В. Л. Быков

Основы программирования на языке *Visual Basic 6.0*

УДК 681:3.06:800.92
ББК 22.183.492
Б 95

Рецензенты:
профессор кафедры “Вычислительные методы и программирование” Белорусского государственного университета информатики и радиоэлектроники, доктор физико-математических наук Колосов С. В.

доцент кафедры “Математическое обеспечение ЭВМ” факультета прикладной математики Белорусского государственного университета, кандидат педагогических наук Белецкая Л. В.

В. Л. Быков
Б 95 Основы программирования на языке Visual Basic 6.0: пособие – Брест: БГТУ, 2002.

ISBN

Учебное пособие разработано в соответствии с рабочей программой по дисциплине “Информатика” Брестского государственного технического университета и предназначено для студентов ВУЗов. Рассматриваются общие сведения о языке программирования Visual Basic 6.0 и приемы работы. Учебный материал разбит на семь тем: общие сведения о языке программирования; форма и ее свойства; программирование в VB; графические средства VB; дополнительные средства для разработки интерфейса; работа с файлами данных; проверка и обработка пользовательского ввода. Обработка ошибок. Учебное пособие содержит большое число примеров и задания для закрепления материала. Последовательное изучение тем позволит читателю, переходя от простого к сложному, приобрести первичные навыки для самостоятельной разработки простых, но функционально полных приложений.

ISBN

УДК 681:3.06:800.92
ББК 22.183.492

Содержание

Введение

Конец 20 столетия был отмечен бурным развитием вычислительной техники и средств передачи информации по компьютерным сетям. Персональный компьютер стал неотъемлемой составной частью рабочего места инженера, специалиста любого профиля, руководителя организации, учреждения, надежным средством оперативного сбора и обработки информации.

Изменение технических возможностей вычислительной техники способствовало разработке и новых систем программирования, позволяющих инженеру - не программисту самостоятельно разрабатывать пользовательские программы достаточно высокого качества для решения текущих задач. Одним из таких языков программирования высокого уровня является язык Visual Basic (VB).

К настоящему времени существует уже несколько версий языка программирования Visual Basic: VB1, VB3, VB4, VB5 и VB6.

Время появления VB относят к 1991 году. До появления VB разработка приложений Windows была намного тяжелее процесса создания приложений для DOS. Программисты должны были позаботиться буквально обо всем, например, о работе с мышью, обработке событий меню, и даже отслеживать, щелкнул пользователь один либо два раза мышью в конкретном месте экрана. С появлением VB ситуаци[1]я значительно изменилась. VB позволяет разрабатывать сложные приложения Windows и Windows NT за меньший период времени. Значительно улучшились возможности по поиску и локализации ошибок в

[1] Программы, работающие в среде Windows принято называть приложениями

разрабатываемых программах. С VB программирование в Windows не становится более эффективным, но ***оно становится более простым***.

Visual Basic 6.0 опирается на язык высокого уровня – Basic. Синтаксис и операторы, этого языка по структуре близки к алгоритмическому языку программирования, изучаемого в школах, что позволяет гораздо легче усваивать его начинающим пользователям по сравнению с такими профессиональными языками программирования как Pascal, Delphi, C. Актуальность изучения языка VB обусловлена не только тем, что он достаточно прост в изучении и обладает большими возможностями по разработке прикладных программ, но также тем, что он используется для написания макросов во всех популярных приложениях Windows: Word Basic – для текстовых редакторов; Basic for Application – для электронных таблиц; Visual Basic – для баз данных.

Visual Basic сочетает в себе возможности транслятора интерпретирующего типа (интерпретатора) и компилирующего типа (компилятора).

Как интерпретатор VB позволяет запускать программы непосредственно из среды разработки. Интерпретатор принимает непосредственное участие в выполнении программы. Он разбирает каждую строку, проводит анализ ошибок и выдает сообщение пользователю при их обнаружении. После устранения некритических ошибок он продолжает работу. Большая часть времени процессора тратится именно на разбор и трансляцию каждой строки программы при каждом ее повторном исполнении. Поэтому программы с трансляторами-интерпретаторами работают значительно медленнее, чем скомпилированные программы. Но у трансляторов-интерпретаторов больше возможностей в организации диалогового режима отладки программ. Visual Basic 6.0 использует новую технологию Threaded-p-Code, при которой каждая введенная строка текста программы преобразуется в промежуточный код – Threaded-p-Code. Это не совсем машинный код, но такой код выполняется быстрее, чем при работе с обычным интерпретатором. Эта технология имеет и другие преимущества: во-первых, VB сразу же проверяет синтаксис программы и выдает сообщение об обнаруженной ошибке; во-вторых, эта технология позволяет осуществлять поиск ошибок.

Как компилятор VB позволяет создавать EXE-файлы, правда не сразу при запуске из среды разработки, а с помощью команд меню. EXE-файлы выполняются под управлением операционной системы без участия транслятора. Процессы написания, отладки и выполнения программы разделены во времени.

Последняя версия VB – Visual Basic 6.0 имеет три различные редакции, рассчитанные на различные группы пользователей и отличающиеся набором возможностей и комплектом поставляемой документации: VB для начинающих, VB для профессионалов и промышленное издание VB. При этом синтаксис языка VB остается неизменным и не зависит от издания. Промышленное издание представляет собой расширенное издание для профессионалов и предназначено для разработчиков корпоративных систем. Набор элементов управле-

ния и средств этой версии позволяет разрабатывать не только простейшие программы, но и достаточно сложные клиент серверные приложения.

Visual Basic 6.0 предъявляет достаточно высокие требования к техническим характеристикам персонального компьютера. Для установки VB требуется не менее 10 Мбайт дисковой памяти, процессор не хуже 80486 или Pentium, 16 Мбайт памяти оперативного запоминающего устройства. Полная инсталляция самой мощной версии VB требует более 100 Мбайт дискового пространства.

Visual Basic работает в среде Windows (не ниже Windows 95) и позволяет создавать ***приложения*** – программы для работы в этой среде. При этом программы имеют похожий интерфейс и способы управления. В частности, VB позволяет добавлять к окнам поля ввода, меню, командные кнопки, переключатели, флажки, списки, линейки прокрутки, а также диалоговые окна для выбора файла или каталога. Программист может использовать сетку для обработки табличных данных, организовать взаимодействие с другими приложениями Windows и доступ к базам данных. Версии VB5 и VB6 могут поддерживать элементы ActiveX. ***ActiveX*** - компонент представляет собой технологию Microsoft для активизации работы с Internet и корпоративными Internet-сетями, причем данная технология может использоваться в обычных приложениях Windows для повышения продуктивности работы. Технология связана с вставкой и внедрением объектов. С помощью VB6 можно создавать 32-х разрядные приложения для работы в среде Windows 95/98 и Windows NT. Однако рассмотрение этих богатых возможностей VB выходит за рамки предлагаемого учебного пособия.

Учебное пособие разработано в соответствие с рабочей программой по дисциплине “Информатика” Брестского государственного технического университета и предназначено для студентов технических специальностей – не программистов. В учебном пособии рассматриваются общие сведения о языке программирования Visual Basic 6.0 и приемы работы в его среде. Учебный материал разбит на семь тем.

В первой теме описана среда разработки и основные приемы работы в ней; даны понятия об объектно-ориентированном программировании, классе, объекте, событии, свойстве, методе; рассмотрены основные свойства объектов.

Во второй теме рассмотрена форма и ее свойства. Рассмотрен вопрос о создании меню пользователя, контекстного меню, создании MDI-формы.

Третья тема посвящена вопросам программирования в VB. Рассмотрен порядок ввода и вывода данных. Даны понятия о переменных, константах, массивах, объявлении переменных, их типах, области видимости и времени жизни. Рассмотрены подробно процедуры и функции пользователя, основные структуры языка программирования. Значительное внимание уделено использованию массивов элементов управления и сетки для ввода данных и представления результатов вычислений.

В четвертой теме рассмотрены графические возможности VB: графические объекты и их свойства, графические методы. Большое внимание уделено масштабированию графических объектов и анимации.

Пятая тема посвящена рассмотрению дополнительных средств для создания интерфейса: флажков, переключателей, линеек прокрутки, списков и др.

В шестой теме рассматриваются файлы данных: текстовые файлы с последовательным и прямым доступом. Приведены примеры разработки баз данных с использованием указанных типов файлов.

В седьмой теме рассматриваются вопросы контроля ввода данных и обработки ошибок периода выполнения.

Учебный материал иллюстрируется достаточным количеством примеров. Для закрепления материала предлагаются контрольные вопросы и задания для самостоятельной работы. Ввиду того, что основное внимание в данном пособии уделено разработке пользовательского интерфейса, а не решению прикладных задач, примеры выбраны простые и понятные. Предполагается, что читатели знакомы с основами программирования и языком программирования Бейсик.

Последовательное изучение тем позволит читателям, переходя от простого к сложному, приобрести первичные навыки для самостоятельной разработки простых, но функционально полных приложений.

Автор выражает благодарность сотрудникам кафедры “Информатики и прикладной математики” Брестского государственного технического университета: заведующему кафедрой кандидату физико-математических наук, доценту Ракецкому В. М. и кандидату технических наук, доценту Ашаеву Ю. П. вычитавших пособие в рукописи, другим сотрудникам кафедры, оказавшим помощь в подготовке учебного пособия.

Особая благодарность рецензентам профессору кафедры “Вычислительные методы и программирование” БГУИР, доктору физико-математических наук Колосову С. В. и доценту кафедры МО ЭВМ факультета прикладной математики Белгосуниверситета кандидату педагогических наук Белецкой Л. В. чьи ценные замечания способствовали значительному улучшению структуры и содержания учебного пособия.

1. Общие сведения о языке программирования Visual Basic 6.0

1. 1. Среда разработки

1.1.1. Запуск программы

Запуск программ VB можно осуществить разными способами.

Если программа установлена на компьютере, то ее можно запустить через команду главного меню: ***Пуск \ Программы \ VB6***[2], либо через программу ***Проводник*** или ***Мой компьютер***.

При запуске программы через Проводник, Мой компьютер или меню кнопки Пуск на экране появляется окно запуска ***New Project***. В этом окне можно выбрать тип создаваемого приложения. Окно содержит три закладки:

New – начать с “нуля”;

Existing – работать с существующим проектом;

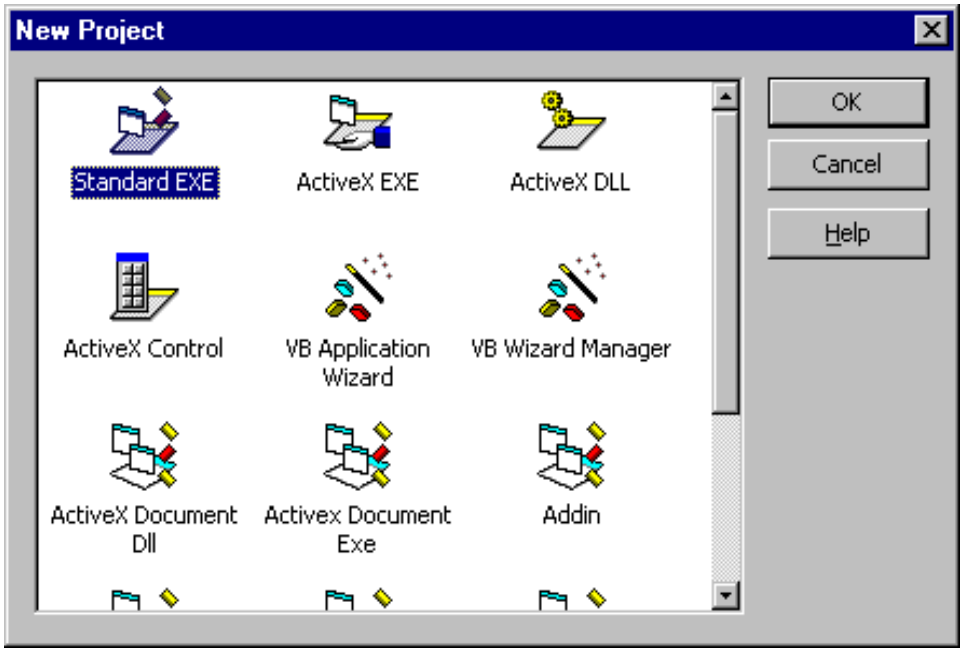


Рис. 1.1. Так выглядит диалоговое окно New Project при запуске из среды разработки

Recent – работать с проектом, разработка которого проводилась недавно.

[2] Указанная запись означает: щелкнуть по кнопке Пуск панели задач рабочего стола Windows, выбрать в меню кнопки Пуск пункт Программы, найти в его подменю пиктограмму приложения Visual Basic 6.0 и щелкнуть по ней дважды мышью. Таким способом будем обозначать последовательный выбор команд из меню верхнего уровня, вложенных подменю и опций окон диалога.

То же самое происходит при запуске новой программы из среды разработки, если выбрать команду ***File \ New Project,*** однако в этом случае окно не имеет закладок (рис.1.1).

Для начала работы с новым пректом следует выбрать закладку ***New*** и выбрать тип ***Standard.EXE***.

Можно выбрать также Мастера приложений – ***VB Application Wizard***. В процессе работы мастера создается почти готовое приложение с различными формами, соответствующей рабочей средой, меню, панелью инструментов и т.д., которое можно потом совершенствовать. Однако, пользоваться этим мастером рекомендуется после приобретения определенного опыта работы с Visual Basic. В ином случае пользователь получит больше проблем по совершенствованию проекта, чем пользы. Поэтому начинающему пользователю рекомендуется начинать работу со стандартного окна Standard.exe.

Выход из программы

Выход из VB осуществляется комбинацией клавиш ***Alt-F4*** или командой ***File\Exit.***

Чтобы составить представление о возможностях языка программирования Visual Basic 6.0 достаточно ознакомиться с назначением программ, представленных в окне **New Project**. При выборе нового проекта в этом окне Visual Basic создаст соответствующий ему шаблон.

- Standard.EXE – этот тип проекта используется для создания стандартной программы для Windows;
- ActiveX.EXE – это сервер автоматизации, который функционирует как часть многосвязных приложений;
- ActiveX.DLL – программа удаленной автоматизации, которая создается как динамически связываемая библиотека (DLL);
- ActiveX.Control – эта опция позволяет создавать собственные элементы управления. Их можно использовать в программах на Visual Basic и любых других ActiveX-совместимых приложениях;
- VB Application Wizard – мастер приложений;
- Data Project – тип проекта, который содержит оболочку приложения для работы с данными. Используя эту оболочку и настроив ее компоненты, можно быстро разработать программу для работы с базами данных;
- IIS Application – эта опция создает проект, который может использоваться на сервере Windows NT совместно с Internet Information Server (IIS) для работы в Web-ориентированной среде;
- Add-IN – эта программа привносит дополнительную функциональность в сам VB, например, можно добавить программу Visual Data Manager, которая используется для создания SQL - запросов;
- ActiveX Document DLL – этот тип проекта создает DLL, которая может использоваться приложениями, работающими в среде Microsoft Internet Explorer;
- ActiveX Document EXE - этот тип проекта создает приложение, работающее в среде Microsoft Internet Explorer;

- DHTML Application – эта опция создает заготовку приложения динамического HTML, которое может работать в Web-броузере.

1.1.2. Рабочее окно

После выбора типа проекта появляется рабочее окно программы (рис. 1.2). Рабочее окно представляет собой интегрированную среду разработки – интерфейс самого продукта Visual Basic. Эта среда может настраиваться с помощью диалогового окна, вызываемого командой ***Tools\Options.*** Рабочее окно предлагает целый набор инструментальных средств, которые можно использовать при создании программ.

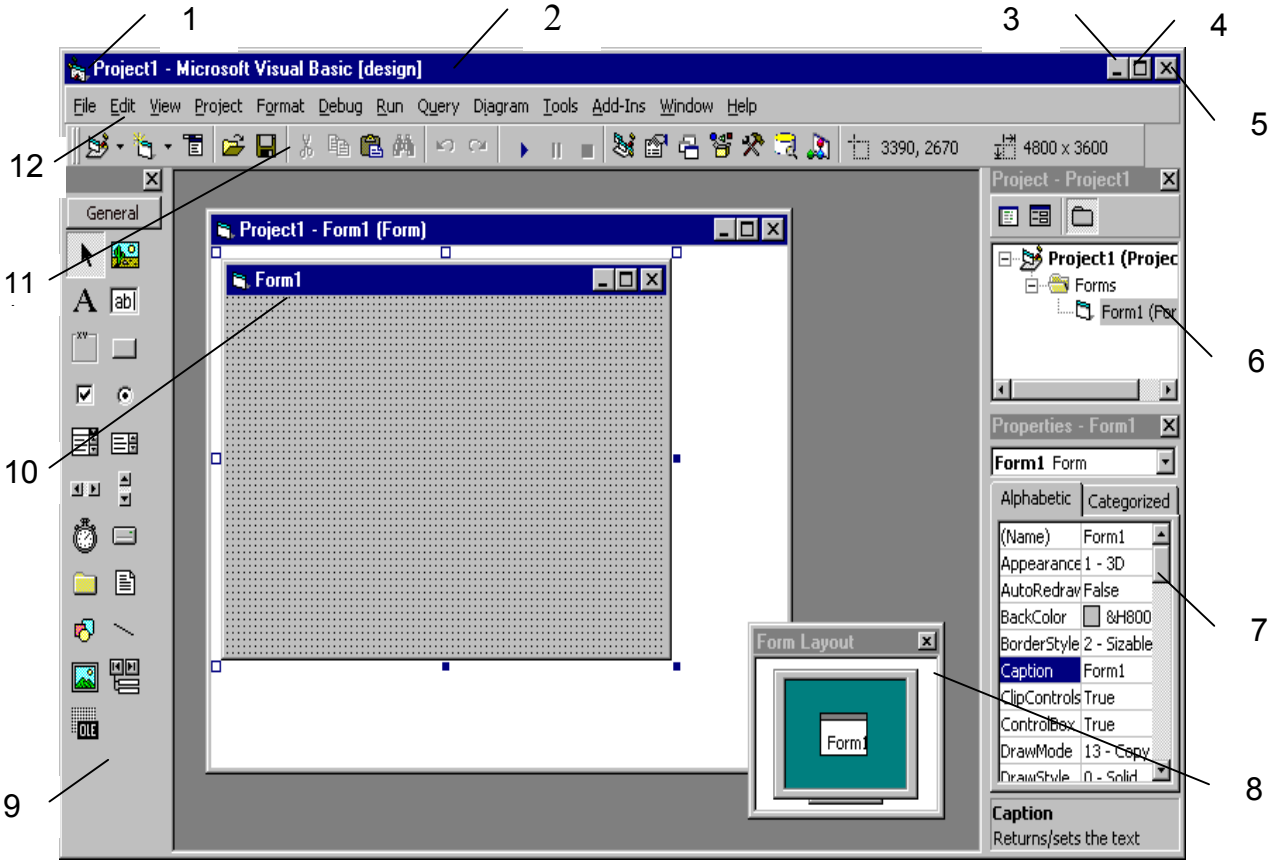


Рис. 1.2. Окно программы Visual Basic

1 – кнопка системного меню, 2 – заголовок, 3 – кнопка свертывания окна в пиктограмму, 4 – кнопка развертывания окна, 5 – кнопка закрытия окна, 6 – окно проекта, 7 – окно свойств, 8 – окно позиционирования формы, 9 – панель элементов управления, 10 – форма, 11 – стандартная панель инструментов, 12 – меню

Большинство элементов рабочего окна, характерны для приложений Windows: заголовок (2), меню (12), панели инструментов: стандартная панель инструментов -Toolbar (11) и панель элементов управления Toolbox (9).

