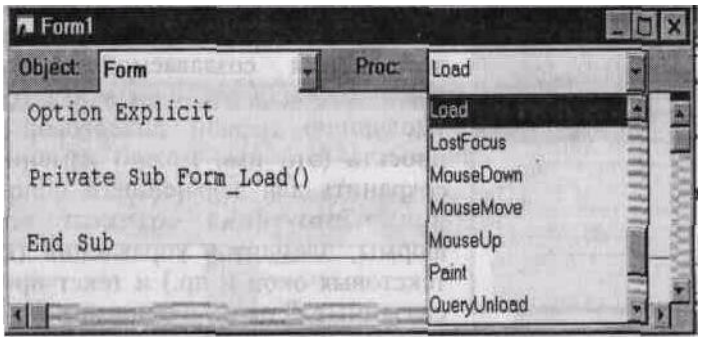


Рис 1 5

Каждому диалоговому элементу в Visual Basic поставлен в соответствие определенный набор событий (эти события перечислены в раскрывающемся меню: **Load, LostFokus** и др.), происходящих в период выполнения программы. Например, событие **Load** (Загрузка) происходит при загрузке формы, событие **Clik** (Щелчок) вызывается щелчком кнопки мыши, событие **DblClick** (Двойной щелчок) вызывается двойным щелчком кнопки мыши и т.д.

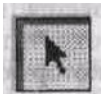
В свою очередь, событию ставится в соответствие процедура обработки события (процедура **(procedure)** в языках программирования состоит из набора операторов, выполняемых при вызове процедуры). То есть событие может вызывать программируемое изменение данных. Visual Basic содержит заготовки таких процедур в поле для записи программы, соответствующие выделенному событию (строки **Sub Form_Load () и End** определяют первый и

последний операторы процедуры обработки события **Load** (Загрузка)). В поле ввода текста программы автоматически формируется заголовок процедуры и конечный оператор. Имя процедуры формируется автоматически и состоит из двух частей, разделенных подчеркиванием: имя выделенного диалогового элемента и имя выделенного события (строки **Sub Form_Load () и End** определяют первый и последний операторы процедуры обработки события **Load (Загрузка)).**

1.1.3. Окно шаблонов или инструментов

Достоинством GUI является то, что имеется стандартный набор объектов диалога (окна, кнопки, линейки прокрутки и т.д.), позволяющий создать стандартный Windows интерфейс программируемой системы. Диалоговые элементы программной системы создаются на основе стандартных шаблонов или инструментов, которые выбираются из окна шаблонов или инструментов **(Toolbox)** и размещаются на форме.

Имеются следующие основные элементы управления (инструментальные средства):



Указатель (Pointer) не является инструментальным средством. Используется для перемещения или изменения размеров помещенных на форму элементов управления.

Изображение или картинка (Picture Box) используется для показа на форме графических объектов (статических или динамических), которые получаются с помощью графических методов.

Метка (Label) используется для вывода текста, который не должен изменяться пользователем (например, заголовок какого-либо объекта управления).

Текстовое окно (Text Box) используется для ввода, вывода и редактирования пользователем текстовой (символьной) строки информации. 

Рамка (Frame) используется для функциональной или визуальной группировки (выделения) элементов управления. Для того чтобы сгруппировать элементы управления, сначала на форме размещается рамка, а затем внутри нее размещаются элементы

Командная кнопка (Command Button) используется для ввода команды пользователем.

Флажок (Check Box) используется в тех случаях, когда пользователь должен выбрать или отметить на форме некоторое условие (да или нет). Условий, которые помечаются флажками на форме, может быть несколько и они могут выполняться одновременно (т.е. несколько флажков на форме может быть по-мечено одновременно).

Переключатель (Option Button) используется в тех случаях, когда пользователь должен выбрать или отметить на форме некоторое условие (да или нет). Условий, которые помечаются переключателями на форме, может быть несколько и только одно из них может быть выбрано (т.е. если одно условие, заданное переключателем, выбрано или отмечено, то другие переключатели на форме не помечены или не выбраны).

Комбинированный список (Combo Box) является комбинацией списка (см. ниже) и текстового окна. Используется для вывода в специальном окне (списке) нескольких строк текстовой (символьной) информации, из которых пользователь может выбрать одну или несколько, а также для ввода и редактирования пользователем текстовой (символьной) строки информации.

Список (List Box) используется для вывода в специальном окне нескольких строк текстовой (символьной) информации, из которых пользователь может выбрать одну или несколько. Если в списке содержится больше элементов, чем может одновременно быть показано в окне, возможна прокрутка списка.

Горизонтальная линейка прокрутки (Horizontal Scroll Bar) используется для быстрого движения по длинным спискам или строкам и отображения текущего положения в списке или строке и увеличения скорости прокрутки списка.

Вертикальная линейка прокрутки (Vertical Scroll Bar) аналогична горизонтальной линейке прокрутки.

Таймер (Timer) используется для задания моментов (интервалов) времени, в которых должны свершиться какие-либо события. Этот элемент управления невидим на форме во время выполнения ппиложения.

Список дисков (Drive List Box) используется для ото-
бражениями выбора имеющихся в системе дисков.

Список каталогов (Directory List Box) используется для отображения иерархического списка каталогов в пользова-тельской системе.

Список файлов (File List Box) используется для отображения списка файлов в пользовательской системе и управления ими (открытие, удаление, сохранение и др.).

Форма (Shape) используется для отображения простых фигур (прямоугольник, окружность, эллипс) на форме на этапе проектирования формы.

Линия (Line) используется для отображения линий различного вида на форме на этапе проектирования формы.

Изображение (Image) используется для отображения на форме растровых графических изображений, иконок или метафайлов. Эти изображения могут быть только декоративными и требуют меньше ресурсов компьютера, чем Picture Box

Сетка (Grid) представляет собой таблицу данных, состоящую из столбцов и строк. Положение конкретного данного определяется координатой строки и столбца, на пересечении которых оно находится

OLE 2 0 реализует технологию Microsoft OLE (object linking and embending - связь и внедрение объектов) и позволяет создавать в программе объект, содержащий данные из другой внешней программы (приложения), например из электронной таблицы Excel Технология OLE обеспечивает связь с внешним приложением (при изменении данных в источнике автоматически обновляются данные в программе на Visual Basic)

Элемент данные (Data) позволяет получить доступ к конкретной информации в базе данных

Стандартное диалоговое окно (Common Dialog) включает набор диалоговых окон, реализующих стандартные и часто используемые функции Windows (открыть, сохранить как идр)

Кроме перечисленных имеются другие элементы управления, сведения о которых можно получить во встроенной справочной системе Visual Basic и которые можно добавлять к приведенному основному списку

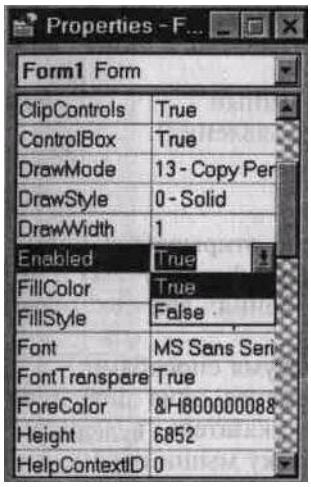
1.1.4. Окно свойств

Каждый шаблон (инструмент) и сама форма обладают набором

специфических свойств, параметры которых определяют их внешний вид и поведение при работе программы Список параметров и их значений определяются в окне свойств (рис 1 6)

Раскрывающийся список объектов в верхней части содержит имена и типы объектов (шаблонов), помещенных на форму, а также самой формы *(Form]* - имя формы, Form - тип) Изначально список содержит только форму, каждый новый шаблон, помещаемый на форму, включается в список

Для фиксированного элемента списка объектов выводится список свойств и их значений (на рис 1 6 список свойств для формы) Значение выделенного подсветкой свойства можно изменять Если значение имеет набор альтернатив, то активизируется стрелка раскрывающегося списка возможных значений. Например, свойство **Enabled** (Доступно) имеет два альтернативных параметра' *True* и *False*



В заключение отметим, что среду Visual Basic можно настроить в соответствии с предпочтениями пользователя (пункт главного меню **Tool** и далее пункты раскрывающегося меню **Environment Options.., Project Options.., Format Options...).**

Рис 16

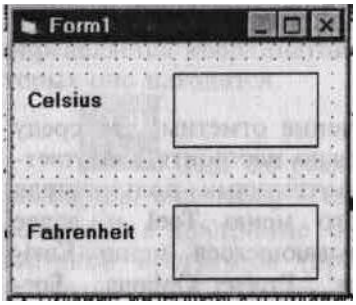
1.2. Пример разработки приложения

Создание любой программной системы (проекта) на Visuai Basic состоит из следующих этапов»

- создания интерфейса (создания окна диалога и размещения в нем элементов управления),
- установки параметров (свойств) элементов управления;
- ввода текста программ и их отладки;
- сохранения проекта в удобном для последующего использования виде

Рассмотрим реализацию этих этапов на примере простого проекта для преобразования значений температуры по Цельсию в значения по Фаренгейту и наоборот Преобразование производится соответственно по формулам F=(C*9/5)+32, C=(F-32)*5/9

Пользовательский интерфейс показан на рис 1 7



При вводе значения температуры по Цельсию (окно Celsius) и нажатии клавиши Ввода (Enter) в окне Fahreheit выводится значение по Фаренгейту.

При аналогичном вводе значения по Фаренгейту выводится значение по Цельсию. Проект включает :

- формы (фон, Windows, диалоговые окна);
- элементы управления ((графические объекты, помещенные на форму);
- программный код (процедуры и объявления).

Рис. 1.7

При запуске Visual Basic по умолчанию открывается новый проект с именем *Project 1* и относящаяся к нему форма *Formi* (рис. 1.8). На форме разместим элементы управления: два текстовых окна и две метки, используя окно шаблонов.

Задать элементы управления можно двумя способами:

1. Щелкните мышью на нужном элементе окна шаблонов (стрелка превращается в»+»), перенесите указатель в верхний левый угол формы, нажав и не отпуская кнопку мыши переместите указатель вправо и вниз формы, отпустите кнопку мыши. На форме в верхнем левом углу создается выбранный элемент управления соответствующего перемещению размера.
2. Дважды щелкнуть мышью на нужном элементе окна шаблонов. При этом создается элемент управления с размерами по умолчанию в центре формы.
   Выберем указанными способами два текстовых окна и разместим их на форме (рис. 1.9).

   Активный в данный момент элемент помечен маркерами (щелчком мыши элемент делается активным) и его можно перемещать мышью и изменять его размеры. Это позволяет отредактировать размер и положение управляющих элементов на форме соответственно желанию разработчика. После ввода всех элементов и редактирования форма принимает вид, показанный на рис. 1.10.

   По умолчанию последовательно вводимым одинаковым элементам присваиваются одинаковые имена, отличающиеся последней цифрой (текстовым окнам последовательно ппсваивают-

   ся имена *Textl* и *Textl,* меткам последовательно присваиваются имена *Label!* и *Label2).*

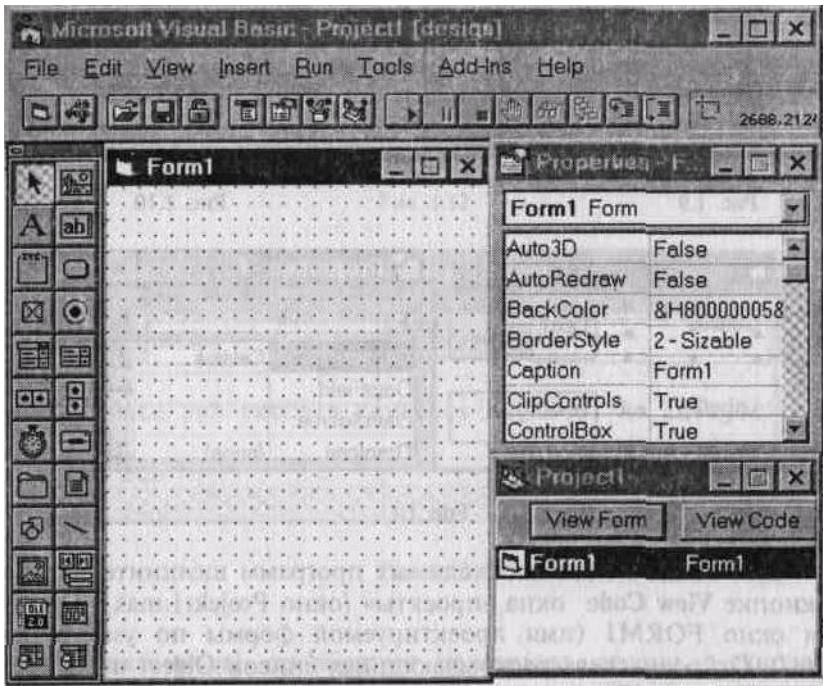


Рис. 1.8

Определим теперь свойства введенных элементов, используя окно свойств **(Properrties).** Активизируем элемент *Label],* при этом активизируются свойства в окне свойств, относящиеся к этому элементу (заголовок **Labell** Label). Выберем свойство **Caption** (Название) и определим его как *Celsiu ,* используя строку ввода. Это название появляется на форме (рис. 1.11).

Свойство **Caption** (Название) для *Label2* зададим *Fahrenheit.* Остальные оставим по умолчанию.

Определим свойства тестовых окон. Активизируем сначала первое окно, очистим окно (удалим значение *Textl* свойства Text) и присвоим ему имя *txtCels* (свойство **Name** (*И*мя), которое будем использовать при написании текста программ (рис. 1.12). Очистим также второе окно и присвоим ему имя *ixtFahr.* Остальные свойства обоих окон оставим по умолчанию.

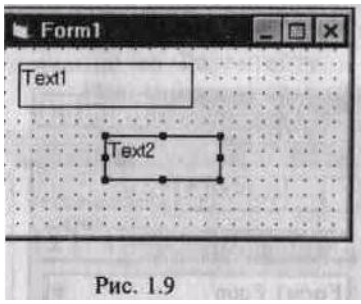


Рис. 1.9

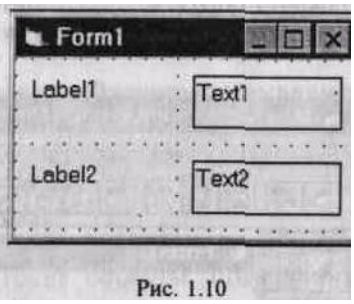


Рис. 1.10

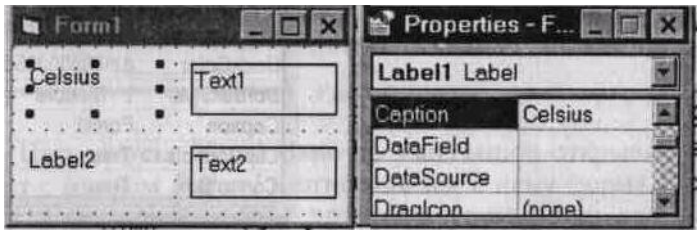


Рис 111

Для ввода текста необходимых программ щелкните мышью на кнопке **View Code** окна «проекты» (окно Piojektl mak) Откроется окно FORM1 (имя проектируемой формы по умолчанию *Formi)* Откроем слева раскрывающийся список **Object** и выберем в нем объект *txtCels* (в качестве объекта выбирается верхнее текстовое окно формы, ранее для этого было задано имя *txtCels)*

Откроем справа раскрывающийся список **Proc** и выберем процедуру обработки события **Keypress** (нажатие клавиши) В поле формы появляются первая и последняя строки процедуры Имя процедуры *txtCels_KeyPress* формируется автоматически (первая часть имени определяется именем выбранного элементом - верхним текстовым окном, вторая - именем выбранной процедуры обработки события) Параметром процедуры является значение кода нажатой клавиши (KeyAscii), определенное как целый тип (Integer) (рис 1 13)

Введем теперь необходимые операторы процедуры (рис 1 14) При нажатии клавиши **Enter,** ASCII-код которой равняется 13 (комментарий в тексте программы), вычисляется значение температуры по Фаренгейту по введенному в окне значению температуры по Цельсию и значение вычисленной температуры выводится в другом окне

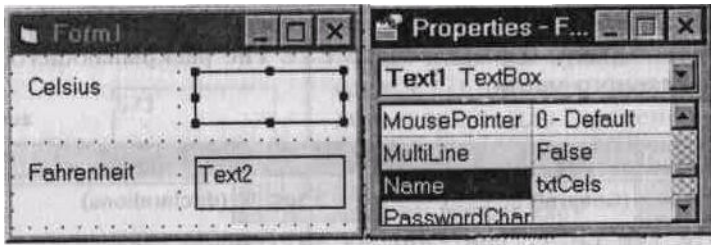


Рис 1 12

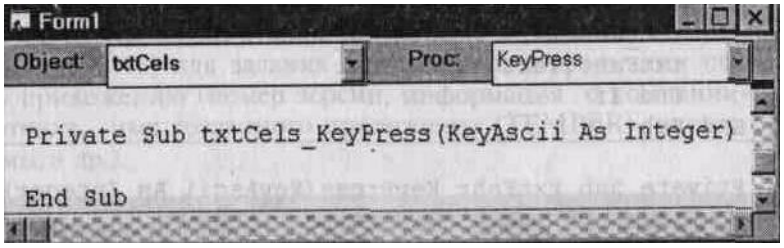


Рис 1 13

Функция **val** преобразует текстовую строку в числовое значение Составные

имена *txtFahr Text* и *txtCels Text* связаны с включением в Visual Basic так. называемых методов для объектов (форм и командных кнопок) Введем также описания переменных *tem_Cels* и *tem_Fahr* в раздел общих (general) описании, что делает их доступными для обеих процедур

Форма и программы обработки событий подготовлены Теперь, щелкнув мышью на кнопке **Run** панели инструментов главного меню, проект можно отправить на выполнение Появляется спроектированная форма, в окнах которой можно вводить информацию (рис 1 15) Кроме ввода и вывода информации, форма соответствует стандарту Windows приложений (реагирует на кнопки минимизации и максимизации, изменяет размеры и др)

Щелчок на кнопке **Конец выполнения** панели инструментов главного меню завершает выполнение проекта

Отлаженный проект можно сохранить двумя способами • используя команду Save **Project** или Save Project as... раскрывающегося меню File главного меню;

- используя команду **Make EXE File** раскрывающегося меню File главного меню.

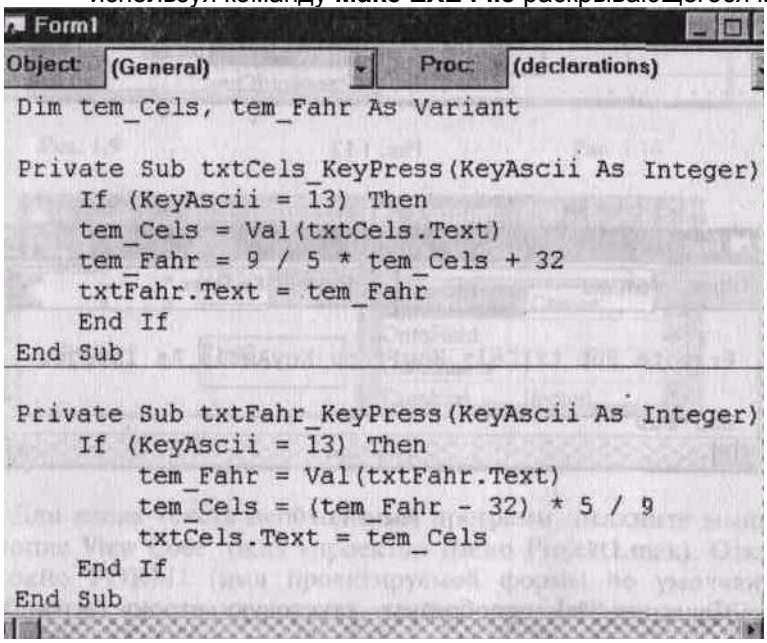
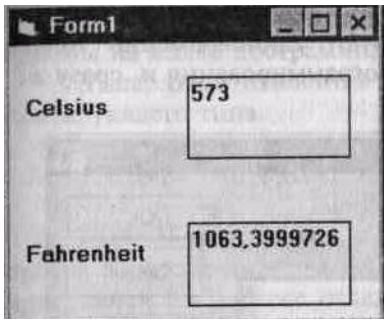


*Рис **1 14***

В первом случае выполнить проект можно только после запуска Visual Basic.

Во втором случае программы проекта компилируются и создается EXE-файл, позволяющий выполнить проект автономно без запуска Visual Basic, даже если на компьютере не установлен Visual Basic.

При выполнении команды Save **Project as...** появляется окно (рис. 1.16), позволяющее задать директорию для хранения и имя проекта (проект сохраняется в директории C:\VB под именем **temper, mak).**



При выполнении команды **Make EXE File** появляется окно (рис. 1.17), позволяющее задать директорию для хранения EXE-файла и его имя (EXE-файл сохраняется в директории C:\VB под именем **temper.exe).**

Рис 1 15

При щелчке по кнопке «Options...» появляется специальное окно (рис. 1.18) для задания необходимой информации по готовому приложению (номер версии, информация о компании- разработчике, имя созданного приложения (TEMPER), иконка для формы и др.).

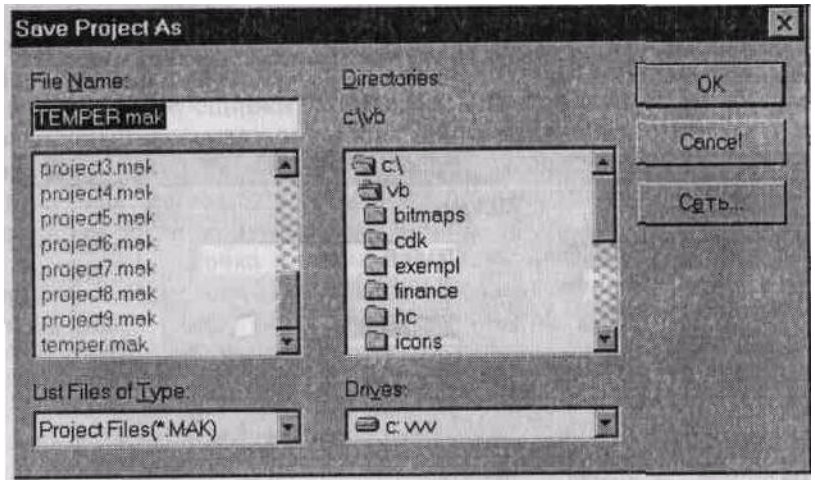


Рис 1 16

Программа на любом языке программирования перед выполнением переводится на машинный язык (набор команд процессора) специальными программами переводчиками. Эти программы принципиально делятся на компиляторы (compilers) и интерпретаторы (interpreters). Первые транслируют всю программу целиком и создают законченный программный модуль на машинном языке Вторые транслируют единовременно только одну строку программы на языке программирования и сразу же

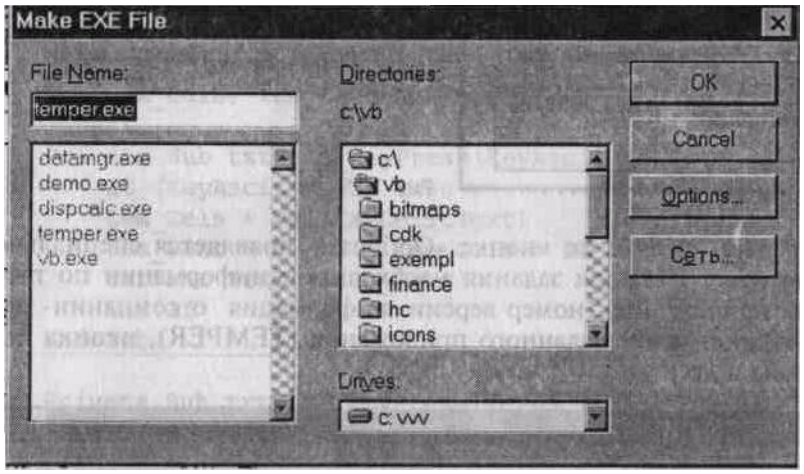


Рис 1 17

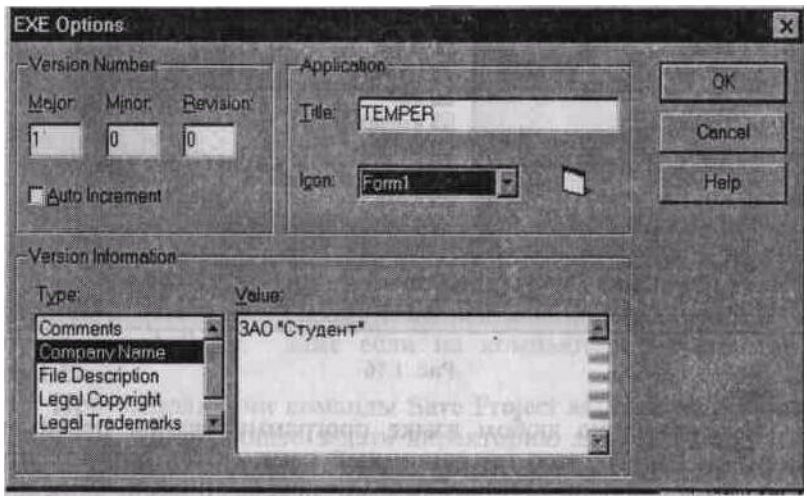


Рис 1 18

исполняют полученный фрагмент программы на машинном языке. Далее интерпретатор возвращается к следующей строке программы на языке программирования, транслирует ее и т. д

Visual Basic относится к языкам программирования интерпретирующего типа

1.3. ОТЛАДКА ПРОГРАММ

Любая программа, написанная даже квалифицированным программистом, содержит ошибки, которые выявляются и исправляются в процессе отладки программы Кроме того, при работе с отлаженной программой пользователь может создать ситуацию, которая программой не обрабатывается корректно Рассмотрим процесс отладки программ и способы обработки ошибок при выполнении программы и имеющиеся для этого средства в Visual Basic

1.3.1. Синтаксический контроль

При ошибке в наборе текста программы в окне кода автоматически (при

активизации в диалоговом окне **Environment Options** (Параметры Среды) флажка **Display Syntax Errors** (Показывать синтаксические ошибки)) инверсной подсветкой выделяется неправильный фрагмент оператора программы На этом этапе Visual Basic отслеживает синтаксические ошибки (неправильно написанные ключевые слова, неверный порядок операндов в операторах, некорректную пунктуацию и т п)

Лишняя точка в операторе уже приведенной выше процедуры автоматически показывается с разъяснением ошибки в окне

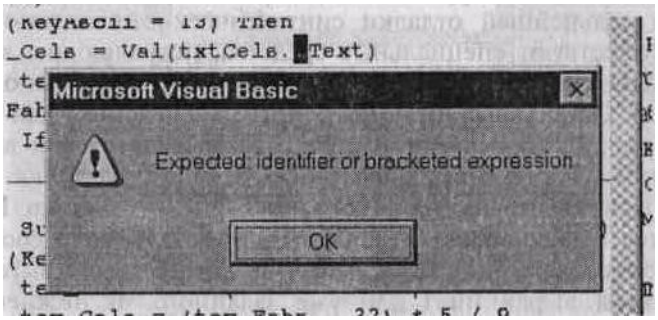


Рис 1 19 Вызов диалогового окна **Environment Options** (Параметры Среды) производится из пункта **Tools** Главного меню. Окно показано на рис. 1.20 и позволяет, кроме указания на выдачу синтаксических ошибок, определить выводимые окна среды (Toolbox, Properties, Project, Debag), задать обязательность объявления переменных (Require Variable Declaration), показа сетки формы (Show Grid) и ее шаг (Width, Height), выравнивания элементов управления относительно сетки (Align Controls to Grid), автоматического сохранения текущих версий файлов форм и проекта перед каждым запуском программы (Save Before Run).

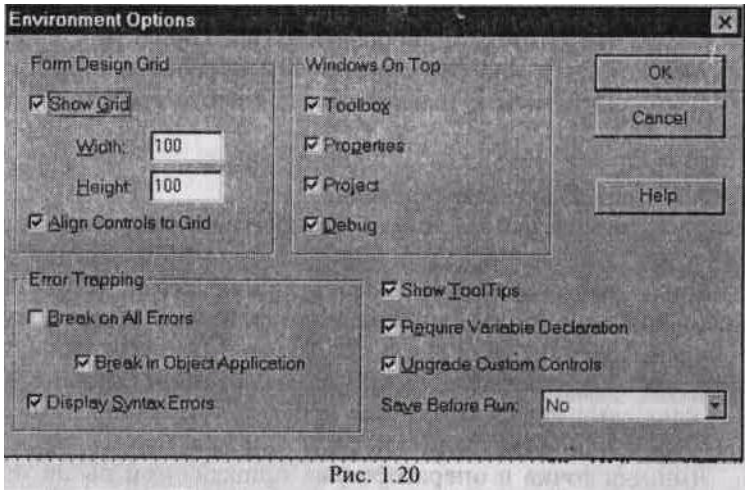


Рис. 1.20

1.3.2. Контроль корректности алгоритма

Для дальнейшей отладки синтаксически правильной программы существуют специальные средства, позволяющие контролировать значения переменных на различных этапах выполнения программы. Окно **Debug** является основным средством для поиска ошибок (рис. 1.21). В верхнем

поле окна выводятся вид выражения (колонка Expression), значение выражения (колонка Value), местонахождение выражения (колонка Context). Кнопки **Immediate и Watch** обеспечивают переключение на просмотр соответственно непосредственно вводимого в этом поле выражения (прямой ввод выражения) или уже заданного. В нижнем поле отображается текст программы.

Вид выражения задается в окне **Add Watch** (вызывается из пункта **Tool** Главного меню) (рис. 1.22). Раскрывающиеся списки модулей (Module) и процедур (Procedure) позволяют задать местоположение выражения в программе. Выражение может быть набрано вручную в текстовом окне. Кроме того, если в тексте программы перед вызовом окна выделен какой-либо оператор или его часть, то оно автоматически появляется в окне (см. рис. 1.22).

Кнопки в области **Watch Type** определяют условия вывода выражения **(Watch Expression** - наблюдение за значением в точках прерывания; **Break When Value Is True** - прерывание выполнения программы, когда значение выражения равно заданному; **Break When Value Changes** - прерывание выполнения программы, когда значение выражения меняется).

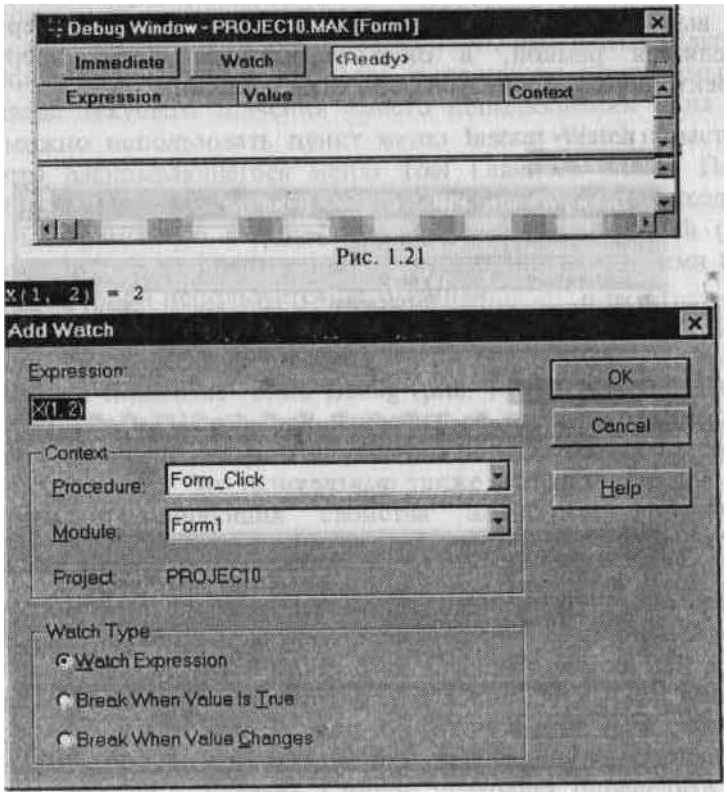


Рис. 1.21

Рис. 1.22 Для вывода в окно **Debug** нужно также установить точки прерывания (моменты, когда программа прекращает работу и выдает требуемую информацию в окно **Debug).** Обычно такими точками являются места программы, проверяемые на корректность работы.

Для установки точки прерывания нужно:

- установить курсор или выделить соответствующий оператор программы;
- щелкнуть мышью по пункту **Toggle Breakpoint** (Поставить точку прерывания) раскрывающегося меню **Run** (Старт) Главного меню.

Установленная точка прерывания выделится подсветкой (рис. 1.23).

После запуска программы значение проверяемого выражения выводится в окне **Debug** (рис. 1.24.). Оператор прерывания выделяется рамкой, в окне указывается место прерывания (проект, форма, процедура обработки события).

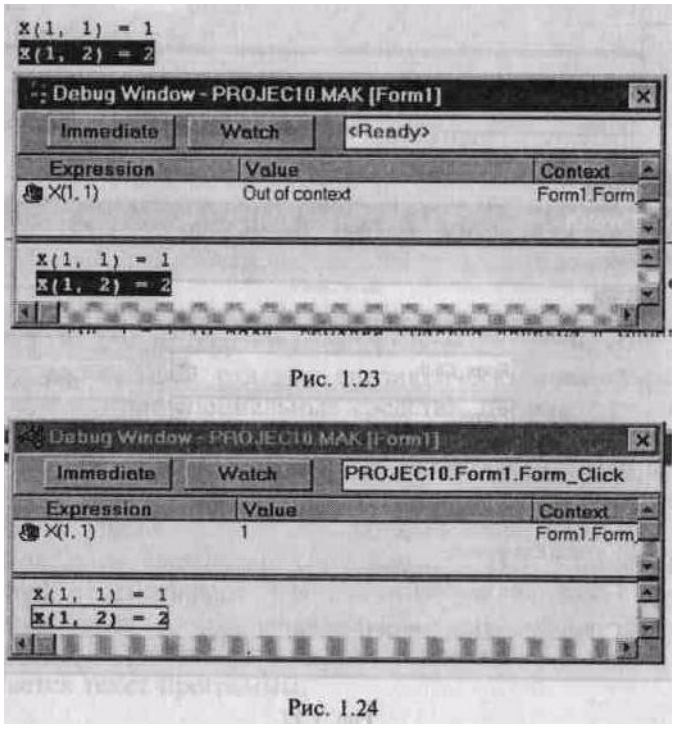


Рис. 1.23

Рис. 1.24

Удалить точку прерывания можно щелкнув мышью по пункту **Clear** Toggle **Breakpoint** (Убрать точку прерывания) раскрывающегося меню **Run** (Старт) Главного меню.

При отладке программ целесообразно комбинировать постановку точек прерывания и точек наблюдения (их установка задается кнопками **Break When Value Is True** (Прерывание выполнения программы, когда значение выражения равно заданному) и **Break When Value Changes** (Прерывание выполнения программы, когда значение выражения меняется) окна **Add Watch).** Точки наблюдения замедляют выполнение программы. Лучше установить точку прерывания в подозрительном месте программы и выполнять программу с нормальной скоростью до этой точки. Далее поставить одну или несколько точек наблюдения и продолжать более медленный поиск ошибок в локализованной области программы.

Если нет необходимости прослеживать изменение значения какой-либо переменной в процессе выполнения программы, то для вывода текущего значения вместо использования окна **Add Watch** можно использовать пункт меню **Instant Watch** (Быстрый просмотр) раскрывающегося меню **Tool** Главного меню. Перед этим, после прерывания выполнения программы,

необходимо выделить переменную в окне кода. В окне **Instant Watch** (рис. 1.25.) выводится имя переменной, текущее значение и имя процедуры, в которой используется переменная.

Щелчком мыши по кнопке **Add** переменную можно добавить в список окна **Debug Windows.**

Кнопка Immediate окна Debug (рис. 1.26) позволяет вывести переменную на форму или изменить ее значение с помощью оператора присваивания (эти изменения не отражаются на содержании программы). Это окно можно также использовать для ввода команд, изменяющих свойства элементов управления (шрифты, цвет фона и др.). Измененные значения выводятся в окне Debug (рис. 1.27).

Если проект включает много процедур, то полезным средством отладки является трассировка вызова процедур (пункт меню **Call...** раскрывающегося меню **Tool** Главного меню или одноименная кнопка панели инструментов). Окно (рис. 1.28.) показывает всю последовательность вызовов от исходной процедуры до текущей (в верхней части расположена последняя вызванная процедура, в нижней - первая). Список позволяет определить, как произошел переход в текущую точку программы.

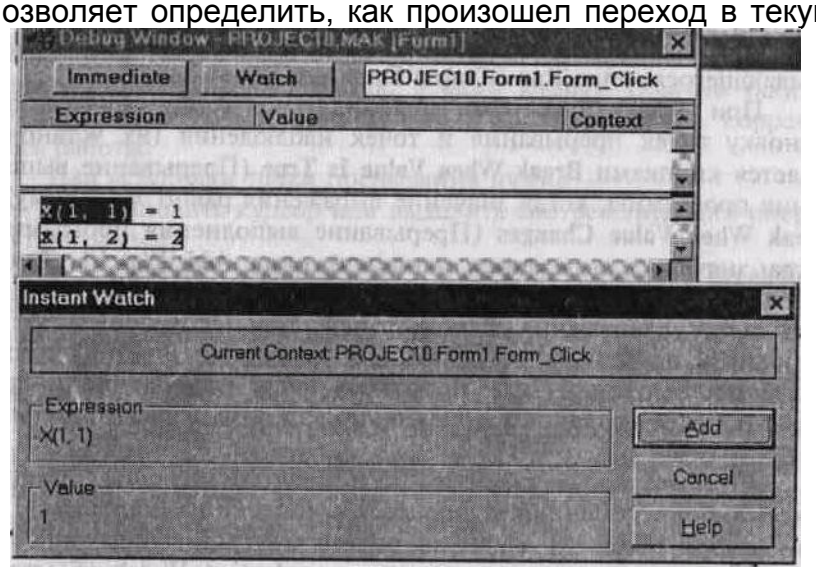


Рис. 1.25

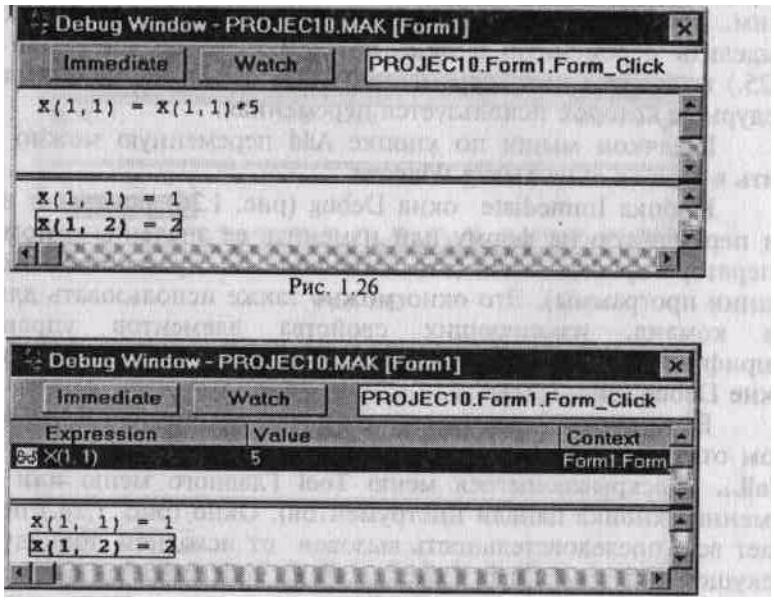


Рис. 1.26

Рис. 1.27

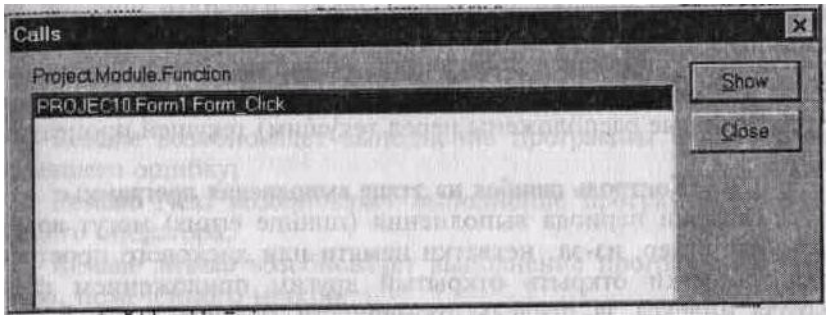


Рис. 1.28

Эффективным средством отладки является также пошаговое выполнение программы и наблюдение результатов выполнения каждого оператора. После приостановки выполнения программы (точки прерывания или кнопка **Break)** необходимо войти в окно редактирования кода. Для выполнения одного оператора используется клавиша F8. При этом будет выполнен оператор, выделенный рамкой, а вьщеление будет перенесено на следующий оператор. Повторяя нажатие клавиши, можно осуществлять пошаговое выполнение программы.

Если текущий оператор содержит вызов процедуры, нажатие клавиши **F8,** щелчок мыши по пункту **Step Into** (Шаг внутрь) раскрывающегося меню **Run** или щелчок мыши по пиктограмме **Step Into** Главного меню вызовет выполнение первого оператора процедуры, которую можно продолжать выполнять по шагам.

Для выполнения всей процедуры (не по шагам) и переходу к оператору, следующему за вызовом процедуры, используется щелчок мыши по пункту Step **Over** (Обойти) или комбинация клавиш Shift+F8.

При пошаговом выполнении некоторый блок операторов можно обойти. Это можно сделать, установив курсор на оператор, на котором необходимо выполнить следующее прерывание выполнения программы, и щелкнув мышью по пункту меню **Step To Cursor** или комбинацией клавиш **Ctrl+F8.**

Пошаговое выполнение можно комбинировать с процедурным выполнением. Отлаженные процедуры одной программы можно выполнять по процедурно, а не отлаженные - пошагово. Средства отладки позволяют также изменять порядок выполнения операторов. При выполнении оператора **Set** Next **Statement** (Установить следующий оператор) раскрывающегося меню **Run** можно перейти к выполнению любого оператора (даже к тем, которые расположены перед текущим) текущей процедуры.

1.3.3. Контроль ошибок на этапе выполнения программы

Ошибки периода выполнения (runtime errors) могут возникать, например, из-за нехватки памяти или дискового пространства, попытки открыть открытый другим приложением файл, выхода индекса за пределы размерности массива и др. В этом случае Visual Basic выводит диалоговое окно с соответствующим сообщением и прекращает выполнение программы (рис. 1.29).

Обработчик ошибок возвращает значение кода ошибки (функция Err), список которых находится в справочной системе **(Help)** и документации к Visual Basic.

Однако такие ошибки можно обрабатывать методом перехвата ошибок **(error trapping).** Перехват ошибок обеспечивается вставкой в текст процедуры оператора

On Error GoTo *метка,* где ***метка*** помечает место входа в обработчик ошибок.

Если в период выполнения ошибка возникнет в одном из операторов процедуры, расположенных за **On Error GoTo *метка,*** то управление передается обработчику ошибок, указанному меткой. Участок обработки ошибок можно закончить оператором **On Error GoTo 0.**

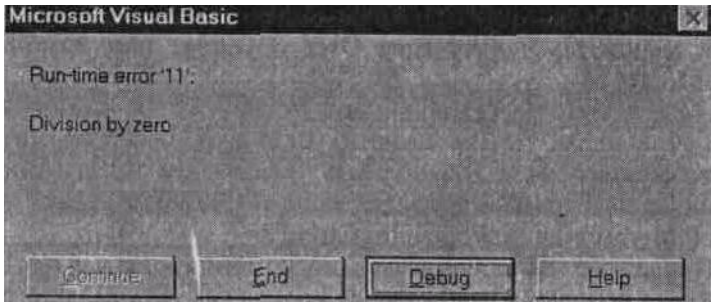


Рис. 1.29

В обработчик ошибок можно включить оператор **Resume** и продолжить

выполнение программы после исправления ошибки операторами, вставляемыми в обработчик ошибок. Оператор **Resume** имеет несколько форм:

Resume возобновляет выполнение программы с оператора, вызвавшего ошибку;
Resume 'Next возобновляет выполнение программы со следующего оператора;
Resume ***метка*** возобновляет выполнение программы с оператора, помеченного меткой.

Пример.

В примере в процедуре **Primer** происходит деление на О (оператор YY = YY/XX, перед выполнением которого переменой XX присваивается значение 0). Этот оператор включен в область обработчика ошибок. При делении на 0 происходит переход на оператор, помеченный меткой Label: (обработчик ошибок), переменной XX присваивается значение 2 и программа продолжает выполнение с оператора, на котором была ошибка на этапе выполнения в результате деления на 0. Программа заканчивает работу и результат распечатывается на форме (рис. 1.30).

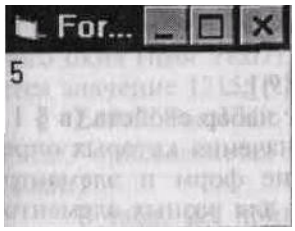


Рис. 1.30

```
Sub Primer()
      Dim XX, YY
      XX = 0
      YY = 10
      On Error GoTo Label
      Y = YY / XX
      Form1.Print YY
      GoTo Label1
      Label:
      XX = 2
      Resume
      Label1:
 End Sub
Private Sub Form_Click()
     Primer
End Sub
```

Таблица 2.1

Название свойства	Для каких элементов управления используется	Описание действия	Возмо жность измнения
Action	Common dialog	Определяет тип диалога	-/+
Action (OLE)	OLE	Определяет тип действия	-/+
Alignment	Check box. label, option button, text box	Определяет способ выравнивания (по левому краю, по правому или по центру)	+/-

Auto Redraw	Form, picture box	Определяет возможность автоматического перерисования	+/-
AutoSize	Label, picture box	Определяет возможность автоматического изменения размера при заполнение	+/-
BackColor, Fore Color	Form, check box, combo box, command button (TWibKoBackColor) data control, directory list box, drive list box, file list box, frame, grid, label, list box, OLE control, option button, picture box, Printer object (только ForeColor), shape (только BackColor), text box	Определяют соответственно цвет фона и цвет выводимой ин4юрмации	+/+
BackStyle	Label, shape	Определяет прозрачность фона	+/+
BorderColor	Line, shape	Определяет цвет рамки	+/+
BorderStyle	Form, grid, image, label, line, OLE control, picture box, shape, text box	Определяет вид рамки	Для Form и text box+/-+/+

Продолжение табл 2 I

Название свойства	Для каких элементов управления используется	Описание действия	Возможность изменения
BorderWidth	Line, shape	Определяет ширину границы	+/+
Cancel	Command button	Определяет, что элемент выполняет функцию Cancel на форме	+/+
Caption	Form, MDI form, check box, command button, data control, frame, label, menu, option button	Определяет текст, выводимый на элемент или возле него Для формы -заголовок	+/+
CellSelected	Grid	Ячейка таблицы выделена строкой и столбцом	-/-
Class	OLE control	Определяет класс объекта OLE	+/-
ClipControls	Form, frame, picture box	Определяет необходимость перерисовки всего объекта или появляющейся части	+/+
Clip	Grid	Определяет содержимое выделенных ячеек таблицы	-/+
Col, Row	Grid	Определяет выде-	-/+

		ленную колонку или строкуа таблицы	
ColAlignment	Grid	Выравнивание данных в колонке таблицы	-/+
Cols, Rows	Grid	Определяет число колонок,столбцов таблицы	+/+
Columns	List box	Определяет число колонок в списке	+/+
ColWidth	Grid	Ширина колонки	-/+
ControlBox	Form	Определяет наличие кнопки системного меню на форме	+/-
CuncntX, CurrentY	Form, picture box, Printer object	Текущие координаты по горизонтали и вертикали (для рисующих или печатающих методов)	-/+

Продолжение табл. 2.1

Название свойства	управления используется		жность изменения
DatabaseName	Data control	Имя и расположение базы данных	+/+
DataChanged	Check box, image, label, picture box, text box	Указатель изменения данных в элементе при чтении записи (несовпадение)	-/+
DataField	Check box. image, box	Определяет связь с полем записи файла	+/+
	Check box, image, label, picture box, text box	Определяет источник данных для элемента управления	+/-
Default	Command button	Определяет, является ли данная командная командной кнопкой по умолчанию	+/+
	box. Printer object, shape.	рисуемой линии (точки)	-/+
DrawWidth	Form, picture box, Printer object	Определяет ширину рисуемой линии (точки)	-/+
Enabled	Form, MDI form, check box, combo box, command button, data control, directory list box, drive list box, file list box, frame, grid, horizontal scroll bar, image, label, list box, menu, option button, picture box, text box, timer, vertical scroll bar	Определяет возможность доступа к элементу (элемент реагирует на действия пользователя)	+/+
FillColor	Form, picture box, Printer object, shape	Определяет цвет заполнения	+/+
FillStyle	box. Printer object,	заполнения	
FixedCols, Fixed Rows	Grid	Число выделенных	+/+ •

колонок,столбцов

Продолжение табл. 2.1

Название свойства	Для каких элементов управления используется	Описание действия	Возможность изменения
FontBold, FontItalic, FontStrikethr. FontTranspar. FontUnderline	Form, check box, combo box, command button, common dialog, data control, directory list box, drive list box, file list box, frame, grid, label, list box, optionbutton, picture box, Printer object, text box	Вид выводимого текста(жирный, курсив, зачеркнутый, «ясный»)	+/+
FontNarne	Form, check box, combo box, command button, common dialog, data control, directory list box, drive list box, file list box, frame, grid, label, list box, optionbutton picture box. Printer object, text box	Тип шрифта выводимого текста	+/+
FontSize	Form, check box, combo box, command button, common dialog, data control, directory list box, drive list box, file list box, frame, grid, label, list box, optionbutton, picture box, Printer object, text box	Размер шрифта выводимого текста	+/+
BackColor, ForeColor	Form, check box, combo box, command button (только BackColor), data control, directory list box, drive list box, file list box, frame, grid, label, list box, OLE control, option button, picture box, Printer object (только ForeColor), shape (только BackColor), text box	Цвет фона и основной цвет выводимой информации	

Продолжение табл. 2.1

Название свойства	Для каких элементов управления используется		жность изменения
Format		Определяет формат получаемых и пере	-/+
GridLines	Grid	определяет видимость сетки таблицы	+/+
GridLineWidth	Grid	Определяет ширину линий сетки таблицы	+/+

Height, Width	Form, MDI form, check box, combo box, command button, data control, directory list box, drive list box, file horizontal scroll bar, image, label	Определяют размеры объекта (высота и ширина)	+/+
Highlight	Grid	ку выделенной ячейки таблицы.	+/+
		иконки формы в ее свернутом состоянии	
Index (Control Arrays)	Check box, combo box, command button, common dialog, data control, directory list box, drive list box, file list box, frame, grid, horizontal scroll bar, image, label, line, list box, menu. OLE control, option button, box, timer, vertical scroll bar	Определяет индекс элемента в массиве однородных элементов управления	+/- .
Interval		миллисекунд в зада-ваемом интервале	+/+
I tern Data	Combo box, list box	Массив значений индексов элементов списка (первоначально значения индексов совпадают с позицией элемента в списке)	-/+

Продолжение табл. 2.1

Название свойства	Для каких элементов управления используется	Описание действия	Возможность изменения
KeyPreview	Form	Определяет, вызываются ли процедуры обработки события клавиатуры формы перед событиями клавиатуры элементов управления	+/+
LargeChange	Horizontal scroll bar, vertical scroll bar	Определяет изменение при щелчке мыши по полю линейки прокрутки между движком и стрелкой	
SmallChange	Horizontal scroll bar, vertical scroll bar	Определяет изменение при щелчке мыши по стрелке	+/+
Left, Top	Form, MDI form, check box, combo box, command button, common dialog, data control, directory list box, drive list box, file list box, frame, grid, horizontal scroll bar, image, label, list box, OLE control, option button, picture box, shape, text box, timer, vertical	Определяют координату верхнего левого угла элемента управления (Left- расстояние до левой границы, Top- расстояние до верхней границы)	+/+

	scroll bar		
LinkItem	Label, picture box, text box	Определяет спецификацию данных, передаваемых, в элемент от других приложений (DDE)	+/+
List	Combo box, directory list box, drive list box, file list box, list box	Определяет список элементов окна	-/+
ListIndex	Combo box, directory list box, drive list box, file list box, list box	Определяет индекс выбранного элемента списка	-/+
Max, Min	Horizontal scroll bar, vertical scroll bar	Определяет максимальное и минимальное значение линейки прокрутки	+/+

Продолжение табл. 2.1

	Для каких элементов управления используется	Описание действия	Возможность изменения
MaxButton, MinButton	Form	Наличие кнопки максимизации и минимизации	+/-
Max Length	Text box	Определяет максимальное число символов	+/+
MousePointer	Form, check box, combo box, command button, data control, directory list box, drive list box, file list box, frame, horizontal scroll bar, image, label, list box, option button, picture box, Screen object, text box, vertical scroll bar	Определяет тип указателя мыши при его перемещении по элементу	+/+
MultiLine	Text box	Определяет многострочное текстовое окно	+/-
MultiSelect	File list box, list box	Возможность и способ множественного выбора	+/-
Name	Все элементы управления и формы	Определяет имя элемента, используемое при написании программы	+/-
NewIndex	Combo box, list box	Индекс добавляемого в список элемента	-/-

Parent	Check box, combo box, command button, directory list box, drive list box, file list box, frame, grid, horizontal scroll bar, image, label, line, list box, menu, OLE control, option button, picture box, shape, text box, timer, vertical scroll bar	Определяет форму, на которой находится элемент	-/-
	Text box	Определяет шифровку выводимых в окне символов	+/+

Продолжение табл. 2.1

Название свойства	Для каких элементов управления используется	Описание действия	Возможность изменения
PasteOK	OLE control	Определяет возможность ввода из Clipboard	-/-
Path	App object, directory list box, file list box	Путь в файловой системе	-/-
Pattern	File list box	Выводимое имя файла	-/+
Picture	Form, image, OLE control, picture box	Определяет выводимую картинку	+/+
Prev Instance	App object	Определяет, что объект - приложение уже запущено	-/-
PrinterDefault	Common dialog (print dialog)	Определяет выбор принтера по умолчанию	
Readonly	Data control, file list box	Определяет указатель на открытие базы данных только на чтение или в списке файлов есть файлы с атрибутом ReadOnly	+/+
RecordCount	Table object. Dynaset object, Snapshot object	Определяет число записей	-/-
Recordset	Data control	Определяет источник данных	-/+
RecordSource	Data control	Определяет источник данных	+/+
RowHeight	Grid	Определяет высоту выделенной строки	-/+
ScaleHeight, ScaleWidth	Form, MDI form, picture box, Printer object	Определяет число единиц измерения по вертикали и горизонтали	+/+
ScaleLeft, ScaleTop	Form, picture box, Printer object	Определяет координаты верхнего левого утла	+/+
ScaleMode	ScaleMode	Определяет единицы измерения координат	+/+
Scroll Bars	MDI form, grid, text box	Определяет наличие линейки прокрутки	+/-
SelCount	List box	Определяет число выделенных элементов списка	-/+

Продолжение табл 2 1

Название свойства	Для каких элементов управления используется	Описание действия	Возможность изменения
Selected	File list box, list box	Определяет выделенные элементы списка	-/+
SelEndCol, SelStartCol, SelEndRow, SelStartRow	Grid	Определяет начало и конец выделенных ячеек	-/+
Sel Length, SelStart, SelText	Combo box, text box	Определяют длину выделения, начальную позицию и выделение	-/+
Shortcut	Menu	Определение клавиш	+/-
SizeMode	OLE control	Определяет преобразование размера для представления	+/+
Sorted	Combo box, list box	Определяет упорядочение элементов в списке по алфавиту или по порядку ввода	+/-
Source Doc	OLE control	Имя файла	+/+
Source I tern	OLE control	Определение данных	+/+
Style 1	Combo box	Определяет тип комбинированного списка	+/-
TabIndex	Check box, combo box, command button, directory list box, drive list box, file list box, frame, grid, horizontal scroll bar, label, list box, option button, picture box, text box, vertical scroll bar	Определяет индекс элемента в массиве элементов управления	+/-
TabStop	Check box, combo box, command button, directory list box, drive list box, file list box, grid, horizontal scroll bar list box, option button, picture box, text box, vertical scroll bar	Определяет быструю установку фокуса	+/+

Продолжение табл. 2.1

Название свойства	Для каких элементов управления используется	Описание действия	Возможность изменения

Tag	Form, MD1 form, check box, combo box, command button, common dialog, data control, directory list box, drive list box, file list box, frame, grid, horizontal scroll bar, image, label, line, list box, menu, OLE control, option button, picture box, shape, text box, timer, vertical scroll bar	Определяет идентификацию внешних данных	+/+
Text	Combo box, list box, text box	Определяет данные текстового окна или выбранного элемента списка	+/+
Top Row	Grid	Определяет максимальное количество выделенных строк	-/+
UpdateOptions	OLE control	Определяет изменение данных элемента при изменении данных в связанном объекте	-/+
Value	Check box, command button, field object, horizontal scroll bar, option button, vertical scroll bar	Значение состояния элемента управления	-/+
Verb	OLE control	Спецификация действий при запуске OLE—объекта	+/+
Visible	Form, MDI form, check box, combo box, command button, common dialog, data control, directory list box, drive list box, file list box, frame, grid, horizontal scroll bar, image, label, line, list box, menu, OLE control, option button, picture box, shape, text box, vertical scroll bar	Определяет видимость элемента	+/+

Окончание табл. 2.1

Название свойства	Для каких элементов управления используется		жность изменения
WindowState	Form, MD1 form	Определяет видимость формы при выполнении программы	+/+
Wordwrap	Label	Определяет направление (вертикальное или горизонтальное) изменения поля метки при ее заполнении текстом	+/-
XI, YI, X2, Y2	Line	Определяют начальные и конечные координаты	+/+

Каждому диалоговому элементу (форме и элементам управления) в Visual Basic поставлен в соответствие определенный набор событий,

происходящих в период выполнения программы. Например, событие **Load** (Загрузка) происходит при загрузке формы, событие **Clik** (Щелчок) вызывается щелчком кнопки мыши, событие **DblClick** (Двойной щелчок) вызывается двойным щелчком кнопки мыши и т.д.