Кнопка (1) в строке заголовка – кнопка системного меню. Кнопки (3, 4, 5) предназначены для свертывания, развертывания и закрытия окна, соответственно. После разветывания окна появляется новая кнопка для восстановления пер-

воначальных размеров окна. Внутри окна программы открываются вторичные окна: окно формы - ***Form1*** (10), окно проекта – ***Project-Project1*** (6); окно свойств – ***Properties*** (7); окно позиционирования формы ***Form Layout*** (8).

Меню (Menu)

Строка Меню содержит список команд, предназначенных для управления разработкой проекта, пункты меню содержат несколько уровней вложения:

File (Файл) - содержит команды для работы с файлами создаваемых приложений, загрузки, сохранения, вывода на печать.

Edit (Правка) - содержит команды редактирования, предназначенные для создания исходного текста программы, включая средства поиска и замены.

View (Просмотр) - обеспечивает доступ к различным частям приложения и среды разработки VB.

Project (Проект) - предназначен для добавления новых объектов VB к разрабатываемым проектам, добавления или удаления элементов управления на панель элементов управления, настройки свойств проекта.

Format (Формат) – дает доступ к различным настройкам элементов управления, размещенных на создаваемых программистом формах.

Debug (Отладка) – содержит средства, предназначенные для отладки программ или поиска ошибок.

Run (Выполнение) – служит для запуска и остановки программ непосредственно из среды разработки.

Tools (Инструменты) – обеспечивает доступ к работе с процедурами и меню внутри приложения. Этот пункт меню имеет важную команду ***Options,*** которая открывает одноименную диалоговую панель с закладками ***Options***, где настраивается практически вся среда разработки Visual Basic.

Add-Ins (Добавить в …)– дает доступ к инструментам, которые могут быть добавлены к окружению VB: мастера, ActiveX – элементы и другое.

Diagram (Диаграммы) – содержит средства для оформления диаграмм.

Window (Окно) – используется для работы с окнами в среде разработки.

Query – доступ к внешним базам данных.

Help – справочная система.

Выбор пунктов меню осуществляется мышью или клавишами. При управлении с помощью клавиатуры для входа в меню используется клавиша ***Alt***. Выбор пунктов меню осуществляется с помощью клавиш управления курсором или с использованием горячих клавиш (подчеркнутые символы в командах меню): [***Alt- клавиша***].

Панели инструментов (Toolbars)

VB имеет четыре стандартные панели инструментов: *Standard* – стандартная, *Edit* - редактирования, *Debug* – отладки и *Form Editor* – редактор форм.

Стандартная панель инструментов содержит команды, дублирующие основные команды меню.

Панель редактирования используется при вводе и редактировании текста программы. Содержит кнопки для вывода всплывающих списков свойств и методов объекта, констант, синтаксиса для процедур и методов, а также содержит кнопки для управления редактированием текста.

Панель отладки – служит для отладки программ в процессе ее выполнения и обеспечивает запуск программы на выполнение, временную остановку (пауза), выход из программы, установку точек останова, пошаговое выполнение в режиме пошагового выполнения, вычисление выделенного выражения.

Панель редактора форм – применяется при разработке форм. Она позволяет изменять порядок элементов управления, выравнивать их по горизонтали и вертикали, уравнивать размеры выделенных элементов управления по ширине и высоте, а также блокировать или разблокировать элементы управления в форме.

Панель инструментов элементов управления и компонентов пользователя (Toolbox).

Панель Toolbox содержит элементы управления. ***Элементы управления*** – это элементы, которые используются при разработке интерфейса[3] создаваемых приложений. Общее количество доступных к использованию элементов управления зависит от того, какая версия VB используется.

Для добавления новых компонентов к панели инструментов Toolbox из числа зарегистрированных необходимо:

- ввести команду ***Project\Components***, выбрать закладку ***Controls***;
- найти в списке нужный элемент управления и установить напротив него флажок;
- выйти из окна диалога, щелкнув кнопку ***Ok***.

Форма (Form)

Создаваемые в Visual Basic окна называются *формами*. ***Форма*** – это основное окно интерфейса разрабатываемой программы, форма - это также основа для создания окон диалога. При загрузке Visual Basic в рабочем окне открыта одна форма, которой по умолчанию присваивается наименование Form1 (рис. 1.2). Форма имеет строку заголовка, кнопку системного меню и кнопки управления развертыванием и свертыванием окна. На ней размещаются все необходимые элементы управления, для удобства размещения которых на форму выводится сетка. В простейшем случае проект может содержать одну форму. Форма настраивается так, как ее будет видеть пользователь при выполнении программы. Перемещают окно формы и изменяют его размеры с помощью мы-

[3] Интерфейс – от латинского interface (связь), средство общения пользователя с программой

ши во время настройки или программным путем в процессе выполнения программы.

Окно Проект (Project)

В окне проекта (броузер проектов) (рис. 1.3.) отображаются все элементы приложения: формы, модули, классы и т.п., сгруппированные по категориям. В VB все разрабатываемые приложения называются ***проектами***. Проект представляет группу связанных файлов и может содержать несколько групп компонентов (формы, модули и т.д.). Все проекты VB строятся по модульному принципу, поэтому и текст программы состоит не из одного большого файла, а из нескольких частей - процедур. Несколько проектов также могут объединяться в группы. Ниже заголовка окна проекта размещены три кнопки: кнопка просмотра текста программы; кнопка просмотра объекта (формы, MDI-формы, управляющего элемента, страницы свойств, модули и классы не имеют визуальных компонентов); кнопка изменения режима просмотра: в виде списка компонентов или в виде дерева групп компонентов.

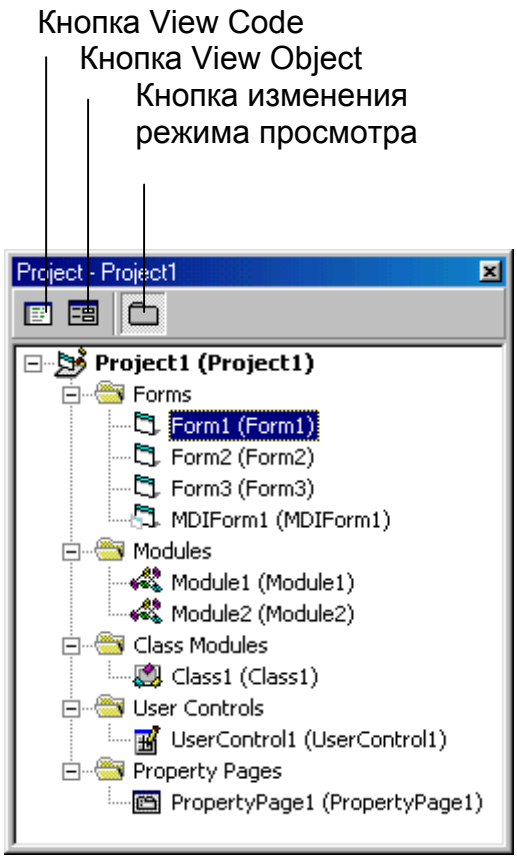


Рис.1.3. Окно Проект

Все элементы проекта сохраняются как отдельные и независимые файлы. Поэтому их можно в любое время загружать и сохранять. Это дает возможность использовать в проекте формы и тексты программ, созданные для других проектов, что экономит рабочее время.

Типы файлов проекта Visual Basic приведены в таблице 1.1.

Таблица 1.1

Типы файлов Visual Basic

Тип файла	*Расширение*
Проект Visual Basic (Project)	.VBP
Форма Visual Basic (Form)	.FRM
Программный модуль (Module)	.BAS
Модуль класса (Class Module)	.CLS
Пользовательский элемент управления(User Controls)	.CLT
Файл формы документа ActiveX (Property Pages)	.DOB
Группа проектов	.VBG

Файлы форм сохраняются с расширением ***.FRM***. Содержимое окна проекта сохраняется в специальном файле с расширением ***.VBP***. Файл проекта содержит список элементов, которые надо загрузить в среду разработки. При сохранении проекта сохраняются и все его компоненты. Если несколько файлов проектов объединяются в группу, то их имена сохраняются в файле с расширением ***.VBG***.

Для добавления в проект нового элемента, необходимо вызвать команду ***Project\Add***, а для удаления элемента нужно выделить его в окне проекта и затем выбрать команду меню ***Project\Remove***.

Окно Свойства (Properties)

В окне свойств (рис. 1.4.) задаются свойства выбранного элемента управления. В строке заголовка окна свойств рядом с текстом Properties указывается имя формы, которой принадлежит элемент управления. Поле со списком под строкой заголовка позволяет выбрать требуемый элемент управления. В списке, расположенном ниже, перечислены свойства этого элемента. Эти свойства могут быть упорядочены в алфавитном порядке (Alphabetic) или расположены по категориям (Categorized). Набор свойств зависит от типа элемента управления.

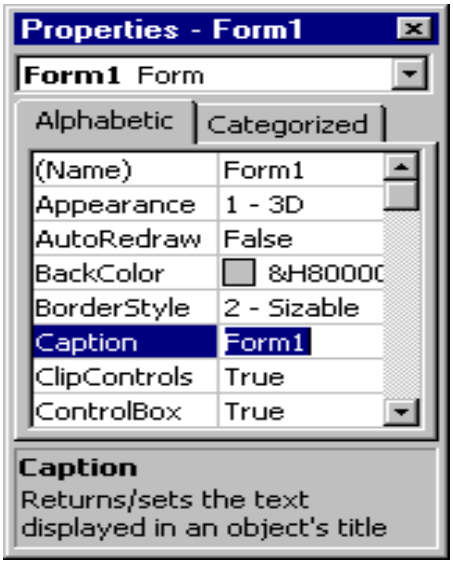


Рис. 1.4. Окно Свойства

Как установить свойства объекта? Имеется три способа:

- ввести значение в поле справа от свойства;
- выбрать из списка, который открывается щелчком мыши по полю;
- установить с помощью окна диалога, при щелчке мышью по полю появляется кнопка (…) – троеточие или эллипсис, щелкните эту кнопку. При щелчке по кнопке троеточие появляется окно диалога для настройки соответствующего свойства.

Окно Программа (Code)[4]

[4] В переводной литературе любая программа называется кодом и дословный перевод наименования окна также – “код”. Однако, в русском языке установилась следующая терминология: программа на исходном языке называется “текст программы”; результат трансляции – “объектный код”; готовая к выполнению программа – “код” или “машинный код” – примечание Колосова С. В..

Сразу после запуска окно Программа не отображается. Текст программы в VB разделяется на части – процедуры и записывается в процедуры обработки

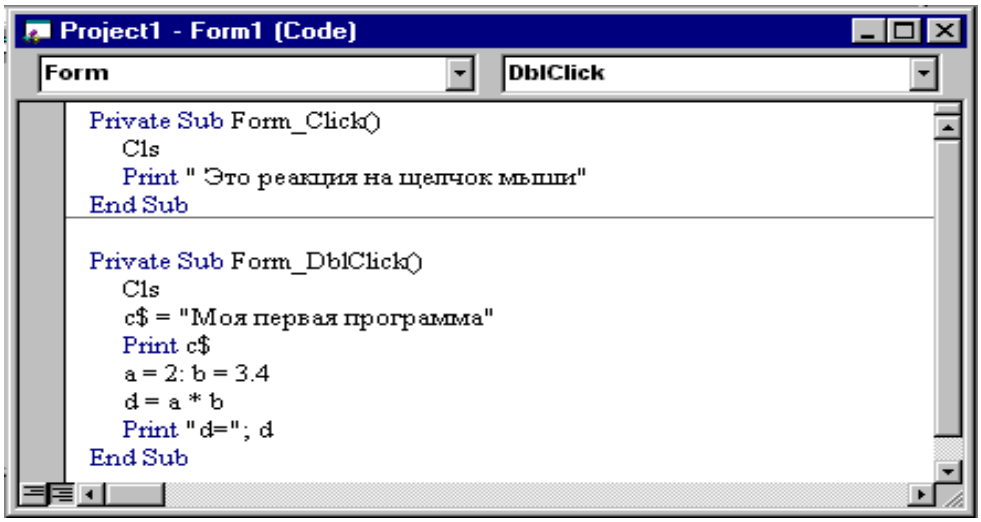


Рис. 1.5. Окно Программы (Code)

событий или процедуры пользователя, поэтому он, как правило, связан с определенным элементом управления. Это позволяет открыть окно диалога двойным щелчком по соответствующему элементу формы или по самой форме. Кроме того, окно программы можно открыть щелчком по кнопке *View Code* в окне *Project*.

В верхней строке окна программы (рис.1.5) располагается строка заголовка, в которой указано имя проекта и управляющие кнопки (системного меню, закрытия окна, свертывания и развертывания окна). Ниже строки заголовка расположены два раскрывающихся списка: список объектов (левый) и список процедур (правый).

Список объектов содержит все объекты текущей формы. В их число входит и специальный объект *General*, содержащий общий код, используемый всеми процедурами формы.

Список процедур содержит список всех событий, распознаваемых текущим объектом. Если для объекта уже написаны процедуры обработки событий, то они выделяются здесь жирным шрифтом. Если выбрать любой пункт данного списка, то VB выведет соответствующую процедуру либо ее шаблон в окне программы и поместит в нее курсор

Окно позиционирования формы (Form Layout)

Данное окно позволяет установить место, где будет размещаться форма при запуске программы.

1.1.3. Работа с внешними устройствами

Сохранение информации и открытие файлов

По окончании работы проект и форму необходимо сохранить на рабочем диске. Для сохранения информации на диске следует выбрать команду ***File\Save Project*** или ***File\Save Project As.*** Если проект сохраняется первый раз, то сначала сохраняется форма, а затем сам проект. После ввода команды на экране появляется окно диалога (рис. 1.6). Откройте раскрывающийся список "Папка" и выберите в нем необходимый диск, например R:, выберите двойным щелчком мыши нужную папку, введите в строке ввода "Имя файла" имя вашего файла и щелкните по кнопке ***Сохранить***. Имя файла следует выбирать осмысленно, чтобы оно отражало содержание хранимой информации. В этом случае его легче будет отыскать на диске. Расширение имени файла предлагается по умолчанию. Как уже упоминалось, проект приложения сохраняется с расширением VBP, а форма – с расширением FRM.

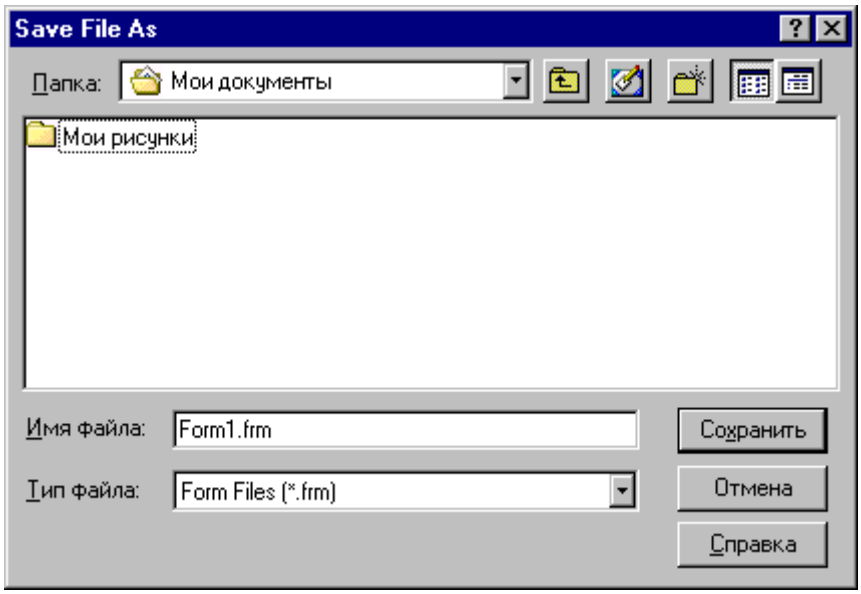


Рис. 1.6. Окно диалога для сохранения файлов

Учитывая, что проект может содержать большое число различных файлов рекомендуется для каждого проекта создавать отдельную папку.

> ***Не рекомендуется*** *применять другие расширения имен файлов, так как при открытии окна диалога для открытия файлов* ***Open Project*** *они не попадут в список файлов, формируемый по умолчанию.*

Чтобы загрузить в программу уже существующий файл, используется команда ***File\Open Project.*** Окно диалога открытия файлов аналогично окну диалога сохранения файлов, но имеет две закладки: *Existing* (существующий), которая позволяет найти любой файл на компьютере, и *Recent* (недавний), позволяющая загрузить недавно использовавшиеся файлы. Откройте список папок и выберите нужный диск и папку (каталог); выберите двойным щелчком мыши файл для запуска, или выделите файл и щелкните по кнопке Открыть.

Вывод информации на печать

Для вывода текста программы на печать служит команда ***File\Print***. После ввода команды появляется окно диалога (рис. 1.7). Это окно позволяет вывести на печать текущую форму, тексты процедур формы, модуль, с которым работает программист или все формы и модули приложения. Переключатели (Selection, Current Module и Current Project) определяют, печатать ли код выделенной части приложения, текст программы активной в настоящий момент формы или текст программы всего проекта, соответственно. Три флажка (Form Image, Code и Form As Text) определяют, выводить ли на печать изображение формы (то, что пользователь видит на экране), текст программы формы - программных модулей или сведения о свойствах формы и элементов управления, установленных на ней. По умолчанию на печать выводится текст программных модулей (установлен флажок Code).

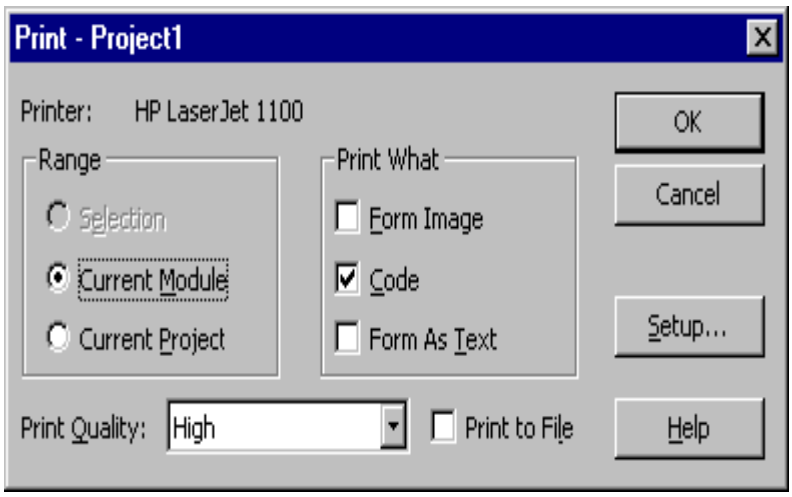


Рис. 1.7. Окно диалога для вывода информации на печать

Команда ***File\Print Setup*** выводит окно с настройками печати. Например, можно выбрать принтер или ориентацию бумаги.

Для организации вывода информации на печать VB взаимодействует с программами Windows более низкого уровня. Для вывода на печать изображения формы со всеми видимыми элементами управления в VB достаточно одной команды – ***PrintForm***.

1.1.4. Упражнение: работа в среде Visual Basic

1. Запустите программу Visual Basic: введите команду Пуск\Программы, найдите в подменю команды Программы пиктограмму Visual Basic 6.0 и щелкните по ней дважды мышью. Если данной программы нет в меню, найдите ее в компьютере командой Пуск\Найти\Файлы и папки. Имя исполняемого файла – Vb6.exe (могут быть и другие способы запуска Vb6 в зависимости от опыта пользователя и установленного программного обеспечения).
2. Щелкните в окне диалога New Project дважды по пиктограмме Standard.EXE для создания стандартного приложения Windows (это окно может не появляться на экране, а сразу появится рабочее окно Visual Basic).
3. Изучите элементы рабочего окна VB и команды меню. Измените размеры и положение формы в окне, перетаскивая его мышью.
4. Закройте окна Проект и Свойства кнопками закрытия окон.

5. Откройте окна Проект и Свойства командами View\Project Explorer и View\Properties Window, соответственно. Найдите в стандартной панели инструментов одноименные кнопки.

6. Переместите окно Проект в нижний правый угол рабочего окна. Переместите окно Проект в первоначальное положение.

7. Закройте панель элементов управления (Toolbox). Выведите панель Toolbox в рабочее окно командой View\Toolbox.

8. Добавьте на панель элементов управления сетку Microsoft Flex Grid. Введите команду Project\Components\Controls, найдите в списке объект Microsoft FlexGrid Control 6.0 (SP3) и щелчком мыши установите напротив него флажок. Щелкните по кнопке Ok.

9. Добавьте к проекту еще одну форму командой Project\Add Form.

10. Создайте новую папку на рабочем диске и сохраните проект в этой папке командой File\Save As.

11. Откройте новый проект командой File\New Project.

12. Откройте сохраненный ранее проект командой File\Open Project.

13. Выведите на печать форму Form1. Введите команду File\Print, установите флажок Form Image и щелкните по кнопке Ok – будет напечатана форма со всеми установленными на ней элементами (или пустая форма).

14. Выйдите из программы Visual Basic.

1.1.5. Закрепление материала

1. Какие версии языка программирования Visual Basic вам известны?
2. Перечислите требования к установке VB.
3. Как запустить программу VB?
4. Назовите основные элементы рабочего окна VB.
5. Как добавить новые элементы управления на панель инструментов Toolbox?
6. Для чего предназначено окно Проект?
7. Каково назначение окна Свойства?
8. Для чего предназначено окно Программа?
9. Как вызвать окно Программа?
10. Где записывается текст программы объекта?
11. Как сохранить программу на диске?
12. Как загрузить существующую программу?
13. Как вывести на печать текст программы?
14. Как вывести на печать сведения об установленных свойствах формы?

1. 2. Основные понятия об объектно-ориентированном программировании

1.2.1. Общие принципы объектно-ориентированного программирования

Объектно-ориентированное программирование (ООП) - вид программирования, при котором код программы и связанные с ним данные организуются в объекты.

Основными принципами объектно-ориентированного программирования являются *наследование*, *полиморфизм*, *инкапсуляция.*

Инкапсуляция или сокрытие информации означает сочетание структур данных с методами их обработки в абстрактных типах данных – классах объектов. При этом обращение к этим объектам и методам скрыто от пользователя. Обращение к данным и функциям выполняется через вызов соответствующих методов или свойств. Сокрытие информации позволяет разработчику объекта изменять внутренние принципы его функционирования, не оказывая при этом никакого влияния на пользователя объекта. Поэтому инкапсуляция – способ работы с объектом как с ''черным ящиком'', повышающий эффективность отладки и повторного использования классов.

Наследование – способность создавать классы, зависящие от других классов. Цель наследования сделать текст программы проще, используя уже готовые объекты и их свойства.

Полиморфизм (многоликий) – способность объекта реагировать на запрос (вызов метода) сообразно своему типу, при этом одно и то же имя свойства или метода может использоваться для различных классов объектов. Суть его заключается в том, что при написании программы, посылающей объекту сообщение, не нужно знать, к какому классу принадлежит этот объект. Все что нужно – это знать имя сообщения и его параметры. Например, оператор + может использоваться как для строковых так и для числовых переменных. Однако это будут совершенно разные операции. В первом случае будут объединяться строки символов, во втором – вычисляться сумма числовых переменных.

Ключевыми понятиями ООП являются *класс, объект, свойства объекта, метод обработки, событие.*

Класс - это абстрактный тип данных, шаблон или план, по которому создаются объекты определенного класса.

В этой интерпретации каждый *объект* – это экземпляр определенного класса. Например, есть *класс форма*, при загрузке VB на основе этого класса автоматически создает и помещает в рабочее окно экземпляр этого класса объект Form1, для которого пользователь может установить требуемые свойства из набора уже известных свойств объекта класса форма. Или другой пример, есть

класс текстовое поле, помещая на форму текстовое поле, мы создаем объект типа текстовое поле. Поместив на форму пять текстовых полей, мы создадим тем самым пять экземпляров объектов класса текстовое поле.

Класс - это абстрактный тип данных, шаблон или план, по которому создаются объекты определенного класса.

Объект – совокупность свойств (параметров) определенных сущностей и методов их обработки (программных средств). Объект содержит инструкции (программный код), определяющие действия, которые может выполнять объект, и обрабатываемые данные.

Объект – это экземпляр класса, характеризующийся определенным набором свойств, методов, событий.

Свойство – характеристика объекта, его параметр. Все объекты наделены определенными свойствами, которые в совокупности выделяют объект из множества других объектов, т.е. объект обладает качественной определенностью. Свойства объектов разных классов могут пересекаться.

Свойство – характеристика объекта, его параметр.

Метод – программа действий над объектом или его свойствами. Метод рассматривается как программный код, связанный с определенным объектом, и который осуществляет преобразование свойств, изменяет поведение объекта.

Методы – это глобальные функции или процедуры, определенные в классе.

Чем отличается метод от свойства? Свойство определяет внешний вид объекта и его местоположение. Свойства задаются на этапе разработки приложения, а также могут изменяться программным путем, например, положение

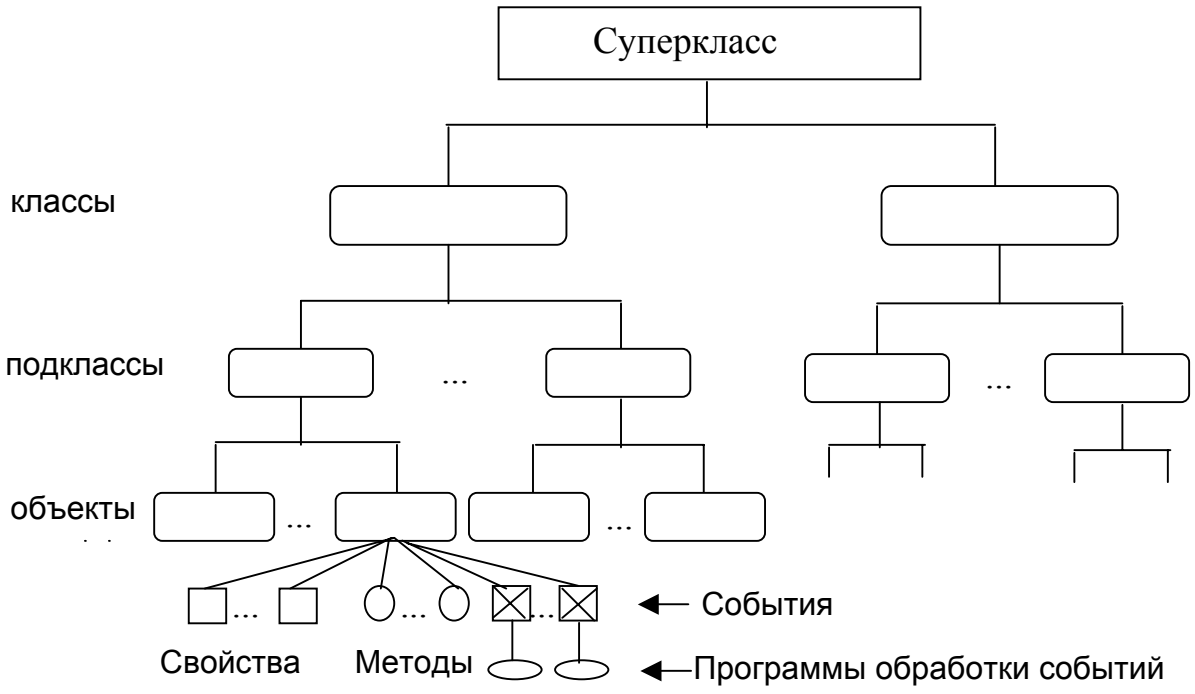


Рис. 1.8. Классификация объектов

объекта определяется значением свойств Top и Left (положение левого верхне-

го угла объекта). Методы используются только в процессе выполнения программы, хотя могут приводить к тем же результатам, например метод Move (перемещение) относительно формы.

Событие – это свойство объекта процедурного типа. События могут генерироваться системой – внутренние события или пользователем – внешние события. Например, щелчок мышью, или нажатие клавиши на клавиатуре – внешние события; загрузка или выгрузка формы – внутренние события.

Событие – это свойство объекта процедурного типа

Взаимосвязь указанных выше понятий представлена на рис.1.8

С каждым событием связана ***процедура***, в которую может быть помещен текст программы. Эта программа выполняется только при наступлении данного события. Например, изменение значения текстового поля – есть событие Change. Для обработки этого события предусмотрена процедура

```
Private Sub Text1_Change ()     - ‘ заголовок процедуры
. . .
End Sub                                  - ‘ конец процедуры
```

Шаблон этой процедуры входит в состав программного кода объекта TextBox. Предположим, что на форму установлен объект TextBox, имя объекта – Text1. Если в окне программы в списке объектов (рис. 1.5) выделить объект Text1, а в списке процедур выбрать событие Change то в окно программы автоматически вставляется шаблон этого события.

Чтобы эта процедура работала, в нее надо поместить текст программы, например:

```
Private Sub Text1_Change ()
    Длина=Val(Text1.Text)
End Sub.
```

Тогда при каждом изменении значения свойства Text объекта Text1 переменной Длина будет присваиваться новое значение.

Программа, созданная с помощью инструментальных средств ООП, содержит объекты с их характерными свойствами, для которых разработан графический интерфейс пользователя. Как правило, работа с приложением осуществляется с помощью экранных форм с объектами управления - *окон диалога*. Экранные формы[5] используются также для выполнения заданий и перехода от одной части программы к другой. При разработке приложений, для объектов управления уточняется перечень событий и создается пользовательский метод обработки – текст программы на языке программирования в виде событийных процедур.

В объектно-ориентированном программировании используется следующий синтаксис операторов для обращения к методам и свойствам объектов:

[5] Формы, представленные на экране монитора не имеют материального аналог, они существуют только в виде программ на диске или в оперативной памяти компьютера, а также в виде визуального изображения на экране монитора. Поэтому их можно называть “экранные формы”. В дальнейшем по тексту, будем писать просто “формы”.

обращение к методу:

ИмяОбъекта.Метод

ИмяОбъекта.Свойство.Метод

изменение свойства объекта в процессе выполнения программы:

ИмяОбъекта.Свойство =”значение”

использование свойства для присвоения значения переменной:

Переменная = ИмяОбъекта.Свойство

Например:

```
Form1.Show    -   ‘  Form1 - имя объекта; Show - метод
Text1.Text =”Учебное пособие”  -  ‘ Text1 – имя объекта; Text – свойство
                                        ‘ объекта; “Учебное пособие” –значение
Художник = Text2.Text   - ‘ Художник – переменная; Text2 – имя объекта;
                                ‘ Text – свойство объекта
```

1.2.2. Основные свойства объектов

При выделении объекта его свойства отображаются в окне свойств (Properties).

Все свойства, события и методы объекта можно просмотреть в окне *Object Browser* (рис.1.9.), который содержит каталог объектов. Для вывода окна диалога необходимо ввести команду ***View\Object Browser***. Выбрать в этом меню в верхнем поле списка необходимую библиотеку либо объекты из всех библиотек. Стандартные элементы управления содержит библиотека VB (Visual Basic objects and procedures). В списке Classes перечислены все классы объектов VB. Здесь Form класс объекта типа Форма, Form1 – объект, экземпляр данного класса. Financial – класс типа Модуль, остальные элементы – константы объекта Form1. После выбора объекта в списке Members of выводятся все свойства и методы, относящиеся к выбранному объекту, в данном случае к форме. В этом окне отображаются свойства (MousePointer, Moveable, Name), события (MouseUp) и методы (Move). События выделяются желтым цветом, методы - зеленым цветом.

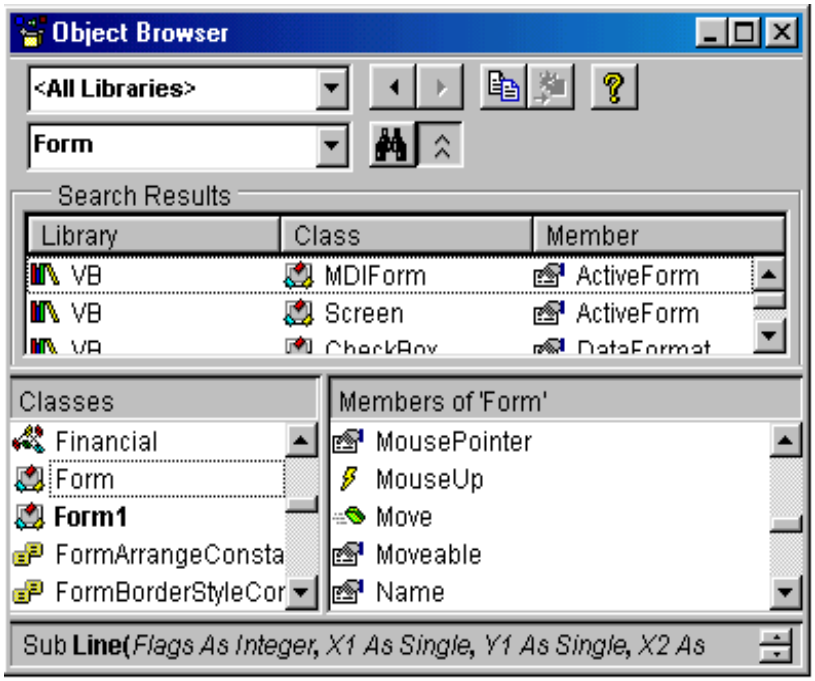


Рис.1.9. Окно диалога для Просмотра объектов

Нажав F1 или кнопку ’’? ” можно вызвать справочную информацию. Чтобы найти конкретное свойство или метод можно воспользоваться кнопкой поиска (бинокль). Введите во второй строке ввода имя объекта, например Form, и щелкните по кнопке поиска. После окончания поиска в окне результаты поиска (Search Results) появится имя библиотеки и искомый объект , а в окне Members of – свойства и методы этого объекта. В нижней части окна дается краткое описание выделенного объекта. В примере на рисунке 1.9 выведено сведение о методе Line.

Свойства объектов

Каждый объект характеризуется определенным набором свойств, методов и событий, связанных с ним. Некоторые из свойств объекта характерны для всех объектов. Такими общими свойствами являются имя, надпись, положение объекта и его размеры, цвет фона, цвет текста, индекс, индекс обхода.

Имя элемента (Name). Каждому элементу управления имя присваивается по умолчанию, например, форме – *Form*, метке – *Label*, полю ввода – *Text* и т.д. Так как однотипных объектов может быть много, то программа нумерует их: Label1, Label2, и т.д. Однако, такие имена мало информативны, поэтому каждому объекту необходимо присвоить оригинальное, достаточно информативное имя. Полям ввода можно присваивать имена в соответствии с элементом данных: длина, ширина, успеваемость. Кнопкам - в соответствии с их функциональным назначением: стоп, пуск, пауза…Имена объектов используются в программе для обращения к ним, поэтому имя объекту необходимо присваивать до написания программы. Имя объекта необходимо писать латинскими символами. В имени объекта не допускается использование пробелов.

Примите за правило: создал объект – присвой ему имя.