В табл. 2.2 приводится список событий, элементы управления, для которых они определены, и краткое описание действия.

Более подробную информацию о приведенных в таблице событиях и особенностях их использования можно найти в встроенной справочной системе Visual Basic.

Каждому приведенному в таблице событию ставится в соответствие процедура обработки события (процедура **(procedure)** - в языках программирования набор операторов, выполняемых при вызове процедуры). Событие может вызывать при наличии в приложении процедуры обработки данного события программируемое изменение данных. Как было показано выше. Visual Basic содержит автоматически формируемые заготовки для таких процедур (первый и последний операторы процедур) в поле для записи программы (вызываются из окна проекты **(Project)** Главного меню). Имя процедуры формируется автоматически и состоит из двух частей, разделенных подчеркиванием: имя выделенного диалогового элемента и имя выделенного события.

Таблица 2.2

Название события	Элементы управления, для которых используется события	Описание действия
Activate, Deactivate	Form, MD1 form	Активизация (окно формы становится активным), дезактивизация формы
Change	Combo box, directory list box, drive list box, horizontal scroll bar, label, picture box, text box, vertical scroll bar	Изменение содержания элементов управления (например, набор символа в текстовом окне)
Click	Form (кроме MD1 form), check box, combo box, command button, directory list box, file list box, frame, grid, image, label, list box, menu, OLE control, option button, picture box, text box	Одинарный щелчок мыши по управляющему элементу
DblClick	Form (кроме MD1 form), combo box, file list box, frame, grid, image, label, list box, OLE control, option button, picture box, text box	Двойной щелчок мыши по управляющему элементу
DragDrop	Form, MD1 form, check box, combo box, command button, data control, directory list box, drive list box, file list box, frame, grid, horizontal scroll bar, image, label, list box, OLE control, option button, picture box,	После перемещения управляющего элемента мышью и отпускания клавиши мыши (определяет результирующую позицию элемента).

	text box, vertical scroll bar	
DragOver t •'•	Form, MD1 form, check box, combo box, command button, data control, directory list box, drive list box, file list box, frame, grid, horizontal scroll bar, image, label, list box, OLE control, option button, picture box, text box, vertical scroll bar	Процесс перемещения управляющего элемента мышью (определяет текущую позицию элемента)

Продолжение табл. 2.2

Название события	Элементы управления, для которых используется события	Описание действия
Drop Down	Combo box (кроме случая, когда свойство Style равно 1)	Результат выделения элементов в комбинированном списке перед каким-либо действием
Error	Data control	Ошибка при обращении к базе данных
GotFocus	Form (кроме MDI form), check box, combo box, command button, directory list box, drive list box, file list box, grid, horizontal scroll bar, list box, OLE control, option button, picture box, text box, vertical scroll bar	Активизация элемента управления (установка фокуса на элемент). В Windows в каждый момент только один элемент экрана является активным, т.е. с ним возможна работа
KeyDown, KeyUp	Form, check box, combo box, command button, directory list box, drive list box, file list box, grid, horizontal scroll bar, list box, OLE control, option button, picture box, text box, vertical scroll bar	Нажатие или отпускание какой-либо клавиши в момент, когда элемент управления находится в фокусе
KeyPress	Form, check box, combo box, command button, directory list box, drive list box, file list box, grid, horizontal scroll bar, list box, OLE control, option button, picture box, text box, vertical scroll bar	Нажатие и отпускание какой-1 либо клавиши
LinkClose	Form, MDI Form, label, picture box, text box	Конец динамического обмена данными (DDE) с другими приложениями
LinkError	Form, MDI Form, label, picture box, text box	Ошибка при динамическом обмене данными (DDE) с другими приложениями
LinkExecute	Form, MDI Form	Командная строка посылается в приложение, с которым устанавливается режим динамического обмена данными (DDE)

Продолжение табл. 2.2

Название события	Элементы управления, для которых используется события	Описание действия
LinkNotify	Label, picture box, text box	Изменение данных в приложении, с которым установлен режим динамического обмена данными (DDE)
LinkOpen	Form, MDI Form, label, picture box, text box	Инициализация связи с приложением для динамического обмена данными (DDE)
Load	Form, MDI form	Загрузка формы
LostFocus	Form, check box, combo box, command button, directory list box, drive list box, file list box, grid, horizontal scroll bar, list box, OLE control, option button, picture box, text box, vertical scroll bar	Дезактивизация элемента управления (потеря фокуса элементом). В Windows в каждый момент только один элемент экрана является активным, т.е. с ним возможна работа (активизация другого элемента)
Mouse Down Mouse Up	Form (кроме MDI form), check box, command button, data control, directory list box, file list box, frame, grid, image, label, list box, option button, OLE control, picture box, text box	Нажатие. Отпускание кнопки мыши
MouseMove	Form (кроме MDI form), check box, command button, data control, directory list box, file list box, frame, grid, image, label, list box, OLE control, option button, picture box, text box	Перемещение мыши
Paint	Form, picture box	Новое представление на экране после изменения размера или после удаления закрывающего объекта на экране
PathChange	File list box	Изменение перехода (установка нового имени файла (FileName) или перехода (Path))
PattemChange	File list box	Изменение модели названия файла (например, .)

Окончание табл. 2.2

	которых используется события	Описание действия
Query Unload	Form, MD1 form	Предшествует закрытию формы или приложения
Reposition	Data control	Запись становится текущей
Resize	Form, MDI form. OLE control, picture box	при изменении размера элемента
RowColChange	Grid	Переход от одной ячейки
Scroll	Horizontal scroll bar. vertical scroll bar	Перемещение движка линейки прокрутки

SelChange	Grid	диапазона ячеек к другому .
Timer	Timer	времени
Unload	Form, MD1 Form	
Updated	OLE control	Изменение данных в объекте OLE
Validate	Data control	Перед тем, как другая запись становится текущей

В языках программирования, поддерживающих объектно-ориентированное программирование, введены так называемые методы. Несмотря на то, что Visual Basic нельзя считать настоящим объектно-ориентированным языком, для него включены методы для форм и элементов управления. Методы работают как процедуры или функции (т.е. обеспечивают выполнение тех или иных действий или изменение данных соответственно стандартному алгоритму, реализующему данный метод), но принадлежат конкретным объектам.

Для того чтобы вызвать метод, указывается имя объекта и через точку имя метода:

{ [имя_формы.] 1 [имя_формы.] имя_элемента_управления. }имя_метода

Например, **Debug.Print** обеспечивает вывод (печать) текста в специальном отладочном окне **(Debug** - имя окна, **Print** - имя

метода).

В табл. 2.3 приводится список методов, элементы управления, для которых они определены, и краткое описание действия.

Таблица 2.3

Название метода	Элементы управления, для которых используются события	Описание действия
AddItem	List box, combo box, grid control.	Добавление элемента в список (List box, combo box) или строки (grid)
AddNew	Data control	Очистка буфера и подготовка создания новой записи
Arrange	MDI Form	Представление окон и иконок на MDI Form
Circle	Form, picture box, Printer object	Рисование дуги, эллипса или окружности
Clear	List box, combo box	Удаление всех элементов списка
Close	Data control	Закрытие базы данных
CIs	Form, picture box	Очистка от графических элементов или текста
DDEMetods (LinkExecute, LinkSend, LinkPoke, LinkRequest)	Label, picture box, text box.	Динамический обмен данных с другими приложениями
Delete	Data control	Удаление текущей записи
Drag	Все, кроме Line, Menu, Shape, Timer	Перемещение объекта
Edit	Data control	Открытие текущей записи для

		редактирования
EndDoc	Printer	Конец передачи документа принтеру
Execute	Data control	Выполнение запроса к базе данных
ExecuteSOL	Data control	Выполнение SQL запроса к базе данных
FieldSize		Определение числа байт в тексте или двоичных знаков
FindFirst, FindLast, FindNext, Find Previous	Data control	Определение первой, последней, следующей или предыдущей записи, соответствующей заданному критерию. Запись делается текущей
GetChunk		Выделенные FieldSize число байтов или двоичных знаков
GetData GetFormat GetText	Clipboard	Работа с буфером обмена

Продолжение табл. 2.3

Название метода	Элементы управления, для которых используются события	Описание действия
Hide	Form, MD1 form	Спрятать форму с экрана без ее выгрузки
Line	Form, picture box. Printer	Рисование линии или прямоугольника
Move	Все, кроме timer и menu	Перемещение элемента
MoveFirst, Move Last, MoveNext, Move Previous	Data control	Переход к первой, последней, следующей или предыдущей записи, соответствующей заданному критерию Запись делается текущей
NewPage	Printer	Переход при печати к следующей странице
Point	Form, picture box	RGB (red-green-blue) цвет точки
PopupMenii	Form	Вывод всплывающего меню в заданной точке формы
Print	Form, picture box. Debug, Printer	Печать строки на объекте
PnntForm	Form	Побитовая распечатка формы
PSct	Form, picture box. Printer	Точка на объекте
Refresh	Все	Немедленная перерисовка на экране
Remove Item	List box, combo box, grid	Удаление элемента списка или ячеек в сетке
Scale	Form, picture box. Printer	Координаты объекта
Set Data	Clipboard	Запись графики в Clipboard с заданным форматом
SetFocus	CheckBox, ComboBox, CommandButton, DiiListBox, DnveListBox, FileListBox, Form, HScrollBar, ListBox, MDIForm, OLE Container, OptionBiitton. PictureBox, Text Box. VScrollBar	Установка курсора
SetText	Clipboard	Запись строки в Clipboard с заданным форматом
Show	Form	Вывод формы на экран

TextHcight	Form, picture box. Printer	Высота текстовой строки при печати с текущим шрифтом
TextWidth	Form, picture box. Printer	Ширина текстовой строки при печати с текущим шрифтом

Окончание табл 2 3

Название метода	Элементы управления, для которых используются события	Описание действия
Update	Data control	Сохранение буфера копирования
UpdateControls	Data control	Контроль изменения данных
Update Record	Data control	Сохранение изменения данных
ZOrder	App, CheckBox, ComboBox, CommandButton, DirListBox, DnveListBox, FileListBox, Form, Frame, Grid, HScrollBar, Image, Label, Line, ListBox, MDIForm, OptionBiitton, PictureBox, Shape, TextBox, VScrollBar	Расположение на переднем или заднем плане

Более подробную информацию об использовании методов можно получить из встроенной в Visual Basic справочной системы.

Рассмотрим более подробно использование свойств, процедур обработки событий и методов для форм и элементов управления при создании графического интерфейса приложений.

2.2. ФОРМА

Каждая форма в период выполнения соответствует отдельному окну. Внешний вид и поведение формы на экране определяется значением свойств. Эти свойства определяются в окне свойств на этапе разработки формы или операторами программы в процессе выполнения приложения. Наиболее часто используемыми свойствами являются:

BorderStyle (Тип границ) принимает одно из четырех стандартных значений, изменение которых допускается только на этапе разработки формы и определяют вид и поведение формы в процессе выполнения приложения (но не влияют на вид формы при разработке):

О - None - границы окна отсутствуют;

/ - *Fixed Single* - окно постоянного размера, определяемого на этапе разработки, с одинарной линией границы; *2 - Sizeable* - окно изменяемого размера с двойной линией границы;

3 - Fixed Double - окно постоянного размера, определяемого на этапе разработки, с двойной линией границы.

Caption (Название) - заголовок окна.

ControlBox (Кнопка системного меню) принимает одно из двух стандартных значений, изменения которых допускается только на этапе разработки формы:

True - кнопка системного меню в левом верхнем углу окна;

False - кнопка отсутствует.

Enabled (Доступ) принимает одно из двух стандартных значений:

True - форма доступна (по умолчанию);

False - форма недоступна (блокирована от воздействия любых событий, связанных с мышью или клавиатурой).

FontName (Имя), **FontSise** (Размер), **FontBold** (Полужирный), **FontItalic** (Курсив), **FontStrikethru** (Зачеркнутый), **FontUnderline** (Подчеркнутый) определяют шрифты выводимого на форму текста.

Размер шрифта определяется в пунктах (point) - стандартная типографская единица измерения (1 пт равен 1/72 дюйма или 0,035 см).

BackColor (Цвет фона) и **ForeColor** (Основной цвет) определяют цвет фона формы и выводимого на форму текста или изображения.

Height (Высота), **Width** (Ширина) определяют высоту и ширину формы. Единица измерения 1 twip равна 1/1440 дюйма или 0,0018 см.

MaxButton (Кнопка развернуть), **MinButton** (Кнопка свернуть) опеделяют наличие на форме соответствующих кнопок.

Name (Имя) - имя формы. Используется при написании текста программы и изменяться не может.

Top (Верхняя координата), **Left** (Левая координата) определяют координаты левого верхнего угла формы.

Visible (Видимость), **WindowsState** (Состояние окна) определяют видимость формы на экране (True - видима, False - невидима) и отображение (0 - нормальное, 1 - свернутое в значок, 2 -развернутое).

Наиболее часто используются следующие события:

Click (Щелчок) или **DblClick** (Двойной щелчок) мышью в любом месте формы вызывает процедуры обработки события Form_Click или Form_DblClick.

KeyPress (Нажатие клавиши) вызывает процедуру обработки события Form_KeyPress, на вход которой подается значение ASCII-кода нажатой клавиши.

Load (Загрузка) - событие происходит при загрузке формы (например, при запуске приложения) и удобна для инициализации свойств и переменных при запуске программы.

Из методов рассмотрим:

Cls очищает форму от всех изображений и текста. Синтаксис:

[имя формы.] Cls;

Print выводит текст на форму и очень удобен для вывода на форму простой информации. Синтаксис:

[имя формы.]Print [[выражение][{;!,}]]...

Если после выражения стоит «;», то за последним символом предыдущего выражения сразу выводится следующее. Если стоит «,», то вывод производится по зонам, каждая из которых имеет размер 14 символов. Отсутствие после последнего выражения «;» или «,» переводит позицию вывода в начало следующей строки.

Пример программы.

Создадим новый проект с именем project2.mak, в котором открывается форма Forml. Используя кнопку **View Code,** откроем окно программы и введем тексты программ для процедур обработки событий **Click и KeyPress** (рис. 2.1) для объекта Form.

При щелчке кнопки мыши в любом месте формы исходное положение и размер формы (заданы по умолчанию) изменяются, задается полужирный шрифт и его размер для вывода на форму начала текста.

При нажатии клавиши **Enter** изменяются исходное положение и размер формы, задается шрифт «курсив» и его размер, которым выводится на форму продолжение текста. Запустив проект на выполнение кнопкой **Run** Главного меню, увидим пустую форму. Щелкнув на ней мышью, увидим изменение положения и размеров и начало текста. Нажав клавишу **Enter,** увидим измененную форму с окончанием текста (рис. 2.2).

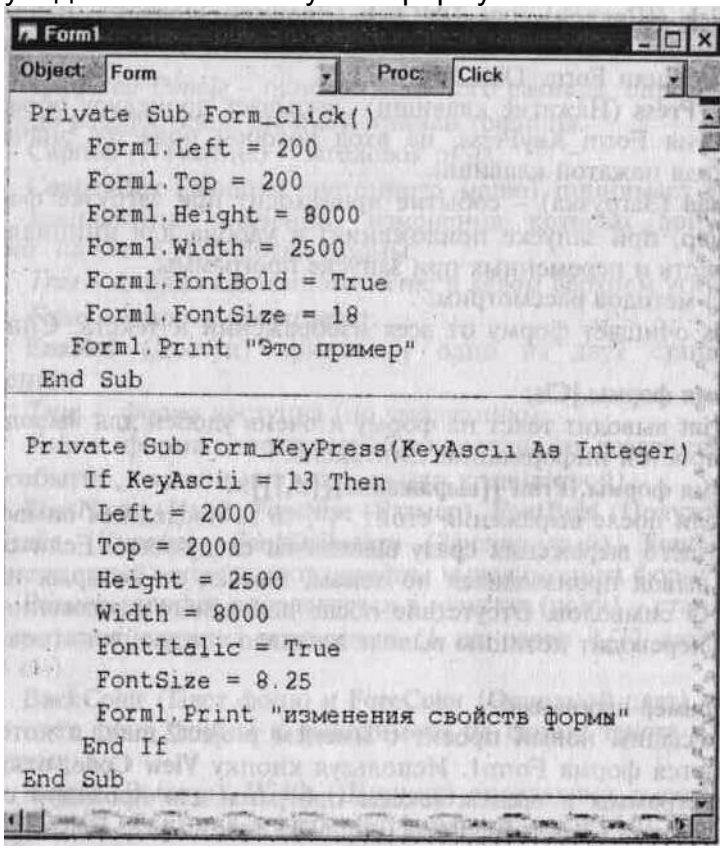


Рис. 2.

2.3. МЕТКА

Label (Метка) - поле, заполняемое текстовой информацией, которая может изменяться только программно. Наиболее часто используемыми свойствами являются:

Alignment (Выравнивание) определяет размещение названия метки. По умолчанию значение равно *0* - выравнивание по левой границе *(Left Justify).* Значение равно / - выравнивание по правой границе *(Right Justify), 2* — выравнивание по центру *(Center).*

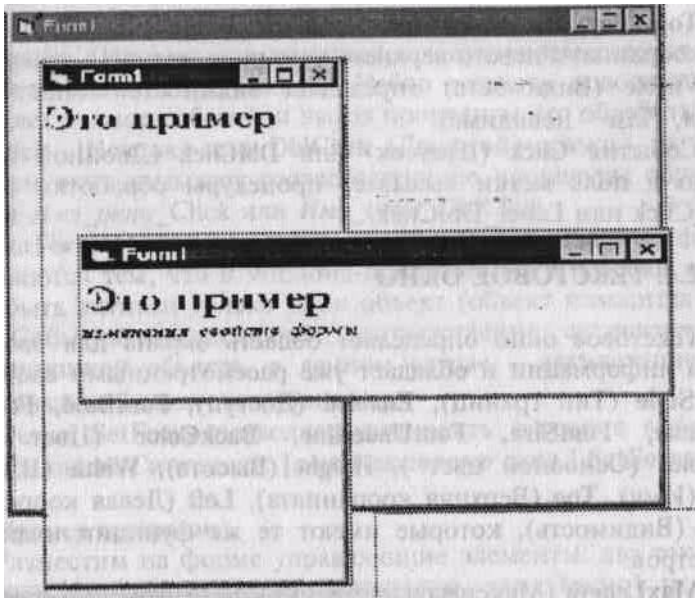


Рис. 2.2

AutoSize (Автоподстройка размера) позволяет автоматически-подогнать размер поля метки под размер текста, заданный свойством **Caption** (значение *True).* При значении False размер не меняется, лишние символы отсекаются.

BorderStyle (Тип границ) принимает значение *0 (по* умолчанию) - контур поля метки отсутствует или / - контур поля метки очерчивается одинарной линией.

Caption (Название) - текст поля метки.

Enabled (Доступ) принимает по умолчанию значение *True.* При значении *False* текст метки поблекнет и обработка событий, связанных с действием мыши, блокируется.

FontBold, FontItalic, FontName, FonfSize, FontUnderline определяют шрифты текста метки.

BackColor (Цвет фона) **ForeColor** (Основной цвет) определяют цвет фона поля метки и выводимого текста.

Height (Высота), **Width** (Ширина) определяют высоту и ширину поля метки.

Name (Имя) - имя метки. Используется при написании текста программы и

изменяться не может. **Top** (Верхняя координата). **Left** (Левая координата) определяют координаты левого верхнего угла поля метки.

Visible (Видимость) определяет видимость метки *(True* -видима, *False* - невидима).

События **Click** (Щелчок) или **DblClick** (Двойной щелчок) мышью в поле метки вызывает процедуры обработки события Label_Click или Label_DblClick.

2.4. ТЕКСТОВОЕ ОКНО

Текстовое окно определяет область экрана для ввода или вывода информации и обладает уже рассмотренными свойствами **BorderStyle** (Тип границ), **Enabled** (Доступ), **FontBold, FontItalic, FontName, FontSize, FontUnderline, BackColor** (Цвет фона), **ForeColor** (Основной цвет), **Height** (Высота), Width (Ширина), **Name** (*И*мя), **Top** (Верхняя координата), **Left** (Левая координата), **Visible** (Видимость), которые имеют те же функции и значения параметров.

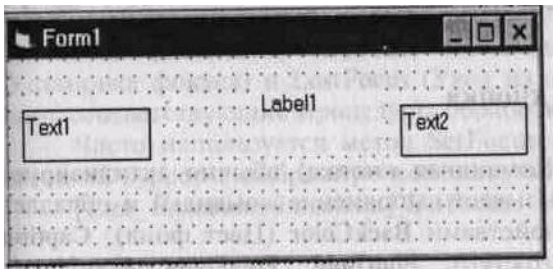
MaxLength (Максимальная длина) по умолчанию принимает значение *0,* что позволяет вводить или выводить любое количество символов. Другое значение определяет возможное число символов.

MultiLine (Несколько строк) может принимать значение *False* (позволяет вводить или выводить одну строку текста) или *True* (позволяет вводить или выводить несколько строк). При вводе, нажимая клавишу **Enter,** можно продолжать набор текста с новой строки.

ScrollBars (Линейки прокрутки) принимает значения *О* (линеек прокрутки в текстовом окне нет), *7* (появляется горизонтальная линейка прокрутки), 2 (появляется вертикальная линейка прокрутки), *3* (появляются горизонтальная и вертикальная линейки прокрутки).

SelLenght (Количество, выделенных символов), **SelStart** (Начало выделеннго блока), **SelText** (Выделенный текст) доступны только при написании текста программы и не содержатся в списке свойств. Определяют соответственно число выделенных символов, позицию первого символа выделяемого блока (О—первый символ, 1 - второй и т.д.), выделенную строку.

Text (Текст) также не содержится в списке свойств. Определяет вводимую или выводимую в текстовое окно строку.



Из событий чаще используются:

Change (Изменение) вызывается изменением свойства **Text** пользователем или программно. Набор каждого нового символа определяет данное событие и вызов процедуры его обработки.

Click (Щелчок) или **DblClick** (Двойной щелчок) мышью в текстовом окне вызывает соответствующие процедуры обработки события *Имя_окна_С\\с\(.* или Ямя_о/сна_ОЬ1СИсР.

GotFocus (Установка фокуса) и **LostFocus** (Уход из фокуса) определяются тем, что в Windows-приложениях в каждый момент может быть активен только один объект (объект находится в фокусе). События определяются соответственно активизацией и дезактивизацией объекта, в данном случае - активизацией или дезактивизацией текстового окна.

Метод **SetFocus** позволяет установить курсор в выбранное текстовое окно. Синтаксис: ***[имя текстового окна.}*** **SetFocus**

Пример программы.

Разместим на форме управляющие элементы: два текстовых окна (OKHOI и окно2) и метку (рис. 2.3).

Свойства управляющих элементов заданы по умолчанию. Введем для первого текстового окна (Text 1) процедуры

обработки событий GotFocus и LostFocus (рис. 2.4). В первой и второй процедурах первые два оператора изменяют размеры текстового окна. Третий оператор изменяет цвет фона окна (функция QBColor с параметром *0* определяет черный цвет фона окна, параметр 7 определяет белый цвет). Четвертый оператор определяет текст метки.

Щелкнув мышью по кнопке **Run** Главного меню запустим программу. Если щелкнуть мышью по первому текстовому окну (активизировать данный управляющий элемент), то первоначальная форма на рис. 2.3 примет другой вид (рис. 2.5). Щелчок мышью по второму окну (первое окно становится не активным) изменяет вид формы (рис. 2.6).

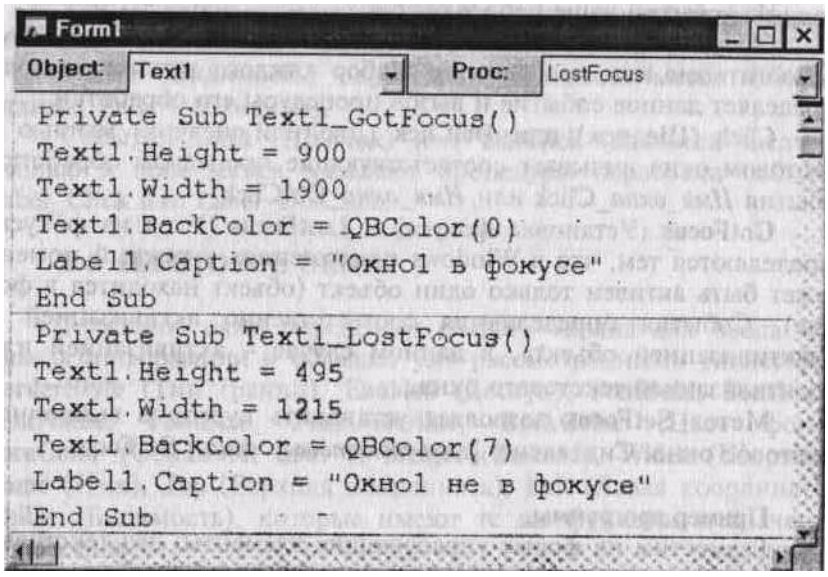


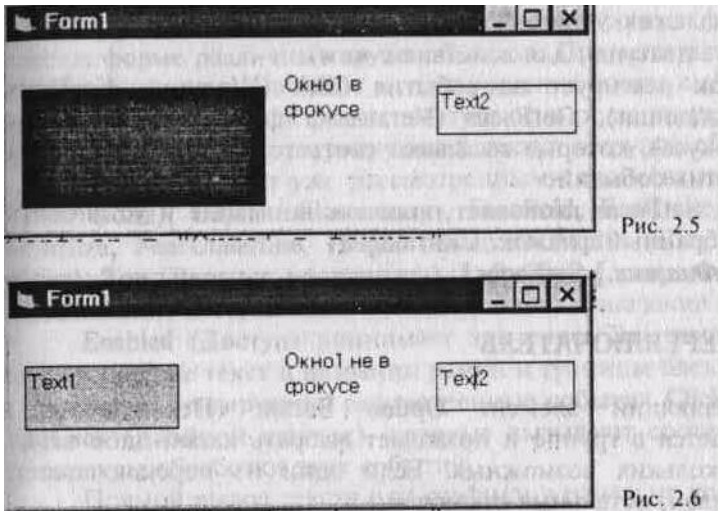
Рис. 2.4

2.5. Командная кнопка

Command button (Командная кнопка) обычно активизирует какую-то операцию (вызывает выполнение команды) и обладает уже рассмотренными свойствами **BackColor** (Цвет фона). **Caption** (Заголовок), **Enabled** (Доступ). **FontBold, FontItalic, FontName, FontSize, FontUnderline, Height** (Высота), **Width** (Ширина), **Name** (Имя), **Top** (Верхняя координата). **Left** (Левая координата). **Visible** (Видимость), которые имеют те же функции и значения параметров.

Cancel (Отмена) принимает значения *True* или *False.* Присвоение этому свойству значение *True* даст при нажатии клавиши **Esc** тот же эффект, что и щелчок мышью по кнопке. Значение *True* может иметь только одна командная кнопка на форме.

Default (По умолчанию) принимает значения *True* или *False.* Присвоение этому свойству значение *True* даст при нажатии клавиши Enter тот же эффект, что и щелчок мышью по кнопке (например, щелчок мышью по кнопке ОК диалогового окна эквивалентен нажатию клавиши Enter). Значение *True* может иметь только одна командная кнопка на форме.



Командная кнопка реагирует на уже рассмотренные события **Click** (Щелчок), **Keypress** (Нажатие клавиши), **GotFocus** (Установка фокуса) и **LostFocus** (Уход из фокуса), которые вызывают соответствующие Процедуры обработки этих событий.

Часто используется метод SetFocns, который позволяет установить курсор в выбранную командную кнопку. Синтаксис:

[имя_Командной кнопки.] SetFocus.

2.6. ФЛАЖОК

Управляющий элемент **Check box** (Флажок) устанавливает или сбрасывает определенный параметр: если на квадратике флажка обзднетач-зттачок X, то-параметр включен (активен), нет значка - параметр выключен. Щелчок мышью по флажку устанавливает или сбрасывает параметр. Любой флажок функционально независим от других флажков.

Флажок обладает уже рассмотренными свойствами:

BackColor (Цвет фона). **Caption** (Заголовок), **Enabled** (Доступ), **FontBold, FontItalic, FontName, FontSize, FontUnderline. Height** (Высота), **Width** (Ширина), **Name** (Имя). **Top** (Верхняя координата), **Left** (Левая координата). **Visible** (Видимость), которые имеют те же функции и значения параметров. **Value** (Состояние) принимает значения: *0* - флажок не помечен; 7 - флажок установлен; 2 - флажок затенен (имеет блеклый вид и не доступен для действия с ним).

Флажок реагирует на события **Click** (Щелчок), **Keypress** (Нажатие клавиши), **GotFocus** (Установка фокуса) и **LostFocus** (Уход из фокуса), которые вызывают соответствующие процедуры обработки этих событий.

Метод **SetFocus** позволяет привлечь внимание и установить курсор в выбранный флажок. Синтаксис:

[имя_ Флажка.] SetFocus.

2.7. ПЕРЕКЛЮЧАТЕЛЬ

Управляющий элемент **Option Button** (Переключатель) обычно задается в группе и позволяет выбрать какой-либо вариант из нескольких возможных. Если один из переключателей группы активен, остальные отключены.

Переключатель обладает уже рассмотренными свойствами:

BackColor (Цвет фона), **Caption** (Заголовок), **Enabled** (Доступ), **FontBold, FontItalic, FontName, FontSize, FontUnderline, Height** (Высота), **Width** (Ширина), **Name** (Имя), **Top** (Верхняя координата), **Left** (Левая координата), **Visible** (Видимость), которые имеют те же функции и значения параметров.

Value (Состояние) принимает значения: *True* - переключатель активен, в центре кружка жирная точка; *False* - не активен.

Переключатель реагирует на уже рассмотренные события **Click** (Щелчок), **DblClick** (Двойной щелчок). **Keypress** (Нажатие клавиши), **GotFocus** (Установка фокуса) и **LostFocus** (Уход из фокуса), которые вызывают соответствующие процедуры обработки этих событий.

Метод **SetFocus** применяется аналогично рассмотренному для флажка.

Прямой вывод текста или графики на командные кнопки, флажки и переключатели не допускается.

2.8. РАМКА

Управляющий элемент **Frame** (Рамка) служит для разделения на форме различных групп объектов. Применительно к переключателям, рамки влияют и на поведение кнопок. Для остальных элементов формы рамки выступают в роли визуального разделителя и функции, регулирующей доступ к группе объектов.

Рамка обладает уже рассмотренными свойствами: **BackColor** (Цвет фона). **Caption** (Заголовок), **FontBold, FontItalic, FontName, FontSize, FontUnderline, Height** (Высота), Width (Ширина), **Name** (Имя), **Top** (Верхняя координата), **Left** (Левая координата), **Visible** (Видимость), которые имеют те же функции и значения параметров.

Enabled (Доступ) принимает значение *True* или *False* (при значении *False* текст в названии рамки и границы блекнут).

Рамка реагирует на рассмотренные события **Click** (Щелчок), **DblClick** (Двойной щелчок), которые вызывают соответствующие процедуры обработки этих событий.

Прямой вывод текста или графики в рамку не допускается. **Пример программы.**

Гипотетическая форма предназначена для управления в диалоге платежами (оплата наличными или с использованием кредитных карточек, вид используемой карточки, оплата рублями). Она содержит командную кнопку для начала работы с формой, рамку, объединяющую два

переключателя для выбора наличной или безналичной оплаты, рамку, выделяющую флажок для определения оплаты рублями, рамку, объединяющую два переключателя для выбора вида кредитной карточки, и имеет вид, показанный на рис. 2.7.

Названия управляющих элементов (Начало, Метод платежа, Наличными, Кредитная карта, Наличными, Рубли, Кредитная карта, Мостбанк, VISA) заданы значением свойств **Caption** перечисленных управляющих элементов формы.

Пои выводе формы на экран все элементы управления кроме командной кнопки должны быть неактивными (не реагировать на какие-либо события). Для этого свойству Enabled всех элементов управления, кроме командной кнопки, в окне свойств каждого из них присваивается значение *False.*

Рис. 2.7

Щелчок мышью по кнопке **Начало** должен активизировать рамку выбора метода платежа и находящиеся в ней переключатели. Щелчок мышью по переключателю **Наличными** должен активизировать соответствующую рамку и находящийся в ней флажок. Щелчок мышью по переключателю **Кредитная карта** должен активизировать соответствующую рамку и находящиеся в ней переключатели (рамка **Наличными** и находящийся в ней флажок должны при этом стать не активными).

Такая логика работы диалоговой формы обеспечивается заданием следующего программного кода (используемые имена управляющих элементов Commandl (Управляющая кнопка), Checkl (Флажок), Framel, Frame2, Frame3 (Рамки), Optioni, Option2, Option3, Option4 (Переключатели) вводятся Visual Basic по умолчанию):

```
Private Sub Commandl Click() Framel.Enabled = True Optioni.Enabled = True
    Option2.Enabled = True
End Sub
Private Sub Optionl_Click()
Frame2.Enabled = True
Checkl.Enabled = True
Frame3.Enabled = False
Option3.Enabled = False
Option4.Enabled = False End Sub
Private Sub Option2_Click()
Frame3.Enabled = True
Option3.Enabled = True
Option4.Enabled = True
Frame2.Enabled = False
Checkl.Enabled = False End Sub
```

Процедура Sub Commandl_Click обработки события, связанного с щелчком мыши по командной кнопке, обеспечивает активизацию рамки Метод платежа и расположенных в ней переключателей.

Процедура Sub Optionl_ciick обеспечивает активизацию рамки «Наличные» и расположенный в ней флажок «Рубли». Процедура Sub Option2_ciick обеспечивает активизацию рамки

«Кредитная карта» и расположенных в ней переключателей. Последние два оператора в каждой процедуре обеспечивают деакти-визацию соответственно рамок «Кредитная карта» и «Наличные» и элементов в них, если они ранее были активны.

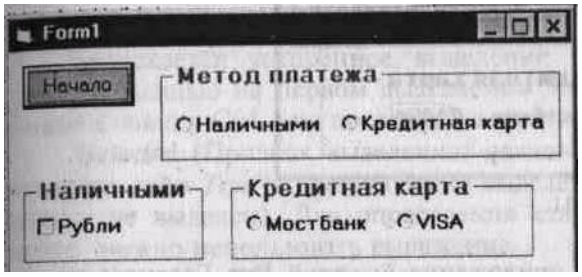


Рис. 2.8

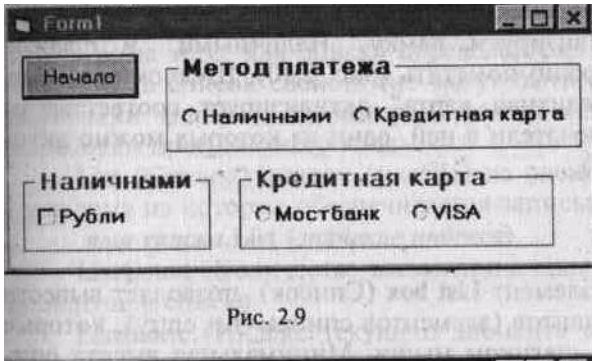


Рис. 2.9

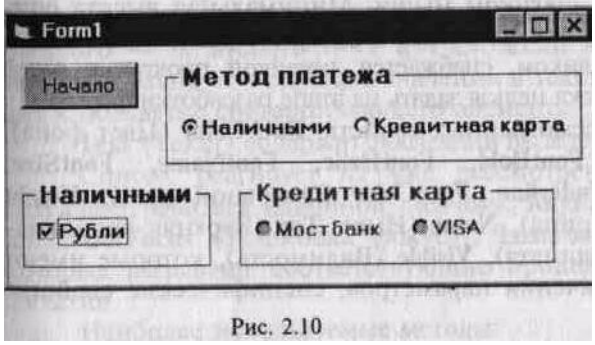


Рис. 2.10

Запустив данное приложение кнопкой **Run** Главного меню, увидим исходную форму (неактивные элементы обозначены бледно) (рис. 2.8). Щелкнув мышью по командной кнопке «Начало», увидим форму (рис. 2.9). Щелкнув мышью по переключателю «Наличными», активизируем рамку «Наличными» и флажок «Рубли», который можно пометить (рис. 2.10). Щелчок мыши по переключателю «Кредитная карта» активизирует соответствующую рамку и переключатели в ней, один из которых можно активизировать (рис. 2.11).

2.9. СПИСОК

Управляющий элемент **List box** (Список) позволяет вывести на экран список вариантов (элементов списка—list entry), которые могут быть выбраны щелчком мыши. Минимальная высота списка составляет три строки. Длинный список, который не может быть выведен на экран целиком, снабжается линейкой прокрутки (scroll bars). Содержимое списка нельзя задать на этапе разработки формы.

Кроме уже рассмотренных свойств: BackCoIor (Цвет фона), Enabled (Доступ), FontBold, FontItalic, FontName, FontSize, FontStrikethru, FontUnderline, ForeColor (Основной цвет), Height (Высота), Width (Ширина), Name (Имя), Top (Верхняя координата), Left (Левая координата), Visible (Видимость), которые имеют те же функции и значения параметров, специфические свойства следующие:

Columns (Колонки) по умолчанию принимает значение *0* и элементы списка располагаются в одну колонку. Если значение больше 0, то элементы списка размещаются в соответствующем

числе колонок. Необходимая линейка прокрутки добавляется автоматически.

MultiSelect (Множественное выделение) определяет, сколько элементов можно одновременно выделить в списке. Принимает значения: *0* — *None* - допускается выделение толька одного дле-мента; / — *Simple* - допускается выделение нескольких элементов;

2 - допускается ускоренное выделение нескольких элементов (щелчок мышью на первом выделяемом элементе, а затем, удерживая клавишу Ctrl, —на последний элемент).

Selected (Признак выделения) является массивом, значения которого либо *True* (элемент списка выделен), либо *False* (элемент списка не выделен). Для определения статуса конкретного элемента можно использовать выражение

имя списка. Selected (значение индекса).

Sorted (Сортировка) принимает значения *True* (элементы в списке располагаются по английскому алфавиту) или *False* (элементы в списке располагаются в порядке их ввода).

Кроме того, для списка определен ряд свойств, которые не включены в список свойств (не могут быть определены на этапе разработки формы), но которые можно использовать на этапе разработки программного кода:

List (Список) список (массив) из элементов списка, доступ к каждому из которых обеспечивается записью:

имя cnucica.List (значение индекса).

ListCount (Количество элементов в списке) равно количеству элементов в списке.

ListIndex (Индекс текущего элемента в списке) определяет номер последнего выделенного подсветкой элемента в списке (первого — *0,* второго — / и т.д.). Если элемент не выделен, значение равно /. Установка значения в тексте программы приводит к подсветке соответствующего элемента.

Text (Текст) содержит последний выделенный элемент списка.

Список реагирует на уже рассмотренные события: **Click** (Щелчок), **DblClick** (Двойной щелчок), **KeyPress** (Нажатие клавиши), **GotFocus** (Установка фокуса), **LostFocus** (Потеря фокуса), которые вызывают соответствующие процедуры обработки этих событий.

Наиболее используемые методы:

AddItem включает элемент (строку текста) в список. Синтаксис:

*имя **спнс/со.AddItem** строка {.индекс}.*

Рис. 2.11

Если индекс отсутствует, то элемент (строка) автоматически ставится в конец списка. Если значение свойства **Sorted** равно *True,* то индекс должен отсутствовать—элемент размещается в соответствии с алфавитным порядком.

Clear удаляет из списка все элементы. Синтаксис:

имя списка.(Леш

Removelen убирает из списка элемент с заданным индексом. Синтаксис:

*имя списка. **Removelen** индекс.*

2.10. КОМБИНИРОВАННЫЙ СПИСОК

Управляющий элемент **Combo Box** (Комбинированный список) объединяет текстовое окно (поле ввода) с обычным списком в один элемент управления. Комбинированный список не позволяет размещать элементы в несколько колонок.

Комбинированный список обладает уже рассмотренными свойствами: **BackColor** (Цвет фона). **Enabled** (Доступ), **FontBold, FontItalic, FontName, FontSize, FontStrikethru, FontUnderline, ForeColor** (Основной цвет). **Height** (Высота), Width (Ширина), **Name** (Имя), **Top** (Верхняя координата), **Left** (Левая координата), Visible (Видимость), **Sorted** (Сортировка), которые имеют те же функции и значения параметров.

Style (Тип) определяет тип комбинированного списка и принимает значения *0* (раскрывающийся комбинированный список), *7* (простой комбинированный список), 2 (раскрывающийся список).

Раскрывающийся список вначале отображает одну строку со стрелкой справа. Щелчок мыши на стрелке открывает весь список, в котором можно выбрать один из элементов, как в обычном списке. Примером раскрывающегося списка является список Object в окне Properties. Не позволяет вводить информацию в текстовое окно.

Простой комбинированный список содержит поле для ввода и нераскрывающийся список под этим полем в окне постоянного размера. Позволяет либо вводить текст в поле, либо выбирать элемент из списка.

Раскрывающийся комбинированный список выглядит как раскрывающийся, но позволяет и выбирать элемент и вводить текст.

Text определяет либо текст выделенного в списке элемента, либо текст поля ввода.

Комбинированный список реагирует на уже рассмотренные события **Click** (Щелчок), **DblClick** (Двойной щелчок), **Keypress** (Нажатие клавиши), **GotFocus** (Установка фокуса), **LostFocus** (Потеря фокуса), которые вызывают соответствующие процедуры обработки этих событий.

Методы:

AddItem включает элемент (строку текста) в комбинированный список. Синтаксис:

*имя **списка AddItem** строка [, индекс}*

Если индекс отсутствует, то элемент автоматически ставится в конец комбинированного списка. Если значение свойства **Sorted** равно *True,* то индекс должен отсутствовать - элемент размещается в соответствии с алфавитным порядком.

Clear удаляет из списка все элементы. Синтаксис:

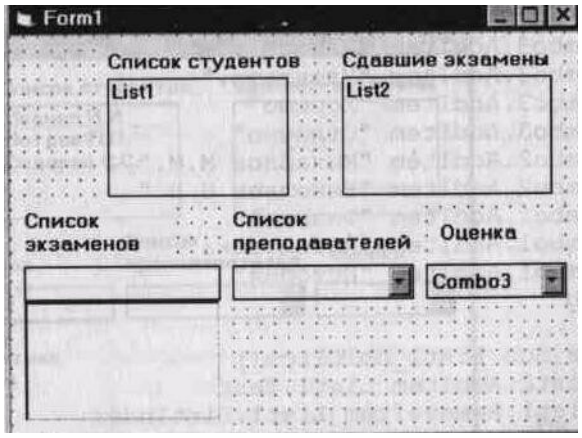
*имя списка. **Clear***

Removelen убирает из списка элемент с заданным индексом. Синтаксис:

*имя списка. **Removelen** индекс*

Пример программы.

Создадим новый проект и разместим на форме следующие управляющие элементы: два списка наверху и три комбинированных списка внизу, снабдив их заголовками (метки).



Имена элементов задаются по умолчанию (последовательно слева направо списки: *List!* и *List2,* комбинированные списки:

Combol, Combo 2, СотЬо3).

Значения свойства Style: для комбинированного списка *Combo!* зададим *I*, для *Combo2 - 0,* для *Combo 3 - 2.* Размер окна для комбинированного списка *Combol* нужно определить при размещении элемента на форме. Форма имеет вид, показанный на рис. 2.12.

Определим следующие процедуры обработки событий:

Listl.AddItem List2.Text List2.RemoveItem List2.ListIndex End Sub

Процедура Form_Load обеспечивает формирование исходной информации списков. Процедуры Listl_DblClick и List2_DblClick обеспечивают при двойном щелчке мыши по одному из элементов списка перенос этого элемента в другой список. Процедуры Combo l_Key Press и Co mbo2_Key Press обеспечивают ввод набранного текста в строке ввода при нажатии клавиши **Enter** в соответствующий список. После запуска программы выводится исходная форма (рис. 2.13). Двойной щелчок мыши по одному из элементов верхних списков переносит этот элемент в другой список.

Элементы левого комбинированного списка сразу же выведены в окно. Имеется возможность выбора элемента из списка и ввода нового элемента в список, набрав его в строке ввода.

Элементы центрального комбинированного списка появляются в раскрывающемся окне после щелчка мыши по стрелке. Имеется возможность выбора элемента из списка и ввода нового элемента в список, набрав его в строке ввода.

Элементы правого комбинированного списка появляются в раскрывающемся окне после щелчка мыши по стрелке. Имеется возможность только выбора элемента из фиксированного списка (рис. 2.14, 2.15).