Рекомендуется имя объекта начинать с префикса. Например, для текстового поля – txt, для метки – lbl, для кнопки – cmd и т.д.: txtDlina, lblDlina, cmdStart. При этом префикс начинается со строчной буквы, а базовое имя (Dlina) - с прописной буквы.

Надпись (Caption) – это текстовое сообщение, которое выводится на соответствующий объект в форме. Каждому объекту программа автоматически присваивает имя и аналогичную надпись. Надпись на элементе управления может совпадать с именем объекта. В отличие от имени, надпись можно писать на любом языке, т.к. она не используется в программе. Надпись на объекте можно создавать в процессе разработки программы, а также при ее выполнении. Текст программы в этом случае представляет собой строку следующего вида:

```
ИмяФормы.ИмяОбъекта.Свойство =”<текст>”
frmForm1.cmdExit1.Caption = ”Резервная кнопка”
```

Положение объекта. Положение объекта в форме определяется координатами верхнего левого угла (рис.1.10). Для этого используется два свойства: **Top** – расстояние от верхнего края формы и **Left** – расстояние от левого края формы. Все расстояния задаются в ’’твипах ’’. ***Твип*** – экранно-независимая единица измерения. Один твип равен 1/20 точки принтера. Это гарантирует независимость отображения элементов приложения от разрешения дисплея.

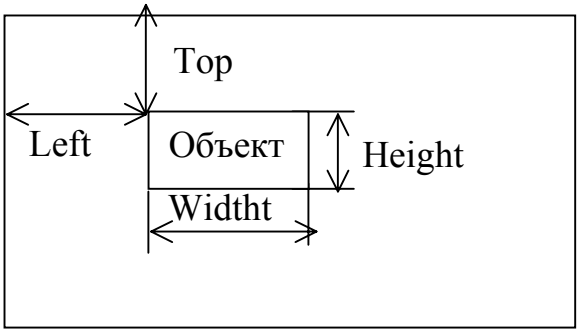


Рис.1.10. Положение объекта в

Размеры объекта определяют также два свойства: **Height** и **Width** (высота и ширина).

Цвет. Управление цветовым оформлением элементов осуществляется с помощью свойства **BackColor**, **FileColor** и **ForeColor**. Этим свойством по умолчанию назначаются стандартные цвета Windows.

Свойство *BackColor* определяет цвет фона. При проектировании цвет выбирают из палитры в диалоговом окне настройки, а в программе цвета задаются либо с использованием цветовой схемы RGB, либо константами библиотеки VBRUN (см. тему 4).

Свойства *ForeColor* и *FillColor* определяет цвет текста и цвет рисованных объектов, соответственно.

Enabled – доступность элемента управления. Может принимать два значения: *True* – истина и *False* ложь. Если значение свойства установлено в *False*, то элемент управления будет недоступен пользователю. Такой элемент управления при выполнении программы подсвечивается блеклым серым цветом.

Visible – видимость элемента. Это свойство также имеет два значения: *True* и *False*. Если установлено значение False, то элемент управления будет невидим.

Index – индекс. Определяет номер элемента в массиве элементов управления или в списке.

TabIndex – индекс обхода. Присваивается программой автоматически при помещении элемента управления на форму в порядке их создания. Этот индекс определяет порядок перемещения *фокуса* по элементам управления при нажатии клавиши *Enter*. Порядок обхода может быть изменен пользователем на этапе разработки программы.

События объектов

Основные события связаны с мышью и клавиатурой.

Click. Событие Click вызывается, как только пользователь выполнит щелчок мышью на элементе управления.

DblClick. Событие DblClick вызывается двойным щелчком мыши на элементе управления. Временной интервал между двумя щелчками устанавливается в панели управления Windows;

KeyDown – нажатие клавиши;

KeyUp - отпускание клавиши;

KeyPress – возвращает код ASCII нажатой клавиши.

При нажатии и отпускании клавиши возвращаются два параметр: KeyCode и *Shift*. KeyCode содержит клавиатурный код, а Shift – информацию о нажатии клавиш Shift, Ctrl или Alt. Например:

```
Control_KeyUp(KeyCode As Integer, Shift As Integer)
```

После нажатия клавиши события наступают в такой последовательности: KeyUp, Key Press, KeyDown.

KeyPreview – определяет порядок передачи событий. Если свойству KeyPreview формы присвоить значение True, то событие, связанное с клавиатурой передается сначала форме, а затем текущему элементу управления. KeyPreview - это свойство присуще только форме. Оно определяет порядок обработки событий, связанных с нажатием клавиш.

Одним из важных понятий при обращении к элементам управления в Windows является понятие ***фокус*.** При нажатии клавиш на клавиатуре управление получает активный элемент, то есть элемент, имеющий фокус. Например: при нажатии клавиши Enter выполняются те команды, которые связаны с командной кнопкой, имеющей фокус; текст вводится в поле ввода, на которое установлен фокус, и т.д. Не все объекты могут получать фокус, например Надпись. Если элемент получает фокус, то он выделяется особым образом, например, текстовое поле отмечается мигающим курсором, командная кнопка - пунктирной рамкой по периметру и т. д. С установкой и потерей фокуса связаны два события и один метод:

GotFocus – событие, установка фокуса;

LostFocus – событие, отмена (потеря) фокуса.

SetFocus – метод**,** установка фокуса.

Фокус на элемент управления может быть установлен и в процессе выполнения программы с помощью метода *SetFocus*. Например: txtText1.SetFocus

Методы объектов

Объекты имеют различное число методов, например, объект типа надпись имеет 10 методов, объект типа форма – 22 метод. Общими для этих объектов являются методы *Move* – перемещать, *OLE Drag* - перемещать с использованием механизма OLE (вставка и внедрение), *Refresh* – перерисовать объект, и *SetFocus* - установка фокуса.

1.2.3. Элементы управления Label, TextBox, CommandButton

Создание Windows-приложений практически невозможно без использования элементов управления, так как они позволяют пользователю взаимодействовать с этими приложениями. Набор таких элементов управления не ограничен и может расширяться за счет так называемых пользовательских элементов управления (custom controls).

В данном разделе рассматриваются три элемента управления, которые позволяют создавать простейшие приложения: Командная кнопка (CommandButton), Надпись (Label) и Текстовое поле или Поле ввода (TextBox).

Командная кнопка (Command Button)

Кнопка используется для управления процессом: начало, окончание, прерывание и т. д.

Основными свойствами являются имя, название, положение, размеры, цвет, доступность, видимость.

Дополнительно можно указать следующие свойства:

Default – определяет, является данная кнопка активной по умолчанию или нет. Свойство имеет два значения: *True* и *False*. По умолчанию – False. Если установлено значение True, то фокус установлен на данной кнопке, то есть кнопка активна. Нажатие клавиши Enter перехватывается и передается этой кнопке.

Cancel - используется подобно свойству Default. Оно обеспечивает перехват нажатия клавиши Esc и вызов события Click для соответствующей кнопки. Например, если свойству Cancel кнопки cmdEnd присвоить значение True, то при нажатии клавиши Esc будет выполнена программа, записанная в обработчике события этой кнопки.

Appearance – позволяет придать кнопке трехмерный вид.

ToolTipText – позволяет ввести текст, который отображается в подсказке, появляющейся при зависании указателя мыши на элементе управления.

Надпись (Label)

Надпись предназначена для отображения текста, который пользователь не может изменить с клавиатуры. Она обладает всеми перечисленными выше общими свойствами. Дополнительно можно указать еще ряд свойств:

BorderStyle - позволяет отображать текст с рамкой или без рамки.

Font - это свойство позволяет оформлять шрифты, используя все возможности Windows.

AutoSize - автоматическое приведение ширины объекта в соответствие с

длиной текста. Если свойство AutoSize равно False и длина вводимого текста больше ширины надписи, то текст усекается. Если свойство AutoSize равно True, то размер объекта приводится в соответствие с длиной текста. При этом значение свойства WordWrap должно быть равно False.

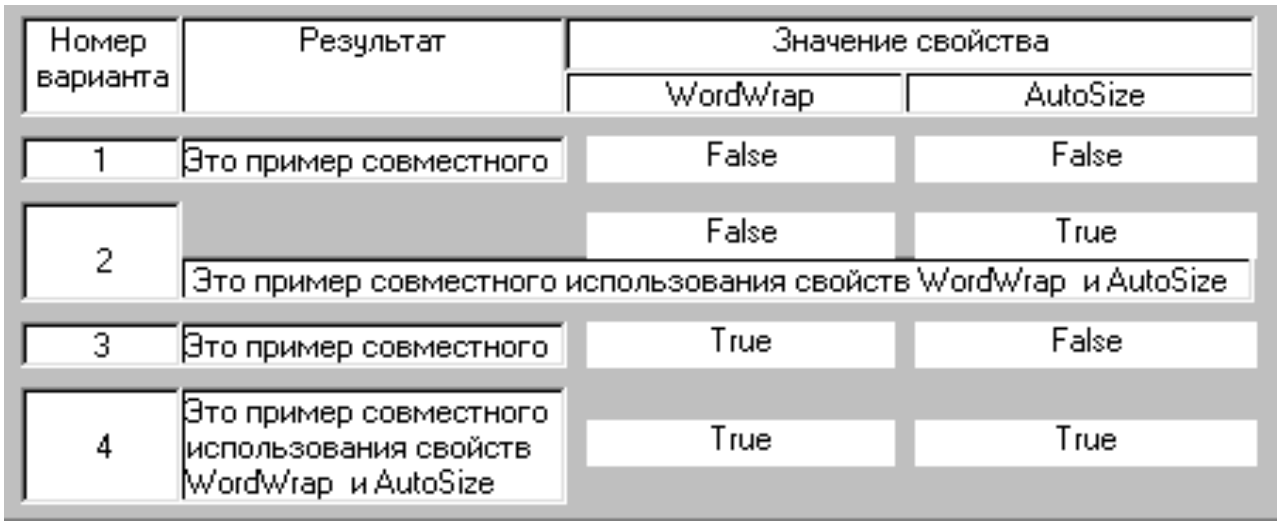


Рис.1.11. Влияние свойств WordWrap и AutoSize на изменение размеров надписи

WordWrap - автоматический перенос длинного текста на другую строку. При установке значения этого свойства в True длинный текст будет автоматически переноситься на новую строку независимо от значения свойства AutoSize. Обычно свойства WordWrap и AutoSize используются совместно. Пример взаимодействия этих свойств приведен на рис. 1.11.

В первом варианте длинный текст усекается в соответствие с размерами элемента управления. Во втором варианте размер надписи приводится в соответствие с длиной текста. В третьем варианте текст автоматически переносится на следующую строку, но этого не видно, так как высота надписи автоматически не изменяется. В четвертом варианте текст автоматически переносится на следующую строку и увеличивается высота надписи в соответствии с размером текста.

Для получения требуемого результата в четвертом варианте установите вначале свойство WordWrap в True, а затем значение свойства AutoSize также установите в True.

Текстовое поле (TextBox)

Текстовое поле является основным элементом управления, предназначенным для ввода и вывода данных.

Основные свойства текстового поля совпадают с перечисленными выше, но есть и особенные свойства:

Text – аналог свойства Caption. Через это свойство осуществляется, как правило, ввод данных в программу и вывод данных на экран;

Alignment – выравнивание текста. Имеет три значения: 0- выравнивание по левому краю, 1 - по правому краю, 2 - по центру;

ScrollBars – вывод линеек прокрутки. Свойство Scrollbars позволяет устанавливать горизонтальную, вертикальную линейки прокрутки или обе.

MultiLine – определяет может ли поле содержать более одной строки текста. Обычно совмещается с установкой свойства ScrollBars.

В текстовом поле можно выделять и заменять текст. Это осуществляется программным путем с помощью свойств *SelStar*t, *SelLength, SelText.*

SelStart – начало выделения.

SelLength – длина выделяемого текста.

SelText – замена текста.

Например:

```
Text1.SelStart=2          ‘начать выделение со второго символа
Text1.SelLength=6         ‘выделить шесть символов
Text1.SelText = “Привет”  ‘заменить выделение на слово  “Привет”
```

MaxLength – определяет максимальное число символов – По умолчанию – 0, что означает максимальное значение – 32 тысячи символов.

PassWordChar – позволяет заменять вводимые символы звездочками. Это свойство используется для ввода пароля.

Locked – запрещает пользователям изменять содержимое поля. Поле можно просматривать, но нельзя редактировать или удалять. Однако, значение поля можно изменить программным путем.

Объект TextBox может обрабатывать 23 события. Основные события текстового поля связаны с вводом данных: Click, DblClick, KeyDown, KeyUp, GotFocus, LostFocus, которые рассмотрены были ранее;

Change - изменение значения текстового поля.

При вводе данных с клавиатуры в активное текстовое поле программа не делает различий между буквами и цифрами: все вводится как текст. Поэтому для перевода текста в числа и обратно чисел в текст, используются функции преобразования символьных переменных:

Val(C) – преобразование текста в число ;

Str(N) – преобразование числа в текст или **Str$(N)** - для преобразования в символьную переменную переменных типа Variant. Например:

```
A = Val(Text1.Text)   ‘перевод значения текстового поля в число
Text2.Text = Str$(a)   ‘перевод числа в текст
```

1.2.4. Приступая к программированию

Прежде чем приступить к разработке программ в среде Visual Basic, необходимо выработать общий план. Отсутствие продуманного плана, программирование “с листа”, приводит к неоправданной потере рабочего времени, перерасходу машинного времени и, как правило, снижает качество проекта. Можно предложить следующий порядок работы:

- изучите задание и составьте список задач, которые должна выполнять программа;
- выберите метод решения задачи;
- разработайте математическую модель (совокупность формул, необходимых для решения задачи);
- выберите переменные программы и опишите их;
- разработайте укрупненную схему алгоритма, указав на ней взаимосвязь отдельных блоков программ;
- разработайте детальные схемы алгоритмов;
- определите количество и состав форм и модулей;
- разработайте эскизы экранных форм с указанием всех элементов и их расположением;
- продумайте порядок тестирования программы и варианты тестов;
- определите место хранения программы, имена файлов и каталогов;
- после разработки программы проведите ее тестирование и оформите их документально. В отчете о тестировании программы необходимо привести исходные данные, указанные в задании или подготовленные в качестве теста, и результаты работы программы при каждом наборе исходных данных;
- создайте исполняемый файл программы.

Для простых программ не все эти этапы обязательны, но чем сложнее задача, тем тщательней должна быть проработка плана проекта. Переделка программы, ее интерфейса занимает порой больше времени, чем написание новой программы.

1.2.5. Упражнения: начало работы в Visual Basic

Рассмотрим несколько задач.

Задача 1.1. Изучение событий Click и DblClick.

Требуется написать программу для вывода текстового сообщения при одиночном и двойном щелчке мышью по форме. Результаты выводить непосредственно в форму.

Порядок работы:

- Запустите программу Visual Basic.
- Откройте новый проект командой File\New Project (если форма не открывается автоматически при запуске программы).
- Откройте окно Программы двойным щелчком по форме.
- Откройте щелчком мыши список объектов и выберите объект *Form1*.
- Откройте список процедур и выберите свойство *Click*. В окне программы появляется шаблон процедуры обработки события Click (рис. 1.5.).
- Запишите в этой процедуре текст программы, приведенный ниже:

```
Private Sub Form_Click( )
    Cls
    Print " Это реакция на щелчок мыши"
```

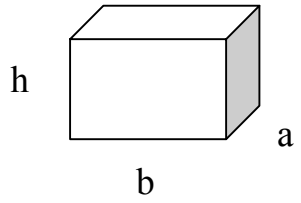
```
End Sub.
```

- Откройте список процедур и выберите свойство *DblClick*. Запишите текст программы:

```
Private Sub Form_DblClick( )
   CLS
   PRINT ”Это реакция на двойной щелчок мыши”
End Sub.
```

Оператор **CLS** очищает форму.

- Запустите программу командой ***Run\Start***. После запуска программы на экране появляется пустая форма.
- Проверьте правильность работы Вашей программы: щелкните по форме мышью – появится первое сообщение, щелкните дважды мышью по форме – появится второе сообщение.
- Сохранить программу на диске командой File\Save Project или File\Save Project as ...



Задача 1.2. Вычисление площади поверхности параллелепипеда и его объема.

Даны размеры сторон параллелепипеда. Требуется вычислить площадь поверхности параллелепипеда и его объем. Результаты выводить по щелчку мыши непосредственно в форму.

Порядок работы:

- Разработайте математическую модель для решения данной задачи:
  $S=2(a+b)h+2ab,\ V=abh$
- Подготовьте контрольные данные, например: a=1; b=1; h=2. Ответ: S=10; V=2.
- Напишите текст программы в обработчик события Click, чтобы при щелчке мышью по форме программа была выполнена:

```
Private Sub Form_Click ( )
   Let  a= 2: Let b=3 : Let h=50
   S=2*(a+b)*h+2*a*b
   V=a*b*h
   Print “a=”;str$(a), “b=’; str$(b), “h=”;str$(h)
   Print ”Площадь поверхности равна “;str$(s)
   Print ”Объем тела равен “;str$(V)
End Sub
```

- Запустите программу.
- Проверьте правильность работы программы по результатам выведенным на форму.
- Сохраните программу на диске.

Задача 1.3. Табулирование функции одной переменной.

Протабулировать функцию Sin x на отрезке [a,b] с шагом dx, если a = $\pi/2$; b = $3\pi/2$; dx=0,1. Для управления проектом на форму поместить кнопку Command1 (название и имя кнопке присваивается по умолчанию).

Порядок работы:

- Поместите на форму командную кнопу:

- щелкните мышью по значку кнопки на панели ToolBox (кнопка выделяется светлым тоном);

- переместите указатель мыши на форму, нажмите левую клавишу мыши и, протягивая мышь, установите требуемые размеры кнопки;

- Откройте окно программы двойным щелчком мыши по форме и выберите объект Command1;
- Откройте список объектов и выбрите свойство Click этого объекта;
- Подготовьте контрольные данные для тестирования программы.
- Запишите в обработчик события Click текст программы:

```
Private Sub Form_Click( )
    PI= 4*ATN(1)
    a=PI/2:   b=3/2*PI
    dx=0.1
    Print "X","Y"
    For   x=a To b+dx/2 step dx
          y=sin(x)
          Print  x,y
     Next  x
End Sub
```

- Запустите программу;
- Щелкните мышью по кнопке Command1 для выполнения расчетов.
- Проверьте правильность работы программы.
- Сохраните программу на диске.

В приведенных примерах использованы операторы LET и PRINT . Оператор LET служит для присвоения начальных значений переменным, он может быть опущен. Оператор PRINT выводит результаты вычислений непосредственно в форму.

Задачи к заданию для самостоятельной работы.

Отрезок табулирования и шаг табулирования выберите самостоятельно в области определения функции.

1. $\dfrac{x^2-1}{\sqrt{(x^2+1)(x^4+1)}}$
2. $\mathrm{Sin}(x)\mathrm{Ln}(\mathrm{Tg}(x))$
3. $\dfrac{e^x(1+\sin(x))}{1+\mathrm{Cos}(x)}$
4. $\dfrac{1}{(3\mathrm{Sin}(x)+2\mathrm{Cos}(x))^2}$
5. $\dfrac{\mathrm{Ln}(x^3)}{x}$
6. $\dfrac{x^3}{3+x}$
7. $\dfrac{x}{x^4+3x^2+2}$
8. $\dfrac{1}{\sqrt{(x+1)(x^2+1)}}$
9. $x+\mathrm{Ln}(x^2-4)$

10. $\frac{2(x+1)^2}{x-2}$ 11. $xLn^2(x)$ 12. $\frac{(4e^{2x}-1)}{e^{2x}}$

13. $x2^{-x^2/2}-e^{2+x}$ 14. $xe^{1/x}$ 15. $\frac{Ln(x)}{(x+1)^2}$

Задача 1.4. Установка объектов на форму и исследование их свойств.

- Откройте новый проект командой File\New Project. Присвойте форме имя – свойство Name, и название – свойство Caption.
- Установите на форму Надпись, Текстовое поле и Командную кнопку:

- выделите щелчком мыши по нужной пиктограмме на панели элементов управления;
- переместите указатель мыши на форму, нажмите левую клавишу мыши и протащите мышь по диагонали. Установите требуемые размеры элементов управления с помощью мыши или с помощью окна Свойств.
- присвойте объектам имена и названия (свойства Name для всех объектов, свойство Caption - для Надписи и Командной кнопки и свойство Текст – для Текстового поля).

- Выпишите свойства объектов по форме (указывать только не “пустые” свойства):

Свойство	Объект			
	Форма	Надпись	Текстовое поле	Кнопка

- Выведите свойства формы и объектов, установленных на ней на печать:
  - введите команду File\Print;
  - установите флажок Form AsText и снимите флажок Code.
  - сравните полученные результаты с Вашей таблицей.
- Создайте несколько копий элементов Надпись, Текстовое поле и Командной кнопки.
- Исследуйте свойства Alignment, Apperanse, BorderStyle и другие свойства, по Вашему усмотрению.
- Исследуйте свойства AutoSize и WordWrap (повторите рис. 1.11).
- Сохраните программу на диске.

Задача 1.5. Разработка простейшей формы.

Студент сдавал во время сессии экзамен по четырем предметам: физике, математике, информатике и физкультуре. Создать форму для вычисления среднего балла успеваемости студента за сессию. Данные вводить с клавиатуры в текстовые поля, результаты вычислений выводить в текстовое поле.

***Порядок работы*:**

1. Уточните состав задач программы.

Программа должна обеспечивать ввод данных с клавиатуры в текстовые поля и вывод данных в текстовое поле. Кроме того, на форму надо поместить элементы управления для управления процессом вычисления и выхода из программы.

2. Разработайте математическую модель для вычисления среднего балла за сессию: средний балл равен сумме оценок по предметам обучения, деленной на число этих оценок:

СрБалл=(Физика+Математика+Информатика+Физкультура)/4

Опишите переменные (табл.1.2).

3. Разработайте интерфейс программы. Изобразите форму и определите состав элементов управления и их размещение на форме (Рис.1.12).

Рис.1.12 Пример формы

Здесь “Успеваемость” - название формы. “Введите оценки по предметам обучения”, “Физика”, “Математика”, “Информатика”, ”Физкультура”, “Средний балл” – надписи. Белые прямоугольники - поля ввода. Ввод и Выход - кнопки. Кнопка Ввод служит для выполнения расчетов, а кнопка Выход – для очистки полей ввода и завершения работы.

4. Выполним описание элементов интерфейса.

Описание свойств элементов управления выполним в соответствии с табл. 1.3.

Таблица 1.2.

Описание переменных

Наименование программы _____	**Тема проекта*: Успеваемость***		**Лист** _____ **Листов** _____
Обозначение переменной	***Имя переменной***	***Тип***	***Примечание***
Физика	F	Целая	Оценка по физике

Математика	M	Целая	Оценка по математике
Информатика	INF	Целая	Оценка по информатике
Физкультура	FIZ	Целая	Оценка по физкультуре
SB	SB	Вещественная	Средний балл

Таблица 1.3

Описание свойств элементов управления формы “Успеваемость”

Тип элемента	*Элемент*	*Свойство*	*Значение*
Форма	Forma1	Name Caption	frmForm1 Успеваемость
Надпись (метка)	Label 1	Name Caption Alignment	lblText Введите оценки по предметам обучения 2
	Label 2	Name Caption Alignment	lblFizika Физика 2
	Label 3 и т. д.	Name Caption Alignment	lblMatematika Математика 2
Поле ввода	Text 1	Name Caption Alignment	txtFiz Пусто 1
	Text 2 и т. д.	Name Caption Alignment	txtMat пусто 1
Командная кнопка	Command1	Name Caption Alignment	cmdOK OK 2
	Command2	Name Caption Alignment	cmdEnd Выход 2

5. Запишите в обработчики событий Click кнопок Ввод и Выход тексты программ для этих кнопок:

```
Private Sub CmdOK_Click ()
    Dim Физика As Integer,  Математика As Integer
    Dim  Информ. As Integer,  Физкультура As Integer
    Dim СреднийБалл As Single
     F = Val (txtFiz.Text)
     M = Val(txtMat.Text)
     INF =  Val(txtInf.Text)
     FIZ = Val(txtFizkult.Text)
     SB = (F + M + INF + FIZ)/4
     txtSredBall = Str(SB)
```

```
End Sub
-------------------------------------------------------------------------
Private Sub cmdEnd_Click()
    End
End Sub
```

Примечание: для переноса длинной строки на другую строку необходимо вставить пробел и символ подчеркивания.

6. Порядок разработки формы.

- Загрузите программу VB.
- Введите имя формы Name – “Успеваемость” и надпись для формы Caption – “Успеваемость”.
- Создайте 6 надписей, разместите их в форме в соответствии с рис. 1.12 и установите свойства согласно таблице 1.3.
- Создайте пять текстовых полей. Присвоить им имена (Name), похожие на соответствующие надписи. Например: txtF – физика, txtM – математика, txtInf – информатика, txtFiz – физкультура, txtSB – средний балл.
- Создайте две кнопки и присвойте им имена и надписи в соответствии с табл. 1.3.
- Перейдите в окно Code. Для чего щелкните дважды мышью по полю формы или щелкните мышью по кнопке Code в окне Project.
- Выбрите в левом списке объект cmdOK, а в правом списке его свойство Click. Запишите в теле этой процедуры текст программы.
- Выбрите в левом списке объект cmdEnd , а в правом списке его свойство Click и запишите текст программы.

6. Сохраните программу на диске.

7. Запустите программу.

8. Проведите отладку программы:

- введите в поля текста контрольные данные: 5, 4, 5,4
- щелкните по кнопке ОК для получения результата;
- сравните полученный результат с контрольной цифрой - 4,5.

1.2.6. Закрепление материала

1. Что понимается под объектно-ориентированным программированием?
2. Поясните основные принципы ООП.
3. Дайте определение класс, объект, свойство, событие, метод.
4. Перечислите основные свойства объекта.
5. Перечислите основные методы объектов и дайте пояснение, как они используются.
6. Как создать элемент управления на форме?
7. Для чего предназначен объект Label?
8. Поясните назначение элемента управления TextBox?
9. Поясните назначение элемента управления CommandButon?

10. Как присвоите значение полю TextBox? Приведите формат команды присвоения значения свойству Text объекта Text1 программным путем.

Задание для самостоятельной работы

Разработать программу для вычисления площади поверхности и объема тела (параллелепипеда, конуса, призмы, цилиндра и т. д.).

Справка.

Основные формулы вычисления площадей поверхности (S) и объема тел (V).

Параллелепипед: S=2(ab+bc+ca); V=abc. S – площадь; V – объем, a, b, c – стороны параллелепипеда; a, b, c – размеры ребер прямоугольного параллелепипеда.

Призма: S= M+2F; V=Fh. M – площадь боковой поверхности, F –площадь основания, h – высота.

Пирамида: S= pb/2+F; V=Fh/3, p – периметр, b – высота боковой грани (апофема),

Круговой прямой цилиндр: $S = 2\pi R(R+h)$, $V = \pi R^2 h$.

Конус: $S = \pi R(R+l)$, $V = \pi R^2 h/3$, где l – образующая конуса, $l = \sqrt{R^2 + h^2}$.

Шар: поверхность сферы $S = 4\pi R^2$, объем шара $V = 4\pi R^3/3$,

2. Разработка интерфейса прикладных программ

2.1. Принципы разработки интерфейса пользователя

В разделе 1.2.5 мы уже создавали интерфейс пользователя на примере задачи вычисления площади параллелепипеда. В данном разделе мы познакомимся с общими принципами разработки пользовательского интерфейса.

Интерфейс пользователя – связующее звено между пользователями и функциональностью приложения.

Основные пользователи приложения называются целевой аудиторией. Зная их потребности, можно достаточно легко создать пользовательский интерфейс. Хорошо продуманный интерфейс упрощает освоение приложения и работу с ним.

Базовые принципы дизайна такие, как композиция и цвет, применимы и к изображению на мониторе компьютера. Для создания эффективного интерфейса, как говорят специалисты, не надо быть художником: главное соблюдать базовые принципы. И тогда интерфейсом будет легко пользоваться. От внешнего

вида интерфейса и заложенных в него концепций прямо зависит и то, какой будет поддерживающая его программа.

Основным элементом пользовательского интерфейса любого приложения, разрабатываемого на Visual Basic является форма. На нее добавляются элементы управления и меню, которые обеспечивают доступ к функциональности приложения.

К базовым принципам дизайна относятся: *композиция; цвет; изображения и значки; шрифт; меню.*

Композиция – размещение элементов интерфейса, не только приятное на глаз, но и создающее максимальные удобства в использовании приложения. Композиция должна учитывать такие факторы, как *простота, разметка элементов, единообразие, узнаваемость, легкость восприятия.*

Простота – интерфейс не должен быть тяжеловесным. Он не должен копировать реальный объект. Необходимо использовать такие элементы, как списки, предлагать значения отдельных полей по умолчанию , группировать поля.

Разметка – часто используемые элементы должны бросаться в глаза, находясь в самых выгодных позициях; менее значимые элементы можно сделать менее заметными; самый важный элемент должен находиться в левом верхнем углу экрана.

Кнопки OK или Next обычно размещаются в нижней правой части экрана. В Диалоговых окнах кнопки располагаются справа или внизу формы.

Логически группируйте информацию по назначению или взаимосвязанности. Например, поля для имен и адресов располагают рядом. Для логического группирования элементов во многих случаях удобно пользоваться рамкой.

Единообразие или ***согласованность.*** Во всем приложении должен быть единым стиль. Рекомендуется придерживаться стиля в существующих клиентских приложениях вроде Microsoft Word. Следует ограничиваться в выборе элементов управления. Стараться использовать их по назначению.

Узнаваемость - определяется визуальными элементами, подсказывающими назначение компонентов пользовательского интерфейса. Например, поля ввода имеют рамку и белый фон, кнопки имеют трехмерное оформление.

Легкость восприятия. Элементы пользовательского интерфейса должны быть отделены достаточным пространством, чтобы они не выглядели слишком нагроможденными друг на друга и их можно было легко воспринимать. Если на форме слишком много элементов, то найти нужные будет непросто. Выравнивание элементов по горизонтали и вертикали тоже улучшает восприятие. Этой цели служат команды *Align* - выравнивание, *Make Same Size* – установка одинакового размера, *Horizontal Spacing* – горизонтальные промежутки, *Vertical Spacing* – вертикальные промежутки, *Center in Form* – центрирование меню *Format*.

Цвет - цвет оживляет интерфейс, но только если используется в меру. Важные участки пользовательского интерфейса можно выделить контрастным

цветом. Не рекомендуется сочетание таких цветов, как красный и зеленый, так как некоторые люди, страдающие дальтонизмом, не смогут прочесть красный цвет на зеленом фоне.

Количество используемых цветов лучше ограничить и придерживаться во всем приложении определенной цветовой схемы.

Рис.2.1. Пиктограммы

Изображения и значки – картинки и значки тоже оживляют приложение, но как и в случае любых других интерфейсных элементов, главное – целесообразность и чувство меры. Изображения могут передавать информацию без текста, но разные люди воспринимают это по-разному. Значки, помещаемые на панель инструментов, могут давать представление о скрытой за ними функциональности. Целесообразно использовать известные, ставшие привычными значки, например, такие, как на рис. 2.1: корзина - папка для удаленных файлов; часы – напоминание о времени; желтый прямоугольник - папка, диск с дисководом – устройство для чтения компакт дисков; принтер – печать документов и т. д.

Шрифты – несут важную информацию пользователю. Некоторые шрифты легко читаются при разных разрешениях экрана и на разных типах мониторов. Выбирайте один или два простых шрифта. Рекомендуются Arial или Times New Roman. Декоративные шрифты хорошо выглядят обычно только на бумаге.

Меню – меню и панели инструментов позволяют структурировать доступ к командам и инструментам. Должное планирование и правильный дизайн меню и панелей инструментов помогут пользователям быстрее понять назначение и возможности Вашего приложения. Если меню хорошо продумано, то уже одно знакомство с ним позволит пользователю составить представление о его возможностях.

Рекомендуется придерживаться стиля, принятого сейчас в клиентских приложениях типа Microsoft Word, Microsoft Excel и др. Рекомендуется также контролировать структуру меню и редактировать ее в зависимости от контекста работы приложения, динамически добавляя или удаляя пункты меню, включая или отключая какие-либо команды.

2.2. Форма и ее свойства

Форма (рис. 2.2.) - это средство общения программы с “внешним миром”, т.е. с пользователем. Она выполняет роль контейнера. Это значит, что в форму можно помещать другие объекты. Свойствами контейнера обладают и такие элементы управления как Frame – рамка, PictureBox - картинка и ToolBar – панель инструментов.

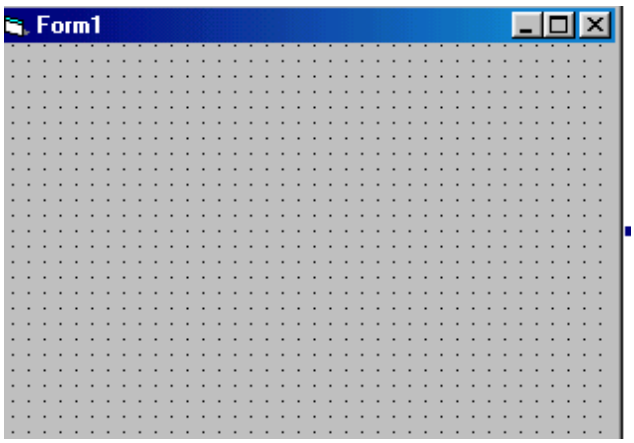


Рис. 2.2 Форма

Форма имеет все элементы стандартного окна Windows: строку заголовка, в которой указано наименование формы; кнопку системного меню – в левой части строки заголовка; кнопки свертывания, развертывания и закрытия окна – в правой части строки заголовка.

Кнопка свертывания сворачивает форму в значок (режим Minimize) и помещает его в нижней части экрана. Кнопка развертывания разворачивает окно на весь экран (режим Maximize). После развертывания окна кнопка развертывания заменяется кнопкой восстановления первоначального состояния окна. Кнопка закрытия закрывает окно.

Для удобства размещения элементов в форме, на нее можно установить сетку. По умолчанию, сетка выведена на форму. При необходимости, сетку можно удалить или изменить расстояние между ячейками. Для вывода сетки на форму используется команда ***Tools\Options*** (или комбинация клавиш [***Alt+T,O***]). После появления окна диалога *Option* (рис. 2.3) выберите закладку *General* и установите флажки *Show Grid* – показать сетку и *Align Controls to Grid* – "привязать" объекты к сетке. Привязку элементов управления к сетке можно выполнить и командой ***Format\Align\To Grid***. Расстояние между соседними ячейками устанавливается в строках ввода опции *Grid Units: Widht* – ширина и *Height* – высота. Расстояние измеряется в твипах.

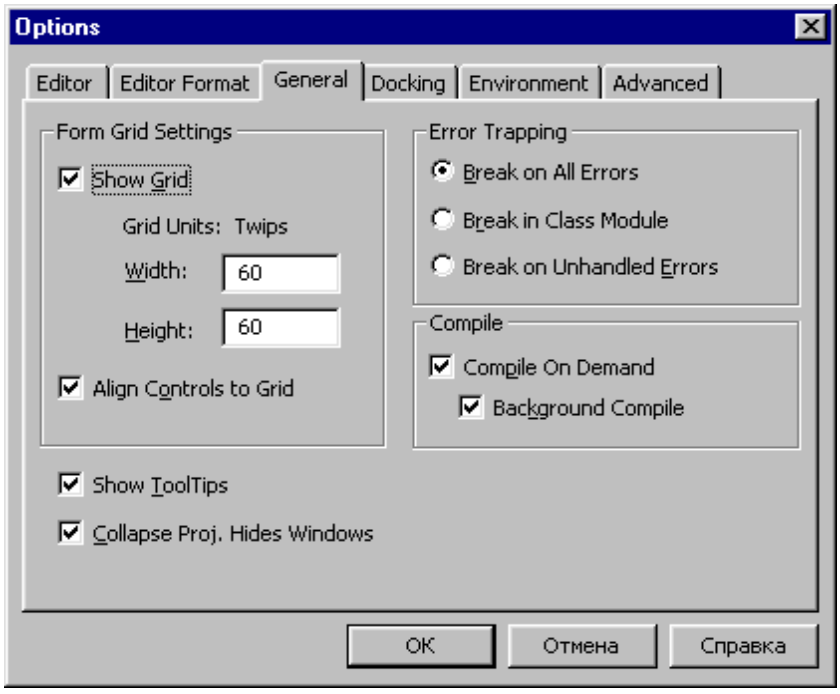


Рис.2.3. Окно диалога настройки параметров сетки

Каждая форма сохраняется в проекте в виде отдельного файла с расширением имени файла ***FRM***. Этот файл содержит описание рабочей среды и тексты программ, относящиеся к элементам управления и форме. Формы сохраняются как обычные текстовые файлы .