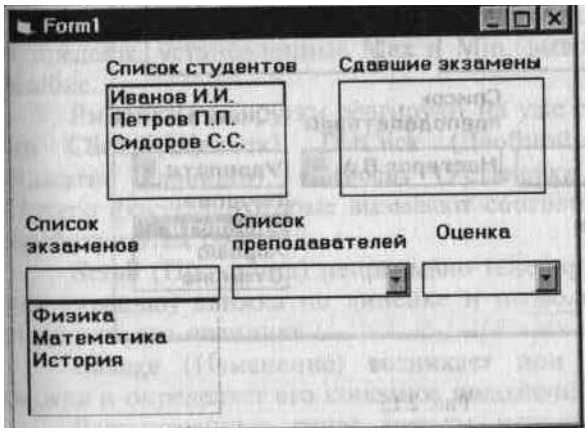


Рис. 2.13

```
Private Sub Combol_KeyPress(keyascii As Integer)
If keyascii = 13 Then
combol.Addltem combol.Text
End If End Sub
Private Sub Combo2_KeyPress(keyascii As Integer)
If keyascii = 13 Then
combo2.Addltern combo2.Text
End If End Sub
Private Sub Form_Load ()
Listl.AddItem «Иванов И.И.» Listl.AddItem «Петров П.П.» Listl.AddItem «Сидоров С.С.»
сотЬо3.AddItem «Неудовл.» combo 3.Add!tern «Удовлетв.» combo3.Addltem «Хорошо» сотЬо3.Addltem «Отлично»
combo2.Addltem «Михайлов М.М.» combo2.Addltem «Николаев Н.Н.» combol.Addltem «Физика» combol.Addltem
«Математика» combol.Addltem «История» End Sub
Private Sub Listl_DblClick() List2.Addltem Listi.Text Listi.Removeltem Listi.ListIndex
End Sub
Private Sub List2_DblClick()
```

2.11. ЛИНЕЙКИ ПРОКРУТКИ

Horizontal Scroll Bar и Vertical Scroll Bar (Горизонтальные и вертикальные линейки прокрутки) позволяют задавать значение какой-либо величины в определенном диапазоне и узнавать значение этой величины по позиции движка на линейке.

Линейки прокрутки обладают уже рассмотренными свойствами: **Enabled** (Доступ), **Height** (Высота), **Width** (Ширина), **Name** (Имя), **Top** (Верхняя координата), **Left** (Левая координата), **Visible** (Видимость), которые имеют те же функции и значения параметров.

LargeChange (Постраничное изменение) определяет изменение текущей позиции движка и соответственно задаваемую величину (Value) при щелчке внутри линейки прокрутки.

Max (Максимум) определяет максимальное значение задаваемой величины (соответствует крайней правой или нижней позиции движка). Диапазон значений: от -32768 до 32767.

Min (Минимум) определяет минимальное значение задаваемой величины (сответствует крайней левой или верхней позиции движка). Диапазон значений: от -32768 до 32767.

SmallChange (Построчное изменение) определяет изменение текущей позиции движка и соответственно задаваемую величину (Value) при щелчке

по одной из стрелок направления прокрутки.

Value (Текущая позиция) отражает текущее значение и позицию движка. При изменении свойства программным путем движок перемещается в соответствующую позицию. При выходе за пределы, установленные Max и Min, выводится сообщение об ошибке.

Линейки прокрутки реагируют на уже рассмотренные события **Click** (Щелчок), **DblClick** (Двойной щелчок), **Keypress** (Нажатие клавиши), **GotFocus** (Установка фокуса), **LostFocus** (Потеря фокуса), которые вызывают соответствующие процедуры обработки этих событий.

Scroll (Прокрутка) непрерывно генерируется при перемещении (мышью) движка по линейке и позволяет динамически отслеживать его позицию.

Change (Изменение) возникает при изменении позиции движка и определяет его конечное положение.

Рассмотренные ранее методы неприменимы к линейкам прокрутки.

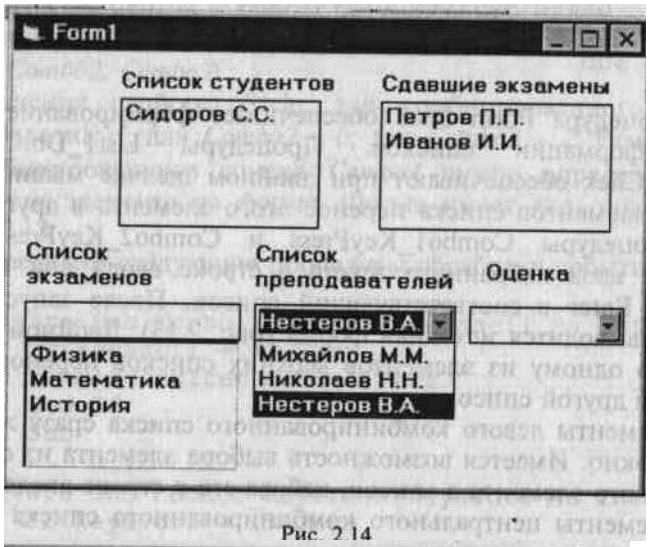


Рис. 2.14

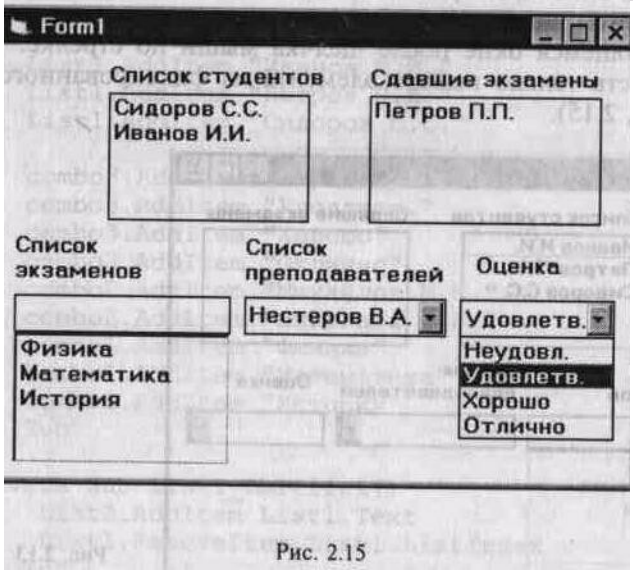


Рис. 2.15

Пример программы.

Откроем новую форму и разместим на ней два текстовых окна и горизонтальную линейку прокрутки (рис. 2.16).

В окне Properties (Свойства) для объекта Hscrolll (имя присваивается по умолчанию) зададим следующие значения свойств:

LargeChange = 10, Max = 100, Min = 0, SmallChange = 5.

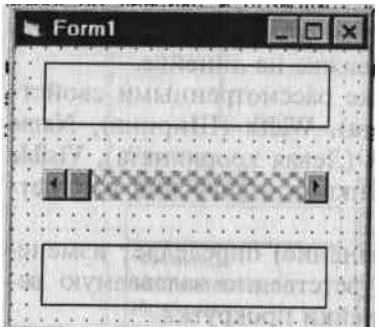


Рис. 2.16

Определим процедуры обработки событий *HScrolll_Change* и *HScrolll_Scroll:*

```
.Private Sub HScrolll_Change()
Textl.Text = HScrolll.Value End Sub
Private Sub HScrolll_Scroll()
Text2.Text = HScrolll.Value End Sub
```

Событие Change отражается в верхнем окне, событие Scroll - в нижнем окне. После запуска программы щелчок мыши по полю вызывает перемещение движка и изменение значения в верхнем окне на 10 единиц (Large Change), щелчок мыши по стрелкам вызывает перемещение движка и изменение значения в верхнем окне на 5 единиц (Small Change). Эти изменения не отражаются в нижнем окне (рис. 2.17). Перемещение движка мышью вызывает изменение значения, которые отражаются в обоих окнах (рис. 2.18).

2.12. ПРИМЕР СОЗДАНИЯ ФОРМЫ

Создадим форму со следующими элементами управления (рис. 2.19):

- Комбинированный список типа 1, содержащий названия фирм-производителей автомобилей.

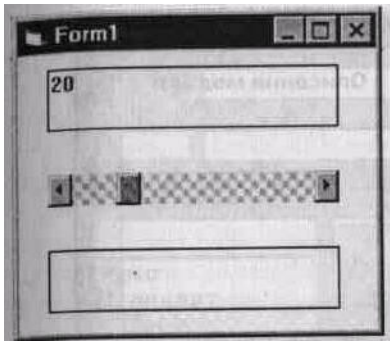


Рис.2.17

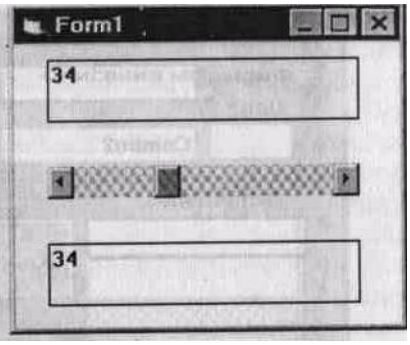
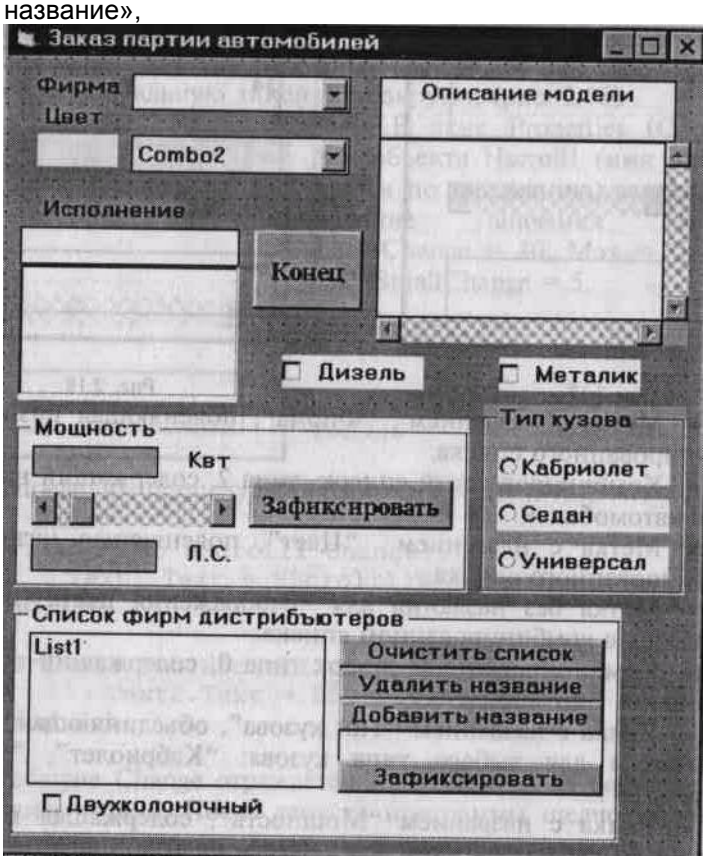


Рис. 2.18

- Метка с названием «Фирма», поясняющая назначение комбинированного

списка.

- Комбинированный список типа 2, содержащий названия цветов автомобилей.
- Метка с названием «Цвет», поясняющая назначение комбинированного списка.
- Метка без названия для отображения цвета, который выбирается в комбинированном списке.
- Комбинированный список типа 0, содержащий названия исполнения автомобилей.
- Рамка с названием «Тип кузова», объединяющая три переключателя для выбора типа кузова: «Кабриолет», «Седан», «Универсал».
- Рамка с названием «Мощность», содержащая горизонтальную линейку прокрутки для изменения мощности двигателя, две метки для отображения мощности двигателя в киловаттах и лошадиных силах, две поясняющие метки с названиями **«кВт»** и **«Л.С.»** (для горизонтальной линейки прокрутки на этапе проектирования формы задаются значения свойств: минимальное значение - 30 (min = 30), максимальное значение - 500 (max == 500), текущее значение - 75 (value = 75), малый шаг - 1 (SmallChange = 1), большой шаг - 10 (LargeChage = 10), недоступна (enabled = false)) и командную кнопку «Зафиксировать».
- Рамка с названием «Список фирм - дистрибьюторов», объединяющая список, флажок, метку с названием «Двухколоночный», поясняющую назначение флажка, и четыре командных кнопки («Очистить список», «Удалить название «, «Добавить название», «Зафиксировать»).

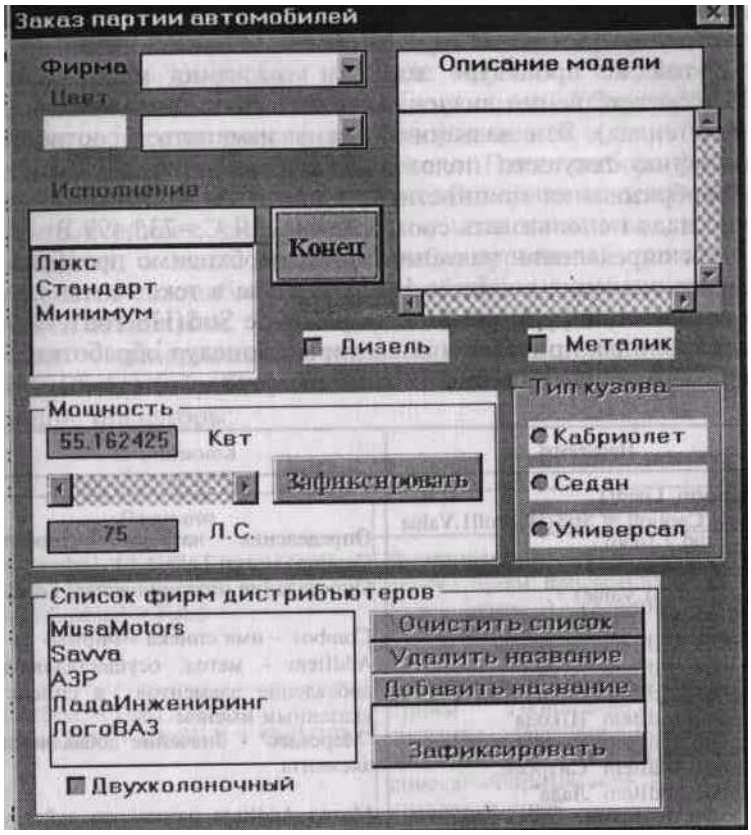


- **Флажки с названиями** «Дизель» и «Металлик».
- Текстовое окно для описания модели с горизонтальной и вертикальной линейками прокрутки.
- Метка с названием «Описание модели».
- Командная кнопка «Конец».

При проектировании формы значения свойств, определяющих основные цвета, цвета фона, вид и размер шрифтов, размеры и расположение управляющих элементов, задаются разработчиком или определяются по

умолчанию.

Элементы управления и процедуры обработки событий должны обеспечить следующую логику работы с формой.



На этапе загрузки формы заполняются комбинированные списки «Фирма», «Цвет», «Исполнение», а также простой список «Фирм продавцов»:

Список «Фирмы»	Список «Цвет»	Список «Исполнение»	Список «Фирмы-продавцы»
Мерседес	Красный	Люкс	АЗР
Ауди	Белый	Стандарт	Musa Motors
Фольксваген	Черный	Минимум	ЛадаИнжинеринг
Шкода	Синий		Sawa
Рено	Зеленый		ЛогоВАЗ
Ситроен			
Лада			

Заполнение производится в процедуре Form_Load с помощью методов Additem.

В той же процедуре задаются названия меток Label4 и Label5 (соответственно значение мощности в киловаттах и в лошадиных силах). Эти названия должны изменяться соответственно значению текущего положения движка линейки прокрутки. Для преобразования мощности из киловатт в лошадиные силы и обратно надо использовать соотношение:

1Л.С.=735,499 Вт.

Для определения названия метки необходимо преобразовать численное значение свойства HScrolll. Value в текст с использованием встроенной функции Str$ (выражение Str$(HScrolll. Value)).

В таблицах приведены операторы процедур обработки событий примера и поясняющие их комментарии.

Процедура	Комментарии
Sub Form Load()	
LabeM.Caption = Str$(HScrolll. Value * 735.499 / 1000)	Определение названия (свойство
Label5.Caption = Str$(HScrolll. Value)	Caption) метки Label4 Определение названия метки Label5
Combo 1.Add Item «Мерседес»	Combol - имя списка «Фирмы»
Combo 1.Add I tern «Ауди» Combo 1.Add Item «Фольксваген» Combol.Additem «Шкода» Combo 1.Additem «Рено» Combol.Additem «Ситроен» Combol.Additem «Лада»	Additem - метод, осуществляющий добавление элементов в список с указанным именем «Мерседес» - Значение добавляемого элемента
	Метод Additem одинаково действует
Combo2.AddItem «Красный» Combo2.AddItem «Белый» Combo2.AddItem «Черный»	и для комбинированных списков и для простых
Combo2.AddItem «Синий»	
Combo2.AddItem «Зеленый»	
СотЬо3.Additem «Люкс»	
Combo3.AddItern «Стандарт»	
Combo3.AddItem «Минимум»	
Listl.Additem “Musa Motors”	
Listl.Additem «ЛадаИнжинеринг»	
Listl.Additem “Sawa”	
Listl.Additem “А3Р”	
Listl.Additem «ЛоговАЗ»	
End Sub	

В результате выполнения процедуры выводится форма (рис. 2.20). На форме активны только четыре элемента: метка с названием «Фирма»,

командная кнопка «Конец», текстовое окно «Описание модели» и комбинированный список «Фирма».

При щелчке по списку он разворачивается и из него можно выбрать название фирмы-производителя. После выбора фирмы её название заносится в текстовое окно «Описание модели», комбинированный список фирм становится неактивным вместе с меткой «Фирма», а метка «Цвет», комбинированный список для выбора цветов автомобиля и метка отображения выбранного цвета становятся активными.

Для выполнения указанных действий процедура Combo l_Click() (обработка события щелчок по списку) содержит следующие операторы:

Процедура	Комментарии
Sub Combo lClick()	Деактивизация комбинированного
Combol. Enabled == False	списка путем присвоения значения
Label 1. Enabled = False	False(ложь) свойству Enabled (Активность)
	Добавление к значению текстовой
Text 1.Text = Text 1.Text + Combol.Text + Chr$(13) + Chr$(10)	строки «Описание модели» значения текущего (выбранного) элемента из комбинированного
	списка «Фирмы». «+» - операция
	сцепления строк
	Chr$(10), Chr$(13) - функция, ре
	зультатом вычисления которой
	является символ с кодом 10 и 13
	13, 10 - комбинация символов для
	перевода курсора в начало следую
	щей строки текстового окна.
	Активизация комбинированного
	списка «Цвета» и его метки
Combo2. Enabled = True	
Label2. Enabled = True	
End *Sub*	

После завершения выполнения данной процедуры на форме изменятся

элементы (рис. 2.21).

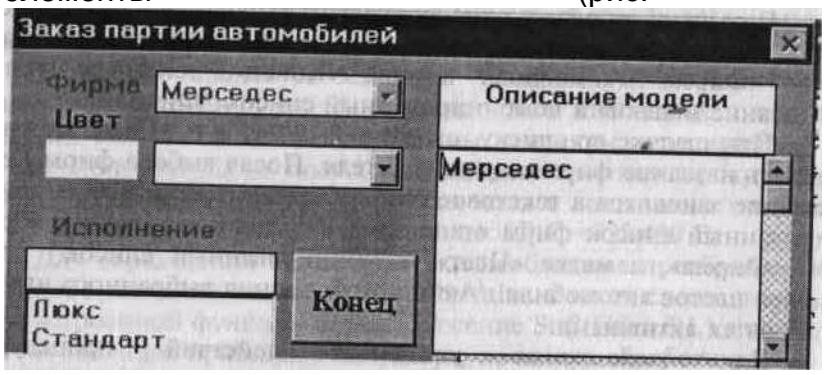


Рис. 2.21

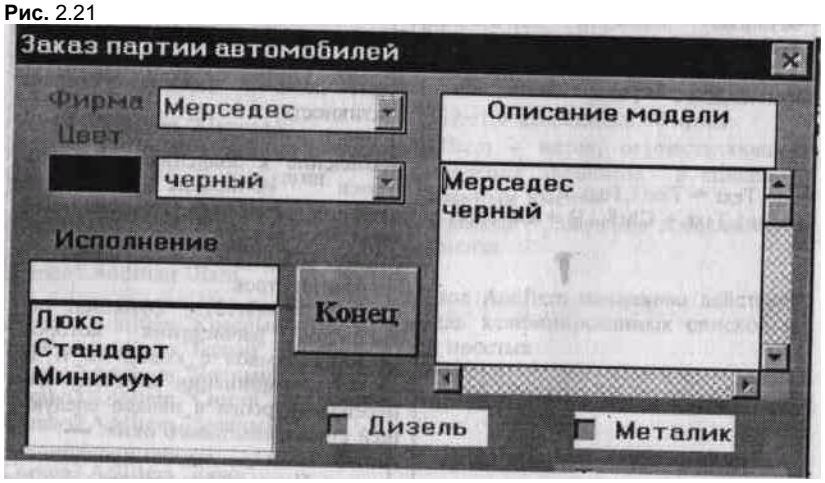


Рис. 2.22

На форме активны только четыре элемента: метка с названием «Цвет», командная кнопка «Конец», комбинированный список для выбора цвета и текстовое окно «Описание модели». При щелчке по списку он разворачивается и из него можно выбрать цвет для машины.

После выбора цвета его название заносится в текстовое окно «Описание модели», комбинированный список цветов становится неактивным вместе с меткой «Цвет», а метка «Исполнение» и комбинированный список для выбора варианта исполнения автомобиля становятся активными (рис. 2.22).

Для выполнения указанных действий процедура Combo2_Click() (обработка события «Щелчок по списку») содержит следующие операторы:

Процедура	Комментарии

Sub Combo2_Click()
 	
Combo2.Enabled = False
Label2.Enabled = False
 	
Text1.Text = Text1.Text + Combo2.Text + Chr$(13) + Chr$(10)
 	
If Combo2.Text = "Красный" Then
Label8.BackColor = код цвета
ElseIf Combo2.Text = "Белый" Then
Label8.BackColor = код цвета
ElseIf Combo2.Text = "Черный" Then
Label8.BackColor = код цвета
ElseIf Combo2.Text = "Синий" Then
Label8.BackColor = код цвета
ElseIf Combo2.Text = "Зеленый" Then
Label8.BackColor = код цвета
End If
 	
Combo3.Enabled = True
Label3.Enabled = True
 	
End Sub
	Деактивизация списка цветов и метки с названием «Цвет»
	
	Запись названия цвета в текстовое окно и подготовка позиции для ввода следующей строки
	
	Проверка выбранного цвет из списка и, в зависимости от выбранного цвета, изменение цвета фона метки с названием «Цвет» (коды цветов приведены ниже в таблице)
	
	Активизация комбинированного списка вариантов исполнения и метки с названием «Исполнение»

Коды цветов приведены в следующей таблице:

Цвет	Кол цвета
	&H 000000 FF&
	&HOOFFOOOO&
	&HOOCOCOCO&
	&HOOOOFFOO&
Голубой	&HOOFFFFOO&
Черный	&HOOOOOOOOO&
Белый	&HOOFFFFFF&
Желтый	&HOOOOFFFF&

Коды цветов заданы в шестнадцатеричной системе счисления, в которой для представления чисел используется 16 символов: цифры от 0 до 9 и буквы от A (эквивалент 10) до F (эквивалент 15). Значимость каждой п-й позиции в шестнадцате-ричном числе возрастает пропорционально степени *\6”n.* Например, десятичное число 419 в шестнадцатеричном представлении записывается 1А3:

*1*16”2 + 10*16»! + 3*16^0 = 419.*

В двоичной системе число 15 записывается 1111, т.е. любой символ

шестнадцатеричного числа можно кодировать четырьмя битами. Память компьютера разделена на байты (1 байт равен 8 битам) и каждый байт может хранить любое из 256 *(16^1)* шестна-дцатеричных чисел: от 0 до FF. Такое компактное представление информации на основе шестнадцатеричной системы счисления обусловливает ее широкое использование в программировании.

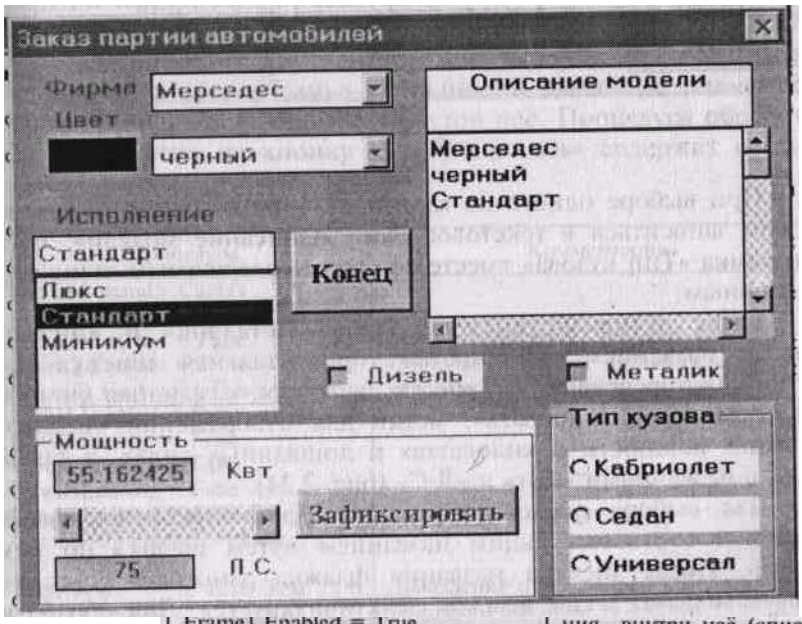
Шестнадцатеричному числу в Visual Basic предшествуют символы &H.

Как видно из рис. 2.22, на форме активны четыре элемента:

комбинированный список, соответствующий метке с названием «Исполнение», командная кнопка «Конец» и текстовое окно «Описание модели». При щелчке по элементу комбинированного списка можно выбрать соответствующий вариант исполнения для машины. После выбора варианта исполнения его название заносится в текстовое окно «Описание модели», комбинированный список становится неактивным вместе с меткой, а рамка с названием «Тип кузова» и три переключателя внутри неё - активными (рис. 2.23). Для выполнения указанных действий процедура Combo3_Click() (обработка события «Щелчок по списку») содержит следующие операторы:

Процедура	Комментарии
Sub Combo3_Click() Combo3.Enabled = False Label3.Enabled = False	Деактивизация комбинированного списка «Исполнение» и его метки
Text1.Text = Text1.Text + Combo3.Text + Chr$(13) + Chr$(10)	Дополнение текстового окна «Описание модели» выбранным значением исполнения
Frame4.Enabled = True Option1.Enabled = True Option2.Enabled = True Option3.Enabled = True End Sub	Активизация рамки с названием «Тип кузова» и переключателей

В результате выполнения данной процедуры на форме произойдут изменения, показанные на (рис. 2.23). На форме активны шесть элементов: кнопка «Конец», рамка «Тип кузова», текстовое окно «Описание модели» и три переключателя в рамке.




```
Frame1.Enabled = True
List1.Enabled = True
Text2.Enabled = True
Command4.Enabled = True
Check3.Enabled = True
Command1.Enabled = True
End Sub
```

ния внутри неё (список фирм, текстовое окно для ввода имени нового продавца, Флажок с названием «Двухколоночный», командная кнопка с названием «Очистить список», командная кнопка с названием «Удалить название», командная кнопка с названием «Добавить название» и командная кнопка с названием «Зафиксировать»)

Рис. 2.23

В результате форма примет вид, показанный на рис. 2.25. На форме активны: текстовое окно с названием «Описание модели», рамка «Список фирм-дистрибъютеров», командная кнопка с названием «Очистить список», командная кнопка с названием «Добавить название», текстовая строка для добавления нового имени в список продавцов и кнопка «Конец».

При щелчке по флажку «Двухколоночный» список в зависимости от состояния флажка преобразуется в двухколоночный и обратно. Реализующая процедура обработки события Check3_Click() содержит следующие операторы:

Процедура	Комментарии
Sub Check3 Click() IfCheck3.Value = 0 Then List!.Columns = 1 Else List 1.Columns = 2 End If End Sub	Свойство Value элемента Check равно 0, если флажок не взведен. В зависимости от значения этого свойства определяется свойство списка. (Listl.Columns = 1 - список одноколоночный, Listl.Columns = 2 -список двухколоночный)

При щелчке по кнопке «Очистить список» из списка фирм удаляются все элементы с помощью метода Clear, а затем кнопка деактивизируется. Реализующая процедура Command l_Click() содержит следующие операторы:

Процедура	Комментарии
Sub Command I ClickQ List [.Clear	Очистка списка Деактивация кнопки очистки

```
Command L Enabled = False списка Деактивация кнопки удаления
Command3.Enabled = False End элемента списка
Sub
```

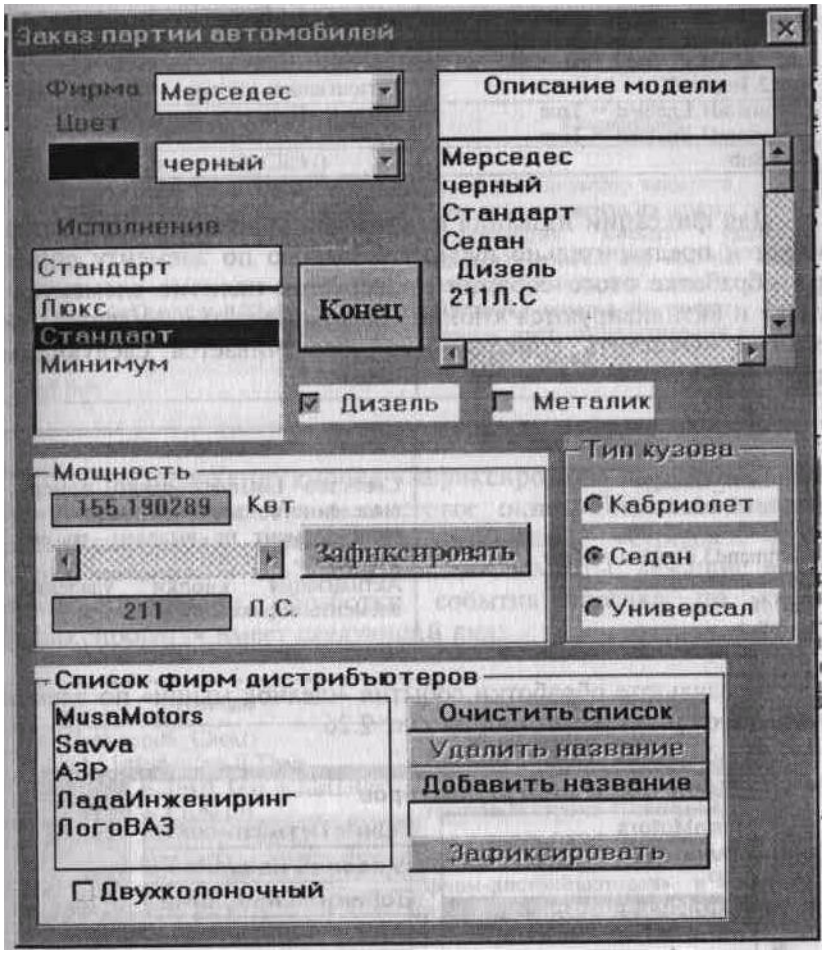


Рис. 2.25

При щелчке по кнопке «Добавить элемент» содержимое текстовой строки добавляется в список с помощью метода Additem. Список обладает свойством автосортировки (добавление происходит в соответствии с упорядоченностью по алфавиту). Происходит очищение текстовой строки и активизация кнопок «Удалить элемент» и «Очистить список». Это обеспечивается следующими операторами:

Процедура	Комментарии
Sub CoiTimand4Click() Listl.AddItem Text2.Text Text2.Text = “” Command [.Enabled = True Command3. Enabled = True End Sub	Добавление нового имени в список Очистка текстовой строки. Активизация кнопок «Удалить элемент» и «Очистить список»

Для фиксации названия фирмы или удаления его из списка требуется предварительно щелкнуть мышью по элементу списка. При обработке этого события проверяется наличие элементов в списке и активизируются кнопки «Зафиксировать», «Удалить элемент» и «Очистить список». Это обеспечивается следующими операторами:

Процедура	Комментарии
Sub Listi Click() If List 1. Listi ndex >= 0 Then Command3.Enabled = True Command6. Enabled = True End If End Sub	Свойство ListIndex равно номеру выделенного элемента. Если ни один элемент не выделен, то его значение - 0 Активизация кнопки удаления элемента и фиксации элемента

В результате обработки события «щелчок мыши» по элементу списка форма примет вид на рис. 2.26.

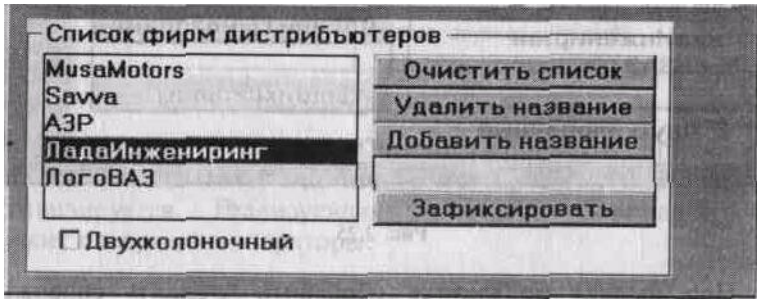


Рис. 2.26

При щелчке по кнопке «Удалить название» удаляется текущий выбранный элемент из списка, деактивируется кнопка удаления списка и, при условии отсутствия других элементов в списке, деактивируется кнопка «Очистить список». Проверка на пус

тоту списка осуществляется с помощью свойства ListCount, значение которого равно числу элементов в списке. Это обеспечивается следующими операторами:

Процедура	Комментарии
Private Sub Command3 Clic() If Listi. Listi ndex >= 0 Then List 1. Remove Item List 1. List! ndex Command3.Enabled = False End If If Listi. ListCount = 0 Then Command [.Enabled = False End If End Sub	Проверка на выбор элемента Удаление элемента из списка Деактивация кнопки удаления элемента списка, так как нет выделенных элементов в списке Проверка списка на пустоту. Если список пуст, то деактивация кнопки очистки списка

При щелчке по кнопке «Зафиксировать» выбранное название фирмы помещается в текстовое окно «Описание модели», деактивизируется рамка с названием «Список фирм-дистрибьюторов» и все управляющие элементы, лежащие внутри рамки. Процедура обработки события щелчка по кнопке «Зафиксировать» имеет следующий вид:

Процедура	Комментарии
Sub Command6_Click() If List1.ListIndex >= 0 Then Text1.Text = Text1.Text + List1.Text + Chr$(13) + Chr$(10)	Проверка выбора элемента из списка При выборе - добавление элемента к текстовой строке «Описание модели»
Frame1.Enabled = False List1.Enabled = False Text2.Enabled = False Command4.Enabled = False Check3.Enabled = False Command1.Enabled = False Command6.Enabled = False Command3.Enabled = False End If End Sub	Деактивизация рамки «Список фирм-дистрибьюторов» и управляющих элементов внутри неё

В результате выполнения этой процедуры форма примет вид, показанный на рис. 2.27.

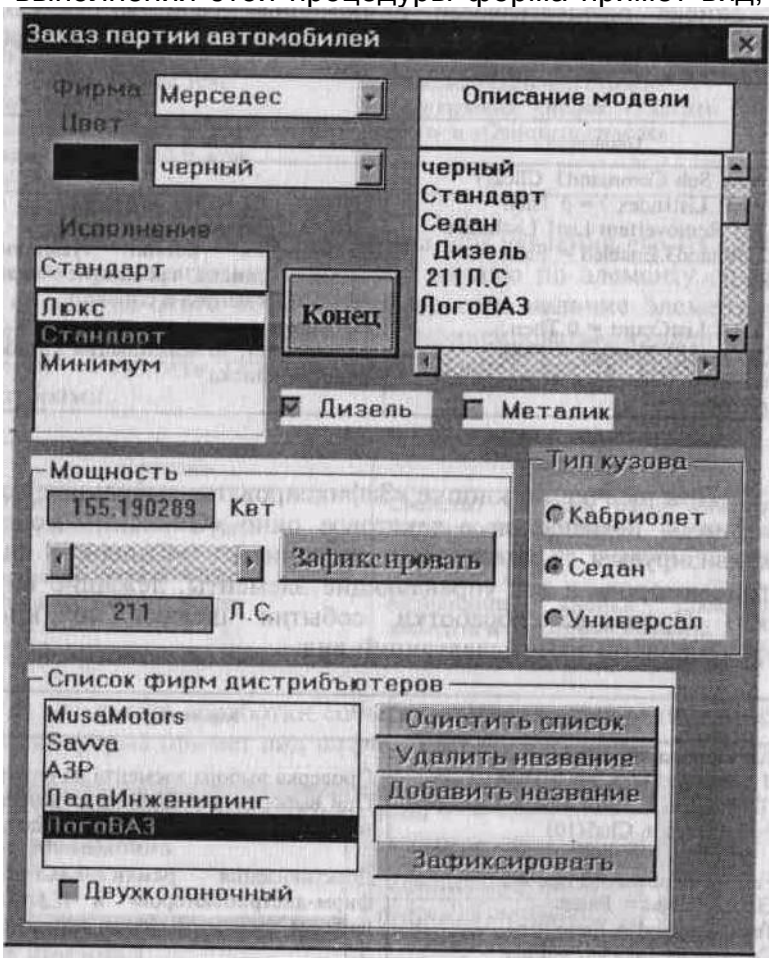


Рис. 2.27

На форме остались активными только два управляющих элемента: командная кнопка «Конец» и тестовое окно «Описание модели». В текстовое окно можно добавлять собственный текст и редактировать имеющийся.

При нажатии на кнопку «Конец» работа программы завершается. Это обеспечивается операторами:

Процедура	**Комментарии**
Sub Command2 Click() End End Sub	**End - оператор завершения программы**

Тексты всех процедур обработки событий примера:

```
Private Sub Check2_Click() If Check2.Value = 1 Then
Textl.Text = Textl.Text +          Check2.Caption + Chr$(13) + Chr$(10)
Check2.Enabled = False End If End Sub
Private Sub Check3_Cli.ck() If Check3.Value = 0 Then
Listi.Columns = 1 Else
Listi.Columns = 2 End If End Sub
Private Sub Check4_Click () If Check4.Value = 1 Then
Check4.Enabled = False
Textl.Text = Textl.Text + Check4.Caption + Chr$(13) + Chr$(10) End If End Sub
Private Sub Combol_Click() Combol.Enabled = False
Textl.Text = Textl.Text + Combol.Text + Chr$(13) + Chr$(10)
Labell.Enabled = False Combo2.Enabled = True Label2.Enabled = True End Sub
Private Sub Combo2_Click() Combo2.Enabled = False Label2.Enabled = False  Private Sub Cornmand4_Click ()
Listl.AddItem Text2.Text Text2.Text = "" Commandl.Enabled = True End Sub
•           Private Sub Command2_Click() End End Sub
Private Sub Command5_Click() Check2.Enabled = False Check4.Enabled = False Frame2.Enabled = False
HScroill.Enabled = False Label4.Enabled = False Label5.Enabled = False Label6.Enabled = False Label7.Enabled =
False Lab.el8 .Enabled = False Commands.Enabled = False
Textl.Text = Textl.Text + Label5.Caption + «Л.С» + Chr$(13) + Chr$(10)
Framel.Enabled = True Listi.Enabled = True Text2.Enabled = True Command4.Enabled = True Check3.Enabled = True
Commandl.Enabled = True End Sub
Private Sub Command6_Click() 'If Listl.ListIndex >= 0 Then
Textl.Text = Textl.Text + Listl.Text + Chr$(13) + Chr$(10)
Framel.Enabled = False Listi.Enabled = False Text2.Enabled = False Command4.Enabled = False Check3.Enabled =
False Commandl.Enabled = False Command6.Enabled = False Commands.Enabled = False End If End Sub
Textl.Text = Textl.Text + Combo2.Text + Chr$(13) + Chr$(10)
If Combo2.Text .= «Красный» Then LabelS.BackColor = &HFF& Elself Combo2.Text = «Белый» Then LabelS.BackColor
= &HFFFFFF Elself Combo2.Text = «черный» Then LabelS.BackColor = &HO Elself Combo2.Text = «Синий» Then
LabelS.BackColor = &HFFOOOO Elself Combo2.Text = «Зеленый» Then LabelS.BackColor = &HCOOO&
End If
Combo3.Enabled = True
Label3.Enabled = True
End Sub
Private Sub Combo3_Click() Combo3.Enabled = False Label3.Enabled = False
Textl.Text = Textl.Text + Combo3.Text + Chr$(13) + Chr$(10)
Frame4.Enabled = True          '   " Optioni.Enabled = True Option2.Enabled = True Option3.Enabled = True End Sub
Private Sub Commandl Click () Listi.Clear        ~ Commandl.Enabled = False End Sub
•           Private Sub Command3_Click() If Listl.ListIndex >= 0 Then
Listi.RemoveItem Listl.ListIndex
Command3.Enabled = False End If If Listi.ListCount = 0 Then
Commandl.Enabled = False End If End Sub  Private Sub Form_Load()
Label4.Caption = Str$(HScrolll.Value *   735.499 / 1000)
Label5.Caption = Str$(HScrolll.Value) Combol.AddItem «Мерседес» Combol.AddItem «Ауди» Combol.AddItem
«Фольксваген» Combol.AddItem «Шкода» Combol.AddItem «Рено» Combol.AddItem «Пежо» Combol.AddItem
«Запорожец» Combo2.AddItem «Красный» Combo2.AddItem «Бельм» Combo2.AddItem «черный» Combo2.AddItem
«Синий» Combo2.AddItem «Зеленый» Combo3.AddItem «Люкс» СотЬо3.AddItem «Стандарт» Combo3.AddItem
«Минимум» Listl.AddItem "MusaMotors" Listi.AddItem "Savva" Listl.AddItem "А3Р" Listl.AddItem «ЛоговАЗ» Listl.AddItem
«ЛадаИнжениринг» End Sub
Private Sub HScrolll_Change()
Label4.Caption = Str$(HScrolll.Value * 735.499 / 1000)
```

```
Labels.Caption = Str$(HScrolll.Value) End Sub
Private Sub Listl_Click() If Listl.ListIndex >= 0 Then Command3.Enabled = True Command6.Enabled = True End If End Sub
Private Sub Optionl_Click() Frame4.Enabled = False Optioni.Enabled = False Option2.Enabled = False Option3.Enabled = False
Textl.Text = Textl.Text + Optioni.Caption Chr$(13) + Chr$(10)
Check2.Enabled = True
Check4.Enabled = True
Frame2.Enabled = True
HScrolll.Enabled = True
Label4.Enabled = True
Labels.Enabled = True
Label6.Enabled = True
Label7.Enabled = True
Labei8.Enabled = True
Commands.Enabled = True End Sub
Private Sub Option2_Click() Frame4.Enabled = False Optionl. Enabled = False Option2.Enabled = False Option3.Enabled = False
Textl.Text = Textl.Text + Option2.Caption + Chr$(13) + Chr$ (10)
Check2.Enabled = True Check4.Enabled = True Frame2.Enabled = True HScrolll.Enabled = True Label4.Enabled = True
Labels.Enabled = True Label6.Enabled = True Label7.Enabled = True Labels.Enabled = True Commands.Enabled = True
End Sub
Private Sub Option3_Click() Frame4.Enabled = False Optioni.Enabled = False Option2.Enabled = False Option3.Enabled = False
Textl.Text = Textl.Text + Option3.Caption > Chr$(13) + Chr$ (10)
```

Check2.Enabled = True Check4.Enabled = True Frame2.Enabled = True HScrolll.Enabled = True Label4.Enabled = True Ldbel5. Ei-icioled = True Ldnei6.EnDied = True LdDpl7 . L.Jt,ied = Tr.Je Lctbel8 . Er.dbied = True CoiTimand . £.ndbled = Tru Ei-ici Sub

2.13. МЕНЮ ПОЛЬЗОВАТЕЛЯ

Важным элементом GUI является стандартное меню пользователя. Visual Basic имеет специальное средство для создания меню пользоваюлеи приложениями - Menu Editor (Проектировщик меню). Это срелс-жо нозволяег построить меню соответственно общим принципам рассмо1ренного событпиного управления для форм и элементов управления. Меню пользоваге-ля создается выстраиванием элементов меню в соответствии с необходимой их иерархией с помощью Проектировщика меню.

Для вызова проектировщика меню нужно вызвать команду **Menu Editor...** из меню **Tools** или щелкнуть кнопку **Menu Editor** на панели инструментов Главного меню. При этом появится окно Menu Editor (рис.2.28). элемешы которою имеюг следующее назначение:

- Caption примеияется для ввода названии элементов списка меню, используемых в диадою (например, Фаил, Edit и т.н.):

для создания разделительной черш (разделяет функциональные группы пунктов меню) вместо имени вводтся символ «-«;

- Name используегся для ввода имен элементов списка меню, используемых при написании нро1раммы (аналогично свойству Name для форм и элементов управления);
- Index - индекс элемеша списка диало1а в массиве (при использовании массива элемента списка диалога);
- Shortcut - раскрывающийся список комбинаций клавиш, из которого можно выбрагь комбинацию «горячих» клавиш эле-менюв списка меню;
- Windows List - флажок, при установке которого меню содержит список (л крытых

окон в многооконном приложении (MDI-приложения):

- HelpConiextID - индекс для поиска в HelpFile;
- Checked - флажок, при установке когорого помечается пункт выбранною меню;

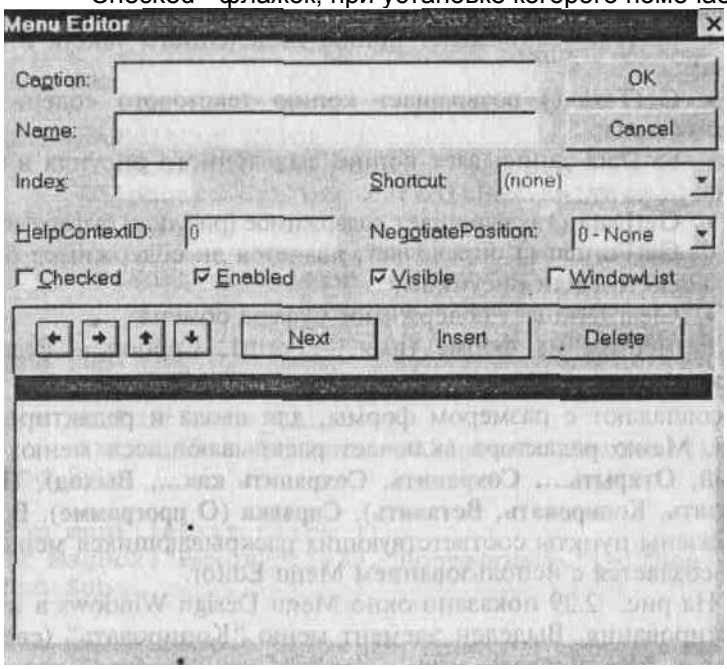
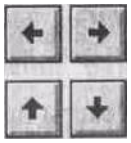


Рис 2 2S

- Enabled - флажок, при установке которого пункт меню доступен;
- Visible - флажок, при усгановке которого пункт меню виден;
- кнопки изменения уровня пунктов меню;



- кнопки изменения позиции пунктов меню на одном и том же уровне:
- Кнопки Next, Insert. Delete обеспечивают переход к следующему пункгу меню, вставку и удаление пункта.

Рассмотрим использование проектировщика меню на примере разработки простого редактора, который обеспечивает ввод и редактирование строк символов, запись выделенного фрагмента в буфер и всгавку из буфера. Доступ к буферу обмена можно получить через объект **Clipboard,** для которого определены следующие методы:

- SetText записывает копию выделенного текста в буфер обмена;
- GetText () возвращает копию текстового содержимого буфера обмена;
- SetData записывает копию выделенного рисунка в буфер обмена;
- Get Data () возвращает содержимое (рисунок) буфера обмена;
- GetFormat () определяет, является ли содержимое буфера обмена текстом или рисунком;
- Clear очищает содержимое буфера обмена.

Разместим на форме (имя - Formi, Caption - Редактор) многострочное текстовое окно (имя - *txlEdilBox),* размеры которого совпадают с размером формы, для ввода и редактирования текста. Меню редактора включает раскрывающиеся меню: **Файл (Новый, Открыть..., Сохранить, Сохранить как.... Выход), Правка (Вырезать, Копировать, Вставить), Справка (О**

программе). В скобках указаны пункты соответствующих раскрывающихся меню. Это меню создается с использованием Menu Editor.

На рис. 2.29 показано окно Menu Design Windows в момент проектирования. Выделен элемент меню «Копировать» (свойство Caption - ^.Копировать, имя - *mnuEdifCopy,* «горячая» клавиша -Ctrl+C). Вставка символа «&» означает задание «горячей» клавиши. Символы «....» обозначают переход на следующий уровень иерархии пунктов меню («Копировать» является подпунктом меню «Файл») и задаются кнопками изменения уровня пунктов меню.

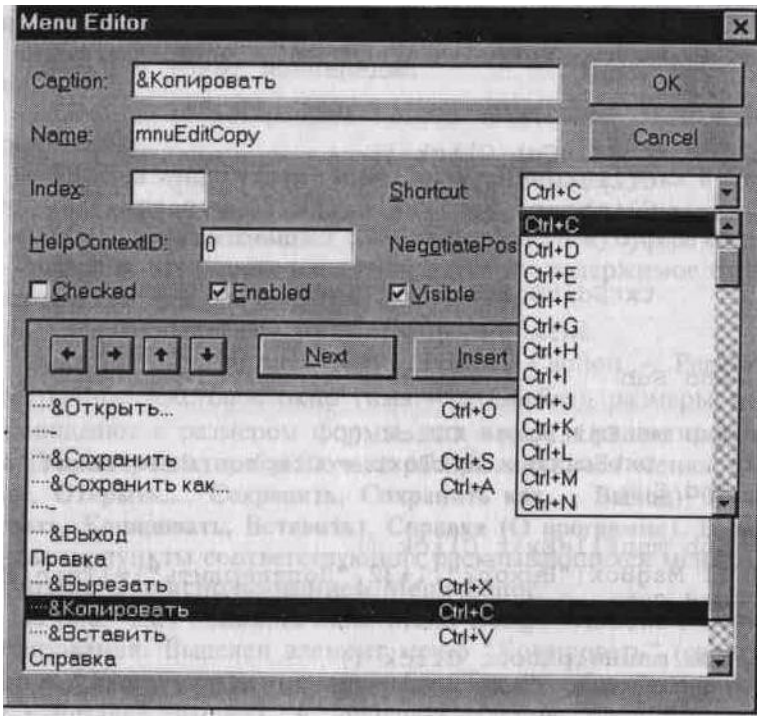
Аналогично имени mnuEditCopy образуются имена остальных пунктов меню, которые используются для записи программы (nmuEditCut, mnuEditPaste, mnuFileExit, mnuHelpAbout, mnuFileNew, mnuFileOpen, mnuFileSave, mnuFileSaveAs, mnu File Exit}.

Тексты процедур обработки событий следующие:

```
Sub Form_Resize ()
txtEditBox.Height = ScaleHeight txtEditBox.Width = ScaleWidth
End Sub
Sub mnuEditCopy_Click ()
Clipboard.Clear 'очистить буфер
Clipboard.SetText txtEditBox.SeiText
'поместить 'выделенный текст в буфер End Sub
Sub mnuEditCut_Click ()
Clipboard.Clear 'очистить буфер Clipboard.SetText txtEditBox.SeiText
'поместить 'выделенный текст в буфер txtEditBox.SeiText = «»
'удаление выделенного
'текста End Sub
Sub mnuEditPaste_Click ()
txtEditBox.SeiText = Clipboard.GetText() End Sub
Sub mnuFileExit_Click ()
If MsgBox(«Выход?»,1+32,»Подтвердить»)=lThen End
End Sub
Sub mnuHelpAbout_Click ()
MsgBox «Мой редактор на Visual Basic”,64,»О программе» • End Sub
Sub Form_Unload (Cancel As Integer)
Cancel = 1
mnuFileExit_Click End Sub
```

Процедура *Form_Resiy* (обработка события изменения размера формы) обеспечивает автоматическое изменение размеров текстового окна при изменении размеров формы.