Формы могут быть нескольких видов: *обычные* формы, *модальные* формы и *MDI* – формы, *дочерние* и *диалоговые*.

Модальность означает, что выполнение приложения возможно только после закрытия окна формы. Установки модальности осуществляется при загрузке формы и выполняется командой

ИмяФормы.Show vbModal или ИмяФормы.Show 1

Здесь vbModal – константа Visual Basic, которая имеет значение 1.

MDI - формы служат для организации совместной работы нескольких форм, которые называются дочерними.

Диалоговые формы служат для организации взаимодействия пользователя с программой.

Свойства формы

Основные свойства формы приведены в табл. 2.1.

. Таблица 2.1

Основные свойства формы

Свойства	***Значение по умолчанию***	***Комментарий***
Name	‘’Form 1’’	Имя формы. Присваивается при разработке. Префикс *frm* (например, *frmProject1*)
Apperance	1	Внешний вид формы: 0 – плоская, 1 – объёмная.
BorderStyle	2	Внешний вид и возможность изменения размеров формы при помощи мыши.
		0 - без рамки. Нет кнопок и заголовка. Используется для экранов с заставками. Изменять размеры и перемещать нельзя.
		1 - нельзя изменять размеры. Возможны операции Minimize и Maximize. Имеется две кнопки: кнопка системного меню и кнопка закрытия окна.
		3 - толстая рамка, размеры которой менять нельзя. Используется для создания диалоговых панелей. Имеется две кнопки: кнопка системного меню и кнопка закрытия окна.
		4 – нельзя изменять размеры. Используется для вывода окна с кнопкой Close. Имеется одна кнопка закрытия окна.
		5 - то же, что и 4, но можно изменять размеры окна.
Caption	“Form 1”	Текст заголовка. Устанавливается при разработке.
ControlBox	1	1 - есть кнопка системного меню. 0 - нет кнопки системного меню (плохая идея).
Enabled	True	Доступность формы. Если значение свойства установлено True, то форма реагирует на события, False – форма не реагирует на события.

Свойства	*Значение по умолчанию*	*Комментарий*
Font	MS Sans Serif	Возможна настройка параметров шрифта с помощью окна диалога: тип, стиль, размер, эффекты, размещение. При щелчке мышью по свойству Font в строке ввода [•••] появляется значок - эллипсис (или троеточие). Если щелкнуть мышью по этому значку, то открывается окно диалога для настройки шрифтов.
Height Width	2880 3840	Высота и ширина в твипах. В одном сантиметре 567 твипов.
Icon	Согласно стандартным настройкам Windows	Определяет значок, выводимый при минимизации программы на линейку инструментов или на рабочий стол в случае обычного исполняемого файла Windows. Форму значка можно изменить загрузив новый файл с помощью окна диалога. Предварительно необходимо найти на компьютере папку с файлами, имеющими расширение *.ICO.*
Left Top	0 0	Определяют положение формы: расстояние от левого края экрана до формы, и от верхнего края экрана до формы, соответственно. Другой способ установки положения формы на экране состоит в использовании окна Form Layout. Перетащите мышью значок формы в нужное положение. Это окно работает только после запуска программы. Выведите на экран окно Layout командой ***View\Form Layout Window.*** Запустите программу и закройте окно – в окне формы Layout появится значок формы, переместите его в нужное место экрана.
Mouse Pointer, Mouse Icon	0 (None)	Установка формы курсора мыши. Имеется 17 значений. Наиболее часто используют 11 и 13. 11 – песочные часы, 13 – стрелка с песочными часами. Если установить значение свойства Mouse Pointer 99, то можно использовать любой значок.
Visible	True	Видимость формы на экране. True – форма видима, False – невидима.
WindowsState	0	Определяет вид формы после загрузки. 0 – нормальный; 1 – форма уменьшается до значка; 2 – форма развернута на весь экран, соответствует операции Maximize.
ScaleMode	1 - Twip	Позволяет изменять единицу измерения масштаба. Существует семь вариантов: 0 – собственное значение, 1 – твипы, 3 – пиксели, 6 – мм, 7 – см.
ScaleHeight ScaleWidth	3195 4680	Используют, когда установлена не стандартная единица измерения масштаба. Установка данных значений приводит к присвоению свойству

Свойства	*Значение по умолчанию*	*Комментарий*
		значений приводит к присвоению свойству ScaleMode значения 0.
ScaleLeft ScaleTop	0 0	Описывают значения координат левой и верхней рамок формы относительно экрана.
ForeColor BackColor	Согласно настройкам Windows	Цвет текста и цвет фона, соответственно. Можно установить собственные значения, выбрав из списка. Закладка Palette выводит панель палитры. Закладка System выводит список текущих значений цвета различных элементов Windows.

События формы

Формы могут распознавать более 20 различных событий.

События **Click**, **DoubleClick** служат для обработки одиночного и двойного щелчка мыши.

Как уже известно, с каждым событием , связана процедура - обработчик событий. События генерируются в ответ на действия пользователя – внешние события или генерируются системой - системные события. Например реакция на щелчок мыши по форме реализуется в виде процедуры обработки события Click формы:

```
Private Sub Form_Click ()
    <текст программы>
End Sub.
```

Основные события формы, как правило, обрабатываются в таком порядке: Initialize, Load, Activate, Deactivate, Query Unload, Unload, Terminate.

Событие **Initialize** используется, обычно, для подготовки приложения к работе. В обработчике этого события переменным присваивают начальные значения, расставляют элементы управления на форме и масштабируют их. Данное событие возникает в момент создания экземпляра формы (до её загрузки или отображения). Однако оно генерируется лишь один раз в течение всего сеанса работы приложения.

Внимание! Если какой-то код должен выполняться несколько раз, то его нельзя помещать в обработчик данного события.

Событие Initialize генерируется, например, при загрузке формы, её показе, а также при возвращении значения свойства (то есть какому-то свойству объекта закрытой формы присваивается программным путем некоторое значение) или вызове метода, определённого в форме. Например, событие Initialize генерируется при вводе команд:

```
frmMyForm.Show
```

или

```
Load  frmMyForm
```

Переменные уровня формы доступны всем процедурам внутри программного модуля данной формы. После инициализации эти переменные существуют, пока выполняется приложение, даже если соответствующая форма выгружена.

Событие **Load** используется для выполнения каких-либо действий до вывода формы на экран. Оно также позволяет присвоить исходные значения свойствам формы и её элементам управления. Это событие возникает при каждой загрузке формы в память. При первой загрузке событие Load следует за событием Initialize. Событие Load генерируется также в результате применения оператора Load или Show, а также после ссылки на свойства, методы или элементы управления не загруженной формы.

Пример 2.1.: Добавление элементов к списку

```
Private Sub Form_Load ( )
Dim i As Integer
For i=2 to 5
     Load txtText1(i)
     txtText1(i).Text = “Текстовое окно № ”+Str$(i)
Next i
End Sub
```

События **Activate /Deactivate**. Эти события возникают при работе с несколькими формами. Событие *Activate* возникает, когда фокус ввода переходит на данную форму от другой формы того же приложения. При этом форма должна быть видима. Например, форма, загруженная оператором Load, остаётся невидимой, пока не примените метод Show или не установите свойство Visible формы как True. Событие Activate генерируется от события Got Focus - установка фокуса. Событие *Deactivate* происходит тогда, когда фокус ввода переходит с данной формы на другую форму этого же приложения.

Событие **QueryUnLoad** полезно, если нужно узнать, как именно пользователь закрывает форму. Данное событие происходит перед событием *Unload*. Событие QueryUnLoad происходит в следующих случаях:

- из системного меню формы выбрана команда Close;
- в программе выполняется оператор Unload;
- закрывается дочерняя MDI-форма, так как закрывается основная MDI-форма.

Пример 2.2. Использование события QueryUnload для контроля за закрытием формы.

```
Private Sub Form _QueryUnload(Cancel As Integer, UnloadMode As Integer)
   If   Unload Mode<>vbFormCode Then
          MsgBox ”Если хотите выйти, то нажмите кнопку Close’’
          Cancel = True          ‘форма остаётся открытой’.
   End If
End Sub.
```

Чтобы предотвратить выгрузку формы, аргументу *Cancel* присвоено значение *True*.

Событие **Unload** генерируется перед событием *Terminate*. Обработчик события Unload можно использовать для проверки того, надо ли выгрузить форму или для определения операций, выполняемых при выгрузке формы. Можно также включить программу проверки уровня формы, необходимую для закрытия формы или сохранения данных в файле. В обработчик события Unload можно добавить оператор End – он гарантирует выгрузку всех форм до завершения программы.

Присвоение аргументу Cancel любого ненулевого значения предотвращает удаление формы, но не запрещает другие события вроде выхода из среды Windows. Чтобы не допустить выхода из Windows, необходимо использовать событие QueryUnload.

Событие **Terminate** генерируется, когда из памяти удаляются все ссылки на экземпляр формы. Чтобы убрать из памяти переменные этой формы и освободить занимаемые системные ресурсы, присвойте объектной переменной формы значение Nothing:

```
Set frmMyForm = Nothing
```

Для всех объектов, кроме классов, событие Terminate генерируется после события Unload.

Методы формы

Метод выполняет над объектом какую-либо операцию. Знание методов форм позволяет разработать приложение, эффективно использующее системные ресурсы компьютера. Для управления формами в программах на Visual Basic предназначены методы: Load, Unload, Hide, Show.

Оператор ***Load*** инициализирует и загружает форму в память, не выводя ее на экран. Любая ссылка на форму вызывает автоматическую загрузку ресурсов этой формы, если они еще не загружены в память:

```
Load frmMyForm      'форма загружается, но не выводится на экран'
```

Оператор ***Unload*** удаляет форму из памяти. Для ссылки на текущую форму можно использовать константу ***Me***.

```
Unload frmMyForm
```

Или

```
Unload Me
```

Оператор ***Hide*** – убирает форму с экрана, не удаляя ее из памяти. Хотя элементы управления скрытой формы не доступны пользователю, к ним можно обращаться программно. Когда форма скрыта, пользователь не может взаимодействовать с соответствующей частью приложения.

Если на момент вызова метода Hide форма еще не загружена в память, она загружается, но на экране не появляется.

```
FrmMyForm.Hide
```

Или

```
Me.Hide
```

Оператор ***Show*** выводит форму на экран. Если на момент вызова оператора форма еще не загружена в память, то VB сначала вызывает метод Load. Метод Show позволяет показывать формы либо как модальные, либо как не модальные. Если на экран выводится модальная форма, то весь ввод с клавиатуры или с помощью мыши будет относиться только к модальной форме и работают лишь процедуры из модуля этой формы.

а) показать форму:

```
frmMyForm.Show
```

или

```
Me.Show
```

б) показать модальную форму

```
frmVvodData.Show vbModal
```

или

```
frmVvodData.Show 1
```

Управление формами

Добавление формы в проект

Добавление формы в проект осуществляется командой *Project, Add Form*. После ввода команд появится диалоговое окно ***Add Form***. Щелкнуть мышкой по значку *Form*, затем – по кнопке *Open*.

В проект будет добавлена новая форма.

Установка стартовой формы

По умолчанию первая форма в проекте считается стартовой. Чтобы сменить стартовую форму необходимо:

- выбрать из меню ***Project*** команду ***Project1 Properties***. Появится диалоговое окно *Project1 Properties*;

- раскройте список ***Startup Object***, выберете имя формы, которую хотите сделать стартовой и щелкните кн. ***Ок***.

Печать формы

Для простейшего вывода на печать всего содержимого формы имеется команда ***PrintForm.***

```
Private Sub Form_Click ()
    PrintForm
End Sub.
```

2.3. MDI – форма

Создание MDI-формы

При разработке приложений для повышения их функциональности и удобства использования разрабатывается много форм. Организацию взаимодействия с этими формами удобно осуществлять с помощью системы меню, помещенных в одну из форм – MDI-форму.

MDI–форма - это многодокументный интерфейс, предназначенный для организации взаимодействия нескольких независимых форм.

MDI-форма является родительской формой или контейнером для других (дочерних) форм. В MDI-форме можно размещать только элементы управления, имеющие свойство выравнивания, такие как окно с рисунком PictureBox, картинка – Image. Можно поместить непосредственно в MDI-форму фоновое изображение.

MDI-формы применяются чаще всего для обслуживания однородных форм. Примерами их использования являются редактор Word или электронная таблица Excel. Но они с успехом могут применяться для организации взаимодействия и разнотипных форм.

Создание MDI-формы осуществляется командой ***Project\Add MDI Form.***

При запуске программы с MDI-формой программа автоматически устанавливает размеры дочерних окон, которые могут оказаться меньше, чем при настройке и поэтому часть объектов активной формы может быть невидимой. Чтобы избавиться от этого недостатка, необходимо в обработчике события Load каждой формы явно указать размеры и положение формы в окне, например:

```
Private Sub Form_Load()
    Me.Height = 2745
    Me.Width = 3090
    Me.Top = (MDIForm1.ScailHeigth – Me.Heigth) / 2
    Me.Left = (MDIForm1.ScailWidth – Me.Width) / 2
End Sub
```

Если одна из форм максимизируется, то и все последующие открываемые формы будут развернуты на все окно. Чтобы этого не происходило, необходимо при закрытии максимизированной формы приводить ее размеры в нормальное состояние. Для этого в обработчик события кнопки ВЫХОД надо поместить следующий код:

```
Private Sub mnuExit_Click()
    Me.WindowState = 0
    Unload Me
End Sub
```

Работа с дочерними формами

Чтобы обычная форма стала подчиненной MDI-форме (дочерней формой), значение ее свойства **MDIChild** необходимо установить в True.

Дочерние формы показываются с помощью метода **Show**, например:

```
Private Sub mnuVvod_Click()
    frmVvod.Show
End Sub
```

Дочерние формы могут иметь собственное меню. При развертывании дочерней формы ее заголовок заменит заголовок родительской формы, а меню дочерней формы заменит меню родительской формы.

Каждое приложение, имеющее MDI- форму должно иметь пункт меню ***Window*** (Окно), позволяющее пользователю выводить дочерние окна каскадом (*Cascade*) или в виде мозаики (*Tile*). При размещении каскадом не минимизированные формы размещаются так, чтобы каждая предыдущая форма немного выступала из - за следующей. При размещении в виде мозаики формы могут размещаться горизонтально (*Horizontal*) или вертикально (*Vertical*). В первом случае каждая не минимизированная форма принимает ширину, равную ширине родительской формы, во втором случае каждая не минимизированная форма принимает высоту, равную высоте родительской формы. Меню Window должно содержать для этой цели список открытых дочерних форм. Чтобы список открытых дочерних форм формировался автоматически, необходимо при разработке меню для элемента меню *Window* установить флажок *WindowList*.

Управление размещением дочерних форм MDI-форма осуществляет с помощью метода ***Arrange***. Кроме того, VB имеет четыре константы для управления окнами: ***VbCascade*** – размещение окон каскадом, ***VbTileHorizontal*** – размещение окон горизонтально, ***VbTileHorizontal*** - размещение окон вертикадьно, ***VbArrangeIcons*** – пиктограммы всех минимизированных окон располагаются по нижнему краю родительской формы.

Для управления размещением открытых окон в меню Window требуется ввести элементы меню второго уровня: каскадом, горизонтально, вертикально, - а затем поместить в обработчики события Click этих пунктов меню объектный код следующего вида:

```
имяMDI-формы.Arrange константаVisualBasica
```

Например:

```
Sub mnuCascade_Click
      MDIForm1.Arrange VbCascade
End Sub
Sub  mnuHorizontal_Click
      MDIForm1.Arrange VbTileHorizontal
End Sub
Sub  mnuVertical_Click
      MDIForm1.Arrange VbTileVertical
End Sub
```

Свертывание открытых окон в значок осуществляется щелчком мыши по кнопке закрытия окна.

2.4. Разработка меню пользователя

Многоуровневые меню

Разработка меню позволяет сделать приложение с более дружественным интерфейсом. Практически любая программа, написанная для Windows, содержит многоуровневые меню или меню, в которых команды сгруппированы по логическому назначению. При запуске программы в строке меню диалогового окна обычно видны только элементы верхнего уровня. При щелчке мышью по пункту меню открывается меню второго уровня и т.д.

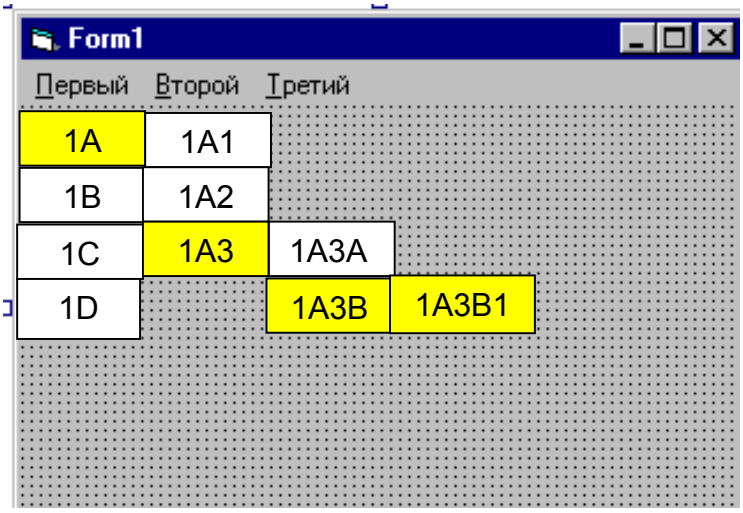


Рис. 2.4. Структура меню

VB позволяет иметь до шести уровней вложенности меню. Большое количество уровней тоже не совсем удобно. Рекомендуется использовать не более 3-х уровней вложенности. На рис. 2.4. представлено четырех уровневое меню. Главное меню: Первый, Второй, Третий. Меню первого уровня для меню Первый –1А, меню второго уровня – 1А1-1А3; меню третьего уровня – 1А3А – 1А3В, меню четвертого уровня – 1А3В1.

Средства для разработки меню

Меню любой конфигурации можно создать с помощью командных кнопок или с помощью текстовых полей. Пример меню сложной структуры приведен на рис. 2.4. Для создания удобного меню с большой функциональностью потребуется приложить немало усилий. Однако эту задачу можно облегчить, если воспользоваться стандартными средствами VB.

Visual Basic 6.0 имеет удобное средство для разработки меню – редактор ***Menu Editor***, которое вызывается командой ***Tools \ Menu Editor*** или комбинацией клавиш ***CTRL – E.***

Диалоговое окно редактора Menu Editor приведено на рис. 2.5.

Строка ввода ***Caption*** служит для ввода наименования пункта меню, выводимого на экран. После нажатия клавиши OK или щелчка мыши введенное наименование появляется в окне редактора. Если перед одним из символов наименования пункта меню поставить символ “&“ - амперсенд, то появится возможность вызывать пункт меню по нажатию данной клавиши (горячей клавиши). Символ, перед которым стоит знак “&”, подчеркивается.

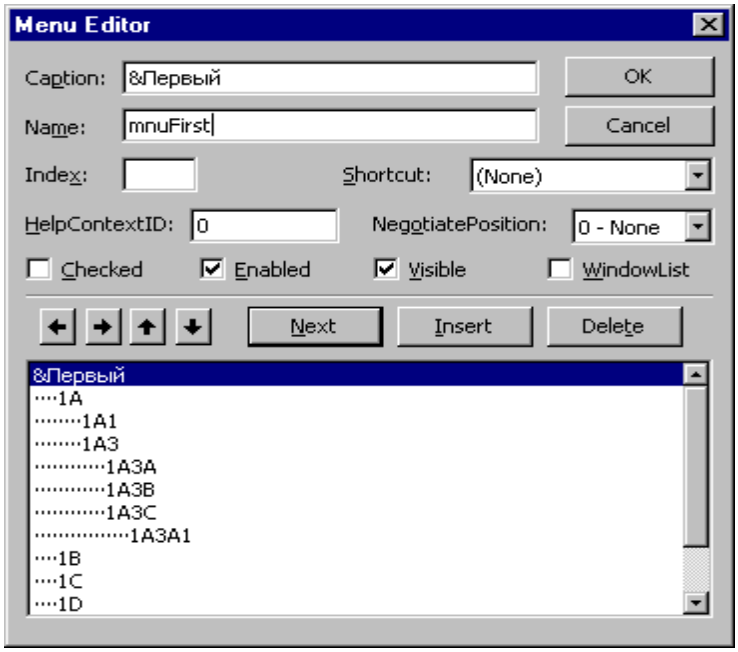


Рис. 2.5. Редактор меню Menu Editor

Строка ввода ***Name*** служит для ввода имени пункта меню, которое будет использоваться в программе для обработки событий. Перед именем пункта меню рекомендуется ставить префикс mnu, например, mnuFile. Программа не позволит пользователю выйти из редактора, пока всем пунктам меню не будут присвоены имена.

Окно ***Index*** используется в том случае, если имеется несколько пунктов меню с одинаковыми именами или надо сделать пункты меню частью массива элементов управления.

Окно ***Shortcut*** позволяет назначить каждому пункту меню комбинацию клавиш для быстрого вызова команд меню: Ctrl + клавиша, Shift + клавиша и др. При открытии списка появится список быстрых клавиш, из которого надо выбрать нужный.

Окно ***HelpContexID*** обеспечивает ввод идентификатора, который используется в электронной справочной системе для выдачи контекстно-зависимой справки по вашему приложению.

Окно ***NegotiatePosition*** – служит для определения способа отображения меню на экране, когда один из связанных объектов приложения активен: не показывать, слева, справа, по центру.

Флажок ***Checked***. Если значение данного свойства равно True, то возле соответствующего пункта меню появляется галочка. Это сигнализирует о том, что соответствующий параметр выбран.

Флажок ***Enabled***. Определяет доступность данного пункта меню. Если его значение False , то пункт меню недоступен.

Флажок ***Visible***. Данное свойство определяет, будет ли виден на экране соответствующий пункт меню. При разработке приложения можно предусмотреть несколько наборов меню, которые должны появляться на экране в соответствующие моменты времени. Например, если в приложении не открыто ни одно окно, меню Window не должно появляться на экране.

Флажок ***WindowList***. Данный флажок служит для автоматического формирования списка открытых окон, являющихся элементами многодокументного интерфейса (MDI). Установка этого флажка для элемента управления меню верхнего уровня приведет к тому, что в данном элементе будет автоматически формироваться динамический список всех активных дочерних окон.

Кнопка ***Next*** предназначена для добавления новых пунктов меню.

Кнопка ***Insert*** позволяет вставить поле для ввода нового пункта меню.
Кнопка ***Delete*** служит для удаления выделенного пункта меню.

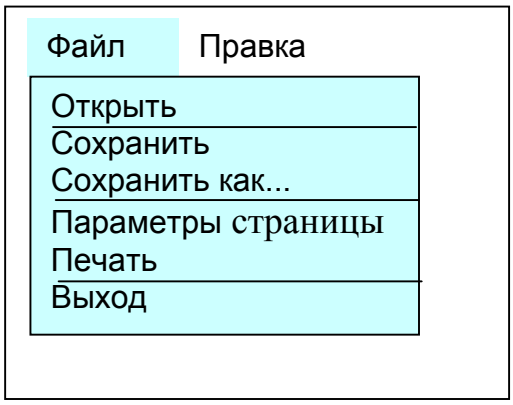


Рис. 2.6.Группировка пунктов меню

Группировка элементов списка пункта меню

Пункты меню, близкие по назначению целесообразно группировать, отделяя их от других пунктов меню горизонтальной чертой – разделительная линия (Separator Bar) (рис.2.6.). Эта черта создается так же, как и другие элементы управления меню, но вместо наименования пункта меню (свойство Caption) вводится дефис (-). Имя данному пункту меню можно присвоить произвольно, например, mnuRaz1 и т. д.

Управление размещением пунктов меню

В редакторе *Menu Editor* изменение уровня вложенности элемента меню осуществляется с помощью кнопок ➡ и ⬅. Первая кнопка понижает уровень, вторая – повышает. Кнопки ⬆ и ⬇ служат для перемещения выделенного пункта меню по вертикали. Уровень вложенности элемента управления при этом не изменяется.

Взаимодействие меню MDI-формы и дочерних форм

Дочерние формы, также как и MDI-форма (родительская), могут иметь меню, созданные с помощью редактора *Menu Editor*. При открытии дочерней формы, содержащей такое меню, оно замещает меню родительской формы. При этом часть пунктов меню MDI-формы может выступать из-под меню дочерней формы, что может вызвать недоразумения. Поэтому программист должен позаботиться о том, чтобы сделать пункты меню родительской формы невидимыми, пока открыта дочерняя форма, а после ее закрытия сделать их снова видимыми.

Контекстное меню

Контекстное (всплывающие) меню появляется, обычно, после щелчка правой кнопкой мыши по объекту. Порядок разработки контекстного меню практически ничем не отличается от порядка разработки обычного меню. Отличие состоит в том, что для меню верхнего уровня свойство Visible устанавливается в False. То есть, в исходном состоянии меню верхнего уровня, а следовательно, и подчиненные ему элементы меню нижних уровней, невидимы.

Для вызова контекстного меню используется метод ***PopupMenu***. Синтаксис команды вызова всплывающего меню:

ИмяФормы.PopupMenu ИмяЭлементаМеню

Команда вызова контекстного меню записывается в обработчик события нажатия кнопки мыши формы ***MouseUp*** для соответствующего пункта меню.

2.5. Упражнения: разработка меню пользователя

Задача 2.1.. Требуется разработать меню для исследования свойства BorderStyle формы.

Порядок работы.

1. Запустите программу Visual Basic и откройте новый проект Standard EXE (если форма не открывается автоматически при запуске программы).
2. Опишите пункты меню

Таблица 2.2

Описание элементов меню

Элемент меню	*Свойство*	*Значение*
Форма BorderStyle-0	Caption Name	Форма BorderStyle-&0 mnuForm0
Форма BorderStyle-1	Caption Name	Форма BorderStyle-&1 mnu Form1
Форма BorderStyle-2	Caption Name	Форма BorderStyle-&2 mnu Form2
Форма BorderStyle-3	Caption Name	Форма BorderStyle-&3 mnu Form3
Форма BorderStyle-4	Caption Name	Форма BorderStyle-&4 mnu Form4
Форма BorderStyle-5	Caption Name	Форма BorderStyle-&5 mnu Form5
Окно	Caption Name	&Окно mnuWindow
Каскад	Caption Name	&Каскад mnuCascad
Горизонтально	Caption Name	&Горизонтально mnuHorizont
Вертикально	Caption Name	&Вертикально mnuVertical

3. Добавьте к проекту MDI-форму командой Project\Add MDI Form
4. Создайте меню пользователя первого уровня из шести пунктов (по числу значений свойства BorderStyle формы).

Введите команду Tools\ MenuEditor или щелкните по кнопке MenuEditor стандартной панели инструментов. В поле Caption введите значение свойства Caption первой формы. Набираемый текст будет отображаться в окне просмотра.

Нажмите клавишу Tab для перехода в поле ввода Name и введите значение свойства Name первой формы.

Щелкните по кнопке Next или нажмите клавишу Enter для завершения ввода данных и перехода к описанию следующего пункта меню.

Для завершения работы по созданию меню щелкните кнопку OK.

5. Добавьте пять форм в проект командой Project\Add Form.

Простым формам присвойте имена frmForm0 – frmForm5. Свойству MDI-Child всех простых форм присвойте значение True.

6. Для элементов управления меню предусмотрено только одно событие – Click. Оно возникает, когда пользователь с помощью мыши или клавиатуры выбирает нужный пункт меню.

Щелкните мышью по первому пункту меню – откроется обработчик события Click выбранного пункта меню:

```
Private Sub  mnuForm0_Click()

End  Sub
```

В обработчик событий Click пунктов меню запишите текст программы для вызова формы:

```
Private Sub mnuForm0_Click()
    frmForm0.Show
End Sub
```

7. Сделайте родительскую форму стартовой: введите команду Project\ Project1 Properties... и в окне Startup Object установите значение MDIForm1.

8. Добавьте в меню пункт Окно и установите для этого пункта флажок WindowList.

Добавьте меню второго уровня для пункта меню Окно: Каскад, Горизонтально и Вертикально.

Пункты меню второго уровня создаются так же, как и пункты меню первого уровня. Для изменения уровня пункта меню выделите его и щелкните кнопку с направленной вправо стрелкой. Для перевода пункта меню на более высокий уровень выделите его и щелкните кнопку с направленной влево стрелкой. Для изменения положения пункта меню в структуре выделите его и переместите вверх или вниз с помощью соответствующих кнопок диалоговой панели.

9. Запишите для каждого пункта меню второго уровня Каскад, Горизонтально и Вертикально тексты программ см. раздел 2.3.

10.Проверьте правильность вызова форм в соответствии с их типами и управление расположением окон (откройте все окна и расположите их каскадом, горизонтально, вертикально, сверните в значок).

11.Сохраните программу на диске.

Задача 2.2. Тебуется разработать меню для вычисления площадей поверхностей и объемов геометрических фигур: параллелепипеда, призмы, конуса, цилиндра, шара. Меню должно содержать не более трех уровней вложенности.

Порядок работы.

1. Изобразите структуру будущего меню (рис. 2.7).

2. Составьте таблицу (табл. 2.3) для удобства описания элементов меню.

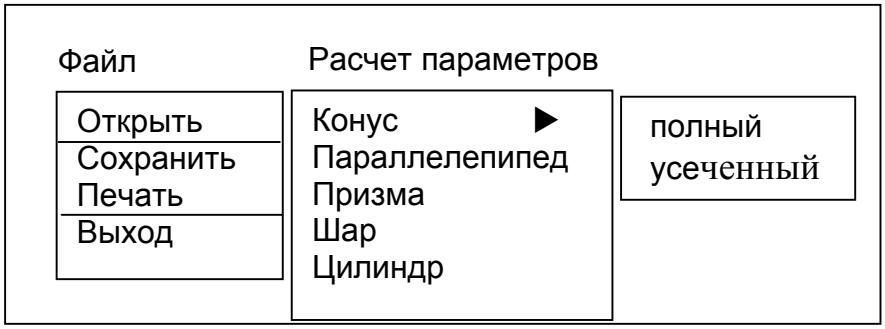


Рис. 2.7. Структура меню

Таблица 2.3

Описание элементов меню

Элемент меню	***Свойство***	***Значение***
Файл	Caption Name	&Файл mnuFile
Открыть	Caption Name	&Открыть mnuOpen
Разделительная полоса	Caption Name	- mnuRaz1
Сохранить	Caption Name	&Сохранить mnuSave
Разделительная полоса	Caption Name	- mnuRaz2
Печать	Caption Name	&Печать mnuPrint
Выход	Caption Name	&Выход mnuExit
Расчет параметров	Caption Name	&Расчет параметров mnuCalc
Конус	Caption Name	&Расчет параметров mnuKonus
полный	Caption Name	&полный mnuFull
усеченный	Caption Name	&усеченный mnuBrif
Параллелепипед	Caption Name	&Параллелепипед mnuParall
Призма	Caption Name	П&ризма mnuPrizma
Шар	Caption Name	&Шар mnuShar
Цилиндр	Caption Name	&Цилиндр mnuZilindr

3. Создайте меню.

4. Напишите текст программ для обработки команд меню.

Войдите в требуемый пункт меню, например Файл, и щелкните по нужной команде, например, *Выход*

Напишите в теле процедуры текст программы, например:

```
‘ объявление типов переменных
Dim nTemp As Inteqer, sTemp As Inteqer
sTemp = ”Выйти из программы”
‘вывод запроса
nTemp = MsqBox (sTemp, VbYesNo, ’’Пример обработки команд меню”)
If nTemp = VbYes Then
    End
End If
```

Для загрузки формы расчета параметров одной из фигур, например, полного конуса необходимо написать следующий текст программы:

```
Private Sub  mnuFull_Click
    frmKonusFull.Show
End  Sub
```

Здесь frmKonusFull – имя формы для расчета параметров полного конуса.

Задача 2.3. Требуется разработать контекстное меню для изменения высоты шрифта.

Рассмотрим для примера меню настройки высоты шрифта для формы. Откроем новую форму и создадим меню следующей структуры (элементы меню отделены друг от друга точкой с запятой): Высота шрифта; ...8; ...12; ...14; ...18; ...24. Для элемента меню “Высота шрифта” введем имя mnuShriftHeight и снимем флажок Visible. Пунктам меню второго уровня присвоим имена mnu8, mnu12 и т. д.

Напишем текст программы.

В обработчик события Click формы запишем оператор печати текста:

```
Private Sub Form_Click()
    Print "Привет"
End Sub
```

В обработчик события MouseUp формы запишем программу проверки условия нажатия правой клавиши мыши. Системная переменная Button возвращает код нажатой клавиши мыши и ее значение сравнивается с константой vbRightButton – код правой клавиши мыши.

```
Private Sub Form_MouseUp(Button As Integer, _
    Shift As Integer, X As Single, Y As Single)
    If Button = vbRightButton Then
        Form1.PopupMenu mnuShriftHeight
    End If
End  Sub
```

В обработчики событий Click пунктов меню второго уровня запишем следующие тексты программ:

```
Private Sub mnu8_Click()
    Form1.FontSize =8
End Sub
Private Sub mnu14_Click()
    Form1.FontSize = 14
End Sub
```

```
Private Sub mnu12_Click()
    Form1.FontSize = 12
End Sub
Private Sub mnu18_Click()
    Form1.FontSize = 18
End Sub
```

```
Private Sub mnu24_Click()
    Form1.FontSize = 24
End Sub
```

Запустим программу. При щелчке левой клавишей мыши по форме появляется текст “Привет” с текущим значением высоты шрифта. Щелкнем правой клавишей мыши по форме – появится контекстное меню формы со списком высоты шрифта. Выберем нужную высоту шрифта щелчком мыши – появится текст с выбранной высотой шрифта.

2.6. Закрепление материала

1. Назовите базовые принципы разработки пользовательского интерфейса.
2. Что входит в понятие композиция при разработке интерфейса?
3. Перечислите основные свойства формы.
4. Перечислите основные события формы
5. В какой последовательности генерируются события формы?
6. Для какой цели используются обработчики событий Load и Unload?
7. Перечислите основные методы формы и их назначение.
8. Что такое MDI-форма, чем она отличается от обычной формы?
9. Как сделать форму дочерней?
10. Как установить стартовую форму?
11. Как разрабатывается меню пользователя?
12. Расскажите назначение окон ввода и флажков окна диалога Menu Editor.
13. Как используется меню?
14. Чем отличается порядок разработки контекстного меню от порядка разработки обычного меню?
15. Как используется контекстное меню?
16. Как обеспечить управление открытыми окнами программы?
17. Что необходимо сделать, чтобы список открытых форм формировался автоматически?

Задание для самостоятельной работы

1. Разработать меню пользователя для управления проектом. Меню должно обеспечивать сохранение программы на диске, загрузку файлов, вывод информации на печать, редактирование текста.
2. Разработать контекстное меню для установки цвета формы.

3. Программирование в Visual Basic

3.1. Основные понятия о программировании в среде VB

3.1.1. Среда программирования

Структура проекта

Все программы, работающие в среде Windows, принято называть ***приложениями***.

Каждое приложение, разрабатываемое в среде VB, называется ***проектом***. Проект состоит обычно из форм и модулей, с которыми связаны тексты программ. Текст программы проекта состоит из множества подпрограмм, оформленных в виде процедур обработчиков событий и пользовательских процедур (рис. 3.1.).

Проекты, формы и модули сохраняются в отдельных файлах и могут самостоятельно загружаться и добавляться к другим проектам, с помощью команд ***Project \ Add Project, Project \ Add Form***.

Текст программы, применяемый во всех формах, можно разделить на несколько модулей, хранимых раздельно.