Процедура *mnuEditCopy_Click* (обработка события щелчка мыши по пункту меню «Копировать») обеспечивает запись выделенного текста в буфер. Используются рассмотренные выше методы для объекта Clipboard и методы для текстового окна (операторы Clipboard. Clear и Clipboard.SetText *txtEditBox.* SeiText). Во втором операторе *txtEditBox.* SeiText присваивается



выделенный текст.

Рис. 2.29

Процедура *mnuEditCut_Click* (обработка события щелчка мыши по пункту меню «Вырезать») обеспечивает перед удалением сохранение выделенного текста в буфере.

Процедура *mnuEditPaste_Click* (обработка события щелчка мыши по пункту меню «Вставить») обеспечивает вставку текста из буфера (в данном случае отсутствие при вставке выделения означает вставку в позицию курсора).

Процедура *mniiFileExit_Click* (обработка события щелчка мыши по пункту меню «Выход») обеспечивает выход из редактора. Оператор End обеспечивает корректный выход из приложения (закрывает файлы и освобождает память). В процедуре используется функция MsgBox для вывода окна сообщения. Синтаксис ее применения следующий:

MsgBox сообщение [, тип окна] [, заголовок окна]

где

сообщение - текст сообщения;

тип окна - это сумма значений, определяющих число и тип кнопок на диалоговом окне, стиль пиктограмм, используемых в окне, и др. спецификации. В таблице приведены некоторые значения аргументов и их описание:

Значение типа	Описание
0	Выводить только кнопку OK
|	Выводить кнопки OK и Cancel
2	Выводить кнопки Abort, Retry, Ignore
3	Выводить кнопки Yes, No, Cancel

4	Выводить кнопки Yes, No
5	Выводить кнопки Retry, Cancel
16	Выводить пиктограмму «Стоп»
32	Выводить пиктограмму «?»
48	Выводить пиктограмму «!»
64	Выводить пиктограмму информации
0	Кнопка по умолчанию - первая
256	Кнопка по умолчанию - вторая
512	Кнопка по умолчанию - третья
0	Программное ведущее диалоговое окно
4096	Системное ведущее диалоговое окно

Первая группа значений (0-5) описывает номер и тип кнопок, отображаемых в диалоговом окне; вторая группа (16, 32, 48, 64) описывает стиль пиктограммы (иконки); третья группа (0, 256, 512) определяет кнопки по умолчанию; четвертая группа (0, 4096) определяет выводимые окна сообщения (0 - окно ввода для окончания работы приложения, 4096 - окно ввода для окончания работы системы).

В качестве значения аргумента может указываться сумма значений типа по одному из каждой группы. Например, в операторе

MsgBox *«Удалить ?», 52*

значение аргумента 52 является суммой 48 и 4 (48 - вывод пиктограммы с восклицательным знаком, 4 - кнопки «Yes» и «No»). Функция MsgBox возвращает следующие значения:

Константа		
vbOK	1	OK
	2	Cancel
vbAbort	3	Abort
vbRetry	4	Retry
vbIgnore	5	Ignore
vbYes	6	Yes
vbNo	7	No

Эти значения можно использовать для обработки результата работы пользователя с окном MsgBox.

Процедура *mnuHelpAbout_Click* (обработка события щелчка мыши по пункту меню «О программе») обеспечивает вывод окна сообщения.

Процедура *Form_Unload* (Cancel As Integer) (обработка события щелчка мыши по кнопке «Закрыть» раскрывающегося стандартного меню формы) обеспечивает выход с подтверждением. Закрытие формы не обеспечивает корректного завершения приложения. В этом случае присвоение Cancel =1 предотвращает закрытие формы и обеспечивает корректный выход через процедуру *mnuFileExit_Click.*

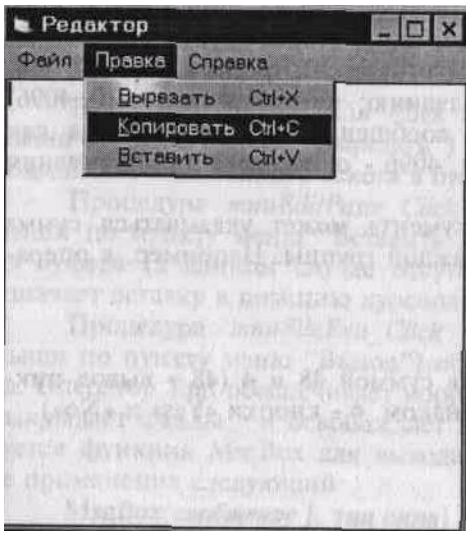


Рис. 2.30

Разработанная программа имеет меню пользователя и обеспечивает функции удаления, вставки и копирования при редактировании и выдачу сообщения о программе (рис. 2.30—2.32).

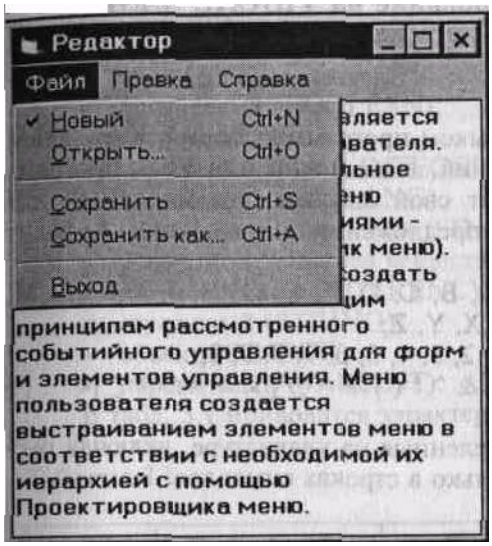


Рис. 2.31

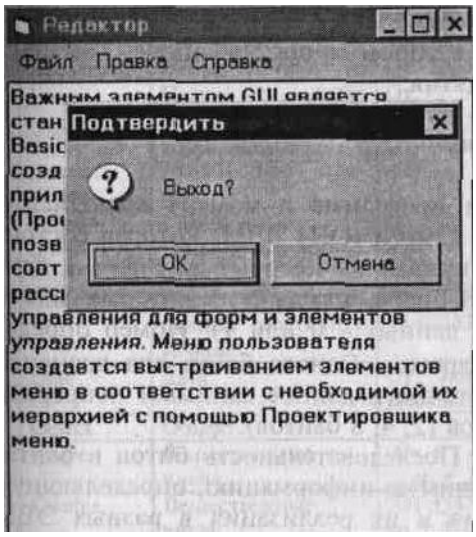


Рис. 2.32 Глава 3. Программирование на VISUAL BASIC 4

3.1. АЛФАВИТ

Visual Basic является языком программирования. Как любой другой язык, например, русский, английский или язык программирования Pascal, он имеет свой алфавит, используемый для написания операторов или предложений Visual Basic. Алфавит Visual Basic включает:

- 26 латинских букв: A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z;
- 10 арабских цифр: 1, 2, 3, 4, 5, 6, 7, 8, 9, 0;
- 26 символов: !#$%&ⁿ*()+-/\<>=?@_|:'.,« » пробел.

Другие символы, определенные на клавиатуре, включая русские, можно использовать только в строках символов.

3.2. ДАННЫЕ

3.2.1. Классификация данных

Данные характеризуются типом и организацией. Тип данных обобщает следующие понятия:

- размещение в ЭВМ;
- способ представления;
- прикладной смысл.

Размещение. Данные и программа в момент выполнения размещаются в оперативной памяти (ОП) ЭВМ, которая состоит из пронумерованных ячеек, каждая из которых вмещает 1 байт данных (1 байт состоит из 8 битов, каждый из которых может содержать элемент двоичных данных - 0 или 1). Номер определенной ячейки называется адресом. Одного байта для хранения данного как правило недостаточно и данное занимает непрерывную последовательность байтов (2, 4, 8 байтов).

Способ представления. Последовательность битов в байтах данного

кодирует всю необходимую информацию, определяющую данное. Правила кодирования и их реализация в разных ЭВМ определяют способ представления данного. Так, для числовых данных существует способ представления с фиксированной точкой и с плавающей точкой.

Прикладной смысл. Определяет возможность использования данных для тех или иных целей (с арифметическими данными можно производить вычисления, из символов можно строить слова и предложения).

В Visual Basic 4 определены следующие типы данных:

- байт (BYTE);
- логический (BOOLEAN);
- дата (DATE);
- короткий целый (INTEGER);
- длинный целый (LONG);
- простой вещественный (SINGLE);
- удвоенной точности вещественный (DOUBLE);
- коммерческий (CURRENCY);
- строковый (STRING);
- объект (OBJECT);
- переменный (VARIANT).

В табл. 3.1 приводятся характеристики этих типов данных:

Таблица 3.1

Тип данных	Описание	Диапазон значении	Занимаемая память
Byte	Двоичные данные	От 0 до 255	1 байт
Boolean	Логический	True или False	2 байта
Integer	Целые числа	От -32768 до 32767	2 байта
Long	Целые числа (длинные)	От -2 147 483 648 до +2 147 483 647	4 байта
String (variable-length)	Символьный (переменной длины)	От О до 2000000000 симво-лов	10 байт + длина строки
String (fixed-length	Символьный (фиксированной длины)	От 0 до 2000000000 симво-лов	1 байт на 1 символ
Currency	Число с фиксиро-ванной десятичной точкой	От -22337203685477,58 до 922337203685477,58	8 байтов
Date	Дата	От January 1, 100 до December 31, 9999	8 байтов
Object	Объект		4 байта
Single	Вещественные числа	От± 1.4* Ю-^{45} до + 3.4*10^{4}-^{38}	4 байта
Double	Вещественные числа	От+ 4.94* Ю-^{324} до+ 1.79* Ю-«^{308}	8 байтов
Variant	Произвольный тип	Любой из перечисленных выше	Зависит от значения

Data (Дата) - определяет дату (месяц, день, год).

Object (Объект) - ссылка на объект (OLE).
String (Строка) - последовательность ASCII-символов.
Саггепсу (Валюта) предназначен для финансовых расчетов и имеет фиксированную точность до четвертого знака после запятой (округляется).

Variant (Варьируемый) способен принимать любые значения (состоят из

двух частей: собственно значения и кода, указывающего исходный тип данного).

Под организацией данных понимается независимость отдельных данных (хранятся в отдельных непоследовательных ячейках памяти) или их связанность (хранятся в связанной последовательности ячеек памяти).

Связанными данными в Visual Basic 4 являются массивы (совокупность связанных данных одного типа) и записи (совокупность связанных данных разных типов).

3.2.2. Константы

Константа - данное, значение которого однозначно определяется написанием и не может быть изменено.

Пример:

3.1415926 6.02E3 (значение 60200) 123 "Visual Basic « «Иванов «

Для хранения постоянных величин Visual Basic позволяет объявить константы, т.е. выделить участки памяти, содержимое которых не меняется (при попытке модификации выдается сообщение об ошибке). Объявление констант осуществляется оператором

[Public | Private] Const *имя* [As type] **= *выражение*** Имя константы принято записывать прописными буквами (правила записи имен см. ниже).

Значения ключевых слов следующие:

Public - константу можно использовать в любых процедурах и функциях;

Private - константу можно использовать только внутри модуля (см. ниже), в котором она определена.

Пример:

```
Const /V= 3.1415926
Const MY_NAME = «Юра «
```

Тип константы можно не объявлять (устанавливается на основе значения — Const *CODE = 35* автоматически получит тип integer). Однако константа *PI* в примере может быть любого из трех типов: single, double или currency. По умолчанию принимается тип, занимающий наименьший объём памяти. Поэтому лучше явно указывать тип специальными символами в операторах объявления констант. Используемые символы показаны в таблице:

Символ объявления типа	Тип данных
%	integer
&	long
1	single
#	double
@	currency
$	string

Пример:

Const *ONE&* = / (резервирует 4 байта)
Const *ONE#* = *1* (резервирует 8 байтов, хранится в виде числа двойной точности с плавающей точкой).

В Visual Basic имеется большое число встроенных констант, значения которых определены заранее и их можно использовать без предварительного определения. Примерами таких констант являются:

vbOKCancel = 1 — аргумент функции MsgBox для вывода в диалоговом окне командных кнопок ОК и Cancel;

vbYesNoCancel = 3 — аргумент функции MsgBox для вывода в диалоговом окне командных кнопок Yes, No, Cancel;

vbOk = 1 — значение, которое возвращает MsgBox, если пользователь щелкнул по кнопке ОК в диалоговом окне;

- vbCancel = 2 - значение, которое возвращает MsgBox, если пользователь щелкнул по кнопке Cancel в диалоговом окне.

Полную информацию о встроенных константах можно найти в Object Browser.

Кроме того, в комплекте Visual Basic имеется файл CONSTANT.TXT, содержащий десятки часто используемых констант. Необходимые константы могут быть скопированы в программу. 3.2.3. **Имена**

Имена используются для обозначения объектов в программе (константа является объектом программы). Правила образования имен:

- первым символом имени должна быть латинская буква;
- имя может включать только латинские буквы, цифры и знак подчеркивания (_);
- имя может содержать не более 40 символов;
- ключевые слова или Reserved word (Зарезервированные слова) не могут использоваться в качестве имен (список ключевых слов содержится в справочной системе Visual Basic в разделе Reserved word).

Примеры:

Правильные имена Неправильные имена *StartTime* *CM*PER*INCH*
A2 *23B*
color *File* (ключевое слово) • *VariableName BMW_360*
Хорошим тоном при программировании на любом языке является осмысленный выбор имен для объектов программы (присваивать объектам имена, соответствующие контексту и несущие описательную нагрузку). В качестве примера можно привести имя процедуры обработки события, связанного с щелчком мыши по командной кнопке, запускающей программу *btnStart_CUck:* первая часть имени состоит из сокращения слова кнопка (button — btn) и слова Start, вторая часть определяет событие — Click.

3.2.4. Оператор объявления

Оператор объявления резервирует в памяти место для хранения данных определенного типа и организации и присваивает ему имя, по которому производится обращение к данным.

Оператор имеет вид:

{Dim | Global} имя [({описатель})} [As [New] тип} [, имя [([описатель])]

[As [New]w«n]] . . .

Dim, Global, As, New — ключевые слова (Global используется для объявления глобальных данных (см. ниже). New используется для создания нового объекта на основе существующих объектов, например формы);

имя — имя объекта (имя переменной, массива);
тип - тип данных;
описатель — определяет организацию данных (например, массива, см. ниже).

Пример:

Dim *Name, YourName* As String, *N* As Integer, *Money* As Currency,

SiirName As String* 15

(переменная *SurName* определена как символьная фиксированной длины в 15 символов).

При описании имен прописные и строчные буквы воспринимаются одинаково. Однако после определения ссылки на эту переменную должны соответствовать последней форме записи (производится автоматическое преобразование текста программы).

Visual Basic допускает использование имен без объявления их типа (в этом случае автоматически определяется тип, требующий для размещения минимальной памяти), однако целесообразно и является признаком хорошего тона явно объявлять типы используемых данных.

Из рассмотренных в предыдущей главе примеров программ для работы с формами и управляющими элементами известно, что любая программа состоит из формы и элементов управления, которым поставлены в соответствие процедуры. Более сложные программы могут включать несколько форм.

Существует понятие области действия (scope) данных, определяющее возможность доступа к тем или данным (например, к переменной) в отдельных процедурах одной формы или в процедурах, относящихся к разным формам одной программы.

Если оператор объявления какой-либо переменной находится внутри процедуры обработки события, то доступ к этой переменной (возможность ее использования) возможен только в рамках данной процедуры. Такая переменная называется локальной (local).

Для того чтобы одна и та же переменная могла использоваться в разных процедурах одной формы, оператор объявления переменных должен быть помещен в раздел общих объявлений (general), доступ к которому открывается щелчком мыши по элементу “general” раскрывающегося списка окна Object формы. Объявленная таким образом переменная имеет статус действующей на уровне модуля (modul-level variable) и может использоваться (доступна) в любой процедуре данной формы. Для того чтобы одна и та же переменная могла использоваться в процедурах разных форм одной программы она должна быть объявлена как глобальная переменная

(global variable). Используется ключевое слово Global вместо Dim.

Пример:

Global *Name, YourName* As String, *N* As Integer, *Money* As Currency,

SurName As String* 75

Операторы объявления глобальных переменных помещаются в-модулях кода (code modules, см. ниже) и эти переменные могут использоваться во всей программе.

Схема, иллюстрирующая области действия переменных, показана на рис. 3.1.

В Форме 1 переменная *P* объявлена в разделе общих объявлений (general) формы и может быть использована как в Процедуре X, так и в Процедуре Z. Изменение значения переменной в одной из процедур влечет за собой изменение и в другой процедуре.

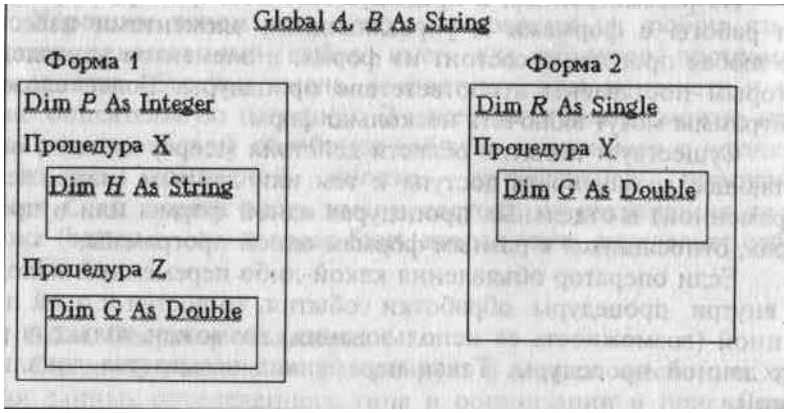


Рис. 3.1

Объявленные переменные *И* в Процедуре X и *G в* Процедуре Z Формы 1 определены только внутри этих процедур, аналогично переменной *G* в Процедуре_У Формы 2. Эти переменные определены только в рамках своих процедур. Более того, для одноименных переменных *G* в Процедуре Z и в Процедуре Y выделяются разные ячейки памяти и изменение значения переменной

G в одной из процедур не влечет изменения значения в другой процедуре.

Переменные *A* и *B* определены как глобальные для всей программы и доступны во всех процедурах Формы 1 и Формы 2. Изменение значения переменных в одной из процедур одной из форм влечет изменение значений в других процедурах и формах.

Рассмотренные области действия переменных справедливы и для других объявляемых данных (констант, пользовательских типов данных, массивов и др.).

Как было сказано выше, описания глобальных данных помещаются .в

модуль кода. Определения формы и все связанные с ней программы хранятся в отдельном файле с расширением .FRM. Программы, состоящие из нескольких форм и соответственно из нескольких таких файлов, размещаются в файлах модулей кода с расширением. Такие файлы создаются при выборе в меню Insert (Вставить) команды Module (Создать модуль) или при щелчке мыши на одноименной кнопке панели инструментов.

3.2.5. Пользовательские типы данных (записи)

Данные различных типов можно сгруппировать по какому-либо признаку в удобную для использования одну структуру. В ряде языков программирования такие структуры называются записями (records).

Оператор объявления пользовательского типа данных (записи) помещается в модуль и имеет вид:

Type имя записи имя элемента записи [(описатель)} As тип

[имя элемента записи [(описатель)} As тип}

End Type

Пример:

```
Type Student
  Familia As String * 20
  Name As String * 10
  Vozrast As Integer
  Birthdate As Double
  Length As Integer
  Wegith As Integer
End Type
```

Оператор создает указанную структуру данных, но не выделяет под неё память. Память выделяется рассмотренным оператором описания, в котором в качестве типа указывается имя записи (имя пользовательского типа данных).

Пример:

Dim *Student_l, Student_2* As *Student* Описаны два имени *(Student_l, Student_2*), для каждого из которых определены заданные в структуре компоненты *(Familia, Name, Voyast, Birthdate, Length, Wegith)* и для хранения значений которых выделяется память соответственно заданному типу (длине и способу представления).

Для обращения к конкретному свойству (элементу структуры) определенного объекта используется составная запись

имя переменной.элемент структуры

Например, *Student'_]. Birthdate* определяет дату рождения студента *Student_7*.

3.2.6. Массивы

Массив - упорядоченный набор однотипных данных, обозначенный одним именем. Массив может строится из однотипных переменных, однородных пользовательских типов данных (одинаковых записей), однотипных элементов пользовательских типов данных.

Массив объявляется уже рассмотренным оператором

{Dim | Global [Static) имя [({описатель})} [As [New] тип} [, имя [([описатель])} [As [NewJ/гаил]] . . . где описатель имеет следующий синтаксис:

[нижняя граница To]верхняя граница[,[нижняя граница To] верхняя граница] . .

нижняя граница определяет минимальное значение индекса массива (целого типа);

верхняя граница определяет максимальное значение индекса массива (целого типа);

To — ключевое слово.

Количество повторений ***[нижняя граница*** **To** ***[верхняя граница*** определяет размерность массива (количество индексов, используемое для определения элементов массива). Максимальное число индексов равно 60.

Static в процедурах и функциях позволяет сохранить значения элементов объявленного таким образом массива между вызовами этих процедур или функций.

Пример различного способа объявления одного и того же массива:

- Dim *A(8,3)* As Double Dim A(6>To *8,* O To *3)* As Double Dim *A(8, OTo 3}* As Double
  Объявляется двухмерный массив (два индекса). Нижняя граница обоих индексов равна 0 (принимается по умолчанию). Верхняя граница первого индекса равна 8, второго — 3. Массив состоит из 36 элементов (9*4=36) одинакового типа (вещественный) и каждый элемент занимает 8 байтов.

  Границы значений индексов: от —32768 до 32767.

Примеры:

Dim *A(—4* To *10}* As Integer
Dim *B(—99* To —5, *—3* To *0)* As String

Элементы массива занимают связанную последовательную область в памяти машины. Массив *A* в примере занимает в памяти машины последовательность из 15 ячеек памяти, каждая из которых имеет длину 2 байта (тип Integer имеет длину 2 байта, рис. 3.2).

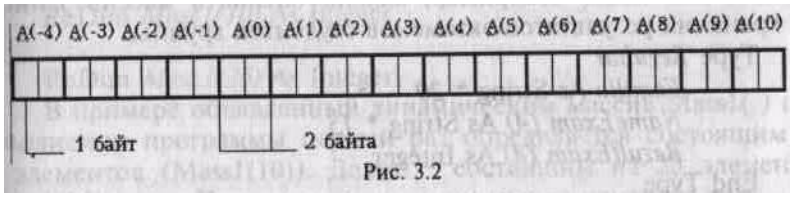


Рис. 3.2

Многомерные массивы также занимают линейную последовательную область памяти. При этом важное значение имеет способ упорядочения элементов многомерных массивов, который отличается для различных языков программирования.

В Visual Basic многомерные массивы упорядочиваются в памяти машины так, что быстрее всего меняется левый индекс. Например, двумерный массив (в прикладных математических программах матрицы представляются двумерным массивом) A (2,3), состоящий из 12 элементов, располагается в памяти машины следующим образом: A(0,0), A(1,0), A(2,0), A(0,1), A(1,1), A(2,1), A(0,2), A(1,2), A(2,2), A(0,3), A(1,3), A(2,3) (если данный
массив представляет матрицу, то в памяти машины она упорядочивается по столбцам).
Для рассмотренного пользовательского типа данных (записи) Student можно также объявить массив оператором Dim *MasStudent (25)* As *Student*

Объявляется одномерный массив *MasStudent,* состоящий из 26 элементов, каждый из которых является одной записью:

Type Student

Familia As String * 20 *Name* As String * 10 *Voyast* As Integer *Birthdate* As Double *Length* As Integer *Wegith* As Integer End Type

рассмотренной выше. Записи располагаются последовательно в памяти машины и занимают каждая 44 байта. 20 байт занимает *Familia* (один символ занимает 1 байт), 10 байт - *Name, 2* байта -*Voyast* (целый тип занимает 2 байта), 8 байт - *Birthdate* (вещественный двойной точности занимает 8 байт) и по 2 байта -*Length* и *Wegith* (целый тип).

Элемент пользовательского типа данных (записи) может являться массивом. Например, можно объявить массив записей для хранения результатов экзаменов студентов группы. Type *Rezultat*

*Familia As String * 20 NameExam (4) As String * /5 RewltExam (4) As Integer End Type Dim Sesia(25) As Reyiltat*

Объявлен пользовательский тип данных *Rewltat,* элементами которого являются *Familia* фиксированной длины 20 символов (фамилия студента не может состоять более чем из 20 символов), массив *NameExam (4)* из пяти элементов (число экзаменов в сессии не более 5) для хранения названия экзаменов (название каждого экзамена не может состоять более чем из 15 символов), массив *RewltExam (4)* из пяти элементов для хранения оценок по каждому экзамену (целого типа). Длина записи *Rewltat* 105 байтов.

На базе пользовательского типа данных *Requital* объявлен массив *Se<iia(25),* каждый элемент которого содержит информацию об экзаменах и оценках для одного студента (не более 26 студентов в группе). Один элемент массива занимает 105 байтов памяти, а весь массив 2730 байт.

В Visual Basic имеется возможность переопределять количество элементов массива в момент выполнения программы, т.е. динамически определять размер массива. Это позволяет эффективно использовать дефицитную оперативную память при создании программ. Для этого используется

оператор ReDim (не объявление, а команда, выполняемая при работе программы) в программе

ReDini [Preserve] имя [([описатель})} [, имя [([описатель})}}

Preserve сохраняет данные существующего массива при выполнении оператора (для многомерных массивов можно применять только для правого индекса).

При использовании оператора ReDim в операторе объявления массива не задается размер (объявляется динамический размер (dynamic array)).

Пример.

```
Dim Mass1( ) As Integer
.....
.....
ReDim Mass1(10) As Integer
.....
ReDim Mass1(25) As Integer
```

В примере объявленный динамическим массив Massl( ) при выполнении программы первый раз определяется состоящим из 10 элементов (Massl(lO)). Далее — состоящим из 25 элементов (Massl(25)). При переопределении нельзя изменять тип.

Элементы объявленного массива могут использоваться в операторах программы в виде

имя массива (значение индекса [значение индекса} ...)

Значение индекса задается арифметическим выражением (см. ниже), нецелочисленные значения выражения округляются до ближайшего целого, **значения выражения должны находиться в пределах возможных значении индекса, задаваемых операторами определения или переопределения массива** (в случае нарушения последнего условия в процессе выполнения программы, выдается предупреждение об ошибке и программа прекращает работу). **Пример.**

Если массив объявлен оператором
Dim *A(10,20)* As Double,
то для обращения к его элементам в операторах программы можно использовать
A(0,0) A(5,J+!) A(1,K) A(2.5, 9.8) (обращение к элементу A(2,10).

. К элементам массива рассмотренных выше пользовательских типов данных обращение записывается:

MasStitdent(J).Name (обращение к имени j-го студента массива *MasStudent,* как к элементу записи *Student)*

Sesia(Ciirrent).NameExam(2) (обращение к названию экзамена для текущего (current) индекса студента массива *Sesia* как к третьему элементу массива *NameExam,* являющегося элементом записи *Reviltat}.*

Пример.

В качестве примера рассмотрим фрагмент программы, реализующий

перестановку элементов массива А, состоящего из 10 элементов, в обратном порядке.

```
Dim A(1 To 10), Rezerv As Single
.....
I = 1          ' присвоение начального значения индекса эле-
мента массива
Povtor:        ' метка
Rezerv = A(I)  ' перестановка каждой пары элементов
A(I) = A(11 - I)  ' осуществляется тремя
A(11 - I) = Rezerv  ' операторами
I = I + 1  ' изменение значения индекса
If I <= 5 Then GoTo Povtor  ' проверка условия и переход к
                    'перестановке следующей пары элементов
```

В данном примере три оператора перестановки элементов, операторы изменения значения индекса и проверки условия выполняются несколько раз при разных значениях I (изменяемый параметр). Реализован так называемый «ручной» цикл, т.е. присутствуют операторы задания начального значения параметра, изменения значения параметра, проверки условия невыхода параметра за заданную границу, которые и организуют цикл вычислений.

В заключение следует отметить, что однотипные элементы управления также могут организовываться в виде массивов, что позволяет делать программы более универсальными и компактными. Если при разработке формы двум одинаковым элементам управления присваиваются одинаковые имена, то открывается специальное окно для подтверждения создания массива управляющих элементов. При подтверждении (ответ «Да»), элементы управления организуются в массив и у процедур обработки событий для этих элементов управления появляется параметр — индекс элемента управления. Изменяя значение индекса, одну и ту же процедуру обработки события можно использовать для разных элементов.

В списке свойств элементов управления имеется свойство Index, значение которого определяет индекс данного элемента в массиве. Индексы присваиваются автоматически последовательно при создании на форме нового одинакового элемента управления (первому элементу — 0, второму — 1 и т. д.) Эта последовательность может быть изменена разработчиком формы, но только на этапе создания формы.

3.3. ВЫРАЖЕНИЯ

Выражения используются для операций над данными. В зависимости от данных и используемых операций выражения можно разделить на арифметические, логические и символьные. Выражение можно определить

операнд **[знак операции** ***операнд}*** **[знак операции** ***операнд]*** **...** где в зависимости от типа выражения используются соответствующие операнды и знаки операций.

3.3.1. Арифметическое выражение

Используются следующие знаки операций:

+ - сложение (2.36+12.5);

- вычитание (231-49);

* - умножение (3*2);

« - возведение в степень (1(^2, результат 100, 10*-2, результат 0.01, 25^.5 или 25»(1/2), результат 5);

/ - деление с плавающей точкой (3/2, результат 1.5);

\ - целочисленное деление (3/2, результат 1);

Mod - вычисление Остатка (7Mod4, результат 3). Приоритет выполнения операции (в порядке убывания приоритета): возведение в степень, умножение и деление с плавающей точкой, целочисленное деление, вычисление остатка, сложение и вычитание. Вычисления в выражении производятся слева направо. Скобки изменяют приоритет.

Пример:

14/5*2 = 5.6 - операции одного приоритета выполняются слева направо;

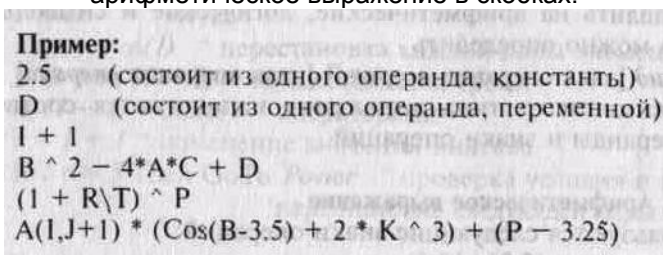
14\5*2 = 1 - умножение имеет более высокий приоритет и при целочисленном делении дробная часть отбрасывается;

27^1/3 = 9 — возведение в степень обладает наивысшим приоритетом;

27-(1/3) = 3 - скобки изменяют последовательность операций.

Операнды выражения:

- константа (Integer, Long, Currency, Single, Double, Variant);
- переменная (Integer, Long, Currency, Single, Double, Variant);
- элемент массива (Integer, Long, Currency, Single, Double, Variant);
- обращение к стандартной функции (см. ниже);
- обращение к процедуре — функции (см. ниже);
- арифметическое выражение в скобках.


Пример:
2.5 (состоит из одного операнда, константы)
D (состоит из одного операнда, переменной)
I + 1
B ^ 2 − 4*A*C + D
(1 + R\T) ^ P
A(I,J+1) * (Cos(B-3.5) + 2 * K ^ 3) + (P − 3.25)

3.3.2. Логическое выражение

Логические выражения используются в математической логике и их также называют Булевыми выражениями, по имени математика Дж. Буля.

Используются следующие знаки логических операций:

- Not- логическое отрицание **НЕ;**
- And — логическое умножение И;
- Or — логическое сложение ИЛИ;
- Xor — исключительное ИЛИ;
- Eqv — логическая эквивалентность;

- Imp — логическая импликация.

Логические операции объединяют логические величины, которые могут принимать два значения: **True (Истина)** или **False (Ложь).** Результат логической операции также принимает одно из двух значений: True (Истина) или False (Ложь).

Результа1 ложческои операции определяется следующей таблицей:

Значения		PCIV.IM.IT ипер.шии						
Оперли.1 (А)	Oiicp.iiLi (В)	Nol A	Not B	A And Ii	A Oi B	A Xoi B	A Еч^ B	A Imp B
T] нс	True	False	False	Tine	True	False	True	True
False	True	Tmc	False	False	True	True	False	F.ilse
Tme	Pulse	F.ilse	Ti не	False	Tine	True	False	False
False	False	Тше	Tme	False	False	False	Tine	True

Приоритет выполнения операций (в порядке убывания приоритета): Not, And, Or, Xor, Eqv, Imp.

Пример:

True And Not False Or False

Вычисляется Not False, результат — True; далее вычисляется And. результат — True; последним вычисляется **Or,** результат — True.

Операндами логического выражения являются:

- логические константы;
- логические переменные;
- обращения к функциям, возвращающим логические значения;

. • выражения отношения;

- заключенные в скобки логические выражения.

Выражения отношения состоят из двух арифметических или символьных (см. ниже) выражений, объединенных знаками операций отношения:

> — больше;
< — меньше;
>= — больше или равно;
<= - меньше или равно;
= — равно;
<> — не равно.

Выражение принимает значение либо True, либо False. Примеры:

3 + 1 > 3 (результат True);

SiirName = «Ивсмов « (если переменная *SiirName* имеет значение « *Иванов»,* то результат True, в противном случае False);

«А» > «В» (результат False, побитово сравниваются значения кодов символов, код символа «Я» больше кода символа 'А»).

Двойные неравенства для правильного их вычисления необходимо записывать с использованием знаков логических операций. Когда арифметические данные преобразуются к логическому типу, то 0

преобразуется в False, а другие значения преобразуются в True. При преобразовании логического типа к арифметическому, False преобразуется к 0, а True к —1.

Рассмотрим неравенство *23 < A < 543.* Хотя синтаксически Visual Basic допускает такую запись, результат в любом случае будет True независимо от значения переменной *A.* Выражение вычисляется слева направо: *23 < A* даст либо True. либо False, т.е. либо —/, либо 0. —/ или 0 всегда меньше *543* и результат будет True. Для того чтобы вычисление было корректным, это двойное неравенство следует записать состоящим из двух операндов и знака логической операции.

Правильная запись двойного неравенства: *23 < A* And *A< 543* (если использовать Or вместо And, то вычисление также будет некорректно).

Составим логическое выражение для набора условий:

Вначале вычисляется выражение в скобках (*И <= Y* And ^<= *Q)* — True, далее слева направо вычисляются *Л+ 2 >= Y And д + B > C* — False, далее

*д*X + B*Y = D* And (// <= *Y* And *Y <= Q)* - False, последней операция Or. которая лает результат — False.

3.3.3. Символьное выражение

В Visual Basic определена одна операция с символьными данными - конкатенация (сцепление), позволяющая объединять несколько строк в одну. Знак операции — «+» или «&».

Операндами символьного выражения могут быть:

- символьная константа;
- символьная переменная:
- элемент символьного массива (string);
- обращение к процедуре-функции, возвращающей символьное значение (см. ниже);
- обращение к стандартной функции, возвращающей символьное значение (см. ниже).

Пример:

NameS = «Иван»

SurName$ = «Иванов»

. Narne$ + SurNameS дает результат «ИванИванов» Следует обратить внимание (см. Пример), что необходимые пробелы нужно расставлять самостоятельно. Visual Basic их не вставляет.

3.4. СТАНДАРТНЫЕ ФУНКЦИИ

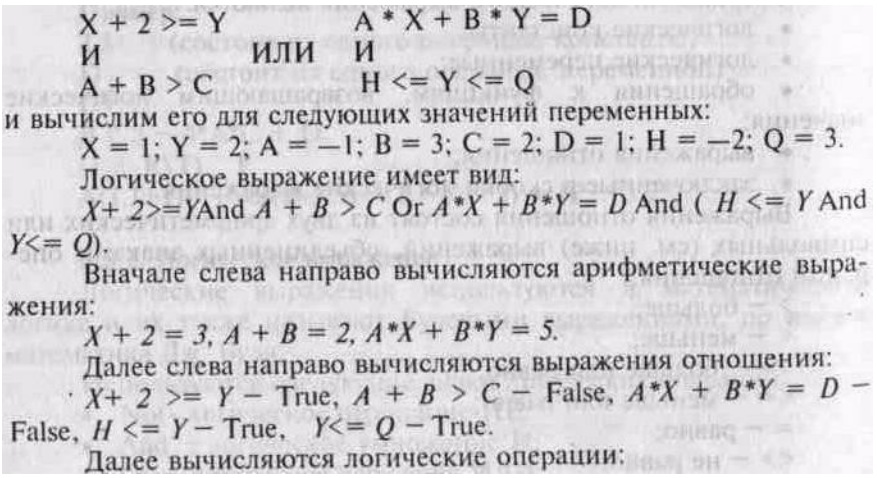
В Visual Basic имеется широкий набор встроенных (стандартных) функций, облегчающий написание программ. Имеются математические функции, для обработки строк, для работы с временем и датами, для финансовых

расчетов.

Встроенные функции различаются тем, что некоторые воз-вращрют вычисленное значение, другие не возвращают. Обращения к функциям, которые возвращают вычисленное значение, является операндом выражения (в рассмотренном выше выражении

A(I,J+1) * (Cos(B+3.5) + 2 * К « 3) + (Р - 3.25) операнд Cos(B+3.5) является обращением к встроенной функции вычисления косинуса угла).

Обращение к встроенной функции, возвращающей значение того или иного типа, должно соответствовать выражению, в кото-


X + 2 >= Y A * X + B * Y = D
И ИЛИ И
A + B > C H <= Y <= Q

и вычислим его для следующих значений переменных:

X = 1; Y = 2; A = —1; B = 3; C = 2; D = 1; H = —2; Q = 3.

Логическое выражение имеет вид:

*X+ 2>=Y*And *A + B > C* Or *A*X + B*Y = D* And (*H <= Y* And *Y<= Q*).

Вначале слева направо вычисляются арифметические выражения:

*X + 2 = 3, A + B = 2, A*X + B*Y = 5.*

Далее слева направо вычисляются выражения отношения:

X+ 2 >= Y — True, *A + B > C* — False, *A*X + B*Y = D* — False, *H <= Y* — True, *Y<= Q* — True.

Далее вычисляются логические операции:

ром к ней обращаются. Например, и арифметическом выражении можно обращаться к функциям, возвращающим значения арифметических типов, в символьном — символьного типа.

Обращения к функциям, которые не возвращают вычисленное значение, являются отдельными операторами программы. Например, запись отдельного оператора

- Beep является обращением к встроенной функции подачи звукового сигнала (в момент выполнения этого оператора компьютер выдает звуковой сигнал).

Для обращения к некоторым встроенным функциям нужно задавать значение аргумента (например, *Sin(X+2),* где *X+2* выражение, определяющее значение аргумента). Для других встроенных функций аргумент задавать не нужно (например. *Now).* Примерами математических функций являются:

- Atn — возвращает арктангенс;
- Sin — возвращает синус;
- Cos — возвращает косинус;
- Tan - возвращает тангенс;
- Exp — возвращает *e^x;*
- Log — возвращает натуральный логарифм;
- Sqr — возвращает квадратный корень;
- Rnd — возвращает случайное число;
- Sgn — возвращает знак числа:
- Fix — возвращает округленное число. Примерами строковых функций являются:
- StrComp — сравнивает две строки;

- Lease - преобразовывает строку в нижний регистр;
- Ucase - преобразовывает строку в верхний регистр;
- Spase — создает строку пробелов;

• • String — создает строку символов;

- Len — определяет длину строки;
- Instr — ищет подстроку;
- Right — выделяет правую часть строки;
- Left — выделяет левую часть строки;
- Asc — возвращает ASCII код символа;
- Сиг — возвращает символ по ASCII коду;
- Str — преобразовывает число в строку;
- Val — преобразовывает строку в число. Примерами функций даты и времени являются:
- Date — устанавливает и возвращает текущую дату;
- Time — устанавливает и возвращает текущее время;
- DateSerial — преобразовывает в последовательную дату три целых числа (день, месяц, год);
- Day — преобразовывает последовательную дату в день месяца;
- Month — преобразовывает последовательную дату в месяц года;
- Year — преобразовывает последовательную дату в год. Полные сведения о встроенных функциях и правилах их применения можно найти в справочной системе Visual Basic.

3.5. ОПЕРАТОРЫ

. Программа на Visual Basic состоит из процедур (любая программа состоит хотя бы из одной процедуры). Процедуры состоят из операторов.

Оператором (Statement) является синтаксически полное описание конкретной команды (аналог предложения на русском или другом языке), которая выражает одно действие или определение.

Одному оператору соответствует одна строка программы. Однако можно использовать разделительный знак двоеточие (:), чтобы поместить больше чем один оператор в строке программы.

Операторы программы выполняются последовательно сверху вниз (слева направо для операторов на одной строке), если другие операторы (перехода, управления, обращения к функциям' или процедурам, см ниже) не вызывают изменения последовательности их выполнения.

Строки программы могут, быть обозначены метками (Linelabel) или номерами (Linenumber).

Метка (Label) обозначает следующую строку программы. Метка может включать не более 40 символов (первый обязательно буква) и заканчивается двоеточием(:), не может быть ключевым словом. Метка может начинаться в любой позиции строки, если ей не предшествует никакой символ.

Номер строки (Lilienumber) обозначает следующую строку программы. Номер строки может включать не более 40 десятичных цифр и **не заканчивается двоеточием.** Номер строки может начинаться в любой позиции строки, если ему не предшествует никакой символ. В рамках одной

процедуры номера строк не могут повторяться. **Пример:**

Routine: ' метка *Num = Num / 2* ' оператор, помеченный меткой Routine:

123 ' Номер строки

Msgbox "Half of your number is " & Num ' оператор с номером 123 Программу легче читать и отлаживать, если операторы программы снабжены комментариями. Комментарии начинаются с апострофа ('), за которым можно размещать любые замечания в тексте программы. Если комментарии располагается на нескольких строках, то каждую строку нужно начинать с апострофа.

3.5.1. Оператор перехода

Оператор перехода имеет вид GoTo { метка | номер строки}

и вызывает переход к выполнению оператора, с указанной меткой или номером строки.

,

Пример:

GoTo *123* 'после этого оператора будет выполняться оператор Msgbox ...
Num = Num / 2
123 ' Номер строки
Msgbox «Половина введенного числа равна « & Num

Следует отметить, что использование оператора перехода в программах является признаком низкой квалификации программиста и его желательно избегать.

Другой оператор перехода позволяет перейти к выполнению выделенной группы операторов (так называемая внутренняя процедура). Синтаксис его использования следующий:

GoSub { метка \ номер строки }

{метка : \ номер строки } Return

этот оператор вызывает переход к выполнению группы операторов, начало коюрои указано меткой или номером строки. Последний оператор группы является оператор Return (Return и GoSub — ключевые слова).

Пример:

- Sub Form_Click ()
  Dim *Num '* Объявление переменной.
  Num = Input Bo\(« Введите число.»)
  GoSub *Routine '* Переход к группе операторов (к
  ' внутренней процедуре).

GoTo *Nextparl '* Обход группы операторов (внутренней
'процедуры). *Routine: '* Начало группы операторов (внутренней
'процедуры). Num = Num / 2

Return ' Конец группы операторов (выход из внутренней ' процедуры). *Nextparl: '* Продолжение программы. Msgbox *«Половина введенного числа равна « & Num* End Sub

3.5.2. Оператор присваивания

Оператор присваивания (assignment statement) имеет следующий вид:

[Let] {переменная | элемент массива} = выражение

Переменной или элементу массива в левой части оператора присваивается значение вычисленного выражения в левой части.

Примеры:

```
C = A(f,J+I) * (Cos(B+3.5) + 2 * K л 3) + (P - 3.25)
StartTime = Now
```

1=1+1

Massiv_J(3,4) =B л 2 - 4*A*C + D

Student_1.Birthdate =DateSerial(1975,6,11) - используется встроенная функция *DateSerial* для преобразования трех последовательных чисел (год, месяц, число) в дату.

*P = X+ 2 >=KAnd A + B > C Or A*X + B*Y=D And (H<=Y And Y<=Q)*

Sesia(Current). Name Exam(2)= «Физика «

При использовании оператора присваивания следует соблюдать следующие правила:

Если в левой части оператора используется переменная или элемент массива символьного типа (String), то выражение в правой части должно быть тоже символьное; Если в левой и правой частях оператора используются арифметические данные (Integer. Long, Single, Double, Currency^ но разных типов, то тип правой части» преобразуется к типу левой части. Результатом присвоения значения вещественной константы 2.5 переменной целого типа (I = 2.5) будет 2 (т.е. в ячейке памяти отведенной для переменной I будет храниться значение 2).

Переменной или элементу массива типа Variant в левой части может соответствовать любой тип выражения в правой части (в ячейке памяти для хранения данных типа Variant сохраняется не только значение, но и его тип). Однако такого присвоения желательно избегать.

Опция Let в операторе используется для присвоения значения одного данного пользовательского типа другому, при условии что типы элементов обоих пользовательских данных совпадают.

3.5.3. Условный оператор

Как правило, алгоритмы обработки информации и реализующие их программы содержат проверки каких-либо условии, от которых зависит последующее действие. Для этого предназначен условный оператор, который имеет вид

If логическое выражение Then then-последователпность [Else else -последовательность} или

If логическое выражение Then

[блок операторов — 1} [ElseIf логическое выражение Then
[блок операторов — 2]] [Else
[блок операторов — п] **] End If где If, Then, ElseIf, Else, End If ключевые слова.**
Во второй синтаксической конструкции **If** и **End If** являются как бы открывающей и закрывающей скобкой группы операторов, образующих структурный логический оператор.

Then-последователность и else-последовательность имеют вид

{операторы **\ [GoTo]** ***номер строки*** **\ GoTo** ***метка } операторы*** — последовательность расположенных на одной строке операторов, разделенных двоеточием.

Пример:

If A > 10 Then A = A + I : B = B + A : C = C + B: GoTo

Vsiavka
If *Name = «Иван»* Then GoTo 555

If *Name = «Иван «* Then 555 (эквивалентно предыдущему оператору)

блок операторов 1, блок операторов **2, ...** ***блок операторов п*** **_ последовательность выполняемых операторов.**

Первая синтаксическая конструкция условного оператора обеспечивает альтернативное выполнение then-последователности или else-последовательности в зависимости от значения логического выражения (принимает значение True или False) If *True* Then *'эти операторы* Else *'эти операторы*

выполняются не выполняются 'следующий оператор . If *False* Then *'эти операторы* Else *'эти операторы*
не выполняются выполняются ' следующий оператор
Вторая синтаксическая конструкция условного оператора обеспечивает альтернативное выполнение блоков операторов (блок операторов — 1, блок операторов — 2,... блок операторов — п) в зависимости от значений логических выражений (принимает значение True или False) If *True* Then

'эти операторы блока выполняются Else эти операторы блока не выполняются End If 'следующий оператор If False Then

'эти операторы блока не выполняются Else 'эти операторы блока

выполняются End If • 'следующий оператор Ключевое слово ElseIf позволяет объединить функции Else и следующего вложенного If. Примеры показывают эквивалентность, с точки зрения реализуемого алгоритма, двух фрагментов программ

```
If Answer = "No" Then                    If Answer = "No" Then
    Mark = Mark -1                           Mark = Mark -1
 Else                                    ElseIf If Num_Answer = Max_Num Then
  If Num_Answer = Max_Num Then               Topic = Topic + 1
    Topic = Topic + 1                      Else
  Else                                       Result = Mark
  Result = Mark                          End IF
  End If
End If
```

Во втором примере отсутствует повторное вложение If — End If.

Пример программы.

Программа должна определять количество десятичных цифр в вводимом числе от 0 до 1000. Запуск программы осуществляется щелчком мыши по форме (операторы программы помещаются в процедуру обработки события Foim_Click):

```
    Private Sub Form_Click()
    Dim X, Y   'объявление переменных . X = InputBox(«Введите число больше 0 и меньше
        1000.»)
If X < 10 Then
Y = 1   • 1 цифра. ElseIf X < 100 Thpn
Y = 2   ' 2 цифры. Else
Y = 3   ' 3 цифры. End If
    If Y > 1 Then Unit = « цифры.» Else Unit = « цифру.»
    MsgBox «Введенное число имеет « & Y & Unit End Sub
```

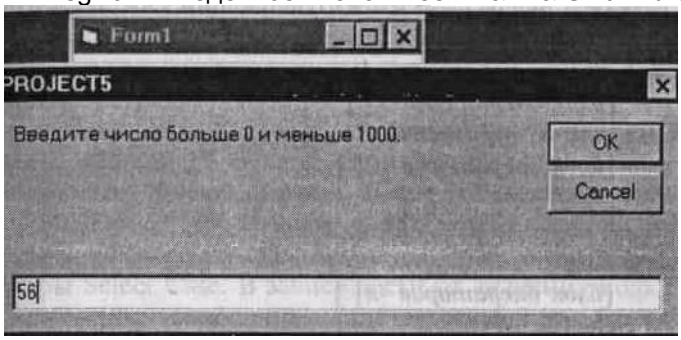


Рис 33

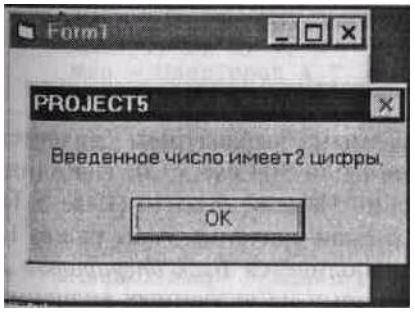
После запуска программы появляется пустая форма Щелчок мыши на форме вызывает появление окна ввода, в котором можно набрать вводимое число и ввести его щелчком мыши по кнопке или клавишей “Enter” (рис. 3 3). После ввода появляется окно с результатом (рис. 3 4)

3.5.4. Управляющая структура Select Case

Структура Select Case применяется, когда одна величина участвует в нескольких логических сравнениях и определяет, какой блок операторов будет выполняться Алгоритм такого множественного сравнения можно запрограммировать и с использованием логического структурного оператора, но применение структура Select Case эффективнее

Наиболее часто структура Select Case применяется в тех случаях, когда сравниваемая величина является целым числом (например, для выбора блоков операторов

программы в зависимости от выбранной альтернативы диалога) *
Рис 34



Структура Select Case имеет следующий синтаксис:

```
Select Case проверяемое выражение
[Case список выражений 1
      [блок операторов-1] ]
[Case список выражений 2
      [блок операторов -2] ]
:
:
[Case Else
      [блок операторов -n] ]
End Select
```

Select Case, Case, Case Else, End Select ключевые слова (Select Case и End Select соответственно оператор начала и конца структуры);
проверяемое выражение — арифметическое или символьное выражение;
список выражении I, список выражении 2...- может иметь одну из следующих форм:

выражение
выражение To выражение
Is знак отношения выражение

Пример:

Select Case *Value*

Case */*, *3*
Блок операторов 1

Case 5 To *10*
Блок операторов 2

Case *12* **Is** >= *15*
Блок операторов 3
Case Else
Блок операторов 4 End Select

В данном примере проверяемым выражением является значение *Value.* Если значение *Value* 1 или 3 (Case */*, *3),* выполняется *Блок операторов I.* Если значение *Value* от 5 до 10 (Case *5* To *10),* выполняется *Блок операторов 2.* Если значение *Value* равно 12 или больше 15 (Case *12* Is >= *15),* выполняется *Блок операторов 3.* Если значение *Value* не равно ни одному из указанных значений

^и не принадлежит ни одному из указанных диапазонов, выполняется *Блок операторов 4.*

Пример программы.

Программа должна определять, является ли вводимый символ большой или малой буквой латинского алфавита или четной или нечетной десятичной цифрой. Проверка символа осуществляется по значению ASCII-кода вводимого символа. Значение ASCII-кода вводимого символа является проверяемым выражением структуры Select Case. В зависимости от значения кода выдается то или иное сообщение (соответствующие блоки

Case-операторов. Запуск программы осуществляется щелчком мыши по форме (операторы программы помещаются в процедуру обработки события Form_Click). Для ввода символа и вывода сообщения используются встроенные функции InpupBox и MseBox.

Текст процедуры:

```
Private Sub Form_Click()
Dim Msg, UserInput   ' Объявление переменных.
Msg = "Введите латинскую букву или число от 0 до 9."
UserInput = InputBox(Msg)   ' Ввод введенного символа.
If Not IsNumeric(UserInput) Then'Это буква или число?
     If Len(UserInput) <> 0 Then
     Select Case Asc(UserInput)   ' Если это буква.
     Case 65 To 90    ' Должны быть большие буквы.
     Msg = "Вы ввели большую букву '"
     Msg = Msg & Chr(Asc(UserInput)) & "'."
     Case 97 To 122   ' Должны быть маленькие буквы.
     Msg = "Вы ввели маленькую букву '"
     Msg = Msg & Chr(Asc(UserInput)) & "'."
     Case Else    ' Не буква и не число.
     Msg = "Вы ввели не букву и не число."
     End Select
     End If
Else
Select Case CDbl(UserInput) ' Если это число.
Case 1, 3, 5, 7, 9   ' Должно быть нечетное число.
Msg = UserInput & " Это нечетное число."
Case 0, 2, 4, 6, 8   ' Должно быть четное число.
Msg = UserInput & " Это четное число."
Case Else'Число не принадлежит диапазону от 0 до 9.
Msg = "Вы ввели число вне "
Msg = Msg & "диапазона от 0 до 9."
End Select
End If
MsgBox Msg   ' Вывод результата.
End Sub
```

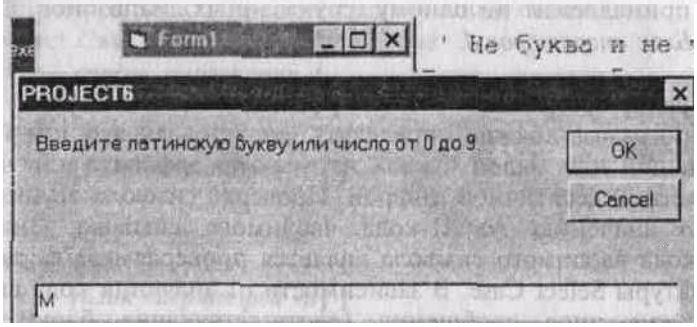


Рис 3 5

В программе используются всгроенные функции Len (определяет длину строки), Cdbl (преобразует тип исходного выражения в тип Double), Is Numeric (проверяет, является ли символ числом), Asc (преобразует символ и ASCII-код), Спг (преобразует ASCII-код в строку).

После запуска программы появляется пустая форма. Щелчок мыши на форме вызывает появление окна ввода, в котором можно набрать вводимый символ и ввести его щелчком мыши по кнопке или клавишей “Enter” (рис. 3.5). После ввода появляется окно, выводящее характеристику введенного символа

Ввод других символов будет вызывать вывод соответствующих сообщений.

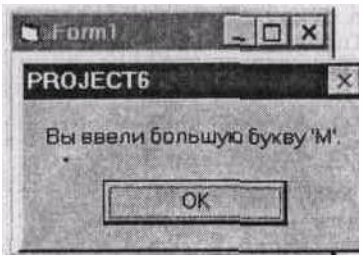


Рис 3 б

3.5.5. **Циклы**

В алгоритмах обработки информации и реализующих их программах широко используются циклы — повторяющиеся одинаковые вычисления.

Для реализации такого рода программ в Visual Basic специальные средства — операторы цикла.

3.5.5.1. Оператор цикла For-Next

Синтаксис оператора следующий:

For параметр цикла = начшьное значение To конечное значение [Step
шаг}
{блок операторов}
[Exit For] ***[блок операторов^*** **Next** ***{параметр цик-га {,параметр цик-ш*** **][, ...]]**

где **For, To, Exit For, Step, Next** ключевые слова. Пара **For-Next** определяют начало и конец оператора цикла. Операторы между ними *(б->ок операторов)* повторяются столько раз, сколько определено. задаваемыми *начальным значением, конечным значением* и

шагом.

Exit For может находится в любом месте между For - Next и используется для прекращения выполнения цикла (управление передается на оператор, который следует сразу за Next) и применяется, например, в логическом операторе, который проверяет альтернативное условие прерывания цикла (например, ошибку).

Параметр цчюш - арифметическая переменная, не может быть элементом массива или элементом пользовательского типа

данных.

Начальное значение, конечное значение **и** ***шаг*** определяют значения, которые принимает параметр цикла при работе программы - на первом шаге параметр цикла принимает начальное значение, после выполнения операторов, входящих в цикл (блок операторов), параметр цикла изменяется на величину шага (выполняется оператор Next), опять выполняются операторы, входящие в цикл, параметр цикла изменяется на величину шага и т. д., пока параметр цикла не примет последовательно все свои

значения.

После того, как параметр цикла примет все свои значения и соответственное число раз выполнится блок операторов в цикле, будет выполняться следующий за Next оператор.

. Операторы, входящие в цикла будут выполняться если:

шаг цикла >= 0 и конечное значение >= начальное значение шаг цикла < 0 и конечное значение <= начальное значение. ***Пример.***

Рассматривается фрагмент программы для нахождения максимального значения элементов одномерного массива из 15 элементов.

```
Dim B (1 To J5), MaxB As Single 'оператор объявления массива и переменной
MaxB = B(1) 'присвоение начального значения MaxB For /= /To 15  'начало цикла
If B(I) > MaxB Then MaxB = B(1) 'этот оператор выполнится 15 раз
Next /       'конец цикла MsgBox Str(MaxB) 'для вывода результата используется
'встроенная функция MsgBox, в которой 'используется встроенная функция
Str 'для преобразования числового значения в строку
```

При использовании оператора цикла необходимо соблюдать правила:

1. Следует избегать изменения значения параметра цикла в каких-либо операторах внутри цикла.
   - 2. Передача управления на операторы внутри цикла (кроме первого) из каких-либо операторов вне цикла запрещена.

Неправильный фрагмент

```
GoTo 555
.....
For K = 1 To 105
.....
555:
C(I) = A(I) + B(I)
Next K
```

Правильный фрагмент

```
GoTo 555
.....
555:
For K = 1 To 10
C(I) = A(I) + B(I)
Next K
```

Внутри одного цикла может находиться другой цикл

```
For I = 1 To 3
   For J= 1 To 4
    C(I,J) = A(I) * B(J)
   Next J
Next I
```

```
For I = 1 To 3
   For J= 1 To 4
    C(I,J) = A(I) * B(J)
Next J, I
```

Оба фрагмента программ эквивалентны, т.е. два цикла могут заканчиваться одним оператором Next.

При выполнении цикла в цикле внутренний цикл выполняется для каждого значения параметра внешнего цикла (в фрагментах программ оператор C(I,J) = A(I) * B(J) выполнится 12 раз со следующей последовательностью индексов: C(1,1), C(1,2), C(1,3), C(1,4), C(2,1), C(2,2), C(2,3), C(2,4), C(3,1), C(3,2), C(3,3),

C(3,4)).

Внутренний («вложенный») цикл должен целиком содержаться во внешнем цикле:

Число вложенных циклов не ограничено.

Пример.

Рассматривается фрагмент программы для формирования вектора A(4). элементами которою являются суммы столбцов матрицы B (5,4).

Dim *A(1* To *4), B(1* To 5, / To *4)* As Single'операТор объявления массивов

```
For J = I To 4
    A(J) = 0 'присвоение начального значения
    'элемента вектора For / = / To 5
    A(J) = A(J) + B(/,J) 'вычисление суммы в цикле Next / Next J
```

Пример программы.

Определен массив записей для хранения результатов экзаменов

Dim *Sesia (5)* As *Resiiitaf* на основе пользовательского типа данных Type *Reyiltat*
Familia As Sti ing * *20 Name Exam (1* To *3)* As Stung * /5 *ReyiltExam* (/ To J) As Integer End Type

Программа должна вводить исходные данные и определять среднюю оценку студента, фамилия которого задается

Поместим описание пользовательской структуры данных в раздел Geneial модуля Module! (рис 3 7) и создадим форму (рис 3 8).

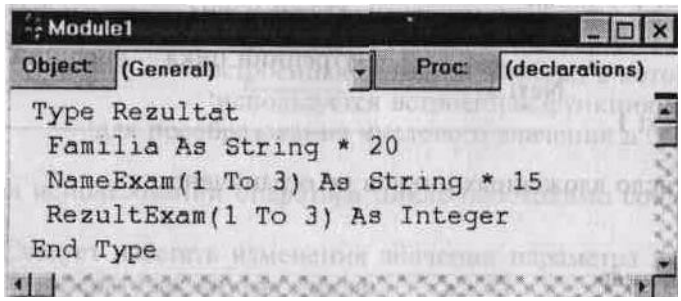


Рис 37

Форма и программы процедур обработки событии обеспечивают следующую логику работы При загрузке формы меткам (Label I, Label2, Label3) присваиваются названия экзаменов и активизируется командная кнопка «Ввод», командная кнопка «Вычисление» не активна Последовательное пятикратное щелканье мыши по кнопке «Ввод» после заполнения информацией текстовых окон обеспечивает ввод необходимых исходных данных После чего командная кнопка «Ввод» деактивизируется, а командная кнопка «Вычисление» активизируется После ввода в текстовое окно под ней фамилии и щелчка мыши по кнопке «Вычисление» производится вычисление средней оценки студента и результат выводится

в текстовое окно

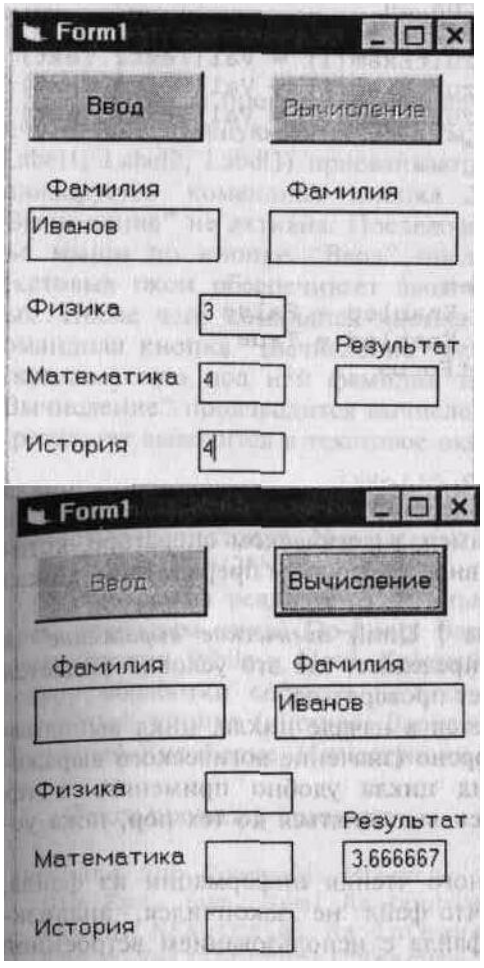
Рис 38

Тексты процедур:

```
Dim Sesia(1 To 5) As Rezultat, I As Integer

Private Sub Command1_Click()
    Sesia(I).NameExam(1) = "Физика"
    Sesia(I).NameExam(2) = "Математика"
        Sesia(I).NameExam(3) = "История"
        Text1.SetFocus
        Sesia(I).Familia = Text1.Text
        Sesia(I).RezultExam(1) = Val(Text2.Text)
        Sesia(I).RezultExam(2) = Val(Text3.Text)
        Sesia(I).RezultExam(3) = Val(Text4.Text)
        Text1.Text = ""
        Text2.Text = ""
        Text3.Text = ""
        Text4.Text = ""
        I = I + 1
        If I > 5 Then
           Command1.Enabled = False
           Command2.Enabled = True
           Text5.SetFocus
          End If
End Sub

Private Sub Command2_Click()
Dim Sredn As Single, Familia As String * 20
Dim I, J As Integer
```

```
Familia = Text5.Text
For I = 1 To 5
If Familia = Sesia(I).Familia Then
     Sredn = 0
     For J = 1 To 3
     Sredn = Sredn + Sesia(I).RezultExam(J)
     Next J
     Sredn = Sredn / 3
End If
Next I
Text6.Text = Str$(Sredn)
End Sub

 Private Sub Form_Load()
     Label2.Caption = "Физика"
     Label3.Caption = "Математика"
     Label4.Caption = "История"
     Command2.Enabled = False
     Command1.Enabled = True
     I= I+1
End Sub
```

Работа с формой показана на рис.3.9, а, б.



3.5.5.2. Оператор цикла Do-Loop

Синтаксис оператора следующий

Do [{While | Until} логическое выражение]

[блок операторов}
[Exit Do]
[блок операторов]

Loop

или

Do

[блок операторов }
[Exit Do]
[блок операторов]

Loop [{While | Until} *логическое выражение*] где Do, While, Until, Exit Do, Loop ключевые слова. Пара Do-Loop определяют начало и конец оператора цикла;

While определяет выполнение операторов, входящих в цикл, пока стоящее следом логическое выражение принимает значение «True»;

Until определяет выполнение операторов, входящих в цикл, до тех пор пока стоящее следом логическое выражение не примет значения “True”; **Exit Do** используется для прекращения выполнения цикла (управление передается на оператор, который следует сразу за Loop) и применяется, например, и логическом операторе, который проверяет альтернативное условие прерывания цикла (например, ошибку).

Запись условия “{While | Until} *логическое выражение»* в начале или в конце цикла определяет, где это условие (задается логическим выражением) будет проверяться.

Когда условие проверяется в начале цикла, цикл выполняется, если условие удовлетворено (значение логического выражения равно True). Такой вид цикла удобно применять в тех случаях, когда цикл не должен выполняться до тех пор, пока условие не будет выполнено.

Пример последовательного чтения информации из файла, когда необходима проверка, что файл не закончился, анализируя наличие записи конца файла с использованием встроенной функции EOF (номер файла):

Do Until EOF(5) операторы чтения файла Loop	Do While Not EOF(5) операторы чтения файла Loop

Операторы чтения файла будут выполняться только в случае значения функции EOF — False (текущая запись не является концом файла).

Запись условия в конце цикла означает, что цикл выполнится хотя бы один раз (при этом первом проходе обычно формируется условие, которое затем будет проверяться).

Пример.

Do I Do *Password* = InputBox («Введите пароль») *Password* = InputBox («Введите пароль»)

Loop While *Password О* «Секрет»] Loop Until *Password* = «Секрет»

В данном примере условие стоит в конце и ввод пароля (обращение к встроенной функции InputBox) выполняется хотя бы один раз (формируется проверяемое в конце условие). В первом случае используется While, определяющее повторение цикла пока «Секрет» не совпадает с введенным значением. Во втором случае используется Until, определяющее повторение цикла до момента совпадения «Секрет» с введенным значением.

Пример программы.

Программа производит вычисление ряда с задаваемой точностью вычисления. М-и элемент ряда вычисляется по формуле

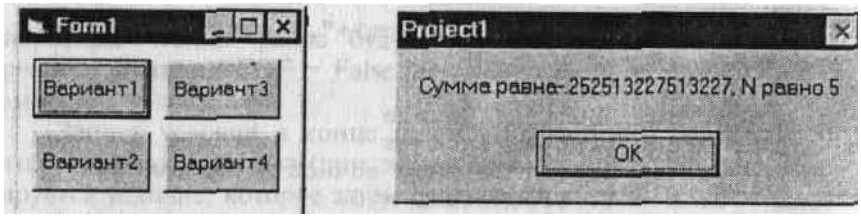
(-irN/(N!*(2*N+I))

Программа реализуется в четырех вариантах с использованием оператора цикла Do-Loop. Варианты определяют место записи условий While и Until. Каждый вариант помещается в процедуру обработки события щелчка мыши по соответствующей командной кнопке (кнопки «Вариант!», «Вариант2», «Вариант3», «Вариант4») на форме. Иллюстрации работы примера на рис. 3.10.

Текст процедур примера:

```
Private Sub Commandl_Click () Dim Eps, Sum, Sumi As Double Dim N, NFactorial As Integer Eps = Val(InputBox(«Введите точность вычисления»).) Sum = 0: N = 1: NFactorial = 1 Do While Abs((-1)”N/(NFactorial*(2 * N + 1))) > Eps
Sum = Sum + (-1) ^ N /(NFactorial * (2 * N + 1)) • N = N + 1
NFactorial = NFactorial * N Loop
MsgBox «Сумма paBHa”+Str$(Sum)+”,N равно» + Str$(N) End Sub
Private Sub Command2_Click() Dim Eps, Sum, Sumi As Double Dim N, NFactorial As Integer
Eps = Val(InputBox(«Введите точность вычисления»)) Sum = 0: N =- 1: NFactorial = 1
Do Until Abs ( (-1)”N/(NFactorial*(2 * N + 1))) < Eps Sum = Sum + (-1) ^ N / (NFactorial * (2 * N + 1)) N = N + 1
NFactorial = NFactorial * N Loop
MsgBox «Сумма равна»+31r$(Sum)+”, N равно»+ Str$(N) End Sub
Private Sub Command3_Ciick() Dim Eps, Sum, Sumi As Double
Dim N, NFactorial As Integer
Eps = Val(InputBox(«Введите точность вычисления»))
Sum =0: N = 1: NFactorial - 1
Do  Sum ^ Sum + (-1;  N / (NFactorial * (2 *  N + 1))'

N = N +  1

    NFactorial = NFactorial * N
Loop Wnile Abs ( (-1)'”N/(NFactorial* (2 * N +  !)))> Eps
MsgBox «Сумма paBHa”+Str$(Sum)+ “, N равно»+ Str$(N) End Sub
Private Sub Command4_Ciick () Dim Eps, Sum, Sumi As Double Dim N, NFactorial As Integer Eps = Val(InputBox(«Введите точность вычисления»))
Sum = 0: N = 1: NFactoriai = 1 Do

Sum = Sum + (-1) ^ N / (NFactorial » (2 * N + 1))

N = N + 1

NFactorial = NFactoriai * N Loop Until Abs((-1)”N/(NFactorial*(2*N + !)))< Eps
MsgBox «Сумма paBHa”+Str$(Sum).”, N равно» + Str$(N) End Sub
```

Циклы Do-Loop позволяют также строить циклы со счетчиком, аналогично циклам For-Next.



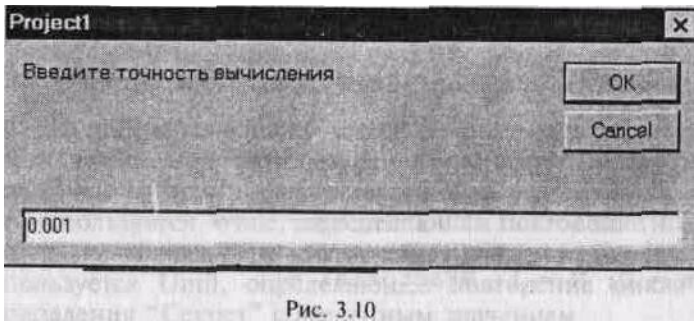


Рис. 3.10

Пример.

Counter = 1 \ *Counter = 1*
Do While *Counter* <= *100* Do While *Counter* > *100*
B(Counter) = ... B(Counter) = ...
Counter = Counter + 1 *Counter = Counter + 1*
Loop Loop

Эти два фрагмента программы эквивалентны следующему фрагменту с использованием цикла For — Next.

For *Counter = 1* To 100
B*(Counter)* = ...
Next Counter

Для организации ввода исходных значений элементов массивов удобно использовать операторы цикла при работе For-Next и Do-Loop. Примеры процедур обработки события щелчка мыши по форме, которые обеспечивают последовательный ввод элементов массива в специальном окне InpiitBox и их вывод в специальном окне MsgBox:

```
    Sub Form_Click ()
    Dim I As Integer
Static A(1 To 5)
I = 1
Do While I <= 5
    A ('I^=Val (InputBox ( «Введите элемент массива» ' Str(-T) ) )
    MsgBox «Значение элемента массива» •+ Str(A(I))
1=1+1
    Loop
    End Sub
    Sub Form_Click ()
    Dim I As Integer
    Static A(1 To 5) 1=1 Do Until I > 5
    A (I)=Val(InputBox(«Введите  элемент массива»  + Str(-I) ) )
    MsgBox «Значение элемента массива» + Str(A('I))
    1=1+1
    Loop
    End Sub
```

Окна для ввода и вывода в приведенных примерах имеют вид на рис. 3.11 и 3.12. Альтернативным способом ввода нескольких значении элементов массива может быть использование текстового окна и его свойств SelLength, SelStart, SelText (см описание текстового окна) В определенных позициях текстового окна можно задавать значения различных элементов массива и с использованием указанных функции эти поля выделять (аналогично тому, как осуществлялся ввод данных в текстовой строке ввода в операционнои системе DOS и др) Создадим форму и разместим на ней

текстовое окно, определив максимальное число символов 80 (свойство MaxLength = 80)

Исторический экскурс' когда-то недавно на перфокарте можно было «набить» 80 символов и строка фортран-программы состояла из 80 позиции Каждый элемент массива будем располагать последовательно в четырех позициях строки (число символов вводимого числа, включая десятичную точку, не превышает 4) Пробелы между числами отсутствуют

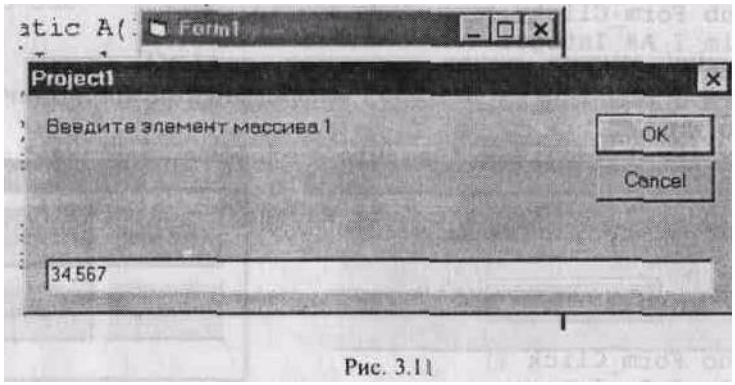
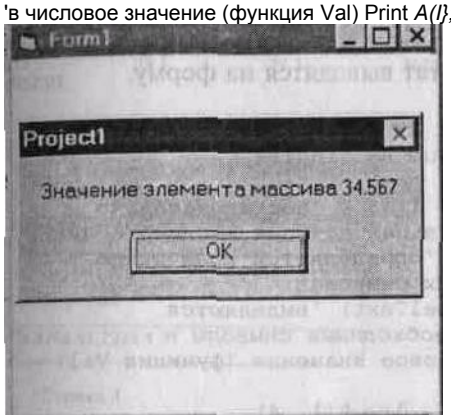


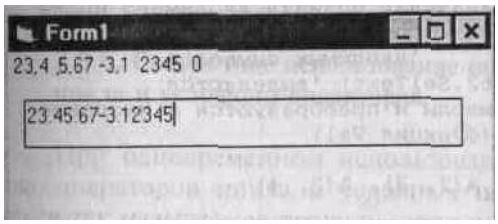
Рис. 3.11

Имя текстового окна Input_Txt Остальные параметры определяются по умолчанию Определим процедуру обработки события нажатия клавиши KeyPiess

```
Sub Inpui:_Txt_KeyPress (KeyAscii As Integer) Static A(1 To 5) As Single If KeyAscii = 13 Then For I = 1 To 5 Input-_txt.
SalStart-f'I 1) *4 'последовательно
'определяется начальная позиция вводимого числа Input_txt.SelLength= 4
'определяется количество ' вводимых символов

A(I) = Val(Input_txt.SelText) 'выделяются 'необходимые символы и преобразуются
'в числовое значение (функция Val) Print A(I}; 'печать введенных значении на форме Next I End If
```



Процедура обеспечивает ввод значении пяти элементов массива при нажатии клавиши “Enter” (ASC1I-код клавиши равен 13). Стоящая «,» после оператора Punt A(I) обеспечивает печать следующего элемента массива на той же строке сразу после предыдущего.



Альтернативно можно использовать «,» (выводимые значения располагаются на одной строке в фиксированных зонах формы) При отсутствии указанных символов выводимые значения располагаются на отдельных строках На рис 3 13 показаны вводимые символы и распечатываемые значения **Пример программы.**

В качестве примера приводится программа нахождения суммы элементов матрицы 3*4 (три строки и четыре столбца), хотя бы один индекс которых кратен трем.

Текст программы размешается в процедуре обработки события щелчка мыши по форме. Для ввода исходных данных (значений каждого из элементов матрицы) используется три текстовых окна (одно окно для каждой строки матрицы). Каждое значение занимает последовательные четыре позиции строки окна. Введенная матрица и результат выводятся на форму.

Текст процедуры:

```
Private Sub Form_Click ()
Static A(1 To 3, 1 To 4), Sum As Single For J = 1 To 4 Input_txtl.SelStart = (J - 1) * 4
    'последовательно
«определяется начальная позиция вводимого числа Input_txtl.SelLength = 4
    'определяется количество
'вводимых символов A(1, J) = Val(Input_txtl.SelText) 'выделяются
'необходимые символы и
'преобразуются в числовое значение (функция Val) Next J
Print A(1, 1), A(1, 2), A(1, 3), A(1, 4) For J = 1 To 4 Input_txt2.SelStart = (J - 1) * 4
    'последовательно
'определяется начальная позиция вводимого числа Input_txt2.SelLength = 4
    'определяется количество
'вводимых символов A(2, J) = Val(Input_txt2.SelText) 'выделяются
'необходимые символы и преобразуются в числовое
'значение  (функция Val) Next J
Print A(2, 1), A(2, 2), A(2, 3), A(2, 4) For J = 1 To 4 Input_txt3.SelStart = (J - 1) * 4
    'последовательно
'определяется начальная позиция вводимого числа Input txt3.SelLength = 4
    'определяется количество
'вводимых символов A(3, J) = Val(Input_txt3.SelText) 'выделяются
'необходимые символы и преобразуются
'в числовое значение (функция Val) Next J
Print A(3, 1), A(3, 2), A(3, 3), A(3, 4) Sum = O
```

For I = 1 To 3 алгоритм For J = 1 To 4 суммирования IfI/3=I\30rJ/3=J\3 Then элементов Sum = Sum + A(I, J) матрицы, один

```
    End If                                  из индексов
  Next J, I                                 которых кратен 3
Print Sum
End Sub
```

Форма с введенными данными и результатом расчета приводится на рис. 3.14.

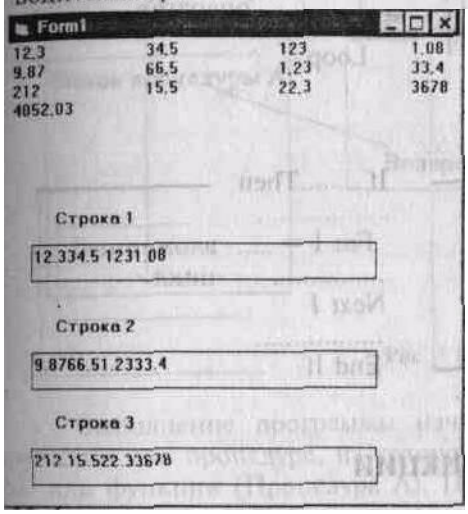


Рис. 3.14

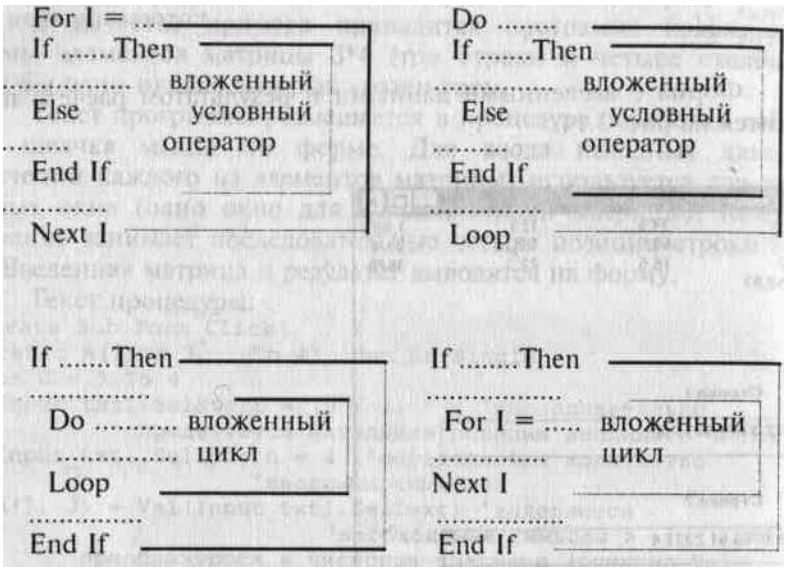
3.5.6. Совместное использование операторов цикла и условного операторов

При одновременном использовании в процедурах и функциях операторов цикла и условных операторов должно выполняться так называемое правило вложенности.

Если среди операторов, выполняющихся в цикле (циклы For-Next и Do-Loop), имеется условный оператор (If-Endlf), то условный

оператор должен целиком содержаться внутри цикла (между операторами For-Next или Do-Loop). Если в Then — блоке или Else ~ блоке условного оператора It'-Endlf имеются выполняющиеся в цикле операторы (циклы For-Next и Do-Loop), то эти циклы Должны целиком содержаться в этих блоках.

Следующие схемы иллюстрируют принцип корректного совместного использования циклов и условных операторов:



3.6. ПРОЦЕДУРЫ И ФУНКЦИИ

- В предыдущих разделах рассматривались процедуры обработки событий, которые в процессе выполнения программы инициировались (запускались) в результате свершения некоторого определенного события (щелчок мыши, нажатие клавиши и т.п.). Однако в Visual Basic имеется возможность выделить неоднократно повторяющиеся фрагменты программы в определенным образом организованные функциональные блоки операторов, которые можно использовать без их повторного написания и отладки. При этом сложные программы получаются более компактными, а функциональные блоки при их правильной организации могут использоваться в других программных разработках. Организовать функциональные блоки в Visual Basic можно в виде процедур (procedure) и функции (function).

Любая программа на Visual Basic является процедурой или функцией.

Схема, показанная на рис. 3.15, иллюстрирует последовательность выполнения программы, состоящей из процедур и функций (рис. 3.15).

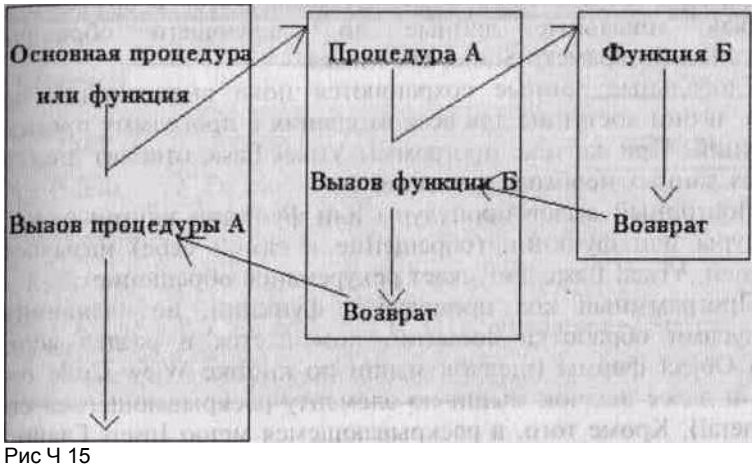


Рис Ч 15

Выполнение профаммы начинается в основной (первой) функции или процедуре, IIJ которой вызываются другие процедуры или функции (Процедура А). После вызова выполняется вызванная процедура (Процедура А), которой передается управление выполнением программы. В

свою очередь в вызванной процедуре могут быть обращения к другим процедурам или функциям (Функция Б) и т.д. (Visual Basic отслеживает до нескольких сот уровней вызовов).

Прекращение выполнения вызванной процедуры или функции (Возврат) означает возвращение в вызывающую процедуру или функцию, выполнение которой продолжается с оператора, следующего за вызывающим оператором.

В программе, состоящей из нескольких процедур или функций, необходимо остановиться на понятии локальные и глобальные данные.

Локальные данные определены только в момент выполнения процедуры или функции и память для них выделяется только на момент выполнения.

После завершения выполнения процедуры или функции локальные данные не сохраняются (выделенная память используется Другими процедурами и функциями). При этом экономится память, так как она выделяется только по мере необходимости и операционные системы современных компьютеров это делают очень эффективно. Однако и Visual Basic имеется и возможность сохранять локальные данные до следующего обращения (специальный параметр Static, см. ниже).

Глобальные данные сохраняются пока выполняется программа и они доступны для всех входящих в программу процедур и функции. При запуске программы Visual Basic отводит для глобальных данных необходимую память.

Повторный вьпов процедуры или функции внутри этой же процедуры пли функции (обращение к самой себе) называется рекурсией. Visual Basic допускает рекурсивное обращение.

Программный код процедур и функций, не являющихся процедурами обработки событии, помещается в раздел general списка Object формы (щелчок мыши по кнопке Wiew Code окна Project и далее щелчок мыши по элементу раскрывающегося списка general). Кроме того, в раскрывающемся меню Insert Главного меню имеется пункт Procedure (рис. 3.16). выбор которого раскрывает специальное окно для выбора процедуры или функции п задания имени (рис. 3.17).

Для заданной процедуры или функции появляется шаблон (заготовка) в окне кода (первая п последняя строки программы), в который можно ввести текст программы.

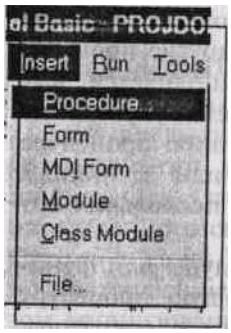


Рис. 3 16

3.6.1. Процедуры

Синтаксически процедура определяется [Private | Public] [Static] Sub *имя процедуры [(список аргумен-*

[операторы объявления] *[операторы}* [Exit Sub] *[операторы}* End Sub

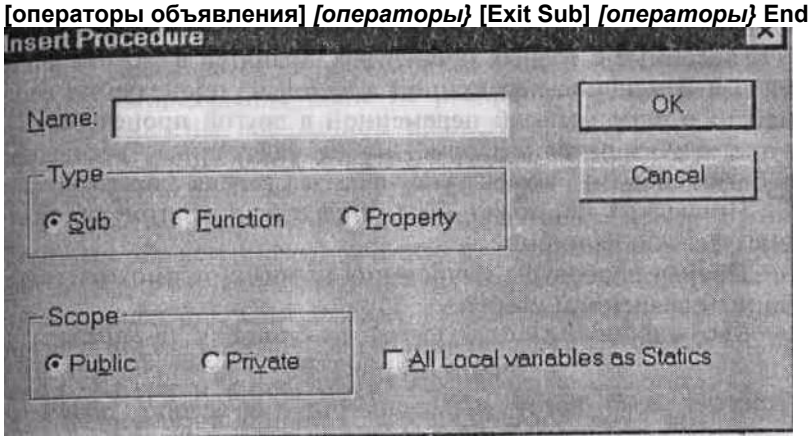


Рис. 3.17

где **Private, Public, Static, Sub, Exit Sub, End Sub -** ключевые слова. **Sub** определяет обязательный первый оператор процедуры. **End Sub** - обязательный последний оператор процедуры;

имя процедуры образуется в соответствии с общими правилами образования имен, но не может иметь описателей типа (имя процедуры не принимает значения);

список аргументов имеет следующий синтаксис:

[Optional] [By Val | ByRef] [ParainArray] *имя [(*)] [As *тип]* где Optional, ByVal, ByRef, ParaniArray и As ключевые слова;

имя — имя переменной, массива (в случае массива используется *имя ( )* без указания границ значения индексов, что позволяет использовать одну процедуру для разного числа элементов массива в каждом конкретном случае), элемента управления или формы (в последних двух случаях тип принимает значения Control и Form).

Тип может быть Byte, Boolean, Integer, Long, Currency, Single, Double, Date, String (только переменной длины). Variant, пользовательский тип или Control и Form. As *тип* необходимо применять для каждого аргумента.

Список аргументов определяет связь по данным между вызывающей (процедура, из которой происходит вызов) и вызываемой (процедура, которая вызывается) процедурами. Локальные данные, определенные внутри процедуры (кроме тех, что определены в списке general для форм), не могут использоваться в других процедурах (например, переменные с одинаковыми именами, но объявленные в разных процедурах, хранятся в разных ячейках памяти и изменение переменной в одной из процедур не ведет к изменению одноименной переменной в другой процедуре). Процедуры, относящиеся к форме, но не являющиеся процедурами обработки событий, помещаются в раздел general формы.

Процедура не может быть определена внутри какой-либо процедуры или функции.

Пример процедуры вычисления площади прямоугольника по заданным значениям сторон:

Sub *SubDemo (RLen, Rwid) '* 'заголовок процедуры
‘SubDemo с двумя аргументами-
‘R1en-длина и Rwid-ширина
Dim *Area '*объявление локальной переменной *Area = RLen * Rwid ' *расчет площади *MsgBox «Площадь равна» & Area* «печать результата End-Sub

Пример процедуры на основе рассмотренного выше фрагмента программы формирования вектора из сумм элементов столбцов матрицы:

Sub SubMatrVektor (A() As Single, B() As Single, .
MaxI As *Integer, MaxJ* As *Integer)’* Dim *J, I* As *Integer For J = 1* To *MaxJ* 'верхняя граница индекса J
'определяется передаваемым 'аргументом процедуры Л*(J) = 0* 'присвоение начального значения
'элемента вектора For *I = 1* To *MaxI* 'верхняя граница индекса
'I определяется
Передаваемым аргументом процедуры *A(J) = A(J) + B(I,J)* 'вычисление суммы в цикле Next I Next J End Sub
Изменяя значения аргументов данную процедуру можно использовать для работы с матрицами произвольного размера.

Рассмотрим, что означают другие используемые ключевые слова в определении процедуры.

Exit Sub вызывает прекращение выполнения процедуры и пыход из нее в вызывающую программу.

Static определяет, что локальные данные (переменные, массивы), определенные внутри процедуры (переменная Area в примере), сохраняются в промежутках между вызовами данной процедуры. Определение Static не распространяется на данные, которые объявлены вне данной процедуры, но используются в ней (т.е. эти данные будут изменяться независимо от определения Static). Нельзя использовать при рекурсивном вызове процедуры.

Private определяет, что процедура может быть вызвана (доступна) только ц том модуле, в котором она объявлена (никакие другие процедуры в других

модулях не могут вызвать данную процедуру). Определение Private не имеет смысла, если процедура определена для какой-либо формы (в этом случае процедура недоступна из вне данной формы).

Public определяет, что процедура может быть вызвана из любого модуля приложения.

Поскольку имя процедуры определяется (глобально распознается) в рамках всех процедур всех модулей одной программы, имя процедуры не должно совпадать с другими глобально распознаваемыми именами программы. Такими глобально распознаваемыми именами являются имена других процедур Visual Basic или процедур динамически подключаемых библиотек (dynamic-link library [DLL|) и данных, объявленных как Global (см. выше).

Чтобы избежать конфликта в объявлении одних и тех же имен, можно использовать определение Private, т.е. сделать процедуру доступной только в рамках модуля. В этом случае имя процедуры не может совпадать с такими именами, объявленными на уровне модуля, как имена переменных, массивов, констант и имена других процедур данного модуля.

Optional определяет, что аргумент не является обязательным. При его использовании все последующие аргументы в списке аргументов также являются необязательными и для них также необходимо использовать данное ключевое слово. Аргументы, объявленные Optional, должны иметь тип Variant. Не может использоваться для аргумента, для которого задан параметр ParamArray.

ByVal определяет, что аргумент передается в процедуру по значению (см. ниже).

ByRef определяет, что аргумент перелается в процедуру по ссылке (см. ниже).

ParamArray относится к последнему аргументу в списке аргументов и определяет, что конечный (заключительный) аргумент является необязательным массивом. Не может использоваться совместно с параметрами ByVal, ByRef, Optional.

Пример.

```
Sub ReturnTwice(ReturnValue, Optional A) If IsMissing(A) Then    'функция IsMissing ()
«проверяет передачу аргумента А в процедуру ReturnValue =Null 'если аргумент не передается,
'вычисляемое значение присваивается О Else ReturnValue = A * 2 'если аргумент передается,
'вычисляется значение от аргумента End If End Sub
```

Синтаксис оператора обращения к процедуре следующий:

имя процедуры[список значении} или

Call имя процедуры[{список значении)}

где Call — ключевое слово;

список значений — список значений аргументов (соответствует списку аргументов в операторе Sub по количеству, порядку следования и типу, кроме случая когда используется параметр Optional).

Кроме того должно выполняться следующее соответствие между списками аргументов и значениями:

Аргумент	**Значение**
Переменная	Константа, переменная, элемент массива, выражение.
Массив	Массив
форма	Форма
Элемент управления	Элемент управления

Следующие примеры показывают правильное согласование между списками аргументов и значении:

Пример 1.

```
Sub Sum (A As Integer, B As Integer, C As Integer)
'список аргументов состоит из переменных C = A + B • End Sub Sub Form Click ()
Static X(3) As Integer X(1) = 1 X(2) =2
Call Sum(X(l), X(1) + X(2), X(3)) 'список
' -значений включает элементы массива и выражение
Print X(1), X(2), X(3) End Sub
```

Пример 2.

```
    Sub Sum (A As Integer, B As Integer, C As Integer)
    'список аргументов состоит из переменных C—A + B End Sub Sub Forrri_Click ()
    Dim X, Z As Integer
X = 5
    Call Sum(X, 4, Z) 'список значений включает
    ' переменные и константу Print X, Z End Sub
```

Пример 3.

```
    Sub Sum (A ( ) As Integer)
    'список аргументов включает массив
    A(3) = A(1) + A(2) End Sub
    Sub Form_Click () Static X(3) As Integer
X(1) - 1
X(2) =2
    Sum X ( ,)'список значений включает массив
    Print X(1), X(2), X(3) End Sub
```

Следует обратить внимание, что в последнем примере обращение к процедуре не содержит ключевого слова Call и отсутствуют скобки, обрамляющие список значений.

Для рассмотренного примера списка аргументов с использованием параметра Optional, обращения могут иметь следующий вид:

```
    Call ReturnTwice(ReturnValue, )
    'значение ReturnValue равно О
```

Call ReturnTwice(ReturnValue, 2)
‘ значение ReturnValue равно 4 **Пример программы.**

Поместим рассмотренную выше процедуру формирования вектора из сумм элементов столбцов матрицы в раздел general формы:

Option Explicit

Sub SubMatrVektor (A() As Single, B() As Single, MaxI As Integer, Max.J As Integer) Dim J, I As Integer For J == 1 To MaxJ A(J) -- 0 For I = 1 To MaxI

A(J) = A(J) + B(I, J) Next I Next J End Sub
Private Sub Form_Ciick()
Static X(1 To 2, 1 To 3) As Single, Y(1 To 3) As Single
X(1, 1) = 1
X(1, 2) = 2
X(1, 3) = 3
X(2, i) = 4
X(2, 2} = 5
X(2, 3) = 6
SubMatrVektor Y(), X(), 2, 3
Print Y(Ii, Y(2), Y(3) End Sub
Результат выводится на форму, показанную на оис. 3.18.

Рис. 3.1S

3.6.2. Функции

Синтаксически функция определяется
[Public | Private] [Static] Function *имя функции [(список аргументов)}* [As тип]
[операторы объявления]
[операторы}
[имя функции = выражение]
[Exit Function] [операторы] [имя функции = выражение] End Function

где Public, Static, Private, Function , Exit Function , End Function —

ключевые слова. **Function** определяет обязательный первый оператор функции. **End Function** — обязательный последний оператор функции:

имя функции образуется в соответствии с общими правилами образования имен и может иметь описатель типа (As **тип** в конце первой строки относится к имени функции). Имя функции принимает значение и хотя бы один оператор

имя функции = выражение

может присутствовать внутри функции и выполняться при выходе из нее. Если никакое значение не присвоено имени функции, то имя функции принимает значение по умолчанию: числовая функция принимает значение 0, функция, объявленная как String, принимает значение пустой строки нулевой длины («»), функция, объявленная как Variant принимает значение Empty (значение, которое принимает имя функции, называют возвращаемым значением функции — возвращаемое значение).

Список аргументов имеет следующий синтаксис:

[Optional] [ByVal|ByRef][ParamArray] *имя[(*)][As *тип]* [.[Optional] [By Val | ByRef][ParamArray] *имя[( )]* [As *тип]] ...* где Optional, ByVal, ByRef, ParamArray и As ключевые слова;

имя — имя переменной, массива (в случае массива используется ***имя ( )*** без указания границ значения индексов, что позволяет использовать одну функцию для разного числа элементов массива в каждом конкретном случае), элемента управления или формы (в последних дйух случаях тип принимает значения Control и Form).

Тип может быть Byte, Boolean, Integer, Long, Currency, Single, Double, Date, String (только переменной длины). Variant, пользовательский тип или Control и Form. As *тип* необходимо применять для каждого аргумента.

Список аргументов определяет связь по данным между вызывающей (функция, из которой происходит вызов) и вызываемой (функция, которая вызывается) процедурами или функциями. Локальные данные, определенные внутри функции (кроме тех, что определены в списке general для форм), не могут использоваться в других процедурах и.'функциях (например, переменные с одинаковыми именами, но объявленные в разных функциях, хранятся в разных ячейках памяти и изменение переменной в одной из функций не ведет к изменению одноименной переменной и другой функции).

Exit Function вызывает прекращение выполнения функции ц выход из нее в вызывающую программу.

Функция не может быть определена внутри какой-либо процедуры или функции.

Пример функции вычисления площади прямоугольника по заданным значениям сторон:

```
Function FunDemo (RLen, Rwid) As Single
'заголовок функции FunDemo с «двумя аргументами- Rlen-длина и Rwid-ширина
«тип имени функции определен как Single Dim Area ' объявление локальной
'переменной Area = RLen * Rwid ' расчет площади MsgBox «Площадь равна» & Area  ' печать
    'результата. FunDemo = Area 'имени функции
    'присваивается вычисленное значение End Function
```

Пример функции для вычисления среднего значения элементов одномерного массива:

```
Function FunSumVecior (A( )} As Single,
MaxI As Integer) As Single
Dim SumVector As Single, I As Integer
Surn Vector = 0
For 1=1 To MaxI ' верхняя граница индекса 'I определяетсяпередаваемым аргументом процедуры
SumVector =SwnVector + A(I)
Next I
FunSuni Vector = SumVector/MaxI 'имени
'функции присваивается вычисленное значение End Function
```

Изменяя значения аргументов данную функцию можно использовать для

работы с вектором произвольного размера.

Используемые ключевые слова аналогичны рассмотренным выше для процедур.

Приведем примеры заголовков функций с использованием параметров ParamArray и Optional:

Function (‘.ilcSur-i (ByVa;: FirstArg As Integer, par^Array OtnerArqri;;^
Function MyFunc.My^tr As String, Optional MyArgI, Opt--^1”1”-‘1 MyArg2)

Обращение к функции является операндом выражения (т.е. в отличии от процедуры для ее вызова не используется специальный оператор), имеет следующий синтаксис

имя функци11([список значений})

список **значении** - список значений аргументов (соответствует списку аргументов в операторе Function по количеству, порядку следования и типу. если не используется параметр Optional). Кроме того должно выполняться следующее соответствие между списками аргументов и значениями:

Аргумент	Значение
Переменная	Константа, переменная, элемент массива, выражение.
Массив	Массив
Форма	Форма
Элемент управления	Элемент управления

Пример 1.

Function *Sum (A* As Integer, B As Integer) As I n t e g e r

‘список аргументов состоит из переменных • C - A + B Sum = C Knd Function Sub Form_Click ;) italic X(3) As Integer X<1) - 1 X(2) - 2 X(3) = Sum(X(l), X(1) - X(2))’список значений

‘ включает 'элементы мас,»'ЛЕ^а и выражение Print X(1) , X(2), X(3) End Sub ***Пример 2. .***

Function *Sum (A* As Integer/ B As Integer) ftg Integer

'список -аргументов состоит из переменных *C = A + B Sum = C* End Function Sub Forrn_Click () Dim *X, Z* As Integer X = 5 Z = Sum *(X, 4)'* список значений включает

' переменную и константу Print *X, Z* End Sub

- Пример 3.

```
    Function Sum (A ( ) As Integer) As Integer 'список аргументов включает массив
A(3) = A(1) + A(2)
Sum—A(3)
    End Function
Sub Form_Click ()
Static X(3) As Integer
X(1) = 1
X(2) - 2
    X.(3 ) = Sum (X() ,)'список значений 'включает массив
Print X(1), X(2) , X(3)
    End Sub
```

Для рассмотренных примеров заголовков функций с параметрами ParamArray и Optional обращения к функциям могут иметь следующий вид:

ReturnValue = CalcSum(4, 3 ,2 ,1) первый аргумент функции FirstArg принимает значение 4, остальные аргументы являются элементами массива и принимают значения OtherArgs(I) = 3, OtherArgs(2) = 2, последнее значение определяет размерность массива (1).

RetVal = MyFunc(“Hollo”, 2, “World”) - В этом обращении заданы все три значения аргумента.

. RetVal = MyFunc (“Test”, , 5) — в этом обращении не задано значение второго аргумента.

RetVal = MyFunc (MyАгд! :-- 7) - в этом обращении зада-но значение второго аргумента с использованием его имени.

Пример программы.

Поместим рассмотренную функцию для определения среднего значения элементов одномерного массива в раздел general (Ьормы и обращение к ней в процедуру обработки события щелчка мыши по форме (рис. 3.19). После запуска программы и щелчка мыши по форме распечатывается среднее значение суммы элементов массива X (рис. 3.20).

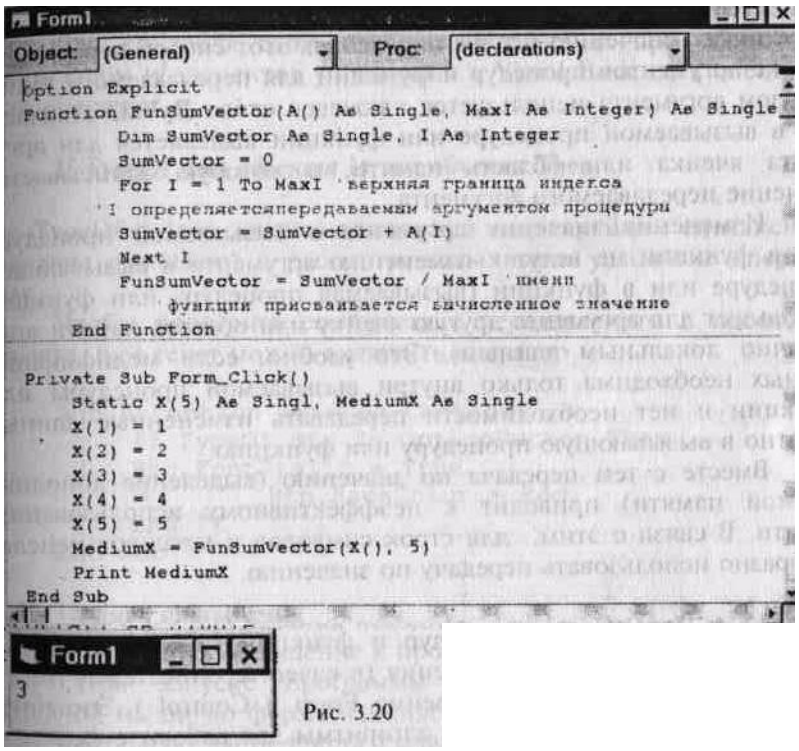


Рис. 3.20

3.6.3. Передача аргументов по ссылке и по значению

Передача значений аргументов при вызове процедуры или функции по ссылке означает, что передается адрес памяти, по которому хранится значение аргумента (т.е. вызываемая процедура или функция для работы с переданным по ссылке аргументом использует одну и ту же ячейку или

область памяти, что и вызывающая процедура пли функция). При этом не выделяется дополнительная намять для работы с переданным аргументом в вызываемой программе. Изменение значения аргумента в вызываемой процедуре или в функции означает изменение значения и в вызывающей процедуре или в функции (используется одна и та же ячейка или область памяти).

Передача цо ссылке определена по умолчанию. Передача значении аргументов при вызове процедуры или функции по значению (чтобы определить этот способ передачи, в списке аргументов процедур и функции для передаваемого таким образом аргумента используется ключевое слово **ByVal)** означает. что в вызываемом процедуре или функции выделяется для аргумента ячейка или область памяти, в которую записывается значение передаваемою аргумента.

Изменения значения аргумента в вызываемой процедуре или в функции не ведут к изменению аргумента в вызывающей процедуре или в функции (вызываемая процедура или функция использует для аргумента другую ячейку или область памяти аналогично локальным данным). Это удобно, если модификация данных необходима только внутри вызываемой процедуры или функции и нет необходимости передавать измененные данные обратно в вызывающую процедуру или функцию.

Вместе с тем передача по значению (выделение дополнительной памяти) приводит к неэффективному использованию памяти. В связи с этим, для строк символов и массивов нецелесообразно использовать передачу по значению.

3.6.4. Использование в качестве аргументов процедур и функции форм и элементов управления

Список аргументов процедур и функций может включать имена форм и элементов управления (в качестве описателей типа для них применяются соответственно Form и Control). Это позволяет создавать универсальные алгоритмы для работы с формами и управляющими элементами. Например, для привлечения внимания пользователя к одной из управляющих кнопок формы можно создать одну универсальную процедуру, изменяющую шрифт и цвет фона,

```
Sub Att^ntio!-. ‘:3^;’. As Control;
Btn. i-‘onrl t.^ e.: ^ True
Bin.BackColor = Red
'костанта Red определена в файле CONSTANT.TXT End Sub
```

аргументом которой является имя кнопки. Обращение к данной процедуре с указанием в качестве значения имени какой-либо конкретной кнопки приведет к изменению ее шрифта и цвета (указывается имя кнопки Btn_Right)

Attention Btn_Riqht

Для контроля соответствия передаваемых при обращении типов форм и

элементов управления в Visual Basic есть специальный оператор

If TypeOf *имя объекта Is* тип объекта Then

где TypeOf и Is — ключевые слова, а последующий синтаксис и действие совпадают с рассмотренным выше условным оператором.

Чтобы проверить соответствие передаваемого в процедуру объекта, в ее текст можно включить оператор

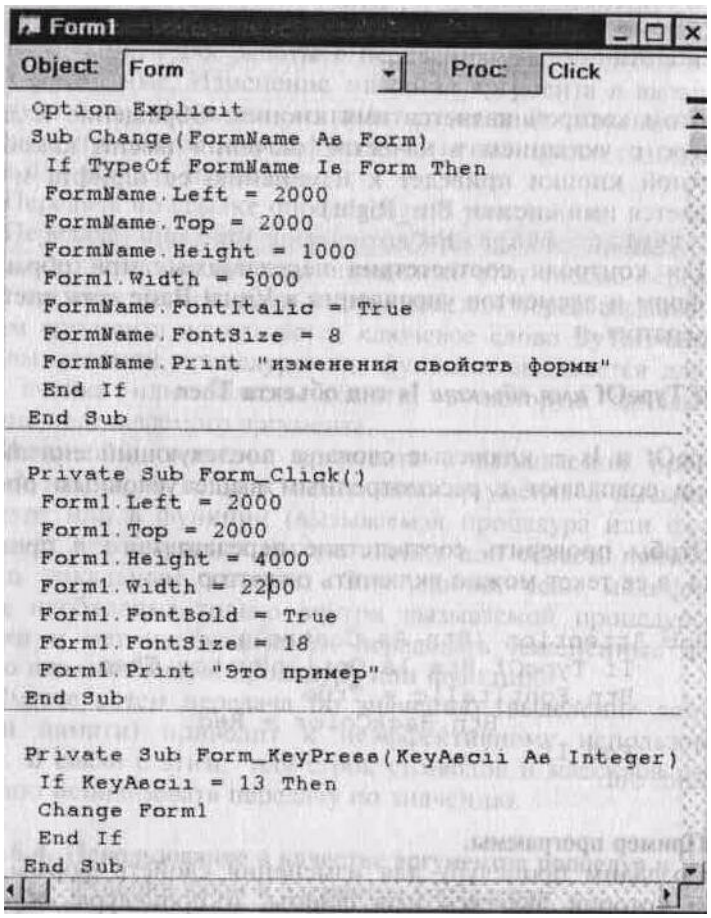
```
Sub Attention (Btn As Control)
If TypeOf Btn Is OptionButton Then
Btn.FontItaiic = True
Btn.BackColor = Red
End If End Sub
```

Пример программы.

Создадим процедуру для изменения свойств формы, аргументом которой является имя формы, и процедуры обработки событий Click для задания исходных свойств формы и KeyPress, в которой имеется обращение к процедуре Change (рис. 3.21)

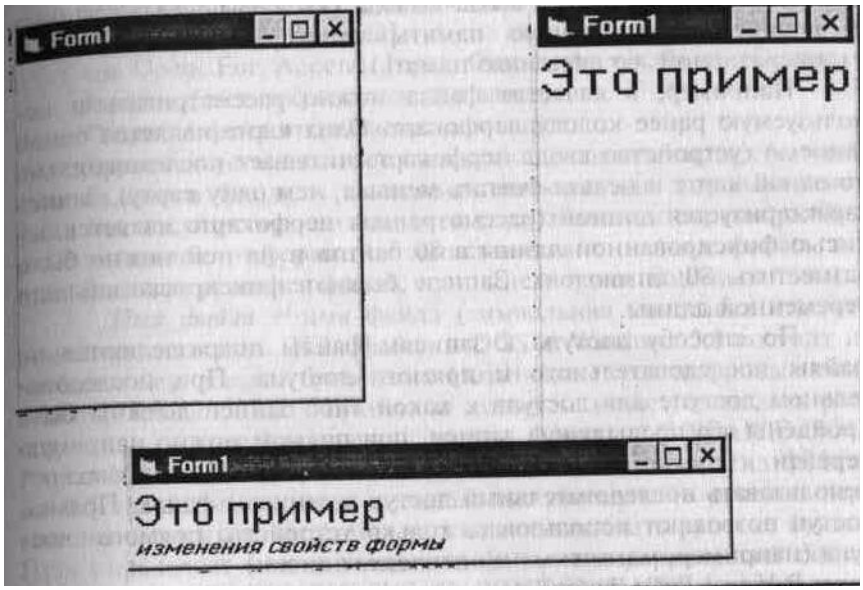
При запуске программы появляется исходная форма. Щелчок мыши по форме изменяет ее размер, положение и выводит начальную часть текста (задается в процедуре обработки события Click).

. Нажатие клавиши Enter приводит к изменению размера, положения формы и вывод заключительной части текста (обращение в процедуре обработки события KeyPress к процедуре Change) (рис. 3.22).



3.7 ФАЙЛОВЫЙ ВВОД-ВЫВОД

Архитектура современных компьютеров предусматривает наличие оперативной и внешней памяти. В оперативной памяти находятся выполняемая в данный момент программа и обрабатываемые ею данные. После выполнения программы ее данные в оперативной памяти не сохраняются, так как используемые ячейки памяти выделяются для данных другой программы. Технически и программно обусловленные ограниченность объема



оперативной памяти и ее относительно высокая стоимость не позволяют применять ее для постоянного хранения больших объемов информации. Для этого используется внешняя память (магнитные ленты, жесткие и гибкие магнитные диски, лазерные диски), позволяющие хранить сотни мегабайт (1 Мбайт = 1048576 байт) и гигабайт (1 Гбаит = 1.073.741.824 байт) информации и стоимость которых относительно невелика.

Хранение больших объемов информации на внешних носителях с учетом того, что время доступа к данным на внешних носителях на один, два порядка ниже времени доступа к данным в оперативной памяти, требует их хорошо продуманной организации. А возможность альтернативного хранения одних и тех же данных на различных внешних носителях требует использования единых принципов организации данных.

Таким универсальным понятием является файл (набор данных), который используется для работы с внешними данными на любом носителе. Под файлом понимается совокупность данных. Файл состоит из записей. Запись состоит из логически связанных данных, которые передаются между оперативной и внешней памятью за одну операцию ввода-вывода (ввод — передача данных из внешней в оперативную память, вывод - передача данных из оперативной во внешнюю память).

Например, в качестве файла можно рассматривать и используемую ранее колоду перфокарт. Одна карта является одной записью (устройство ввода перфокарт считывает последовательно по одной карте и нельзя считать меньше, чем одну карту). Запись характеризуется длиной (рассмотренная перфокарта является записью фиксированной длины в *SO* байтов и на ней можно было разместить 80 символов). Записи бывают фиксированной или переменной длины.

По способу доступа к записям файлы подразделяются на файлы последовательного и прямого доступа. При последовательном доступе для доступа к какой-либо записи должны быть пройдены все предыдущие

записи, при прямом можно напрямую перейти к нужной записи. Все внешние устройства позволяют использовать последовательный доступ к записям файла. Прямой доступ позволяют использовать только устройства прямого доступа (например, магнитные и оптические диски).

В Visual Basic файлы подразделяются на:

- последовательные (sequential), записи которых могут быть переменной длины;
- прямого доступа (random access), записи которых могут быть только одинаковой длины;
- двоичные (Ьупагу) файлы с произвольной группировкой байтов, что позволяет организовать гибкий доступ к данным.

3.7.1. Последовательные файлы

Каждая запись последовательного файла представляет собой строку символов, в конце которой содержится символ <CR>, обозначающий переход к началу строки (carriage return, ASCII-код равен 13), и <LF>, обозначающий переход на другую строку (line feed, ASCII-код равен 10). Последовательное действие этих двух символов обеспечивает переход к новой записи. После последней записи записывается признак конца файла (end of file), который может проверяться встроенной функцией EOF (возвращает значение True. если обнаружен конец файла, и False, в противном случае).

Для получения доступа к файлу для операции ввода-вывода используется оператор открытия файла, синтаксис которого следующий

Open *имя_файла* For {Append Input | Output} As *#номер_фаила* fLen = *размер буфера памяти}*

где Open, For. Append. Input. Output, As — ключевые слова.
Input — файл открывается для ввода.
Output — файл открывается для вывода.

Append — устанавливает считывающе-записывающее устройство на конец файла и выводимая информация записывается в файл после существующих записей (при значении параметра Output выводимая информация записывается с начала файла, т.е. происходит перезапись файла, если в нем существовали записи).

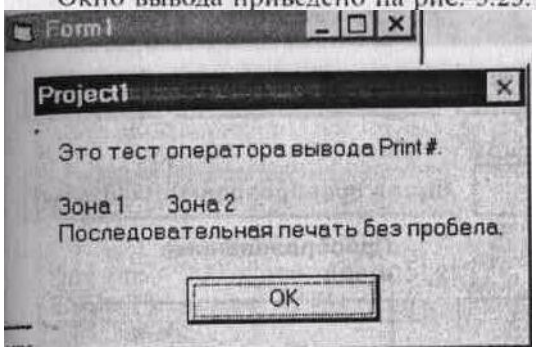
Имя_файла — имя файла (символьная константа или переменная) или путь. Имя файла в DOS и Windows 3.1 состоит из не более чем восьми буквенных (латинских) или цифровых символов (первый символ буква), за которым может следовать расширение, определяющее тип файла (VBPART3.DOC, Project. MAC). В Windows 95 допускаются длинные имена файлов до 255 символов, включая специальные символы (в русской версии для записи имени файла можно использовать русские буквы) Путь определяет местонахождение файла в иерархической системе каталогов (C:\CONF1G.SYS, C: \WINDOWS\VB).
Номер файла — целочисленное выражение, значение которого должно лежать в диапазоне от 1 до 255 (присваиваемый файлу номер).
Len — определяет размер буфера операций ввода-вывода (по умолчанию 512 Кб).
Примеры:
Open «C:\CONFIC.SYS" For Input As tt5 **'открывается**
файл с именем CONFIG.SYS в директории C: для ввода
и ему присваивается номер 5

Doc$ = "a:\Utilits\NC.DOC"
присвоение значения константе
Open *Doc$* For Input A.s ft 5
'открытие файла **NC.DOC** на диске A: в директории Utilits для ввода, файлу присваивается номер 2
Open *"Resulc. t:-:t «* For Output As tt7
'открытие файла для вывода.
Если открывается для вывода несуществующий файл. то он создается при значениях параметров Append и Output. Если для ввода открывается несуществующий файл, то Visual Basic сообщает об ошибке. Если файл скрывается для вывода Visual Basic всегда создает новый файл, перезаписывая в него любой имеющийся на диске файл с тем же именем.
Существование файла перед открытием можно проверить с помощью встроенной функции Dir$ (возвращает строку с копией имени файла, если указанный файл существует, либо пустую строку в противном случае).
Пример.
If Dir$*("FilePrim.Txt")* <> *«»* Then
.Open *"FilePrim.Txt"* For Input As #12
End If
После завершения операций ввода — вывода файл должен быть закрыт. Для этого используется оператор
Close *#номер_фачла* где **Close** — ключевое слово;
номер_ файла соответствует номеру в операторе Open.
Для ввода информации из последовательного файла используется оператор
Line Input *#номер_файла, имя_переменной* где **Line Input** — ключевое слово;
номер_файла — номер файла, совпадающий с номером в операторе Open (вводится информация из открытого файла);
***имя_переменной* —** имя переменной, которая принимает значение записи файла, типа String или Variant.
При выполнении оператора считывается одна запись файла и помещается в ячейку оперативной памяти, адрес которой соответствует имени переменной в операторе ввода (переменная получает значение, совпадающее с введенной записью файла). Последовательное выполнение операторов ввода обеспечивает последовательное считывание записей файла.
Встроенная функция **EOF** (аббревиатура английских слов End Of File - конец файла) позволяет проверять при чтении файла: достигнут конец файла или нет. Значением аргумента функции EOF является номер считываемого файла. Функция возвращает значение True, если достигнут конец файла, и False - в противном случае.
Пример.
Dim *FileYura, Sr.rokaVvoda* As String
'объявление символьных переменных
Open *FileYura* For Input As *#5*
'открытие файла *FileYura* для ввода Do While *Not* EOF(5)'цикл последовательного чтения
'записей файла пока не достигнут конец Line Input *#5, StrokaVvoda'*ввод считанной записи
'в переменную StrokaVvoda LineRead *StrokaVvoda* 'обращение к
'процедуре обработки строки LineRead 'аргументом которой является переменная *StrokaVvoda* Loop
Close *#5 '*закрытие файла
Для вывода информации в последовательный файл используется оператор
{Print* Write} *»номер_файла,[[{8рс(ч)* |Tab(n)}]
[*выражение}[{,\;}}}* где **Print, Write** — ключевые слова.
Print# — обеспечивает вывод в последовательный файл в формате дисплея (т.е. аналогично выводу на печать, например, на форме).
выражение - выражение, значение которого записывается в файл.
Если выражения разделяются «;», то в файл они записываются без пробелов слитно.
Если выражения разделяются «,», то в файл они записываются в фиксированные зоны длиной 14 символов (зонный формат).
Если в конце выражения не стоит «;» или «,», то выведенная в файл строка дополняется символами **<CR>,** обозначающими переход к началу строки (carriage return, ASCII-код равен 13), и **<LF>,** обозначающими переход на другую строку (line feed, ASCII-код равен 10). Т.о. каждому значению соответствует одна запись или одна строка при выводе в

формате дисплея.
Spc(n) и TaЬ(n) определяют соответственно **вставку n пробелов** между выводимыми выражениями и **табуляцию на n колонок** перед списком выражений.
Для удаления с дискового пространства неиспользуемого файла используется оператор
Kill *имя_файла* где **Kill** — ключевое слово.
В заключение следует отметить, что данные любого типа (Boolean, Data, Integer, Single, Double, Currency) записываются в файл в символьной форме. При выводе данные преобразуются к символьной форме, при вводе происходит их преобразование к первоначальному типу, па что затрачиваются ресурсы компьютера. Кроме того. представление данных в символьной форме неэффективно. Например, число 421596 типа Single -занимает в памяти компьютера 4 банта, но при записи в последовательны» файл - 7 байт: 1 байт па каждый символ.
Для того чтобы считать требуемую запись в последовательном файле необходимо последовательно пройти все предыдущие записи; чтобы изменить одну запись в файле, необходимо переписать весь файл заново. От этих недостатков свободны файлы прямого доступа.
Пример программы.
В первой части процедуры обработки события щелчка мыши по форме в файл TE5TFILE выводятся строки символов с использованием оператора Print# (вывод в формате дисплея). Во второй части процедуры записи этого файла читаются и выводятся в окно вывода в том виде, как они записаны в файле.
Текст процедуры:

```
Private Sub Form_Click()
Dim FileData, Msg, Nl As String
Nl = Chr(10) 'перевод строки
Open "TestFile" For Output As #1
                'открыть файл для вывода
Print #1, "Это тест оператора вывода Print #."
Print #1, 'Печать пустой строки.
Print #1, "Зона 1", "Зона 2"
                'Печать в две фиксированные зоны.
Print #1, "Последовательная печать без пробела"; "."
Close #1 'Закрыть файл
Open "TestFile" For Input As #1
                'открыть файл для ввода
Do While Not EOF(1)
Line Input #1, FileData 'Ввод записи
Msg = Msg & FileData & Nl
                'Формирование строки сообщения
Loop
Close #1 'Закрыть файл
  MsgBox Msg 'Выдача сообщения
  Kill "TestFile" 'Удаление файла с диска
End Sub
```

Окно вывода приведено на рис. 3.23.



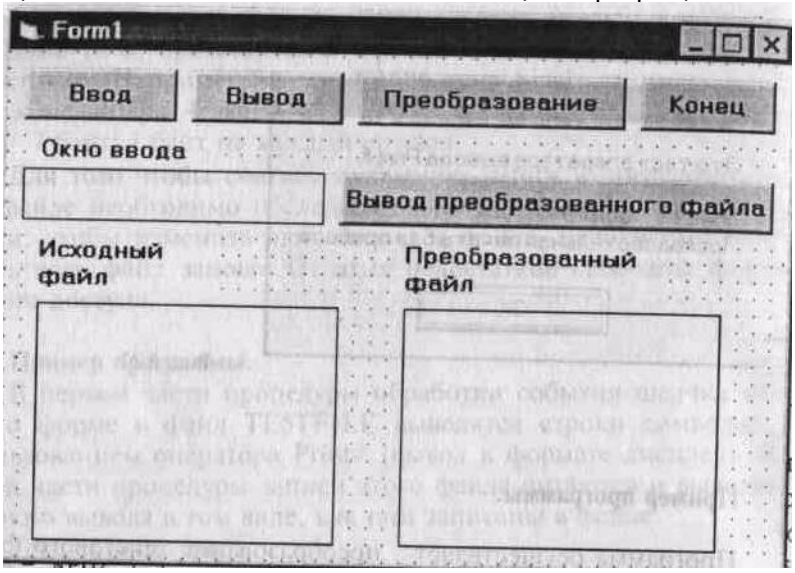
Пример программы.

Программа осуществляет преобразование текстового файла последовательною доступа распаковкой его строк таким образом, чтобы каждая цепочка символов «*п», где п — однозначное целое число (число пробелов), заменялась указанным числом пробелов.

Ввод исходных данных и управление работой программы осуществляется на форме на рис. 3.24.

На форме располагаются:

- командная кнопка «Ввод» (имя — Command 1), щелчок мыши по которой активизирует окно ввода (имя — Textl). Вводимая информация записывается в файл последовательного доступа:
- командная кнопка «Вывод» (имя — Commands), щелчок мыши по которой обеспечивает вывод записей файла в многострочном текстовом окне (имя Text2):
- командная кнопка «Преобразование» (имя — Command2), щелчок мыши по которой обеспечивает чтение записей файла, их преобразование в соответствии с заданием и запись новых записей в другой файл:
- командная кнопка «Вывод» (имя — Command4). щелчок мыши по которой обеспечивает вывод записей повою файла в многострочном текстовом окне (имя Textl);
- щелчок мыши по кнопке «Конец» прекращает выполнение программы.



Тексты процедур обработки событий:

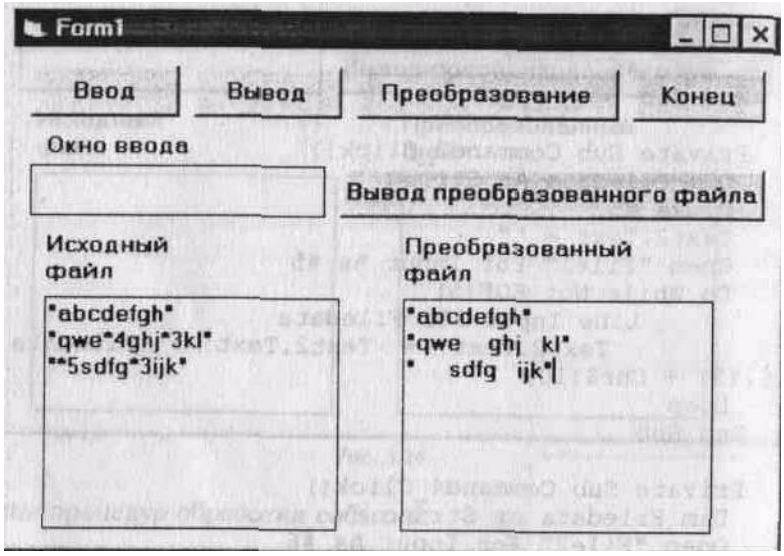
Private Sub Coriurtandl_Click ()
Text3.SetFocus Open “Filel” For Output As #5
Text3.Text = “” End Sub
Private Sub Command2_Click() Dim Filedata, Filedatal As String Dim NPoz, NProbel As Integer
Close #5
Open “Filel” For Input As tt5 Open “File2” For Out-put As #6 Do While Not EOF(5) Line Input #5, Filedata Filedatal = “” 1=1 Do While InStrfl, Filedata, “*”) > 0

NPoz = InStr(I, Filedata, “*”) **Filedatal = Filedatal + Mid$(Filedata, 1, NPoz - I)** NProbel = Val (Mid$ (Filedata, NPoz +1, 1);

Fiiedatal = Filedatal + String$(NProbel, “ I .= NPoz + 2 Loop

```
Filedatal = Filedatal +          Mid$(Filedata, I) Write #6, Filedatal Loop
Close #5 Close ft6 End Sub
Private Sub Command3_Click() Dim Filedata As String Close #5
Text2.Text = ""
Open "Filel" For Input As ft5 Do While Not EOF(5)
Line Input #5, Filedata
Text2.Text  =  Text2.Text  +  Filedata :hr$ (13) + Chr$(10)
Loop End Sub
'Private Sub Command4 Click() Dim Filedata As String Open "File2" For Input As #6 Do While Not EOF(6) Line Input tt6,
Filedata
Textl.Text = Textl.Text + Filedata + Chr$(13) ^ Chr$(10)
Loop Close #6 End Sub
Private Sub Command5_Click()
Kill "Filel"
Kill "File2"
End End Sub
Private Sub Text3_KeyPress(keyAscii As Integer) Dim Filedata  As String If keyAscii = 13 Then Filedata = Text3.Text
```

Write tt5, Filedata Text3.Text •= «»



Описание используемых стандартных (функции для обработки строк

Stiing$(niimbei, chaiactci) возвращает проку одинаковых символов (chaiaciei) заданной длины (niimbel)

Mid$(stung, start|. length|) возвращает строку с заданным количеством символов (length) из CIROKH (stung), начиная с заданной позиции (start) EL in start больше количесгва символов в строке - возвращаечся прока нулевой длины Если отсутствуег параметр length, то воз праща клея симво1ы от заданной позиции до конца строки Если slung равно Null - возвращается Null

lnSti(|stait. |stiingl. stiing2) возвращает первую позицию, с которой одна строка (stiing2) вхолиг в другую (stimgl), как подстрока Start — арифмешческое выражение, значение которого задае! начальную позицию поиска в slungi (если параметр stait не задан, ю поиск иде1 с первой позиции)

Возвращаемое значение' если stiingi имеет нулевую длину — О если string! равно Null - Null если stiing2 имеет нулевую длину — start если stiing2

равно Null — Null если string2 не найдено — O если start > длины stimgl — O

3.7.2. Файлы прямого доступа

Для получения доступа к файлу для операции ввода-вывода используется оператор откргыгия файла, синтаксис которого следующий

Open *имя_фаша* [For Random] [Access {Read] Write I Read

Write}]

[{Shared | Lock Read | Lock Read Write}] As [#] *номер файла* [Len =

длина записи}

где Foi Random, Access. Read, Wiite, Read Wiite, Shaied, Lock Read, Lock Wiite, Lock Read Wiite, Len - ключевые слова.

For Random ***определяет прямои доступ и принимается по умолчанию***

Access определяет допустимые операции с файлом **(Read** — чтение фаипа (ввод). **Write** - вывод в файл. **Read Write** — допускается ввод и вывод)

Shared, Lock Read, Lock Write, Lock Read Write определяют допустимые операции над фаилом для друшх пользователей сеги (только для сетевых приложении)

имя_фа1иа — имя файла (символьная константа или переменная) или путь

Номер файла - целочисленное выражение, значение которого должно лежать между 1 и 255 Другие операторы ввода-вывода используют номер открытого файла

Len определяет длину записи в байтах (по умолчанию длина записи устанавливается в 128 бант)

Длина записи определяется информацией, хранимой в файле, и задается целочисленным выражением и должна быть меньше 32767 байт (см ниже) Все записи одного файла прямого доступа имеют одинаковую длину (записи одного файла последовательного доступа Moiyr иметь разные длины)

Несушес1вующии файл создается при выполнении оператора Open **Примеры:**

Open *“TESTFILE”* For Random *As#l* Len=Len (*Cl ien* CWame) 'открывается файл “TESTFILE” прямого 'доступа с номером 1, длина записи

'определяется с помощью встроенной 'функции Len (определение длины строки)

Open “*MYFILE.TXT”* For Random As #*5* Len = 256

Для файлов прямого доступа можно открыть файл с другим номером, не закрывая файл Файл прямого доступа закрывается оператором

Close *#номер файла* где ***номер фалла*** - указанный при открытии номер

Для ввода и вывода в файлах прямого доступа используются соответственно операторы

Get #номер файла, [номер записи], имя_переменной

Put *#номер файла, [номер записи], имя_переменной* где Get, Put - ключевые слова (Get — ввод. Put — вывод).

номер записи - арифметическое выражение, значение которого должно лежать в диапазоне от 1 до 2247483647, и которое определяет номер читаемой записи при вводе и номер записи, в которую выводится переменная

Если номер записи отсутствует, то ввод — вывод начинается со следующей от текущеи записи (используемый номер в последнем из предыдущих операторов Get и Put) Обратите внимание, что запятые должны присутствовать при отсутствии номера записи (Get #4„FileBufTer)

Имя^переменной — имя любой переменной, **кроме имени массива** (отдельный элемент массива может быть) и **имени объекта,** значение которой записывается в файл при выводе и значение которой считывается из файла при вводе

Примеры

Put *#3, 4, Massiv(5)'*выводится значение 5-го 'элемента массива Massiv в 4-ю запись J-го файла.

Get *#fileNum, I, RecVar* 'ич 1-й -записи файла с 'номером liieNun' считывается 'значение переменной RecVar.

Пример программы.

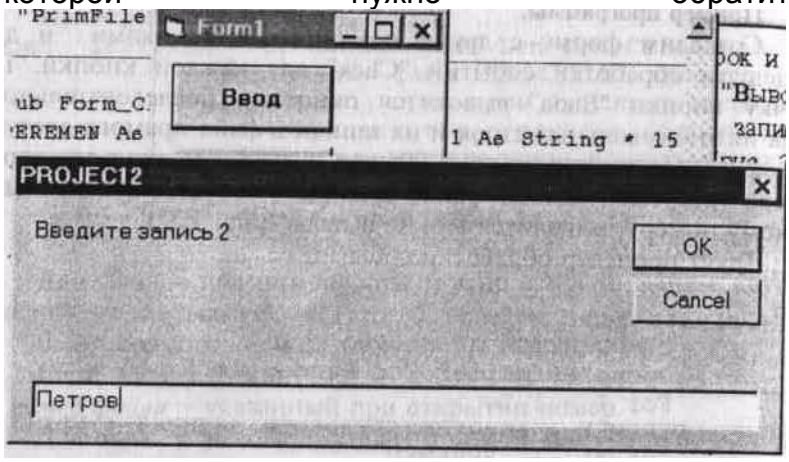
Создадим форму с двумя командными кнопками и две процедуры обработки событии “Click” для каждой кнопки По щелчку кнопки «Ввод» выводится окно для последовательного ввода пяти символьных строк и их записи в фаил прямого доступа (рис 3 26) По щелчку кнопки «Вывод» выводится окно для ввода номера требуемои записи фаила и запись, соответствующая введенному номеру, выводится в окне вывода (рис 3 27)

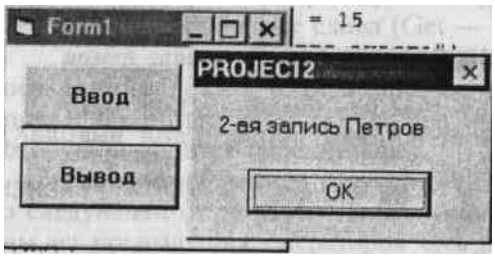
Текст процедур обработки событии'

```
Private Sub Commandl_C-Lick ()
i^im PEREMLN Ач String * 15
Open “Prirrb^le” For Random As 5 Len - 15
For I - 1 ^r b PEREMEN = InputBox(«Введите запись» + Str(I))
Put #1-, T, PEPEMEN
Next r
Close ^ End Sub
Private Sub Commcind2 Click ()
Dim PEREMEN As String * 15
Open “Primflie” For Random As 5 Len = 15 К = InputBox(«Введите номер записи для вывода»)
Get #5, К, PEREMEN
MsgBox Str(K) +' «-ая запись» + « « + PEREMEN
Close #5
Kill “PrimFile” End Sub
```

Еще одним преимуществом файлов прямого доступа является возможность использования индексов записеи Если для реорганизации фаила последовательного доступа (добавление, удаление, перестановка записей) необходимо сначала его прочитать, затем реорганизовать данные, а потом переписать фаил на диск в соответствии с новой реорганизациеи, то используя индекс, можно реорганизовывать индекс без реорганизации самого фаила Это значительно увеличивает скорость обработки больших объемов информации

Чтобы создать индекс, вначале создается массив целых чисел, в котором каждый элемент содержит один (свой) номер записи фаила прямого доступа Этот массив используется для определения номера записи, к которой нужно обратиться Такая





организация хранения записи файла называется индексированием записи. Записи одного файла могут иметь несколько разных индексов для организации различных алгоритмов работы с одним и тем же файлом.

Для массива индексов Index () оператор ввода для связанного с этим индексом файла имеет вид

Get *ftFiieNiim. Imlex(l), RecVar*

Первоначально каждый элемент массива Index () содержит свой собственный номер:

Index (1) = 1
Index (2) = 2 *«.',* Index (3) = 3

Результат использования такого массива для доступа к записям файла идентичен прямому доступу к этому файлу. Для того чтобы переставить местами, например, вторую и третью запись, вместо реальной перестановки записей достаточно изменить значения индексов следующим

образом:

Index (I) = 1

Index (2) = 3

Index (3) = 2

Пример программы.

Добавим к предыдущей программе процедуру обработки события щелчка мыши по форме.

Private Sub Form_C]-ick ( )

Dim PEREMEN As String * 15, PEREMEN1 As String * 15 Static thelndex(I To 5) As Integer Open “PrimFile” For Random As 5 Len = 15 For I = 1 To 5 thelndex(I) = I Next I
11 = Val(InputBox(«Введите номер 1-й записи для перестановки»))
Get #5, thelndex(II), PEREMEN
12 = InputBox(«Введите номер 2-й записи для перестановки»)
Get #5, thelndex(12), PEREMEN1 Print Str(thelndex(II)) + «-ая запись « + PEREMEN Print Str(thelndex(12)) + «-ая запись « + PEREMEN1 thelndex(II) = 12 thelndex(12) = II Get #5, thelndex(II), PEREMEN Get #5, thelndex(12), PEREMEN1 Print Str(II) + **«-ая** запись « + PEREMEN Print Str(I2) + «-ая запись « + PEREMEN1 Close #5

Kill “PrimFile” End Sub

В процедуре вводится массив thelndex для задания индексов записей используемого файла. Первоначально значениям массива последовательно присваиваются значения от 1 до 5 (индекс записи совпадает с ее реальным номером’в файле). Далее вводятся значения двух номеров записей для их перестановки, записи Private Sub Commandl_Click()
Nzapisi = 1
Nzapisil = 1
Open “TestFile” For Random Access Read Write As #10 Len = 70
Text 1.Enabled = True
Text2.Enabled = True
Text3.Enabled = True
Text4.Enabled = True
Text7.Enabled—True
Textl.SetFocus
Commandl.Enabled = False End Sub
Private Sub Command2_Click() Dim Peremen As Primer Textl.Text = “” Text2.Text = “” Text3.Text = “” Text4.Text = “” If Nzapisil < Nzapisi Then Get #10, Nzapisil, Peremen Textl.Text = Peremen.Familia Text2.Text = Peremen.Dolgnost Text3.Text = Peremen.Podrazdeienie Text4.Text = Peremen.Zarplata End If
Nzapisil = Nzapisil + 1 End Sub
Private Sub Command3_Click()
Textl.Text = “”
Text2.Text = “”
Text3.Text = “”
Text4.Text == “”
Text5.Enabled = True
Text6.Enabled = True
Text5.Text = “”
Text6.Text = “”
Text5.SetFocus End Sub
Private Sub Command4_Clicki’ Dim Peremen As Primer
Dim Zaprosi As String * 20, Zapros2 As String * 20, Sum As Single
Dim Pri As Integer, Pr2 As Integer, Kol As integer, Rez As Single
Pri = -1: Pr2 = -1: Nzapisil = 1: Kol = 0: Sum = 0
Zaprosi = Text5.Text: Zapros2 = Text6.Text Do While Nzapisil < Nzapisi Get #10, Nzapisil, Peremen If Peremen.Podrazdeienie - Zaprosi Then Pri = 1 End If

```
If Peremen.Doignost = Zapros2              Then Pr2 - 1 End If
If Peremen.Podrazdeienie = Zdprosi And Peremen.Doignost = Zapros2 Then
Kol -= Kol + 1: Sum =              Sum + Val(Peremen.Zarplata): Rez = Sum / Kol End If
Nzapisil = Nzapisil + 1 Loop If Pri < 0 Then
• • MsgBox «Такого подразделения нет» Text5.Text = "": GoTo Finish6 End If If Pr2 < 0 Then MsgBox «Такой должности нет»
Text6.Text = "": GoTo Finish6 End If
Text7.Text = Str$(Rez) Command5.Enabled =- True Finish6:
End Sub
Private Sub Cun-iniand5_Ciick ()
Cl?Qse„#10
Kill "TeS-tFile"
End End Sub
```

.Private Sub Form_Load() ' Commandl.Enabled = True Command2.Enabled - False Command3.Enabled = False Command4.Enabled ª False Хотя можно указать запись длиной 1 байт, компьютер на самом деле прочитает в буфер памяти один или более секторов. Когда программа читает данные из дискового файла, она читает их из буфера. Когда прочитывается весь буфер, компьютер читает в этот буфер следующие несколько секторов из файла. Аналогично данные записываются в дисковый файл: сначала они записываются в буфер памяти, а когда он заполняется, то записываются на диск.

Предположим, что размер дискового сектора 512 байтов, а длина записи файла 260 байтов. **При** такой длине записи большинство записей файла будут расположены в двух секторах (например, запись 2 использует байты с 261 по 512 первого сектора и с 1 по 8 байт второго сектора) и для чтения с диска такой записи потребуется прочесть два сектора. Использование вместо 260-байтных записей 256-байтных обеспечивает упаковку в один дисковый сектор двух полных записей и позволяет для ввода одной записи читать только один сектор.

Таким образом, выбор длины записи определяется типом и длиной передаваемых данных и размерами дисковых секторов. Возможно, что реальная длина передаваемых данных будет меньше длины записи, указанной параметром Len. Если длина читаемых данных меньше длины, указанной параметром Len, то место до границы следующей записи заполняется содержанием буферного файла. Это нежелательно и следует стремиться к согласованию реальной длины передаваемых данных с задаваемой длиной записи с учетом указанных выше замечаний.

Пример программы.

Записи файла прямого доступа содержат информацию о сотрудниках подразделения предприятия (структура записей файла:

фамилия, должность, подразделение, заработная плата).

Программа обеспечивает ввод исходной информации в файл и получение информации по запросу: средняя заработная плата для заданных должности и подразделения.

Форма имеет вид, показанный на рис. 3.30.

Текстовые окна «Фамилия», «Должность», «Подразделение»,

«Зарплата» (имена соответственно Textl, Text2, Text3, Text4) используются для ввода исходной информации и просмотра записей файла. Текстовые окна «Должность», «Подразделение» (имена соответственно Text5, Text6) для ввода запроса. Текстовое окно «Результат» (имя Text?) используется для

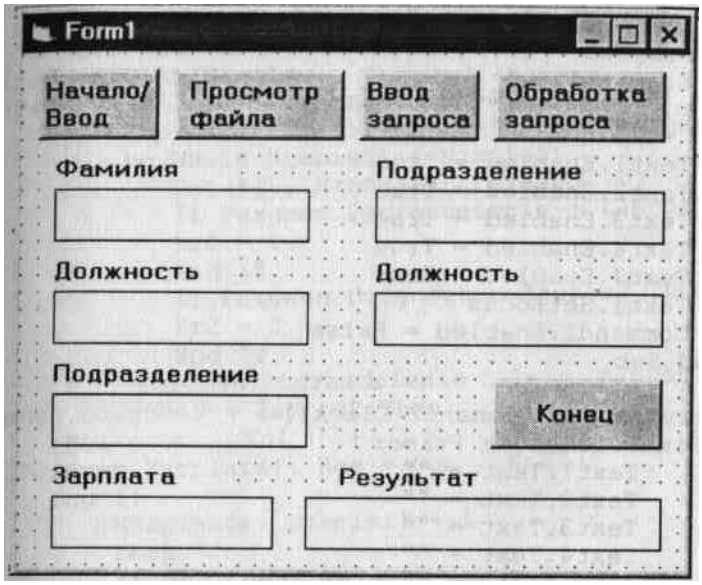


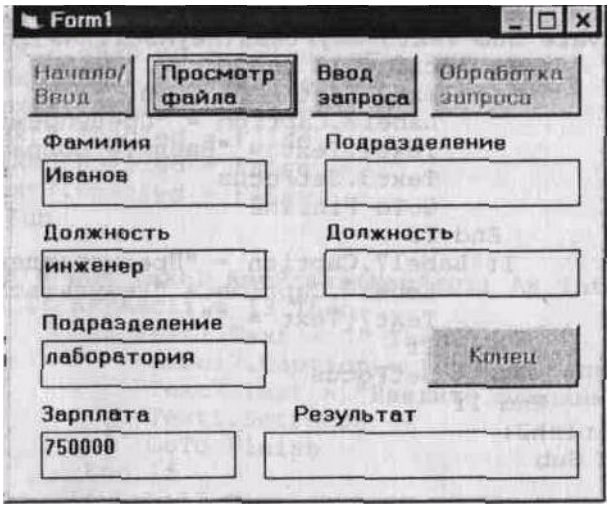
Рис. 3.30

вывода результата и предупреждении о не заполнении необходимых текстовых окон. Коммандные кнопки «Начало/Ввод», «Просмотр файла», «Ввод запроса», «Обработка запроса». «Конец» (имена соответственно Command I, Command2, Command3, Command4, Command5) обеспечивают переход к функциональным алгоритмам. Оператор объявления пользовательского типа данных помещаются в модуль module I.bas. Работа программы иллюстрируется на рис. 3.31—3.33.

Текст процедуры обработки событий:

```
Type Primer
Familia As String * 20
Dolgnost As String * 20
Podrazdelenie As String * 20
Zarpiata As String * 10 End Type
Dim Nzapisi, Nzapisil As Integer Dim Peremen As Primer  Private Sub Coirimandl_Cllck ()
Nzapisi = 1
Nzapibil  1
Open “TestFile” For Random Access      Read Write            As #10 Len = 70
Textl.Enabled - True
Text2.Enabled -= True
Text3.Enabled—True
Text 4.Enablea = True
Text?.Enabled - True
Textl.SetFocus
Commandl.Enabled -= False End Sub
Private Sub Command2_Click ( ) Dim Peremen As Primer Textl.Text = “” Text2.Text - “” Text3.Text - “” Text4.Text = “” If
Nzapisil <” Nzapisi Then Get #10, Nzapisil, Peremen Textl.Text = Peremen.Familia Text2.Text =- Peremen.Dolgnost
Text3.Text = Peremen.Podrazdeleni Text4.Text = Peremen.Zarplata End If
Nzapisil = Nzapisil + 1 End Sub
Private Sub Command3__Click()
Textl.Text = “”
```

```
Text2.Text = “”
Text3.Text =• “”
Text4.Text = “”
Text5-Enabled = True
Text 6.Enabled = True
Text5.Text = “”
TextG.Text = “”
Text5.SetFocus End Sub
Private Sub Conmand4_Clic[1] Dim Peremen As Primer
Dim Zaprosi As baring - ^.u, ^aproь^ AS String * 20, Sum As Sinqie
Dim Pri As Integer, Pr2 As Integer, Koi As Integer, Rez As Single
Pri - -1: Pr2 - -1: Nzapisil = 1: Kol = 0: Sum - O
Zciprosi - Text’).Text: Zapros2 = Text6.Text Do While N^dpibil “•’zapisi ‘^rit #1’, Nzdpisil, Peremen If Peremen. Pcmra^aeienie - Z^prosi Th>.’n t’rl -= 1 End If
If Peremen.Dolgnost = Zapros2 Then Pr2 - 1 End If
If Peremen. Podrazdelenie - Zciprosi And Peremen.Dotqnosr Zdpros2 Then
Kol K’)I + 1 : Sun SUIT + Val ( E’fc r emen . Zdrplata ) : Rez - iiiim / Koi End If
N?apisil - ^Zcioisil + 1 Loop II: Pri < (• T^hen
MsqBox “Tai- ого псдразд&ления нет» Tex^J.rext - “”: GoTo l-mishb End If If Pr2 < 0 Then
    MsgBox «Такой допкно ••ти нет»
r^xtt.rex»- - “”: u^Tu Finishfa End I*
Text /.It >;r Str$ ;Re.i) i”^mmdndb . Fn,jbi( ci =- ‘I rue F i n i a n o :
End Sub
briV.itt S л1 ‘ .:l”LcU!f.t^ CLICK(,
^1оъе #1-;
Kill “Tebtr.i-“ Enci bnd Suo
Private Suo ruriii L’^ad () Comricinal .Enabit ‘•i True Coi’mdnd2 . Er.JDied r.aisp Command i. EnaO-l ed - i-alse Command4 . Eriab i-ed - ‘: ^lae Commands.Enabled = False Textl.Enabled =- False Text2.Enabled = False Text3.Enabled = False Text4.Enabled = False Text5.Enabled = False Text6.Enabled = False Text7.Enabled = False End Sub
Private Sub Textl_KeyPress(KeyAscii As Integer) If KeyAscii = 13 Then
If Textl.Text = “” Then
Label7.Caption = «Предупреждение» Text7.Text = «Введите фамилию» Textl.SetFocus GoTo Finish End If
If Label7.Caption = «Предупреждение» Then Label7.Caption = «Результат» Text7.Text = “” End If Text2.SetFocus End If Finish:
End Sub
Private Sub Text2 Keypress(KeyAscii As Integer) If KeyAscii ^ 13 Then
If Text2.Text = “” Then
Label7.Caption = «Предупреждение» Text7.Text = «Введите должность» Text2.SetFocus GoTo Finishi End If
If Label7.Caption = «Предупреждение» Then Label7.Caption = «Результат» Text7.Text = “” End If
Text3.SetFocus End If Finishi:
End Sub
Private Sub Text3 Keypress(KeyAscii As Integer) If KeyAscii ~= 13 Then
If Text3.Text = “” Then
Label7.Caption = «Предупреждение» Text7.Text - «Введите подразделение Text3.SetFocus GoTo Finish2 End If
If Label7.Caption = «Предупреждение» Then Ldbei7.Caption = «Результат» Text 7.Text = “” End If Text4.SetFocus End If Finish2:
End Sub
Private Sub Text4_KeyPress(KeyAscii As Integer) Dim Peremen As Primer
If KeyAscii =. 1 [3] Then
It Text 4.Text = “” Then
Labei7.Caption = «Предупреждение» Text7.Text = «Введите зарплату» Text4.SetFocus GoTo Finish3 End If
If Labei7.Caption = «Предупреждение» Then Label7.Caption = «Результат» Text7.Text = “” End If Finish3:
Peremen.Familia = Textl.Text Peremen.Dolgnost = Text2.Text Peremen.Podrazdelenie = Text3.Text Peremen.Zarplata = Text4.Text Put #10, Nzapisi, Peremen Nzapisi = Nzapisi + 1 Command2.Enabled = True
```



Command3.Enabled = True

Рис. 3.31

```
  Text1.Text = ""
        Text2.Text = ""
        Text3.Text = ""
        Text4.Text = ""
        Text1.SetFocus
    End If
End Sub

Private Sub Text5_KeyPress(KeyAscii As Integer)
     If KeyAscii = 13 Then
         If Text5.Text = "" Then
             Label7.Caption = "Предупреждение"
             Text7.Text = "Введите подразделение
             Text5.SetFocus
             GoTo Finish4
         End If
        If Label7.Caption = "Предупреждение" Ther
             Label7.Caption = "Результат"
             Text7.Text = ""
         End If
       Text6.SetFocus
      End If
  Finish4:
End Sub
```

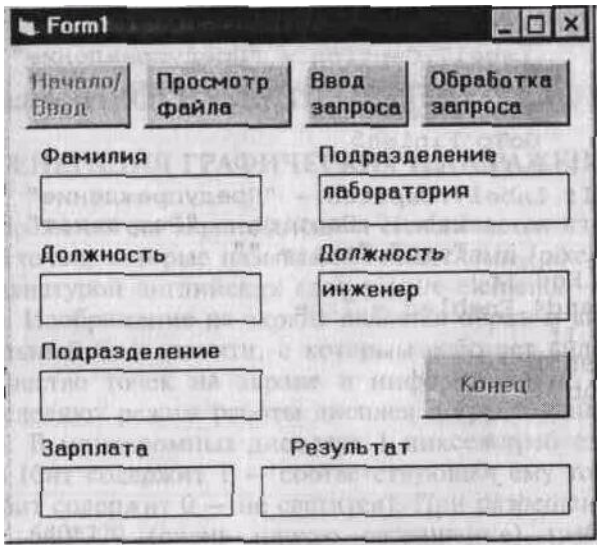
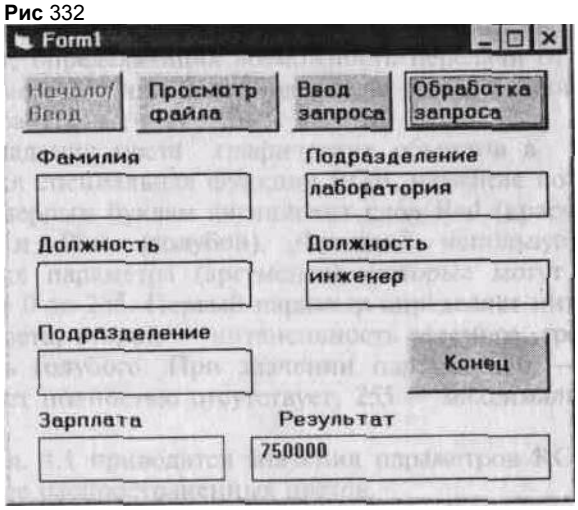


Рис 332




```
    If KeyAscii = 13 Then
      If Text6.Text = "" Then
         Label7.Caption = "Предупреждение"
         Text7.Text = "Введите должность"
         Text6.SetFocus
        GoTo Finish5
       End If
      If Label7.Caption = "Предупреждение" Then
            Label7.Caption = "Результат"
            Text7.Text = ""
       End If
  Command4.Enabled = True
   End If
  Finish5:
End Sub
```

Глава 4. ПРОГРАММИРОВАНИЕ ГРАФИКИ 4.1 ГЕНЕРАЦИЯ ГРАФИЧЕСКИХ ИЗОБРАЖЕНИЙ

Изображение на экране дисплея складывается из множества отдельных точек, которые называются пикселами (pixel — образуется аббревиатурой английских слов picture elements — элементы картинки) Изображение на

экране является образом видеопамяти — специальный блок памяти, с которым работает видеоконтроллер Количество точек на экране и информация по каждой из точек определяют режим работы дисплея и требуемый объем видеопамяти. В монохромных дисплеях 1 пиксел требует 1 бит видеопамяти (бит содержит 1 — соответствующая ему точка экрана светится, бит содержит О—не светится) При разрешении экрана, например, 640*320 (очень плохое разрешение) требуется уже 640*320 = 204 800 бит видеопамяти (25 600 байт).

В цветных дисплеях каждый пиксел кодируется несколькими битами, определяющими цвет Любой цвет создается смешением красного, зеленого и синего цветов различной интенсивности В зависимости от видеоконтроллера на 1 пиксел требуется от 4 до 32 бит, определяющих возможность передачи оттенков цветов (например, 256 или более цветов), и объем видеопамяти достигает 4 Мбайт

Для задания цвета графических объектов в Visual Basic используется специальная функция RGB, название которой образовано по первым буквам английских слов Red (красный). Green (зеленый) и Blue (голубой) Функция использует три целочисленных параметра (аргумента), которые могут принимать значения от 0 до 255 Первый параметр определяет интенсивность красного цвета, второй — интенсивность зеленого, третий — интенсивность голубого При значении параметра 0 — соответствующий цвет полностью отсутствует, 255 — максимальная интенсивность

В табл. 4 1 приводятся значения параметров RGB-функции Для наиболее распространенных цветов.

Теоретически функция RGB позволяет работать с 16 млн. Цветов, но реально цветовая гамма определяется видеоплатой компьютера.

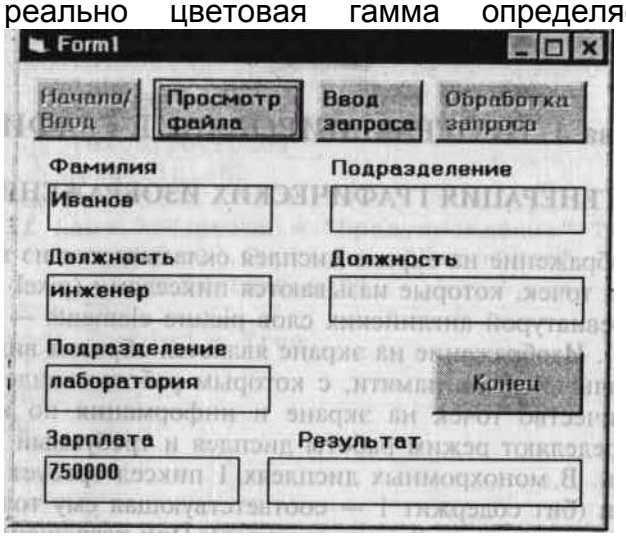


Рис 331

```
Text 1.Text = “”
Text2.Text          = “”
Text3.Text          = “”
Text4.Text          = “” Textl.Setbocus End If End Sub
Private Sub Text5_KeyPress(KeyAscii As Integer) If KeyAscii - 13 Then
If TextS.Text = “” Then
```

Label7.Caption = «Предупреждение» Text7.Text - «Введите подразделение» Texi-S.SetFocus GoTo Finish4 End If
If Label7.Caption = «Предупреждение» Then Label7.Caption = «Результат» Tex!»[7]. Text = “” End If Text6.SetFocus [t]
End If Finish4:
End Sub

Рис 332

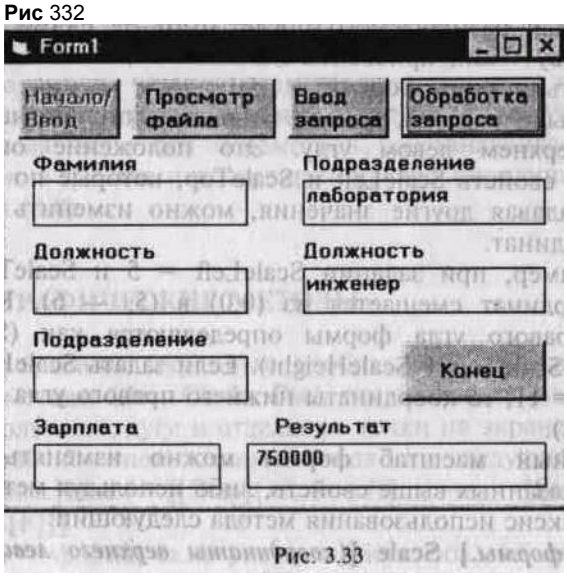


Рис. 3.33

Таблица 4 3

Значения свойства ScaleMode	Единицы измерения
0	Определяются пользователем
1	twips (по умолчанию)
2	Пункты (0 035 см)
3	Пикселы (зависит от разрешения дисплея)
4	Символы (1/6 дюйма по вертикали и 1/12 дюйма по горизонтали)
5	Дюймы
6	Миллиметры
7	Сантиметры

Единицы измерения координат и размеров объектов на форме определяются значением свойства ScaleMode для формы В табл 4 3 приводятся значения свойства ScaleMode и соответствующие им единицы измерения.

При значении ScaleMode равном 0 можно задать собственную систему единиц Для этого также нужно настроить свойства ScaleHeight (шкала по Y) и ScaleWidth (шкала по X) Если, например, ScaleWidth приравнять 100 и ScaleHeight приравнять 200, то ширина формы будет равна 100 единицам, а высота 200 единицам Если для размещаемого после этого на форме текстового окна свойству Width присвоить значение 25, то оно будет занимать четверть ширины формы

Как было сказано, начало координат по умолчанию находится в верхнем левом углу. Это положение определяется значениями свойств ScaleLeft и ScaleTop, которые по умолчанию равны 0. Задавая другие значения, можно изменить положение начала координат

Например, при задании ScaleLeft = 5 и ScaleTop = — 6 начало координат смещается из (0,0) в (5, — 6). Координаты нижнего правого угла формы определяются как (ScaleLeft + ScaleWidth, ScaleTop + ScaleHeight) Если задать ScaleHeight = 18, ScaleWidth = 11, то координаты нижнего правого угла станут (16, 12) (рис 42)

Заданный масштаб формы можно изменять, варьируя значения указанных выше свойств, либо используя метод Scale

Синтаксис использования метода следующий'

[имя формы.} Scale [(координаты верхнего левого угла) — (координаты нижнего правого угла)}

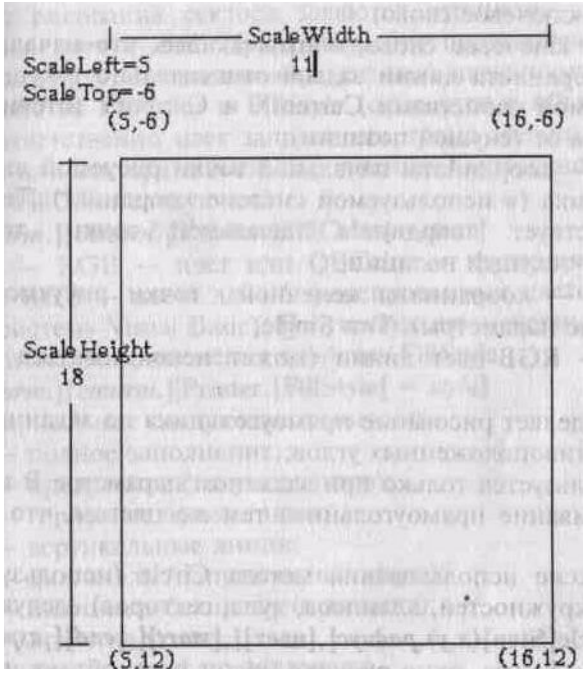


Рис 42

По заданным координатам верхнего левого и нижнего правого угла вычисляются значения ScaleHeight и ScaleWidth. Применение метода Scale

без параметров восстанавливает исходный масштаб окна формы и возвращает начало координат в верхний левый угол (0,0)

4.3. ГРАФИЧЕСКИЕ МЕТОДЫ

Для создания экранных изображений используются графические методы **Line, Circle, Pset,** позволяющие рисовать линии

и прямоугольники, дуги и отдельные точки на экране. ***объект —*** форма, окно рисунка (pictur box), объект печати, на

которых рисуется линия или прямоугольник; **Line** — ключевое слово;

Step — ключевое слово, обозначающее, что начальная или конечная координата линии задана относительно текущей позиции, задаваемой свойствами CurrentX и CurrentY (отсчитывается как смещение от текущей позиции);

xl, yl — координаты начальной точки рисуемой линии или прямоугольника (в используемой системе координат). Тип Single. Если отсутствует координата начальной точки, то линия нчинается от текущей позиции;

x2, y2 — координаты конечной точки рисуемой линии (обязательные параметры). Тип Single;

цвет — RGB-цвет линии (может использоваться функция RGB);

B определяет рисование прямоугольника по заданным координатам противоположенных углов;

F используется только при заданном параметре B и определяет закрашивание прямоугольника тем же цветом, что и линии сторон.

Синтаксис использования метода Circle (используется для рисования окружностей, эллипсов, дуг и секторов) следующий:

[объект.]C1гс\e[81ер](x,y),радиус[, [цвет] [, [start] [,[end}[, aspect]]]} где объект — форма, окно рисунка (pictiir box), объект печати, на которых рисуется окружность, эллипс, дуга или сектор;

Circle — ключевое слово;

Step — ключевое слово, обозначающее, что центр окружности, эллипса, дуги или сектора задан относительно текущей позиции, задаваемой свойствами CurrentX и CurrentY (отсчитывается как смещение от текущей позиции);

x,y — координаты центра (обязательные параметры). Тип Single;

радиус — радиус (в используемой системе координат);

цвет — RGB-цвет линии (может использоваться функция RGB);

start, end используются при рисовании дуги и определяют в радианах начальный и конечный углы дуги (диапазон — от -2 П до 2 П). По умолчанию start равен 0, end — 2 П. Для перевода значений углов из градусов в радианы используется формула ГРАДУ С*П/180.

aspect — соотношение радиусов по осям X и Y при рисовании эллипса.

Для рисования сектора задаются отрицательные значения параметров ***start и end.*** При рисовании проводятся радиальные линии под углом, заданными абсолютными значениями параметра.

Свойства FillColor и FillStyle (относятся к объекту) определяют соответственно цвет заполнения прямоугольника, окружности, дуги или сектора и стиль заполнения. Синтаксис применения свойства FillColor следующий:

{[/or;n.][c<w»-o/.]|Printer.}FillColor[= цвет] где **цвет** — RGB — цвет или QBColor — функция (возвращает значение RGB — цвета для 16 фиксированных цветов, см. Справочную систему Visual Basic). По умолчанию — черный цвет.

Синтаксис применения свойства FillStyle:

{|/<wn.][con/ro/.]|Printer.}FillStyle[= *style]* где *style* принимает следующие значения:
О — полное заполнение;
1 — прозрачное (по умолчанию);
2 — горизонтальные линии;
3 — вертикальные линии;
4 — диагональ (направленная вверх);
5 — диагональ (направленная вниз);
6 — крест на крест;
7- крест на крест по диагонали.
Синтаксис использования метода Pset для высвечивания какой-либо точки определенным цветом следующий
[объект.] PSet[Step](^,^)[,uBeT]
где ***объект*** — форма, окно рисунка (pictur box), объект печати, на которых рисуется точка;
Pset — ключевое слово;
Step — ключевое слово, обозначающее, что координаты точки заданы относительно текущей позиции, задаваемой свойствами CurrentX и CurrentY (отсчитывается как смещение от текущей позиции);

x, y — координаты точки (обязательные параметры). Тип Single;

цвет — RGB-цвет или QBColor-функция (возвращает значение RGB-цвета для 16 фиксированных цветов, см. справочную систему Visual Basic). Если этот параметр отсутствует, то Цвет определяется свойством ForeColor.

Толщина и вид рисуемых линий, а также размер и внешний вид точки определяется значением свойств DrawWidth, DrawStyle и DrawMode. При значении DrawWidth = 1 (измеряется в пикселах) толщина линии и точки равна 1 пикселу. При увеличении значения толшина увеличивается (координата точки определяет ее центр). Значения свойства DrawStyle от 0 до б определяют, какой будет линия:

сплошной, пунктирной, точечной и др.

Свойство DrawMode может принимать одно из 16 значений которые определяют характер взаимодействия вычерчиваемого изображения с уже имеющимся на экране. Например, по умолчанию DrawMode равно 13

(Copy Pen) и означает, что новое изображение закрывает собой существующее.

Метод Point возвращает цвет заданной точки. Синтаксис:

[объект.} Point (x,y)

Возвращает значение цвета в RGB-кодировке пиксела в заданных координатах x и y.

Рассмотрим примеры использования приведенных графических методов. Приводится текст процедуры рисования объектов на форме. Нарисованные объекты показаны на рис. 4.3.

Текст процедуры:

```
Private Sub Form_Pa-Lnt ( )
Scale (-10, 10)-(10, -10) 'установка масштаба
Line (-10, 0)-(10, 0)  'ось X
Line (0, -10)-(0, 10)  'ось Y
```

Line (-8.5, 9.2)-(9.01, -2.1) 'прямая линия Line (-5, 2)-(2, 1), , B 'не закрашен, прямоугольник Line(-8.5,3.5)-(-5.5, -4),,F 'закрашен, прямоугольник

```
Circle (1, 2), 5 'окружность
Circle (-1, -5), 3, , , , 0.4 'эллипс
```

Circle (-7, -9.5), 4, , -0.7, -2.1 'малый сектор Circle (6, -6), 3.5, , -2.1, -0.7 'большой сектор DrawWidth = 10 'изменение ширины линии 'или размера точки

```
PSet (7, 6.5) 'точка
End Sub
```

Методы, осуществляющие вывод информации на форму, позволяют направлять ее непосредственно на принтер. Принтер (объект Printer) рассматривается как особая форма, размер которой -совпадает с размером печатаемой страницы. Метод Scale применим и для объекта Printer и позволяет установить нужную систему координат.

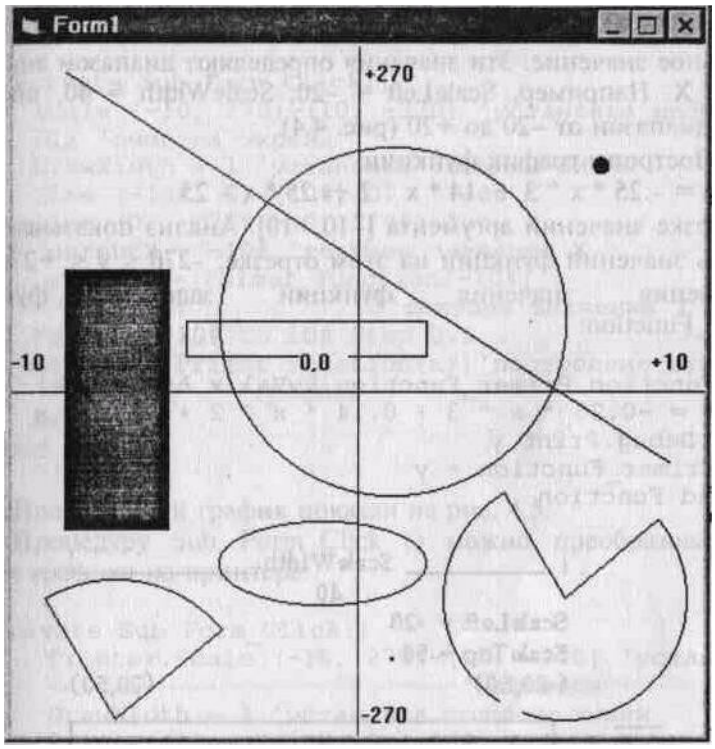


Рис 43 4.4. ПОСТРОЕНИЕ ГРАФИКОВ ФУНКЦИЙ

Использование пользовательской системы координат удобно при построении на экране графиков функций. В этом случае нет необходимости переводить и масштабировать значения аргументов и функций в единицы измерения координатных осей. Достаточно задать систему координат соответственно значениям аргументов и функций.

Для задания стандартной системы координат с точкой (0,0) в середине экрана свойству ScaleTop присваивается некоторое Положительное значение, а свойству ScaleHeight отрицательное значение, которые определяют диапазон значений шкалы Y. На-«ример, ScaleTop = 50, ScaleHeight = -100, шкала Y имеет диапа**зон от -50 до +50. Аналогично свойству ScaleLeft присваивается некоторое отрицательное значение, а свойству ScaleWidth положительное значение. Зти значения определяют диапазон значений шкалы X. Например, ScaleLeft = -20, ScaleWidth = 40, шкала \ имеет диапазон от -20 до +20 (рис. 4 4)**

Построим график функции

y = -.25 * x ^ 3 + .14 * x ^ 2 + .25 * x - 25 на отрезке значений аргумента [-10,+10]. Анализ показывает, что область значении функции на этом отрезке: -270 < y < +270. Для вычисления значения функции зададим функцию Primer_Function:

```
Function Primer_Function(ByVal x As Single) y = -0.25 * x •»• 3 + 0.14 * x - 2 + 0.25 * x - 25
Debug.Print y Primer_Function = y End Function
```

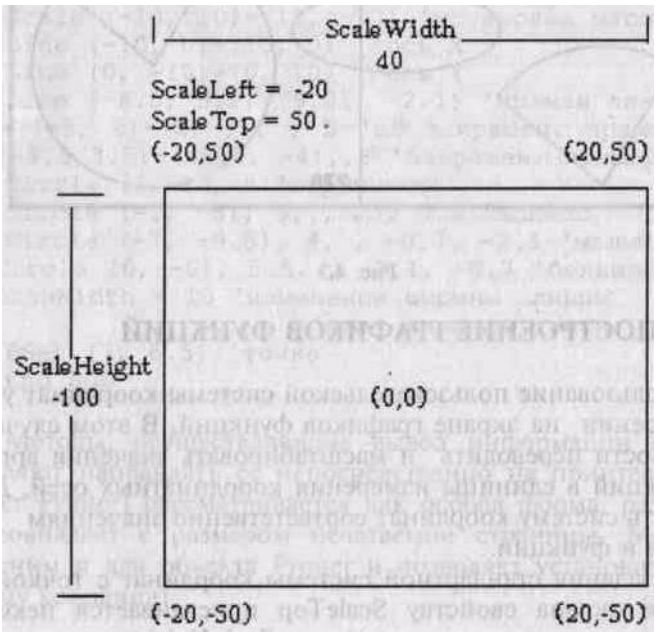


Рис 44

Алгоритм построения графика поместим в процедуру обработки события Form_Click:

Private Sub Form_Ciick()
Scale (-10, 270)-(10, -270) 'установка масштаба CIs 'очистка экрана
DrawWidth = 1 'установка толщины линии Line (-10ft, 0)-(10#, 0) 'ось X Line (0, -270ft)-(0, 270ft) 'ось Y CurrentX = -10# 'текущее значение X CurrentY = Primer_Function(-10ft)

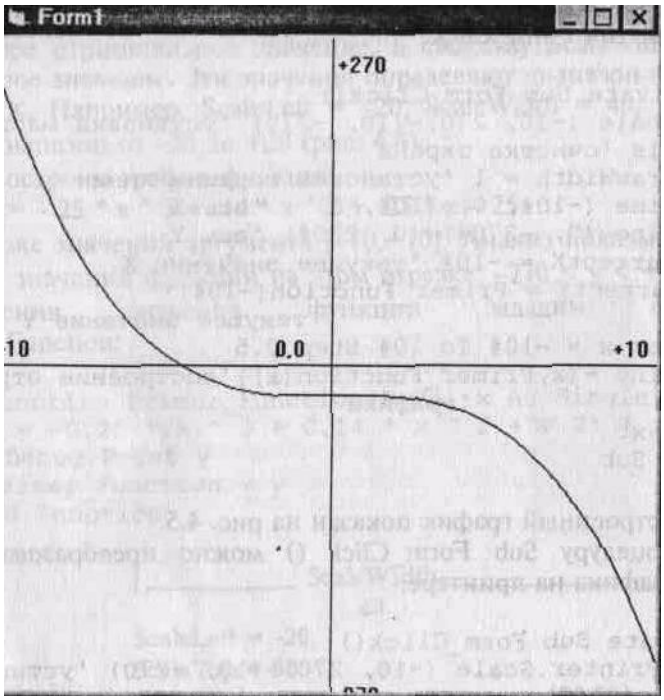
'текущее значение Y • For x = -10ft To 10ft Step 0.5
Line -(x,Primer_Function(x))'построение отрезков
‘графика Next End Sub

Построенный график показан на рис. 4.5. Процедуру Sub Foim_Click () можно преобразовать для печати графика на принтере'

Private Sub Form_Click()
Printer.Scale (-10, 270)-(10, -270) 'установка
'масштаба DrawWidth = 1 'установка толщины линии Printer.Line (-10<t, 0)-(10#, 0)'ось X Printer.Line (0, -270ft)-(0, 270ft) 'ось Y CurrentX = -10ft 'текущее значение X CurrentY = Primer_Function(-10ft) 'текущее

‘значение У
For x = -10ft To 10ft Step 0.5 Printer.Line -(x, Primer_Function(x))
'построение отрезков графика Next Printer.EndDoc End Sub

Кроме того, выведя графическую информацию на форму, можно использовать метод PlintFoirn для распечатки растровой копии



формы на принтере.

Рис 45

4.5. АНИМАЦИЯ ГРАФИЧЕСКИХ ИЗОБРАЖЕНИЙ

Если в последовательные моменты и изменять координаты графических объектов на экране, то объекты будут «перемешаться» по экрану. Для инициирования некоторых событий (в случае анимации изображений в моменты свершения этих событий необходимо изменять координаты графических объектов) через регулярные промежутки времени используется инструмент Таймер (timer). Таймер программируют присвоением определенных значений свойству Interval, которое задает периодичность возникновения событий, связанных с таймером.

Значение свойства Interval задается в миллисекундах (мс) и может принимать значение О...65535 и теоретически способно заставить систему генерировать события со скоростью от 1000 раз

секунду до 1 раза в минуту. Интервал, равный 0 (по умолчанию), отключает таймер. Для того, чтобы события возникали п раз, задаваемое значение свойства Interval можно пассчитывать приблизительно по формуле 1000/п (событие генерируется 2 раза при значении свойства Interval равном 500). Однако необходимо учитывать, что технически для компьютера имеется порог чувствительности разрешения таймера (примерно 18 событий в секунду или 56 мс).

В качестве примера рассмотрим перемещение окружности по форме из нижнего левого угла в правый верхний. Это обеспечивается размещением на форме инструмента таймер, заданием значения свойства Interval отличного от 0 (при выполнении приложения таймер не виден на форме) и следующей программой, включающей описание переменной в разделе general формы, задание начального значения этой переменной в процедуре обработки события Forml_Load и процедуру Timerl_Timer :

```
Option Explicit Dim I As Integer
Private Sub Form_Load()
1=1 End Sub
Private Sub Timerl_Timer() Dim Dx, Dy As Single Beep 'подача звукового сигнала ' Dx = 5 'смещение по оси X Dy •= 5 'смещение по оси Y Scale (0,
```

100)-(100, 0) 'оси координат

'(0,0)-в нижнем левом углу Circle (5 + Dx * (1-1), 5 + Dy * (1-1)), 5
'окружность
I =• I + 1 'переход к следующей точке End Sub
После запуска приложения на форме последовательно рисуются окружности (рис. 4.6).
По такому же принципу можно строить более сложную анимацию (например, движение предметов на экране). В отличии от приведенной картинки движущейся окружности, при движе-чии предметов в каждый момент должно показываться только текущее положение предмета. В каждом конкретном случае это может достигаться комбинацией использования свойств FillColor, FillStyle и DrawMode. В частности, при значении свойства DrawMode, равном 6 (Invert), проведенная еще раз линия или другая фигура через те же координаты рисуется цветом инверсным к первой линии (если первая черная, то вторая белая т.е. при белом фоне экрана исходная линии исчезнет).

Подробно о значениях свойств и их действии при рисовании можно узнать из справочной системы Visual Basic.

Приведем пример программы, обеспечивающей показ движения парохода по бурному морю. Вид движущегося парохода показан на рис. 4.7.

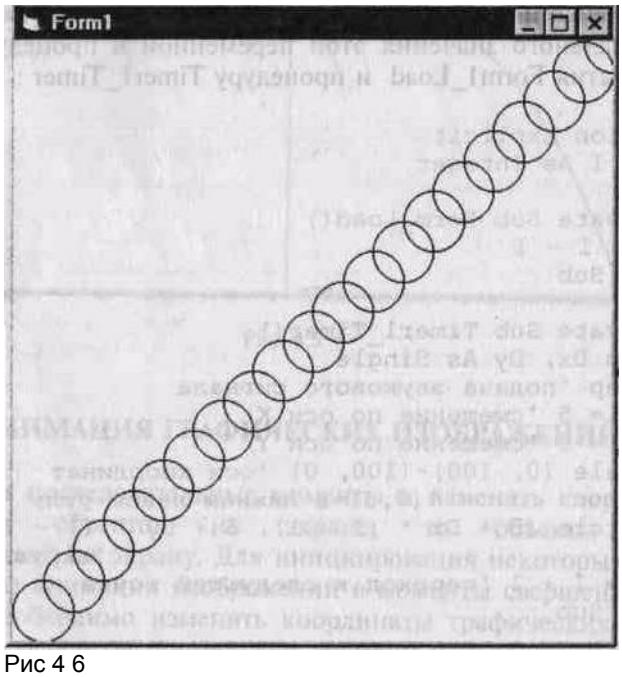


Рис 4 6

Текст программы:

```
Dim I As Integer – в general

Sub Form_Load ()
  I = 0
End Sub

Sub Timer1_Timer ()
 Dim Dx, Dx1 As Single
 Dx = 1# 'шаг перемещения парохода
 Dx1 = 3 'шаг перемещения волны
 Scale (0, 100)-(50, -3) 'задание координат
DrawMode = 16 'инверсия линий рисунка
If I > 0 Then
Line (0# + Dx*(I-1),0#)-(0# + Dx*(I-1),3)        | удаление
Line (0# + Dx*(I-1),3#)-(16# + Dx*(I-1),3)       | изображения
Line (14# + Dx*(I-1),0#)-(16# + Dx*(I-1),3)      | парохода
Line (0# + Dx*(I-1),0#)-(14# + Dx*(I-1),0)       | инверсией
Line (3# + Dx*(I-1),5#)-(11# + Dx*(I-1),3),,B    | изображения
Line (5# + Dx *(I-1),7#)-(6# + Dx*(I-1),5),,B    |

Circle (5# + Dx * (I – 1), 8), .25                | удаление
Circle (5# + Dx * (I – 1), 8), .25                | колец
Circle (3# + Dx * (I – 1), 10), .5                | дыма
Circle (1# + Dx * (I – 1), 12), .75               | инверсией
Circle (-1# + Dx * (I – 1), 14), 1#               | изображения
End If

DrawMode = 13 'рисование заданным цветом
FillStyle = 1 'прозрачный фон

 Line (0# + Dx*I, 0#)-(0# + Dx*I, 3)
Line (0# + Dx*I, 3#)-(16# + Dx*I, 3)      | изображение
Line (14# + Dx*I, 0#)-(16# + Dx*I, 3)     |
Line (0# + Dx*I, 0#)-(14# + Dx*I, 0)      | парохода
Line (3# + Dx*I, 5#)-(11# + Dx*I, 3),,B   |
Line (5# + Dx*I, 7#)-(6# + Dx*I, 5),,B    |

 Circle (5# + Dx * I, 8), .25             |
Circle (3# + Dx * I, 10), .5              | кольца дыма
Circle (1# + Dx * I, 12), .75             |
Circle (-1# + Dx * I, 14), 1              |

For J = 0 To 33
 If I > 0 And I / 2# = I \ 2# Then
   DrawMode = 13
```

```
Circle (0# + Dx1*(2*J + 1),-3),2,,-.5,-2.5 | изображение
Circle (0# + Dx1*(2*J),-3),2.5,,-.5,-2.5    | волн

   DrawMode = 16
   FillStyle = 1
 Circle (0# + Dx1*(2*J),-3),2,,-.5,-2.5      | удаление
Circle (0# + Dx1*(2*J + 1),-3),2.5,,-.5,-2.5 | волн
  Else

    DrawMode = 13
Circle (0# + Dx1*(2*J),-3),2,,-.5,-2.5       | изображение
Circle (0# + Dx1*(2*J + 1),-3),2.5,,-.5,-2.5 | волн

   DrawMode = 16
 Circle (0# + Dx1*(2*J+1),-3),2,,-.5,-2.5    | удаление
Circle (0# + Dx1*(2*J),-3),2.5,,-.5,-2.5     | волн

    End If
    Next J
    I = I + 1
    End Sub
```

В заключение следует отметить разницу в использовании графических элементов управления (Line, Shape), которые используются для улучшения внешнего вида экранных форм, и графических методов. С помощью графических методов графические объекты рисуются на этапе выполнения приложения, а графические элементы управления размещаются на этапе проектирования. Однако, если графические методы используются с

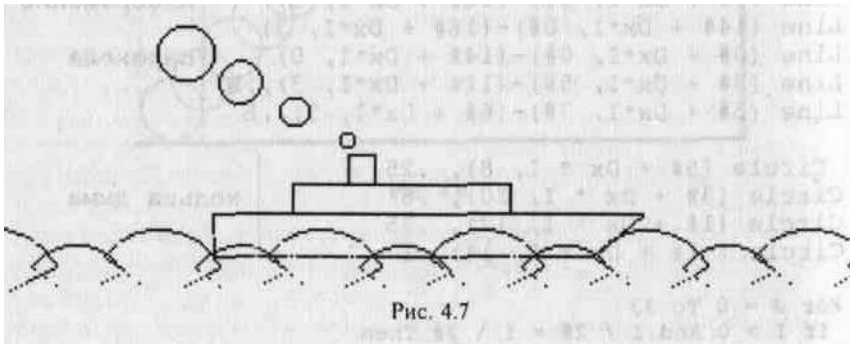


Рис. 4.7

1елью улучшения внешнего вида форм, то они используют мень-пе системных ресурсов, поскольку приложение не хранит описание графических элементов управления. Вместе с тем, результат использования графических элементов управления виден уже на этапе проектирования. **Глава 5. ИСПОЛЬЗОВАНИЕ БАЗ ДАННЫХ**

5.1. МЕХАНИЗМЫ РАБОТЫ С БАЗАМИ ДАННЫХ В VISUAL BASIC

Параллельно с развитием ЭВМ и их широким внедрением во все сферы деятельности увеличивались объемы и усложнялась структура обрабатываемой информации. Для решения проблемы структуризации, хранения и обработки больших объемов информации разработаны системы управления базами данных (СУБД) (database management system — DBMS). Примерами таких систем для персональных компьютеров являются Microsoft Access, Microsoft FoxPro, Borland Paradox, Borland dBase. Универсальной и очень развитой системой для различных платформ

является СУБД Oracle, для больших компьютеров широко используется СУБД DB2 фирмы IBM и др.

Visual Basic использует механизм баз данных Jet фирмы Microsoft для подключения баз данных и доступа к информации, хранящейся в них. Механизм Jet является тем же самым механизмом базы данных, что используется в системе Microsoft Access. Механизм Jet позволяет работать с данными, хранящимися в собственных базах данных Jet (файлы с расширением .MDB), а также с базами данных из других систем, таких как dBase, FoxPro, Paia-dox, BTtrieve, SQL-server.

Доступ к данным из Visual Basic выполняется посредством использования объекта управления данными и связанных с данными объектов управления или объектов доступа к данным. Механизм Jet, объект управления Data (Данные) и связанные объекты управления позволяют легко обрабатывать ввод и вывод элементов данных. Кроме того, имеется два дополнения: Data Manager (Управление данными) и Form Designer (Разработчик данных), которые упрощают построение баз данных и создание форм для доступа к данным.

Типы данных, поддерживаемые механизмом Jet, во многом совпадают с типам данных Visual Basic и приведены в табл. 5.1.

При работе с большими объемами информации необходимо, по возможности, гарантировать правильность (корректность) информации в базе данных (прикладная информация, организованная на основе используемой СУБД). Эта корректность может обеспечиваться написанным программным приложением, средствами СУБД или и тем и другим. Механизм Jet поддерживает два

типа целостности данных (целостность первичного ключа и ссылочную целостность отношений) и две формы правильности данных (правильность на уровне поля и правильность на уровне записи). Эти функции можно реализовать на этапе создания базы данных и они далее поддерживаются и используются при обращениях к базе данных.

Таблица 5.1

Тип данного	Информация	Диапазон значении
Text	Строки символов	255 символов
Memo	Длинные строки символов	До 1,2 Гбайт
Byte	Целые	От 0 до 255
Integer	Целые	От -32768 до 327670
Long	Целые	От -2147483648 до 2147483647
Counter	Длинные целые с приращением	
Single	Вещественные	От-3,4*10-38 до 3.4* 10»38
Double	Вещественные	От -1,8*10»308 до 1.8*10”308
Yes/No	Логическая	

Date	Значение д>пъ1	
Binary	Двоичные	До 1,2 Гбайт
OLE	OLE — объекты	До 1.2 Гбайт

Кроме механизма Jet можно также использовать драйверы связи открытых баз данных (ODBC — Open Database Connectivity) для доступа к другим базам данных. Существуют также разнообразные механизмы баз данных других фирм, которые можно использовать с Visual Basic.

Система Visual Basic позволяет хранить и использовать информацию в реляционных системах управления базами данных (английская аббревиатура — RDBMS).

5.2. РЕЛЯЦИОННАЯ СТРУКТУРА ДАННЫХ

В подавляющем большинстве существующие СУБД построены на основе реляционной модели данных, которая, несмотря на свою внешнюю простоту, базируется на прочном фундаменте масштабных теоретических исследовании, основы которых были заложены Э. Коддом в 1969 г. Результаты этих теоретических исследований позволяют сделать данные полными непротиворечивыми и не избыточными, чтобы все факты оказав лись учтены, но при этом каждый из них хранился только один' раз. Специалисты в области создания и сопровождения баз данных должны знать и уметь использовать эту теорию.

Реляционная база данных это такая база данных, которая состоит из таблиц (и ничего иного, кроме таблиц). Ссылка из одной таблицы на другую через какое-нибудь общее поле (common field) называется отношением (relation) (отсюда и название реляционная).

Рассмотрим пример реляционной базы данных состоящей из трех таблиц или отношений, таблица поставщиков, таблица деталей и таблица поставки деталей.

Таблица поставщиков

Номер постав щика	Фамилия	Состояние	Город
si	Смит	20	Лондон
s2	Джонс	10	Париж
s3	Блеик	30	Париж
s4	Кларк	20	Лондон
s5	Адаме	30	Атенс

Каждый поставщик имеет уникальный номер, фамилии могут повторяться. Каждый поставщик находится только в одном городе.

Таблица деталей

Номер детали	Название	Цвет	Вес	Город
Pi	Гайка	Красный	12	Лондон
p2	Болт	Зеленый	17	Париж
P3	Винт	Голубой	17	Рим
p4	Винт	Красный	14	Лондон
p5	Кулачок	Голубой	12	Париж
P5	Заклепка	Красный	19	Лондон

Каждый вид детали имеет уникальный номер. Город — место хранения детали. Предполагается, что каждый вид детали имеет только один цвет и хранится на складе только одного города.

Таблица поставки деталей

Номер поставщика	Номер детали	Количесшо
Si	Pi	300
Si	P2	200
si	РЭ	400
sl	P4	200
si	P5	100
sl	p6	100
s2	P'	300
s2	P2	400
s3	P2	200
s4	P2	200
s4	P4	300
s4	P5	400

Таблица поставки деталей служит для связи между собой двух первых таблиц. Например, первая строка этой таблицы связывает поставщика «sl» из таблицы поставщиков с деталью «p1» из таблицы деталей (представляет поставку детали вида «p1» поставщиком с номером «sl» в количестве 300 деталей). Предполагается, что в каждый момент существует только одна поставка для заданного поставщика и заданной детали, т.е. комбинация *Номер_поставщика, Номер_детали* является уникальной для заданной поставки относительно множества поставок в данный момент.

Принципиальной особенностью таблиц является следующее:

- данные в таблицах имеют только явные значения, т.е. не существует каких-либо связей или указателей, соединяющих одну таблицу с другой. Связь, например, между поставщиком и деталью задана наличием конкретного номера поставщика (sl) и конкретного номера детали (p1). Такие ссылки из одной таблицы на Другую через какое-нибудь общее поле (Common field) называется отношением;
- таблица состоит из строки заголовков столбцов и нуля или более строк значений данных;
- строка заголовков столбцов специфицирует один или более столбцов, определяя также тип данных для каждого из них (все значения в данном столбце одного типа. Например, имя поставщика типа строки символов, число деталей некоторого арифметического типа и т.п.);
- значения данных являются атомарными, т.е. в каждой ячейке таблицы всегда только одно значение.

Строка таблицы также называется **записью.** Элемент записи называется **полем** (наименьшая единица информации в базе данных)

Прикладная информация может по-разному задаваться таблицами (т.е количество таблиц и их содержание могут быть различными для одной и той же прикладной информации). Например, можно было бы вместо трех таблиц «Таблица поставщиков» «Таблица деталей» и «Таблица поставки деталей» сделать одну объединенную таблицу. В этом случае происходило бы дублирование информации (для каждой детали повторялась бы информация о поставщике). Если бы изменилась какая-либо информация по поставщику (например, состояние), то эту информацию необходимо бы было изменить во всех записях.

Для правильной организации данных в таблицы используется понятие **нормализации. Нормализация данных** — процесс исключения

избыточной информации, при которой достигается то что каждый элемент информации запоминается только один раз. Теория и практические рекомендации по нормализации рассматриваются в книгах по реляционным базам данных.

Связь между таблицами реализуется с использованием ключей данных. Ключи обычно называют **первичными и внешними ключами. Первичный ключ** однозначно определяет запись в таблице (в таблице поставщиков таким первичным ключом является номер поставщика) и должен быть уникальным. **Внешним ключом** является ключ, используемый для связи с первичным ключом другой таблицы (в таблице поставок такими внешними ключами являются номер поставщика и номер детали, которые используются для связи с другими таблицами). Таблицы связываются между собой с помощью ключевых полей.

Ключевое поле может иметь содержательный смысл (например, фамилия, но в этом случае фамилии не могут повторяться) или полем, которое служит специальной цели обеспечения уникальности записи (например, номер поставщика или счетчик для первичного ключа). Поле счетчика является целочисленным полем, которое автоматически увеличивается на единицу системой управления базами данных, когда добавляется новая запись. При этом механизм создания уникальных полей перекладывается на систему.

Приведенные таблицы являются исходными (базовыми) для задания данных и связей между ними. Такие таблицы физически существуют в памяти ЭВМ, хотя их вид необязательно соответствует приведенному (физическое представление данных, т.е. то, как данные реально хранятся на носителях данных, отличается от их логического представления в виде рассмотренных таблиц и зависит от типа ЭВМ и носителя данных).

Кроме того, на основе базовых таблиц могут быть созданы производные от них или представления. Например, таблица поставщиков деталей в количестве больше 200 имеет вид:

Номер поставщика	Номер детали
si	Pi
si	P3
s2	pi
s2	?2
s4	P4
s4	?5

Такая таблица является производной или виртуальной таблицей, те. таблицей которая непосредственно не существует в физической памяти, но для пользователя, получающего информацию, выглядит существующей.

В результате информационного запроса (query) к реляционной базе данных также получается некоторая таблица, которая является представлением существующих данных. Заложенные в исходных таблицах отношения позволяют конструировать различные логические представления данных. В

частности приведенная таблица является результатом запроса о поставщиках и деталях, поставляемых в количестве более 200.

Для более эффективной работы с реляционной базой данных (повышение скорости получения информации и модификации данных) используется индексирование (аналогично рассмотренному индексированию записей файла). Индекс создается для одного или нескольких заданных столбцов таблицы (для одной таблицы и группы столбцов может быть создано несколько индексов) и для заданного упорядочения (например, по убыванию) и автоматически поддерживается средствами СУБД при изменении информации. Для приведенных таблиц могут быть, например, созданы индексы по столбцу «Номер поставщика» таблицы поставщиков, по столбцу «Номер детали» таблицы деталей, по двум столбцам «Номер поставщика» и «Номер детали» таблицы поставки деталей.

Для работы с реляционными базами данных существует стандартный язык запросов SQL (Structured Query Language — язык структурированных запросов). Этот язык используется для создания исходных таблиц базы данных (создание базы данных) формирования запросов, управления базами данных, позволяя использовать стандартные средства для работы с различными базами данных. Visual Basic позволяет использовать язык SQL для работы с базами данных.

5.3. ИСПОЛЬЗОВАНИЕ ЭЛЕМЕНТОВ УПРАВЛЕНИЯ Data u Grid

Элемент управления **Data** служит для установления связи между базой данных и другими элементами управления, используемыми .зля отображения данных из базы данных. Поддерживается связь с базами данных Access, FoxPro, dBase, Paradox (список поддерживаемых связей определяется версией Visual Basic).

Один элемент управления Data всегда обеспечивает доступ только к одной записи в данный момент (такая запись называется **текущей (current)),** позволяя отображать содержимое текущей записи на форме.

Кроме стандартных свойств, элемент управления **Data** обладает следующими специфическими свойствами:

- **BOFAction** (Beginning Of File — начало файла), **EOFAction** (End Of File — конец файла) возвращает или устанавливает значение, указывающее действие при значении BOF или EOF, равном True.

Свойство BOF равно True, если указатель текущей позиции в файле расположен до какой-либо записи (обычно возникает при удалении первой записи). Свойство EOF равно True, если указатель текущей позиции в файле расположен за существующими данными. Если любое из этих свойств равно True, то текущая запись имеет недопустимый номер. Если оба свойства равны True, то в файле не содержится данных.

Синтаксис:

объект. **BOFAction [=** ***значение}*** **объект. EOFAction f=** ***значение}***

BOFAction =1 — переход к первой записи, если BOF = True. EOFAction =1 — переход к последней записи, если EOF =

True.

- **Connect** (Подключение) определяет тип (формат данных) используемой базы данных (Access, FoxPro, dBase, Paradox и др.).
- **Database** объект «описатель базы данных» (профессиональная версия Visual Basic)
- **DatabaseName** (имя базы данных) определяет имя файла базы данных или переход.
- **EditMode** возвращает значение, определяющее состояние редактирования для текущей записи.
- **Exclusive** определяет использование базы данных одни или несколькими пользователями (True — использование одним пользователем. False (по умолчанию) — несколько пользователей).
- **Readonly** определяет возможность редактирования ото-// бражаемого данного в элементе управления (False (по умолчанию) — редактирование возможно, True — невозможно).
- **Recordset** возвращает или устанавливает объект Recordset, определенный свойствами элемента управления Data или существующим объектом Recordset.

Элемент Data автоматически инициализируется при запуске приложения. Если свойства Connect, DatabaseName, Options, RecordSource, RecordSource, Exclusive, ReadOnly и RecordsetType установлены или, если они устанавливаются на этапе выполнения при использовании метода Refresh, механизм баз данных Jet пытается создавать новый объект Recordset, основанный на этих свойствах. Этот объект Recordset доступен через свойства элемента управление Data.

Важно: ссылаться на свойства объекта Recordset, создаваемого элементом управления Data, можно только используя свойство Recordset самого элемента Data. Синтаксис такого определения свойства следующий: о^бетс/и/.свойствоА.свойствоБ определяет свойствоБ некоторого объекта, адресуемое через свойствоА объекта!.

Объект **Recordset** представляет запись в основной таблице или запись, которая является результатом выполнения запроса. Когда используются объекты доступа к данным, то это происходит с использованием объектов Recordset. Таблица. Recordset есть представление основной таблицы. Dynaset-тип.Recordset — динамический набор записей. Кадр. Recordset — статическая копия множества записей (может содержать поля из одной или более таблиц в базе данных, но не может модифицироваться).

- **RecordsetType** определяет тип набора записей, который можно использовать для доступа к данным. Существует три типа наборов записей: таблица. Dynaset (динамическое множество) или кадр (моментальный снимок). Соответственно свойство принимает значения 0, 1,2.

При использовании Data для доступа к базе данных по умолчанию создается набор записей динамического типа, который является набором указателей на информацию. Динамический набор является очень гибким, но имеет ряд ограничений при использовании (в частности не поддерживает созданные индексы, что не позволяет динамически изменять порядок представления записей в наборе).

При задании типа набора «Таблица» возможен доступ только к одной полной таблице с использованием индекса для упорядочения, но обновление данных отображается медленно.

При задании типа набора «Кадр» работа выполняется с копией данных. Достоинством является быстрота, но требует ресурсов памяти и невозможно обновление данных.

- **RecordSource** (источник данных) определяет, откуда извлечь данные. Это может быть имя таблицы либо конкретное логическое представление данных (запрос).

Для объекта Data при работе с базами данных существует понятие **базовых элементов управления** — непосредственно связаны с единственным полем в наборе записей и не требуют дополнительного определения, кроме имени поля, задаваемого для некоторого свойства элемента управления. Например, для текстового окна (TextBox) присвоение свойства Text может содержать значение поля базы данных (тип данных: строка символов, арифметический тип, дата) и определяется заданием свойств DataSource (указывается имя объекта Data) и DataField (указывается имя поля таблицы, установленной свойством Data-Source объекта Data). Окна изображений и рисунков можно использовать для показа картинок, хранящихся в базе данных, флажок — для отображения булевых значений.

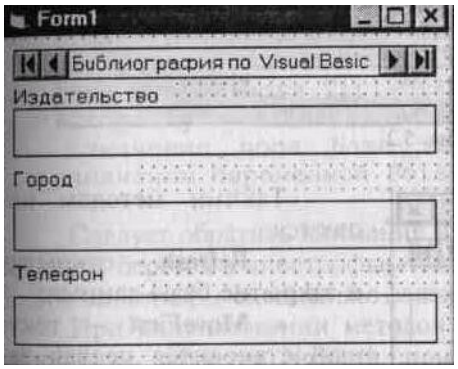
В комплекте Visual Basic имеется база данных B1BLIO.MDB, содержащая библиографические сведения по Visual Basic. Она состоит из трех таблиц. Таблица Авторы (Authors) с полями идентификационный номер (Au_ID) и фамилия автора (Authors). Таблица издательств (Pablishers) с полями идентификационный

номер (PubID), название (Name), компания владелец (Company Name), адрес (Address), город (City), штат (State), код (Zip), теле-tboH (Telephone). Таблица Названия книг (Titles) содержит список опубликованных книг: название (Title), год публикации (Year published) и ссылки на первые две таблицы.

Используем элемент управления Data для доступа к этой базе данных. Создадим форму (рис. 5.1) с элементом управления Data, с тремя метками и с тремя текстовыми окнами для вывода соответствующей информации из базы данных **B1BLIO.MDB.** Свойствам объекта Data присваиваются следующие значения:

DataBaseName = C:\VB4\BIBLIO.MBD
RecordSourse = Pablishers
Connect = Access.

Для подключения базы данных (задание значения свойства DataBaseName = C: \VB4\BIBLIO.MBD) используется специальное окно (рис. 5.2).



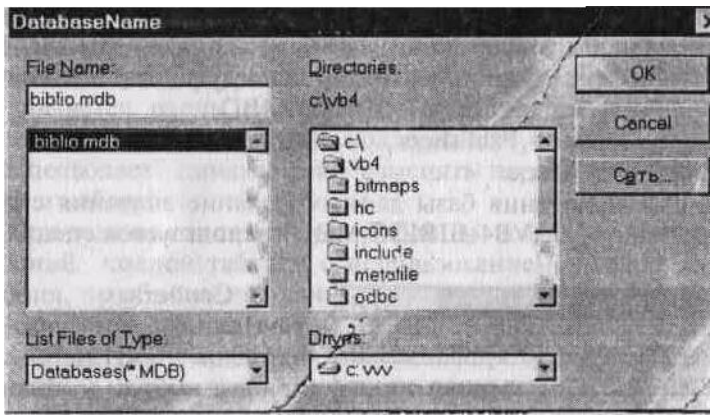
Свойствам объекта Textl (текстовое окно с заголовком «Издательство») присваиваются значения:
DataSourse = Datal (имя элемента Data — Datal), DataField = Name (Name — имя поля таблицы издательств). *
Рис. 5.

Свойствам объекта Text2 (текстовое окно с заголовком «Город») присваиваются значения: DataSourse = Datal (имя элемента Data — Datal), DataField = City (City — имя поля таблицы издательств).

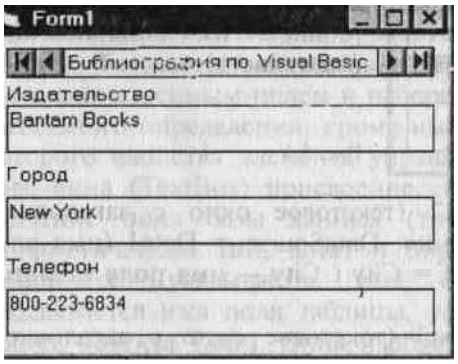
Свойствам объекта Text3 (текстовое окно с заголовком «Телефон») присваиваются значения: DataSourse = Datal (имя элемента Data — Datal), DataField = Telephone (Telephone — имя поля таблицы издательств).

Созданное приложение позволяет просматривать базу данных, получая из нее указанную информацию. Элемент управления Data обеспечивает движение по записям (последовательное Движение к последующей записи или к предыдущей записи при щелчке мышью по правой или левой стрелке объекта и переход « первой или последней записи при щелчке мыши по левой или правой кнопке окна) (рис. 5.3).

Аналогичные действия можно проделать без использования мыши, а программно. Для этого применяются методы объекта Data.



PHL 5 2



Такгцуж методам являются:

- **Refresh** — открытие и закрытие базы данных;
- **MoveFirst —** текущей становится первая запись из множества записей:
- **MoveNext** — текущей становится следующая запись из множества записей;

Рис 53

- **Move Previous** — текущей становится предыдущая запись из множества записей;
- **MoveLast** — текущей становится последняя запись из множества записей;
- **Update** — редактированное поле записывается в базу данных (этот можно применять только для таблиц и динамических множеств);
- **FindFirst** найти первую запись для заданного условия поиска;
- **FindNext** найти следующую запись для заданного усло-„ия поиска:
- **FindLast** найти последнюю запись для заданного условия

поиска;

- **FindPrevious** найти предыдущую запись для заданного условия поиска.

Последние четыре метода требуют задания критерия поиска записи. В общем случае критерием является строка символов, которая может включать логические выражения. Переменной типа строки символов, являющейся критерием, присваивается значение Следующие примеры показывают задание критериев:

Dim Criteriy As String

Criterly = “State = ' NY ' « ' значение поля State (Штат) таблицы должно быть равно NY Criteriy = “Title > ‘A’ And Title < 'B' « ' первой буквой поля Title таблицы должна быть буква A Criteriy = “Name =» & «'» & Poisk & «'» ' значение поля Name таблицы должно быть равно значению переменной Poisk

Следует обратить внимание на то, что задаваемое значение в условиях берется в апострофы « символ — '». В последнем примере эти апострофы специально задаются.

При использовании методов для поиска записей используется также свойство **NoMath,** применяемое для таких объектов, как таблица, динамическое множество, кадр или Recordset. Синтаксис использования свойства следующий:

Объект. NoMath

Свойство принимает значение True (требуемая запись не найдена) или

False (требуемая запись найдена). Если поиск оказался безрезультатным, то свойство имеет значение True и положение текущей записи (в момент применения одного из указанных выше методов поиска) не меняется.

Как было сказано выше, ссылаться на свойства объекта Recordset, создаваемого элементом управления Data, можно только используя свойство Recordset самого элемента Data (синтаксис см. выше). Таким образом, при использовании перечисленных методов вставляется свойство Recoidset элемента Data. Например,

Datal.Recordset.MoveLast Datal.Recordset.MoveNext Datal.Recordset.FindFirst FindCriterly

Пример программы.

Модифицируем приведенную выше форму, добавив командные кнопки и текстовую строку (рис 5 4) Свойству Visible для элемента управления Data присвоим значение False (т е на этапе выполнения окно Data не видимо и его нельзя использовать для диалогового доступа к базе данных) Функции движения по записям базы данных, действии с записями, поиска программируются процедурами обработки событии соответствующих командных кнопок Критерии поиска требуемой записи вводится в текстовом окне, которое активизируется командной кнопкой «Поиск» (отрицательный результат поиска также выводится в этом текстовом окне). В процедурах используются выше рассмотренные свойства и методы

Процедуры обработки событий-

```
Option Explicit Dim Pri, Pr2 As Boolean
Private Sub Commandl Click()
If Not Datal.Recordset.EOF  Then Pr2 = True
Commandl.Enabled = True Datal.Recordset.MoveNext
If Not Pri Then Pri = True: Command2.Enabled = True Else Pr2 = False
Datal.Recordset.Move Previous Commandl.Enabled = False End If End Sub
Private Sub Command2_Ciick() If Not Datal.Recordset.BOF Then Pri - True Command2.Enabled = True
```

```
Datal.Recordset.Move Previous
If Not Pr2 Then Pr2   True: Commandl.Enabled = True Else Pri - False Datal.Recordset.MoveNext Command2.Enabled =
False End If End Sub
Private Sub Command3 Click(\
Datal.Recordset.Move First
Pr2 = True
Command2.Enabled   False
Commandl.Enabled - True
Pri - False End Sub
Private Sub Command4 Click(
Datal.Recordset.MoveLast
Pri = True
Commandl.Enabled = False
Command2.Enabled = True
Pr2 - False End Sub
Private Sub Command5_Click/
Datal.Recordset.Delete
Datal.Recordset.MoveNext End Sub  Private Sub Command6_Click()
Datal.Recordset.Update End Sub
Private Sub Command7_CiicK ( i
Text4.SetFocus End Sub
Private Sub Form_Load()
Pri = True
Pr2 = True End Sub
Private Sub Text4_KeyPress(KeyAscil As Integer)
Dim FindCriteriy, Poisk As Variant
If KeyAscil = 13 Then
Poisk = Text4.Text
FindCriteriy = "Name =" & "'" &              Poisk & "'"
Datal.Recordset.FindFirst FindCriteriy
If Datal.Recordset.NoMatch Then Text4.Text = «Такой записи нет»
End If
End If End Sub
```

Примеры работы с формой показаны на рис. 5.5 — 5.7. Обратите внимание, что элемент Data на форме не виден. При попытке удаления записи с использованием Delete срабатывает защита целостности базы данных (выдается окно предупреждения на рис. 5.7). База данных включает три связанные таблицы. Удаления записи из одной таблицы, на которую ссылается запись в другой таблице, приводит к разрушению информации.

В рассмотренных выше примерах база данных, необходимые таблицы и поля записей определялись на этапе разработки приложения. Однако Visual Basic позволяет открывать при работе некоторого приложения любую существующую на компьютере базу данных, определять состав ее таблиц и записей и выводить для анализа всю таблицу или информацию из нее по запросу (такой способ подключения к базе данных на этапе выполнения приложения называется динамическим доступом).

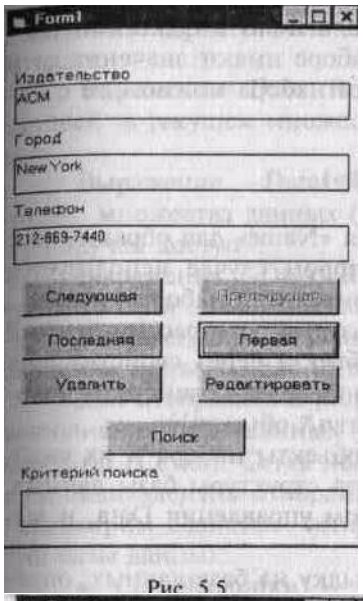


Рис. 5.5

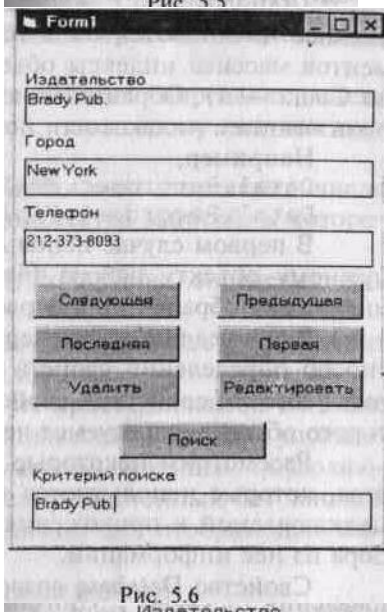


Рис. 5.6

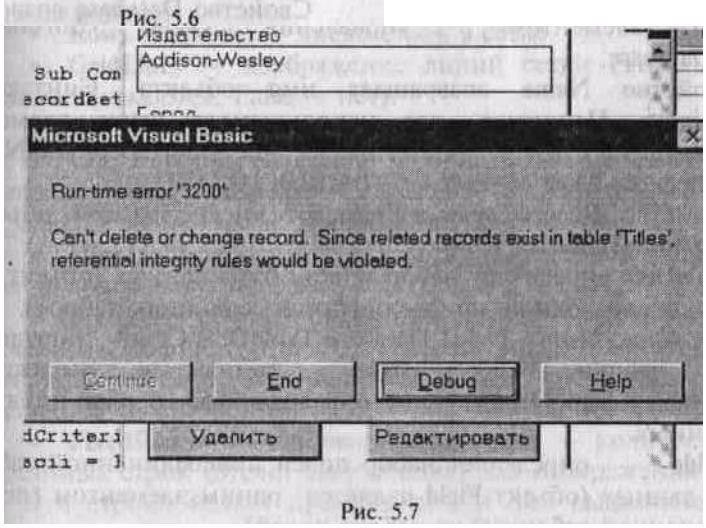


Рис. 5.7

Для дальнейшего изложения рассмотрим еще одно понятие—**наборы**

(collections) Набор — это несколько связанных объектов, для которых определены общие свойства. Общим для всех наборов является свойство **Count,** определяющее число объектов в наборе (аналогично индексации элементов массива индексы объектов в наборе имеют значения от о до Count — 1). Обращаться к элементам набора можно или с использованием индекса или по именам.

Например,
Datal.Recordset.Fields(«Name»)
Data 1.Recordset.Fields(0)
В первом случае используется имя «Name» для обращения к нужному объекту набора Fields. Во втором случае используется индекс для обращения к первому элементу этого набора.

В приведенных примерах используется уже рассмотренный способ определения свойств одного объекта через свойства другого *(объект* /.свойствоА.свойствоБ — определяет свойствоБ некоторого объекта, адресуемое через свойствоА объекта!).

Рассмотрим некоторые наборы, объекты набора и их свойства, которые используются для анализа структуры базы данных, подключаемой к приложению элементом управления Data, и выбора из нее информации.

Свойство **Database** возвращает ссылку на базу данных, определенную элементом управления Data. Синтаксис:
Объект. Database.
Свойство **Name** возвращает имя объекта. Синтаксис:
Объект.^ame. Например, для используемого выше элемента управления Data с именем Datal выражение Datal. Database. Name определит файл базы данных C:\VB4\BIBLIO.MBD.
Свойство **RecordCount** возвращает число записей объекта RecordSet или TableDef.
TableDefs определяет набор таблиц базы данных (объект **TableDef** является одной присоединенной таблицей набора). Например, выражение Datal.Database.TableDefs.Count определит число таблиц в базе данных. Выражение Datal. Database.Table Defs(O). Name определит имя первой таблицы в базе данных.
Fields определяет набор полей присоединенной таблицы базы данных (объект **Field** является одним элементом (полем присоединенной таблицы) из набора полей).
Выражение Datal.Database.TableDefs(0).Fields.Count определит число полей в первой таблице базы данных.

Выражение Datal. Database. Table Defs(l).Fields(0). Name определит имя первого поля второй таблицы присоединенной базы данных.

Выражение Datal. Recordset. RecordCount определит число записеи в текущем множестве записей, к которому получен доступ.

Выражение Datal. Recordset. Fields.Count определит число полей множества данных (текущее множество записей, к которому получен доступ).

Выражение Datal. Recordset. Fields(2). Name определит имя третьего поля на множестве данных.

Выражение Datal. Recordset. Fields(2). Value определит значение третьего поля на множестве данных.

Для отображения информации из присоединяемой на этапе выполнения базы данных удобно использовать элемент управления **Grid** (Сетка). Сетка является двумерной таблицей, позволяющей эквивалентно отображать таблицы реляционных баз данных. Рассмотрим свойства сетки, необходимые для отображения таблиц базы данных:

- **Cols** (Колонки) — число колонок в сетке.
- **Rows** (Строки) — число строк в сетке.
- **GridLines** — изображение линий сетки (True — линии сетки изображаются. False — нет).
- **GridLinesWidth** — ширина сетки таблицы.
- **Col, Row** пределяет или устанавливает активную ячейку (номера колонки и строки). Синтаксис: *имя_сетки.Со*\ [= *номер], 11мя_сетки.Ро^/* [= *номер}.* Доступны только на этапе выполнения.
- **HighLight** определяет подсветку выделенной ячейки сетки (True — подсветка есть. False — нет).
- **FixedCols** (Фиксированные колонки) — количество фиксированных колонок (отсчет от левого края) для отображения заголовков таблицы. Эти колонки нельзя прокручивать и они выделяются серым фоном.
- **FixedRows** (Фиксированные строки) — количество фиксированных строк (отсчет сверху вниз) для отображения заголовков. Эти строки нельзя прокручивать и они выделяются серым фоном.

• • **ScrolBars** (Линейки прокрутки) — горизонтальная и вертикальная линейки прокрутки для отображения невидимых колонок и строк. ScrolBars = *0* — нет линеек прокрутки, 1 — горизонvbOFNPathMiistExist — возможность ввода только существующего пути и др

Например, следующий оператор определяет возможность ввода только имени существующего файла и существующего пути

чмя_станд _окна Flags = vbOFNFileMustExist Or \bOFNPathMustExist

Пример программы.

Приводится пример приложения, которое открывает существующую на компьютере базу данных, выдает список таблиц базы данных и выводит выбранную пользователем таблицу в сеточную форму Будет использоваться уже рассмотренная выше база данных B1BLIO MDB Используются элементы управления Data (имя — Datal), Gild (имя — Giidl, командная кнопка (имя — Commandl). комбинированный список (имя — Combol) и Common Dialog (имя — CommonDialog) Форма на этапе разработки имеет вид, показанный на рис 5 9

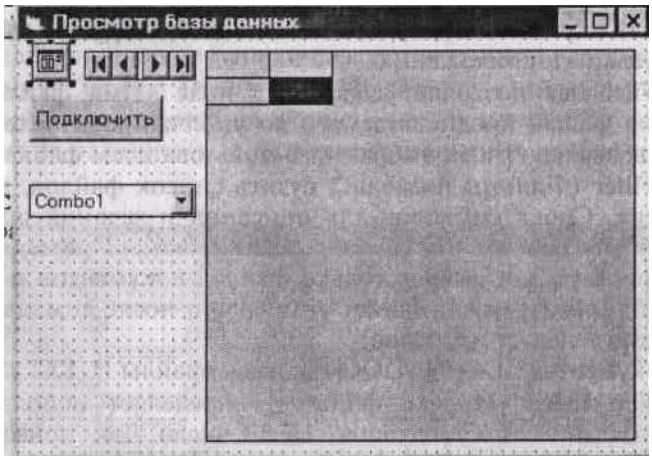


Рис 5 9

Алгоритм работы приложения следующий При щелчке мыши по командной кнопке «Подключить», появляется стандартное диалоговое окно для выбора файла базы данных. После выбора файла базы данных программно формируется список таблиц выбранной базы данных, который выводится в комбинированное окно После выбора элемента списка (таблица базы данных) ч

нажатия клавиши «Ввод», содержимое таблицы выводится на сетку Линейки прокрутки сетки позволяют просматривать все поля сетки

Текст программ'

```
Option Explicit
Const vbOFNPAlhMUSTEXIST - &H800&
Const vbOFNFILEMUSTEXIST   &H1000&
Private Sub Commandl_Click() Combol.Clear
CommonDiaiog.DefaultExt - “MDB” CommonDialog . FileName - “” CommonDialog.Filter =
«Базы данных MS Access(*.MDB)I*.MDB” CommonDialoci. Flags   vbOFNPATHMUSTEXIST CommonDiaiog.Action - 1
If CommonDiaiog.FileName   “” Then Exit Sub OpenDataBase CommonDialog.FileName End Sub
Public Sub OpenDataBase(ByVal DataFile As String)
Dim I As Integer
Datal.Connect - “”
Datal. Dat-abaseName = DataFile
Datal.Refresh
For I = 1 To Datal.Database.TableDefs.Count - 1
Combol.AddItem Datal.Database.TableDefs(I).Name
Next
Combol.Text = “”
End Sub
Private Sub Combol Keypress(KeyAscii As Integer)
If KeyAscii = 13 Then FillGrid Combol.Text
End If End Sub
```

Private Sub FillGrid^ByVal TableName As String) Dim I As Integer, CellWidth As Integer Datal.RecordSource = TableName Gridl.Cols = Datal.Database(TableName).Fields.Count

```
Gridl.Row = 0 For I =, 0 To Datal.Database(TableName).Fields.Count-1     Gridl.Col = I Gridl.Text =
Datal.Database(TableName).Fields(I).Name
Gridl.ColWidth(I) = TextWidth(Gridi.Text) + 100 Next
Datdl.Refresh
```

```
Datal.Recordset.MoveLast
Gridl.Rows = Datdl.Recordset.RecordCount + 1
Datal.Recordset.Move First
Gridl.Row ^ 0
Do While Not Datdl.Recordset.EOF
Gridl.Row = Gridl.Row +• 1 For I = 0 To Datal.Database(TableName).Fields.Count-1
Gridl.Col = I If IsNull(Datal.Recordset.Fields(I)-Value) Then
Gridl.Text = "" Else Gridl.Text - Datal.Recordset.Fields(I).Value
End If
CellWidth = TextWidth(Gridi.Text) + 100
If CellWidth > Gridi.ColWidth(I) Then
Gridi.ColWidth(I) = CellWidth
End If Next I
Datal.Recordset.MoveNext Loop End Sub
```

Процедура Commandl_Click обеспечивает очистку комбинированного списка и обращение к стандартному окну «Открыть файл» с заданным расширением (*.MDB) (рис. 5.10). После выбора файла происходит обращение к процедуре Open DataBase, в которую передается имя выбранного файла (CommonDialog.FileName).

Процедура OpenDataBase обеспечивает подключение к базе данных через элемент управления Data (по умолчанию оператором Datal.Connect = «» выбирается Формат MS Access, имя базы данных определяется передаваемым в процедуру значением Datal. DatabaseName = DataFile) и заполнение комбинированною списка именами таблиц (цикл). Из этого списка можно выбрать нужную таблицу (рис. 5.11).

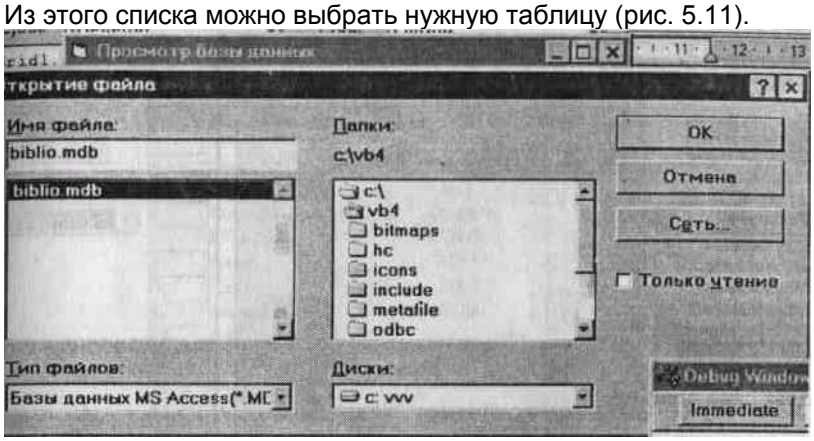


Рис 5 10

После выбора таблицы, процедура обрабогки события Combo l_KeyPress вызывает процедуру FillGrid заполнения сетки, в которую передается имя выбранной таблицы Combol.Text.

В процедуре FillGrid инициализируется множество записей (оператор Datal.RecordSoui-ce = TableName) и определяется количество колонок таблицы (оператор Gridi.Cols = Datal.Database(TableName).Fields.Count), первая строка сетки (Gridl.Row = *0)* заполняется названиями колонок выбранной таблицы (операторы

For I = 0 To Datal.Database(TableName).Fields.Coiint-l

Gridl.Col = 1
Gridl.Text = Data].Database^TableName).Fields(I).Name
Gridi.ColWidth(I) = TextWidth(Gridl.Text) + 100
Next I)

В этих операторах для задания ширины каждой колонки используется обращение к функции TextWidth, которая возвращает длину выводимого текста.

Количество записей и соответственно количество строк сетки определяется операторами
Datal. Refresh
Datal .Recordset.Move Last
Gridl.Rows = Datal. Recordset. RecordCount + 1 (первая строка заполнялась названиями полей).

Возврат на первую запись и к первой строке сетки производится операторами Datal. Recordset. MoveFirst и Gridl.Row = 0.

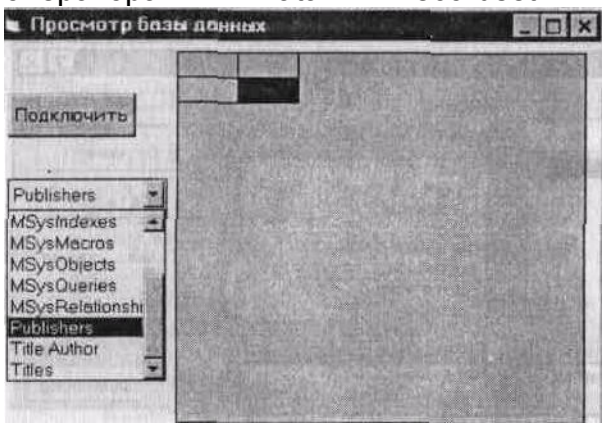


Рис 511

Цикл Do-Loop обеспечивает движение по записям (оператор Data I. Recordset. MoveNext) и выполняется пока функция EOF имеет значение False (т.е. до последней записи). На каждом шаге цикла производится переход к новой строке (Grid I. Row = Grid 1. Row + 1), для которой выполняется цикл (Gridl.Col = I) заполнения всех ее ячеек. Если поле таблицы базы данных не содержит значения (проверяется функцией IsNull(Datal. Recordset. Fields(I). Value)), то соответствующая ячейка сетки заполняется пустой строкой.
Последние операторы обеспечивают согласование ширины ячеек таблицы с максимальной возможной длиной поля записи (операторы
CellWidth = TextWidth(Gridl.Text) + 100 • If CellWidth > Gridl.ColWidth(I) Then
Gridl.ColWidth(I) = CellWidth).
Заполненная сетка показана на рисунке 5.12.

5.4. СОЗДАНИЕ БАЗ ДАННЫХ

Создание новой базы данных может быть выполнено программно или с помощью специальной встроенной подсистемы Data Manager, позволяющей в диалоговом режиме создавать и модифицировать базы данных механизма Jet. Подсистема Data Manager запускается из раскрывающегося меню Add-Ins Главного меню Visual Basic.

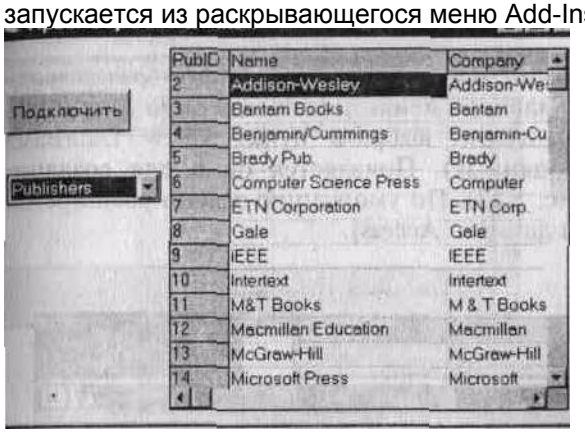


Рис 5 12

Рассмотрим использование подсистемы Data Manager на примере создания

базы данных по товарам на складе и их поставщикам. Таблицы База данных включает две таблицы (таблица товаров на складе, таблица поставщиков).

Таблица товаров на складе

Номер товара	Номер постав-щика	HilJBBUHe roB.tpJ	Стоимость, руб /нiт	Количество на СКЛаДе
1	2	Магнитола	150000	20
2	2	Приемник	200000	5
3	1	Плеер	350000	15
4	3	Кофеварка	175000	34
5	3	Элсктрочаини к	120000	57
6	1	CD — Плеер	750000	8

Таблица поставщиков

Номер постав-шика	Название фирмы	Город	Адрес	Телефон
1	ЗАО «Посредник»	Тверь	ул. Космонавтов 12	12345
2	«Импульс»	Москва	Пр Мира, 5	2334455
3	«Старт»	Серпухов	ул Зеленая 11	345678

Таблица товаров на складе связана с таблицей поставщиков через поле «Номер поставщика» (внешний ключ для таблицы товаров).

После щелчка мыши по пункту Data Managei раскрывающегося меню Add-Ins Главного меню появляется окно Data Manager, в меню «Файл» нужно выбрать пункт «New DataBase» (создание новой базы данных). Появляется окно для создания файла базы данных (рис. 5.13). По умолчанию задано расширение файла .mbd (файл базы данных Access).

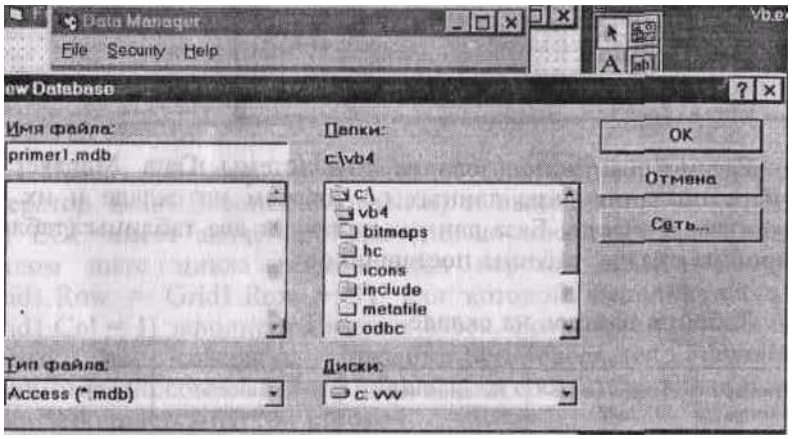
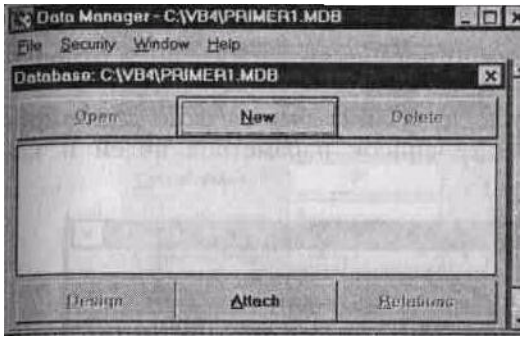


Рис 5 13

После задания имени файла появляется окно проектирования для задания таблиц, полей таблиц, отношений и индексов (рис. 5.14).

Кнопка «New» используется для создания новой таблицы, «Open» — открытие существующей таблицы для ввода данных, «Delete» — удаление

таблицы, «Design» — задание параметров полей таблицы, «Attach» — для подключения используемой СУБД (по умолчанию — СУБД Access). «Relations» — построение отношений между таблицами.



Щелчок мыши по кнопке «New» открывает окно проектирования таблицы для ввода имен полей и задания их свойств (рис. 5.15).

Рис 5 14

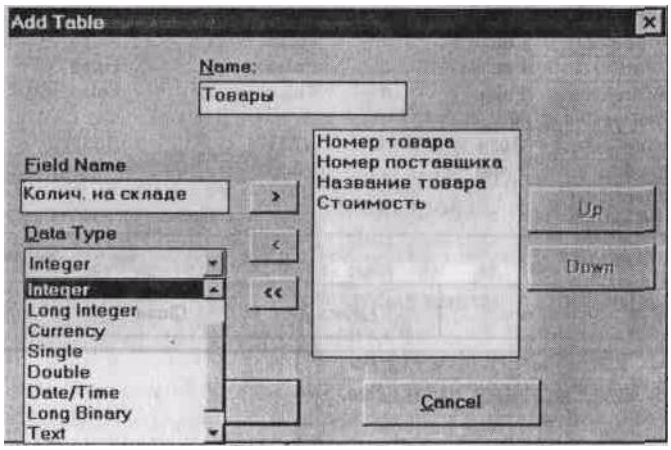


Рис 515

В поле Field Name вводятся имена полей таблицы и в раскрывающемся киже меню задается их тип (при задании типа полей «Номер товара» и «Номер поставщика» следует выбирать Long Integei, так как эти поля будут использоваться в качестве счетчика для первичного ключа). Стрелки « > » и « < » обеспечивают соответственно добавление введенных имен полей или их удаление. Кнопки «Up» и «Down» позволяют изменять последовательность полей таблицы. Аналогично может быть введена таблица поставщиков.

Когда таблицы созданы и выбрана одна из них, активизируются кнопки Design (проекгирование таблиц). Open (открытие таблиц для ввода или редактирования информации). Delete (удаление таблиц) и Relations (отношения между таблицами). Проектирование таблиц должно предшествовать заданию отношений между таблицами и вводу данных

При щелчке по кнопке Design открывается окно редактирования (рис 5 16), содержащее список параметров полей и командные кнопки

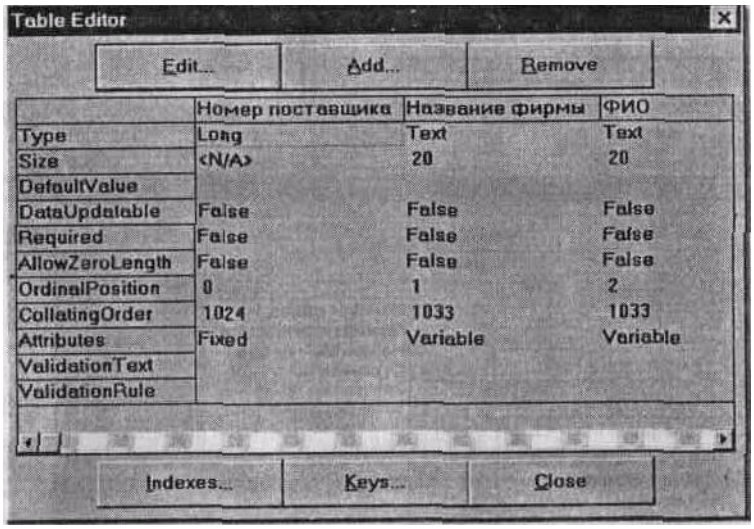


Рис 5 16

Для выбранного поля таблицы (на рисунке выбрано поле «номер поставщика») при щелчке по кнопке «Edit» открывается окно редактирования параметров (рис 5 17). На рисунке для поля таблицы «Номер поставщика» задан параметр «Counter» (Счетчик) Это поле является ключевым и должно иметь уникальные значения При задании параметра «Counter», СУБД автоматически поддерживает уникальность этого поля, увеличивая значение счетчика при вводе новой записи Другие параметры имеют следующий смысл DefaultValue — значение по умолчанию, DataUpdatable — возможность изменения. Required — обязательность ввода, AllowZeioLenght — возможность задания строки нулевой длины, ValidationText и ValidationRule — правила корректности, Fixed или Variable Length — фиксированной или переменной длины.

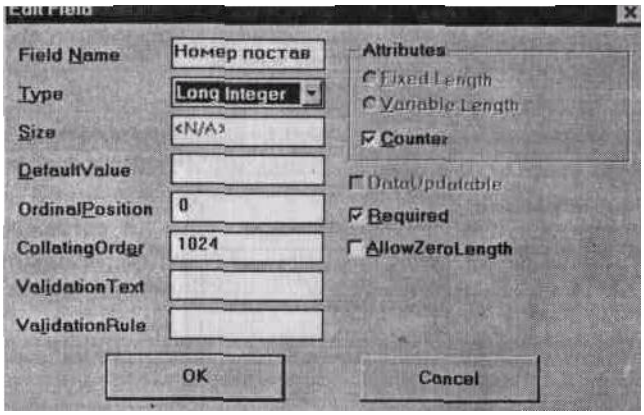
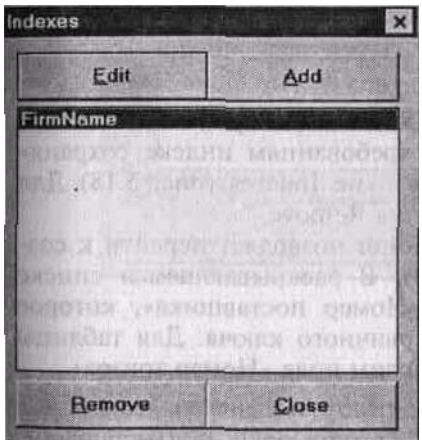


Рис 5 17

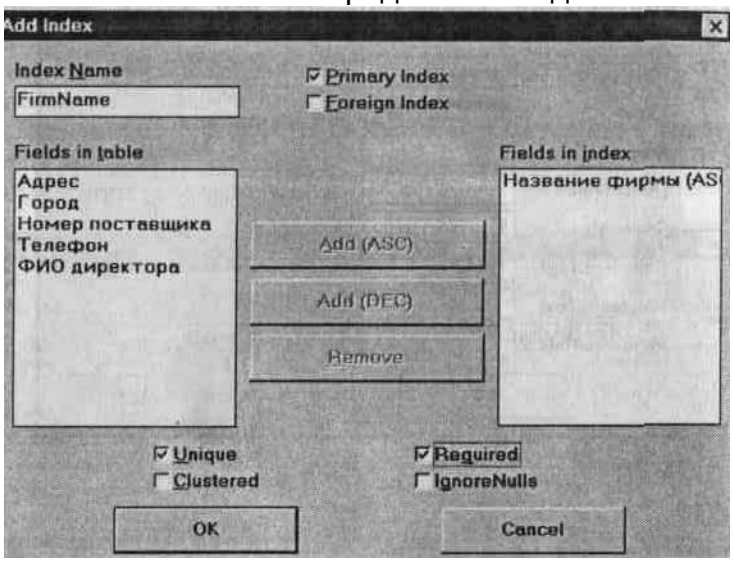


Кнопка «Indexes..» позволяет перейти к созданию и редактированию индексов таблицы (рис 5 18) Для создания или

добавления нового индекса нужно щелкнуть по кнопке «Add » открывается окно для создания индекса (рис. 5.19).

Рис 5 18

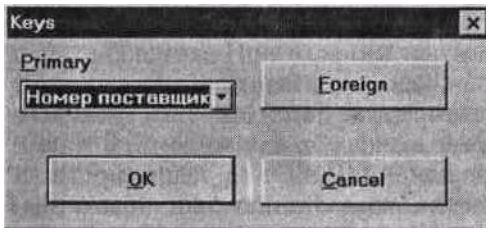
В окне нужно задать имя индекса. Затем выбрать поля таблицы («Fields in tdble»), которые включаются в индекс. Выбранные поля кнопками Add(ASC) или Add(DEC) (в зависимости от того, упорядочиваются записи по возрастанию или убыванию) Добавляются в индексный список («Fields in index»). Кнопка Remove позволяет удалить поле индекса Индикаторы в окне позволяют также определить индекс как первичный (Plimary Index)



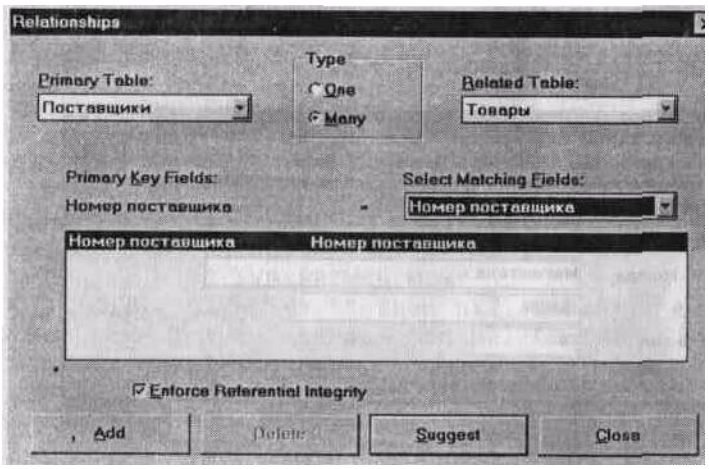
или внешний (Foieign Index), уникальный (Unique) или кластерный (Clustered). Соответствующий требованиям индекс сохраняется кнопкой ОК и отображается в окне Indexes (рис. 5.18) Для удаления индекса используется кнопка Remove

Кнопка «Keys. .» окна Table Editor позволяет перейти к созданию ключей

таблицы (рис. 5 20). В раскрывающемся списке полей таблицы выбирается поле «Номер поставщика», которое будет использоваться в качестве первичного ключа. Для таблицы «Товары» первичным ключом определим поле «Номер товара».



Кнопка «Relations...» окна Table Editor позволяет перейти к созданию отношений между таблицами. В окне Relations» (рис. 5.21) нужно выбрать из раскрывающегося списка первичную таблицу (Prymary Table) и связанную таблицу (Related Table) и установить связь между ними В рассматриваемом примере первичной таблицей является таблица поставщиков, поле «Номер поставщика» которой используется в качестве внешнего ключа для таблицы «Товары». Ключевое поле первичной таблицы (Номер поставщика) оюбражается на поле окна Поле для связи связанной таблицы выбирается в раскрывающемся списке («Select Matching Fields). Связующие поля отображаются в окне.



Вид отношения «один ко многим» (один поставщик может поставлять различные товары) задается переключателем «Many». Переключатель «One» определяет отношение «один к одному».

Флажок «Enfoice Refeiential Integrity» позволяет установить, нужно ли обеспечивать целостность отношении (средства СУБД автоматически проверяют целостность базы данных при модификации информации) Кнопка «Add» вводит заданное отношение в базу данных.

Таблицы и связи между ними созданы и отображаются в списке таблиц базы данных (рис. 5 22). При выборе таблицы становятся доступны кнопки «Open», «Design», «Delete», «Relations» Кнопка «Open» открывает выбранную таблицу для ввода или редактирования данных (рис. 5.23). Данные приведенных выше таблиц примера могут быть введены в базу

данных.

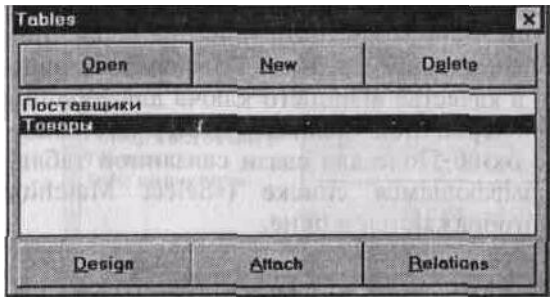


Рис. 5.22

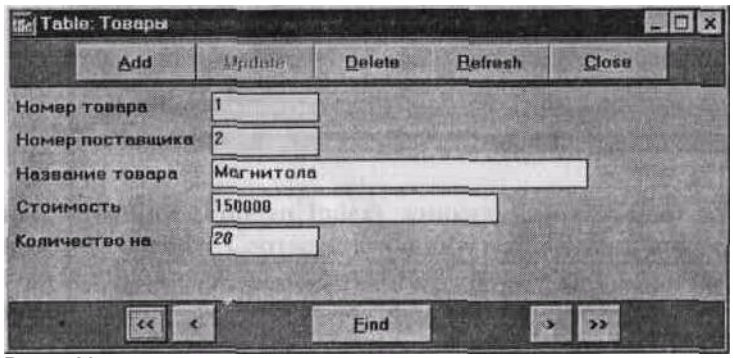


Рис. 5.23

К недостаткам подсистемы Data Manager можно отнести ограниченные возможности по заданию отношений между таблицами. Подсистема позволяет задать отношения только для ключевых полей «Counter» (Счетчик) и не позволяет задать составных ключей. Базы данных сложной структуры целесообразней создавать непосредственно в СУБВ Access, которая имеет развитые и очень удобные средства проектирования баз данных.

Приведенная база данных может быть создана и программно. Приводится текст программы для создания таблиц примера. задания ключей, индексов и отношений между таблицами.

Текст программы:

•

```
Global Const DB_LANG_GENERAL =
";LANGID-Ox0809;CP=1252;COUNTRY=0' Global Const DB_BOOLEAN = 1 Global Const DB_BYTE = 2 Global Const
DB_INTEGER = 3 Global Const DB_LONG = 4 Global Const DB_CURRENCY = 5 Global Const DB_SINGLE = 6 Global
Const DB_DOUBLE = 7 Global Const DB_DATE = 8 Global Const DB_TEXT = 10 Global Const .DB_LONGBINARY =11
Global Const DB_MEMO = 12
Private Sub Form_Click()
CreateNewDB End Sub
Sub CreateNewDB ()
Dim Db As Database, Dbname As String Dbname = "C:\VB\PRIMER.MDB" If Dir(Dbname) <> "" Then Kill Dbname Set
Db = CreateDatabase(Dbname, DB_LANG_GENERAL) NewProduktTabie Db NewProviderTable Db Relations Db
MsgBox «Ваша база данных , " & UCase(Dbname) & ", создана» End Sub
Sub NewProduktTabie(D As Database)
Dim Td As New TableDef, Fld() As New Field Dim Idx() As New Index, I As Integer ReDim Fld(I To 5), Idx(I To 2)
Td.Name = «Товары» ' Имя таблицы.
' Создание полей таблицы. Fid(I).Attributes = DB_AUTOINCRFIELD For I = 1 To 5 ' Задание свойств полей таблицы.
Fld(I).Name = Choose(I, «Номер_товара», «Номер_поставщика», «Название_товара», «Стоимость»,
«Количество»)
Fid(I).Type = Choose (I, DB_LONG, DB_LONG, DB_TEXT, DB_CURRENCY, DB INTEGER) Fld(I).Size = Choose(I,
4, 4, 30, 4, 4) Td.Fields.Append Fid(I) Next I ' Создание индексов. Idx(I).Name = "PrimaryKey" Idx (1) . Fields
```

```
= «Номера/товара» • Idx(I).Primary = True
Idx(1).Unique = True              • Idx(2).Name = “Name” Idx(2).Fields = «Название_товара» For I = 1 To 2
Td.Indexes.Append Idx(I) Next I
D.TableDefs.Append Td ‘ Создание таблицы. End Sub
Sub NewProviderTable(D As Database)
Dim Td As New TableDef, Fid() As New Field
Dim Idx() As New Index, I As Integer
ReDim Fld(I To 5), Idx(I To 2)
Td.Name = «Поставщики»  ' Имя таблицы. ‘ Создание полей таблицы.
For I = 1 To 5 'Задание свойств полей таблицы.
Fld(I).Name  =  Choosefl,  «Номер_поставшика», «Название_фирмы», «Город», «Адрес», «Телефон»)
Fld(I).Type  =  Choose(I,  DB_LONG,  DB_TEXT, DB_TEXT, DB_TEXT, DB_TEXT)
Fld(I).Size = Choose(I, 4, 30, 15, 30, 10)
Next I
Fid(I).Attributes = DB_AUTOINCRFIELD
For I = 1 To 5
Td.Fields.Append Fld(I)
Next I    ‘ Создание инде^ов.
Idx(I).Name == “PrimaryKey”
Idx(1).Fields = «Номер_поставщика»
Idx(1).Primary = True
Idx(1).Unique = True
Idx(2).Name = “Fi rmNarne”
Idx(2).Fields = «Название_фирмы»
For I = 1 To 2
Td.Indexes.Append Idx(I)
Next I
D.TableDefs.Append Td ‘ Создание таблицы. End Sub
Public Sub Relations (D As Database) Dim MyField As Field, MyRelation As Relation
```

Set MyRelation = D.CreateRelation(“MyRelation”) MyRelation.Table = «Поставщики» MyRelation.ForeignTable = «Товары» Set MyField =

MyRelation.CreateField(«Номер_поставщика») MyField.ForeignName = «Номер_поставщика» MyRelation.Fields.Append MyField D.Relations.Append MyRelation End Sub

Константы в программе скопированы из входящего в состав Visual Basic файла DATACONS.TXT. Обращение к процедуре CreateNewDB создания базы данных помещено в процедуру обработки события Form_Click, из которой производится обращение к процедуре NewProduktTable, создающей таблицу товаров на складе, к процедуре NewProviderTable, создающей таблицу поставщиков товаров и к процедуре Relations, создающей отношения между таблицами. Следует обратить внимание на удобство использования встроенной функции Choose для задания свойств полей таблиц.

5.5. ИСПОЛЬЗОВАНИЕ ЯЗЫКА SQL ДЛЯ

СОЗДАНИЯ И РАБОТЫ С БАЗАМИ ДАННБ1Х

Язык структурированных запросов (SQL) является стандартным средством для работы с базами данных и может использоваться как для интерактивной работы с базами данных, так и включаться в языки программирования. Применительно к Visual Basic SQL позволяет:

- создавать, модифицировать или удалять таблицы в базе данных Access;
- создавать или удалять индексы для таблиц в базе данных Access;
- вставлять, удалять или модифицировать записи таблиц;
- получать сводную информацию о данных в таблицах (число записей, суммы, средние,

минимальные, максимальные значения и др.);

- поиск данных в одной или более таблицах по запросу. Язык SQL используется совместно с элементом управления DATA или с объектами доступа к данным (DAO) механизма Jet. Объект Database содержит объекты доступа к данным TableDef, определяющие таблицы, объекты Field, определяющие поля, объекты Index, определяющие индексы и др.

В ранее рассмотренных параграфах данной главы через элемент управления Data подключалась одна таблица (свойству Re-cordSource элемента управления Data присваивалось значение имени таблицы). При использовании языка SQL запрос вводится в свойство RecordSource. В случае использования элемента управления Data, это позволяет работать с информацией из нескольких таблиц.

Следует отметить, что каждое выполнение оператора запроса изменяет набор записеи. В связи с этим необходимо применять метод Refresh для элемента управления Data после каждого присвоения SQL-запроса свойству RecordSource.

Рассмотрим операторы и использование языка SQL для работы с базами данных. Для поиска информации в базе данных используется оператор SELECT. Синтаксис оператора следующий:

SELECT *список имен полей* FROM *список имен таблиц* где SELECT, FROM — ключевые слова;

список имен полей — список имен полей, которые выбираются из одной или нескольких 1аблиц. Для выбора всех полей можно использовать символ «*», вместо перечисления имен всех полей. Если имя поля таблицы содержит пробел, то это имя должно заключаться в квадратные скобки;

список имен таблиц — список имен таблиц, из которых производится выбор.

Имена полей и таблиц нечувствительны к регистру клавиатуры.

' Примеры:

SELECT * FROM *Titles* — выбор всех полей из таблицы Titles рассмотренной выше базы данных по библиографии по Visual Basic.

SELECT *Title, /Year Published/* FROM *Titles* — выбор полей заголовков (Title) и года издания (Year Published) из таблицы Titles. Имя поля года издания берется в квадратные скобки (имеет пробел).

Для поиска информации, соответствующей некоторому условию, используется дополнение к оператору SELECT — WHERE, которое имеет следующий синтаксис:

SELECT *список имен полей* FROM *список имен таблиц* WHERE *условие* где *условие* определяет критерии поиска информации.

В условии используются имена полей, операции сравнения (<> <=, =, >, >=, <>) и специальные операции сравнения IN, LIKE, BETWEN. Эти операции могут объединяться с помощью логических операций и задавать сложные

условия поиска информации.

Примеры:

SELECT /Year Published/, Title FROM Titles

WHERE */Year Published/ > 1991* определяет выбор названий книг, год выпуска которых позже 1991.

SELECT I Last Name], /First Name/ FROM Employees

WHERE */Last Name/ = 'King'* определяет выбор полей имен и фамилий из таблицы служащих, фамилии которые совпадают с фамилией King.

Операция IN сравнивает содержимое поля со списком значений, определяющих критерий поиска информации.

Примеры:

SELECT */Year Published/, Title* FROM *Titles*

WHERE */Year Published/* IN *(1995, 1996)* определяет выбор книг, опубликованных в 1995 и в 1996 гг.

SELECT /Last Name/, /First Name/, City FROM Employees

WHERE *City In ('Interlaken ', 'New York', 'Frankfurt')* определяет выбор служащих, живущих в городах Interlaken, New York или Frankfurt.

Операция LIKE сравнивает содержимое поля со значением образца. Для записи образца используются строковые константы, символы шаблона и списки диапазона символов.

Символы шаблона следующие: *, ?, #. * — соответствует цепочке символов, ? — соответствует одному символу, # — соответствует одной цифре.

Примеры:

R* — возможные результаты поиска right, Roza.
A? — возможные результаты поиска and, any.
12345# — возможные результаты поиска 123455. 123457.

Список диапазона заключается в квадратные скобки и первый и «последний символы диапазона отделяются дефисом (-). Диапазон задается в возрастающем порядке.

Примеры:

SELECT */Last Name/, /First Name/* FROM *Employees* WHERE */Last Name/* Like *'S*'* определяет выбор служащих, фамилии которых начинаются с буквы S. SELECT *Author* FROM *Authors* WHERE *Author* LIKE *fA-K/* определяет выбор авторов, фамилии которых начинаются с букв от А .до К.

SELECT *Title* **FROM** *Titles*

WHERE *Title* LIKE *«database*»* AND *[Year Publisher/ = 1996* определяет

выбор названий, в которых присутствует слово «database» и выпущенных в 1996 году.

Операция BETWEEN проверяет принадлежность значения поля диапазону значении и является включающим значением (выбираются записи, содержащие поле со значением, равным границе диапазона). Границы значений объединяются операцией AND.

Примеры:

SELECT lOnler IDI, I Order Dale/

FROM *Orders* WHERE */Order Date/ Between # I-1-94»* And *#6-30-94#* определяет выбор документов первой половины 1994 г. *C*

SELECT */Last Name/, Salary* FROM *Employees* '

WHERE *[Last Name] Between 'Lon'* And To/' определяет выбор зарплаты служащих, начальные буквы фамилий которых, расположенные в алфавитном порядке, находятся в диапазоне 'Lon' и 'ToГ.

Для определения порядка, в котором представляются результаты поиска записей, используется дополнение ORDER BY, синтаксис которого следующий:

SELECT ***список имен полей*** **FROM** ***список имен таблиц*** **WHERE** ***условие*** **ORDER BY** ***имя поля*** **[DESC]** ***[имя поля*** **[DESC]]** где ***имя поля*** — поле, по которому производится упорядочение. Опция **DESC** устанавливает обратный порядок сортировки. Упорядочение может вестись по нескольким полям (сортировка, например, по фамилии, а затем по имени)

Пример:

SELECT *Title* FROM *Titles* WHERE *Title* LIKE

*«*database*»* AND *[Year Publisher] = 1996* **ORDER BY** *Title* определяет выбор названий книг, в которых присутствует слово «database» и выпущенных в 1996 г., и упорядочивает названия в алфавитном порядке.

При работе с несколькими таблицами, каждое из рассмотренных дополнений условий выбора может быть применено для любой из таблиц. В общем случае при формировании запроса для

нескольких таблиц указывается таблица, в которой ведется поиск полей и связь между таблицами. Синтаксис запроса для нескольких таблиц следующий:

SELECT список имен полей FROM список имен таблиц, список связей

где ***список связей*** определяет, как таблицы в списке имен таблиц связаны между собой. В частности для задания связи используется рассмотренное дополнение WHERE (WHERE *имя_табмщы1.1шя_поля1 = 1шя_таблицы2.имя_поля2).*

Примеры:

- SELECT *Tities.Title, Publishers. [Company Name}* **FROM** *Titles. Publishers* WHERE *Titles. Pubid = Publishers. Pubid* определяет выбор названий книг (Title) из таблицы названий (Titles) и названий издательств (Company Name) из таблицы издательств (Publishers). Дополнение WHERE определяет связь между таблицами (выбирается название книги и номер издательства из таблицы Titles, по номеру издательства в таблице издательств Publishers находится название издательства).

SELECT Tities.Title. Authors. Author, Publishers. [Company Name]

FROM *Titles, Authors. Publishers* WHERE *Titles.auJd = Authors. au_id* AND *Tilles.pubid = Publishers. Pubid* определяет выбор из всех трех таблиц полной информации по книге (название, автор, издательство).

В целом, язык SQL имеет широкий набор средств для организации различных запросов, о которых можно более подробно узнать из встроенной системы помощи Visual Basic.

Пример программы.

Модернизируем приведенную выше программу для просмотра таблиц баз данных для ввода SQL-запросов и отображения полученной информации. Добавим на форму текстовое окно для ввода SQL-запроса (нажатием клавиши ввода запрос вводится). Полученная информация динамически (соответственно запросу) формируется и отображается элементом управления Grid.

Форма показана на рис. 5.24, пример ввода запроса и таблица с информацией — на рис. 5.25.

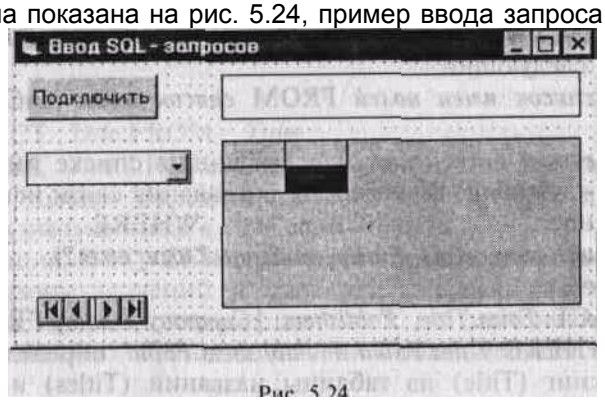


Рис. 5.24

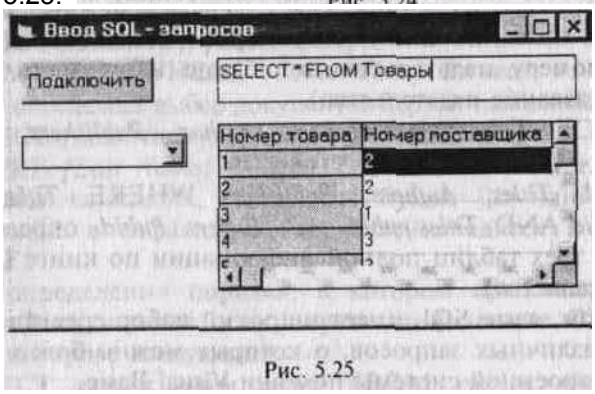


Рис. 5.25

Текст программы:

```
Option Explicit
Const vbOFNPATHMUSTEXIST =      &H800&
Const vbOFNFILEMUSTEXIST =      S.H1000&
```

```
Private Sub Coinmandl_Click () Combol.Clear
CommonDialogI.DefaultExt = “MDB” CommonDiaiogI.FiieName = “”
CommonDialogI.Filter = «Базы данных Access(*.MDB)|*.MDB”
CommonDialogI.Flags = vbOFNPATHMUSTEXIST
CommonDialogI.Action = 1
If CommonDialogI.FileName = “” Then Exit Sub
OpenDataBase CommonDialogI.FileName End Sub
Public Sub OpenDataBase(ByVal DataFile As String)
Dim I As Integer
Datal.Connect = “”
Datal.DatabaseName = DataFiie
Datal.Refresh
For I = 0 To Datal.Database.TableDefs.Count - 1
Combo 1.AddItem Datal.Database.TableDefs(I).Name
Next
Combol.Text = “”
End Sub
Public Sub FillGrid(ByVai Zapros As String)
Dim I As Integer, CellWidth As Integer
Datal.RecordSource = Zapros
Debug.Print Zapros
Datal.Refresh
GridI.Cols = Datal.Recordset.Fields.Count
Debug.Print GridI.Cols
GridI.Row •= 0 ‘
For I = 0 To Datal.Recordset.Fields.Count - 1
GridI.Coi = I
GridI.Text = Datal.Recordset.Fields(I).Name GridI.ColWidth(I) = TextWidth(Gridi.Text) + 100 . Next Datal.Refresh
Datal.Recordset.MoveLast
GridI.Rows = Datal.Recordset.RecordCount + 1 Datal.Recordset.MoveFirst GridI.Row = 0
Do While Not Datal.Recordset.EOF GridI.Row = GridI.Row + 1
For I = 0 To Datal.Recordset.Fields.Count - 1 GridI.Coi = 1 If IsNuil(Datal.Recordset.Fields(I).Value) Then
GridI.Text =• “” Else
GridI.Text = Datal.Recordset.Fields(I).Value End If
CellWidth =.TextWidth(Gridi.Text) + 100 If CellWidth > Gridi.ColWidth(I) Then GridI.ColWidth(I) = CeilWidth
End If Next I
Datal.Recordset.MoveNext Loop End Sub
Private Sub TextI_KeyPress(KeyAscii As Integer)
If KeyAscii = 13 Then FillGrid TextI.Text
End If End Sub
```

ЗАКЛЮЧЕНИЕ

Автор надеется, что прочитавший эту книгу до конца и самостоятельно выполнивший на компьютере приведенные в ней примеры получил достаточные знания и навыки программирования на Visual Basic, позволяющие самостоятельно создавать программные системы.

Вы получаете в свои руки относительно простой, но очень эффективный инструмент для создания программ, работающих в операционной системе Windows. Эти программы могут иметь любую прикладную направленность: от простых баз данных для личного использования на работе и дома до профессиональных систем автоматизации производственной, торговой, банковской, страховой и других видов деятельности.

Вы делаете первые шаги, чтобы стать специалистом в области информационных технологий. Технологий, которые прокладывают дорогу в XXI век и в значительной степени определят дальнейшее развитие человечества.

Желаю Вам успехов на этом сложном, но очень перспективном пути.