Структура проекта отображается в окне *Project* в виде древовидной

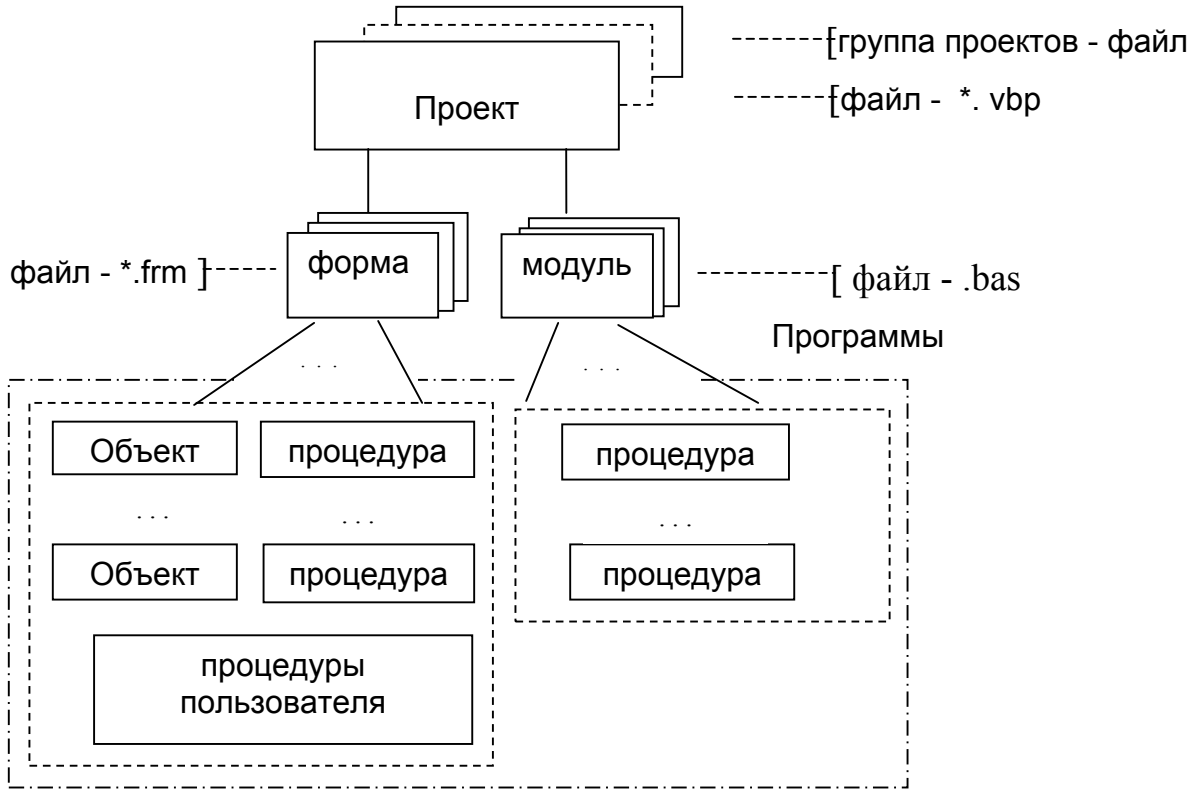


Рис. 3.1.Структура проекта

структуры (рис. 1.3, 3.1).

Когда в VB начинается подготовка больших проектов, очень остро встает вопрос о повторном использовании текста программы. По этой причине целесообразнее хранить процедуры и функции в отдельных модулях, чем подключать их к форме.

Различают стандартные модули, модули классов и модули пользовательских элементов управления.

В ***стандартных модулях*** размещается текст программ, доступ к которым должен иметь весь проект. Эти программы применяются для повторного использования. *Стандартные модули не имеют визуальных компонентов.* Новый модуль создается при помощи команды ***Project \ Add Module***. Для загрузки существующего модуля необходимо ввести команду ***Project \ Add Module*** и далее открыть закладку ***Existing*** или использовать команду ***Project \ Add File***. По умолчанию файлы стандартных модулей имеют расширение ***.bas.***

Окно Программы (Code)

Окно Программы (рис. 3.2) – часть среды разработки VB. Здесь вводится и редактируется текст программы, который позволяет программе реально выполнять какую-либо работу. Эту часть среды разработки VB с полным основанием можно назвать ***средой программирования***.

Каждая форма и модуль имеет собственное, связанное с ней окно программы, которое открывается двойным щелчком мыши по форме или по имени файла в окне проекта. В окне программы расположены два раскрывающихся списка: список объектов и список свойств и процедур. ***Список объектов*** содержит список всех элементов, расположенных на форме. ***Список свойств*** содержит список всех процедур обработчиков событий выделенного объекта и процедур данной формы.

Текст программы формы или модуля состоит из программы раздела Главная (General) и программ обработчиков событий, пользовательских процедур и функций, разделенных пунктирной линией. Эта линия формируется автоматически при создании новой процедуры. Раздел Главная служит для объявления типов переменных и написания пользовательских процедур и функций доступных всем процедурам формы.

На рис. 3.2 в разделе Главная записан один оператор – Option Explicit, а ниже записаны коды обработчиков событий Click и Load формы. Разделительная полоса ***Split Bar*** (ее маркер находится над вертикальной полосой прокрутки) служит для деления окна редактирования на две части по горизонтали. В этом случае в каждом окне можно просматривать разные участки программы, повышается удобство отладки и написания программы. Установка и удаление разделительной полосы осуществляется с помощью мыши. Для установки разделительной полосы зацепите ее мышью и перетащите в нужное место экрана.

Чтобы удалить разделительную полосу передвиньте ее мышью к верхнему краю окна.

Две кнопки слева от горизонтальной полосы прокрутки служат для переключения режима просмотра модулей. Первая кнопка (Procedure View) устанавливает режим просмотра одного модуля, вторая (Full Module View) – просмотр нескольких модулей. Последний режим установлен по умолчанию.

Для редактирования текста можно использовать все стандартные функции редактирования Windows (выделение текста, копирование, удаление, вставка). Кроме того, в VB применена новая технологий *IntelliSense* фирмы Microsoft. Она обеспечивает:

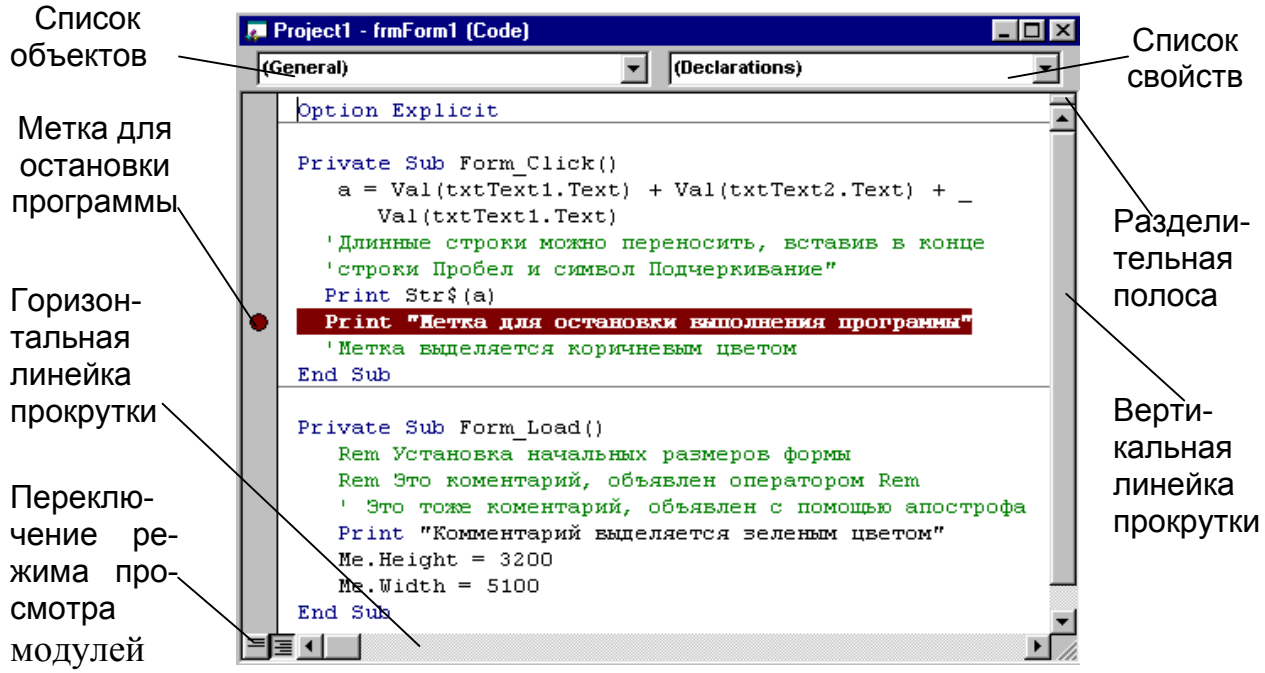


Рис. 3.2. Окно кода

- контроль вводимого кода. Размер кода не должен превышать установленные размеры (64 Кбайт). Выводится всплывающее окно с информацией о текущем объекте;
- выдачу информации о синтаксисе текущего оператора VB (Quick Info). При вводе ключевого слова, за которым следует пробел или открывающаяся скобка, на экране появляется подсказка, где рассказывается о синтаксисе данного элемента;
- вывод на экран списка всех свойств и методов этого объекта, после того как введена точка в конце названия объекта (List Properties/ Methods). Методы выделяются при этом зеленым цветом;
- после ввода служебного слова As при объявлении типов переменных на экран выводится список типов переменных.
- получение списка возможных констант (Available Constants) после ввода знака равно после имени объекта.

Указанные, выше функции можно включить или отключить с помощью команды ***Tools\Options***. После ввода команды выберите закладку ***Editor*** и установите или снимите флажки ***Auto Quick Info*** и ***Auto List Members*** в группе ***Code Setting***.

Комментарии вводятся с помощью оператора ***Rem*** или апострофа. Комментарий выделяется зеленым цветом.

При вводе выражений осуществляется контроль синтаксиса. При наличии ошибок после нажатия клавиши Enter или перемещения курсора на другую строку, строка с ошибкой выделяется красным цветом. Контроль синтаксиса может быть отключен командой ***Auto Syntax Check*** вкладки ***Editor*** диалогового окна ***Options*** меню ***Tools***.

Программная строка может быть достаточно длинной, что неудобно при ее просмотре и редактировании, а также при выводе текста программы на печать. Для переноса программной строки в месте раздела необходимо ввести пробел и знак подчеркивания « _». В одной программной строке допускается до 10 переносов. Максимальная длина строки – 1023 символа.

Допускается размещать в одной программной строке несколько операторов, разделяя их двоеточием.

Для редактирования текста можно использовать меню ***Edit***, контекстное меню, вызываемое щелчком правой клавиши мыши, а также закладку ***Editor Format*** окна ***Options*** меню ***Tools***.

VB имеет мощные средства отладки. Отметим некоторые из них:

- установка метки останова. Щелкните мышью по рамке окна напротив оператора, где вы хотите остановить программу, - на рамке появляется метка и вся строка выделяется коричневым цветом. Для отмены метки щелкните по ней мышью. Метка не устанавливается на пустую строку и на строку комментария;
- после остановки программы можно просмотреть значения всех интересующих вас переменных и тем самым установить причину прерывания программы или неправильной ее работы. Значения переменных высвечиваются при зависании на них мыши.

В синтаксисе языка Visual Basic используются операторы, функции, переменные и константы.

Операторы – синтаксические конструкции, которые управляют вычислительным процессом. Операторы образуют текст программы.

Функции – вычислительные процедуры, предназначенные для выполнения наиболее часто используемых вычислительных , логических операций.

Переменные – именованные области памяти, предназначенные для хранения данных. Значения переменных могут изменяться в процессе выполнения программы.

Константы – постоянные величины, или поименованные области памяти, предназначенные для хранения данных. Значения констант не изменяются в процессе выполнения программы.

3.1.2. Переменные

Переменная – это именованная область памяти, предназначенная для хранения данных.

Значения переменных могут изменяться в процессе выполнения программы. Каждой переменной присваивается идентификатор – имя. Имя переменной начинается с буквы и может содержать до 255 символов. В имени не допускаются пробелы, нельзя использовать также символы “.” и “&”. Имя переменной рекомендуется начинать с одно символьного ***префикса*** (табл. 3.1), характеризующего тип хранимых данных. Тип переменной устанавливается при объявлении переменной или определяется программой автоматически по содержанию присвоенной ей информации. Тип переменной задает определенный формат и размер содержимого переменной.

Способы объявления переменных

Чтобы не было путаницы с типами данных и проблем при поиске ошибок, рекомендуется обязательно объявлять тип переменной. Тип переменной можно объявить несколькими способами:

1. Не явно с помощью специального символа – ***суффикса***, записанного после имени переменной (табл. 3.1). Спецсимвол следует указывать только при первом использовании переменной:

C$ = “текст”: a! = 1.769: B% = 12674

2. По умолчанию с помощью операторов вида ***DefType***. Этот оператор используется только в разделе Главная. Синтаксис оператора:

```
DefType <список символов >!
```

Например:

```
DefStr C
DefInt I-L
```

В данном примере первый оператор объявляет, что все переменные, имя которых начинается с символа C являются переменными строкового типа, второй оператор объявляет, что все переменные, имена которых начинаются с символов I, J, K, L являются переменными целого типа одинарной точности.

3. Явное объявление с помощью операторов ***Dim, Private, Static, Public, Global***. При объявлении переменной указывается ее имя и тип. Синтаксис операторов объявления переменных:

```
Dim <имя_переменной> As <тип_переменной> [,<имя_переменной>, As <тип_переменной >]
```

Соглашение о типах переменных, принятых по умолчанию, можно изменить, используя суффикс или оператор Dim.

Контроль типов переменных

С целью исключения ошибок, связанных с неправильным объявлением имен переменных, рекомендуется объявлять все переменные явно. Контроль типов переменных можно установить с помощью оператора ***Options Explicit.*** Этот оператор необходимо поместить в разделе Главная (General) данной формы. Если переменная не была объявлена, то при запуске программы выдается сообщение об ошибке.

Другой способ обеспечить контроль типов переменных – установить флажок в опции ***Require Variable Declaration*** на вкладке ***Editor*** в меню ***Tools \ Options***. В последнем случае оператор ***Options Explicit*** будет автоматически вставляться в раздел Главная при создании новой формы.

Типы переменных

VB имеет большое число типов переменных – 14. Основные типы приведены в табл. 3.1.

Не допускается использование в программе двух одинаковых имен, отличающихся только типом.

Особо необходимо отметить переменные типа Variant. Если тип переменной не указан, они будут объявлены программой, как Variant. В переменной типа Variant можно хранить переменные любого типа. Преобразование типов переменных в этом случае осуществляется автоматически. Но при этом в некоторых случаях могут возникать ошибки. Кроме того, использование переменной типа Variant приводит к перерасходу оперативной памяти, а также этот тип данных неприемлем для использования в качестве аргументов для тех

Таблица 3.1.

Типы переменных

Суффикс	***Префикс***	***Тип переменной***	***Объявление по умолчанию***	***Занимаемая память в байтах***	***Описание***
$	s	String	DefStr	1 байт на символ	Символьная переменная переменной длины, 2 миллиарда символов для динамически изменяемых срок
		String* длина	DefStr	1 байт на символ	Символьная переменная фиксированной длины, 65400 символов
%	n	Integer	DefInt	2	Целое, одинарной точности (-32768,+ 32768)
&	l	Long Integer	DefLng	4	Целое, двойной точности (от - 2 147 483 648 до + 2 147 483 647)

Суффикс	Префикс	Тип переменной	Объявление по умолчанию	Занимаемая память в байтах	Описание
!	f	Single	DefSng	4	Вещественное, одинарной точности, +-(10^{-45} … $3*10^{38}$)
#	d	Double	DEfDbl	8	Вещественное, двойной точности, 1E127, +-($5*10^{-324}$… $1,8*10^{308}$)
@	c	Currency	DefCur	8	Денежный тип, используется для финансовых расчетов (+- $9*10^{14}$)
-	b	Boolean	DefBool	2	Логическое, используется для представления логических переменных, имеет два значения : True и False
-	-	Byte	Def Byte	1	Целые числа от 0 до 255
-	-	Date	DefDate	8	Дата и время, от 1/1/100 до 12/31/ 9999; от 0:0:0 до23:59:59
-	-	Object	-	4	Экземпляр класса; объект типа OLE
-	v	Variant	DefVar	16 + 1 байт/символ	Вариантный

функций, параметры которых описаны явным образом. Поэтому рекомендуется с самого начала объявлять типы переменных явно.

Объявление типов переменных явно считается хорошим стилем программирования.

Область определения (видимости) переменных

В VB есть три вида областей определения, характеризующих доступность переменной: локальная; контейнера (формы или модуля); глобальная.

Локальная переменная доступна только в текущей процедуре или функции. Локальные переменные объявляются оператором ***Dim***.

Переменная контейнера доступна только в текущей форме, модуле или классе. Объявляются оператором ***Dim*** в разделе Главная (General) формы или модуля. Переменная модуля объявляется оператором ***Private***.

Глобальные переменные доступны во всем проекте. Они объявляются в разделе Главная главного модуля программы оператором ***Global*** или ***Public*** (файл с расширением .bas).

Внимание*: Для каждой переменной ее тип объявляется отдельно. В одном операторе Dim можно объявлять несколько переменных, разделяя их запятыми и указав для каждой переменной ключевое слово As и тип переменной.*

Примеры:

```
Dim  a As Integer, b As Single
Dim  nCount As Integer, I As Integer, J As Integer
```

```
Publuc sText As String, NameFile As String
Global dFiz As Double
```

Время жизни переменных

Переменные, объявленные как локальные, при выходе из процедуры удаляются из памяти, а при новом вызове инициализируются заново.

Глобальные переменные при выходе из программы сохраняют свои значения.

Статические переменные

В некоторых случаях необходимо, чтобы при выходе из процедуры переменные сохраняли свое значение. Например, при использовании процедуры для печати страниц номер страницы должен увеличиваться при каждом вызове процедуры. Поэтому при выходе из процедуры переменная, хранящая значение текущего номера страницы, должна сохранить свое значение, чтобы при последующем вызове процедуры можно было увеличить номер страницы на единицу.

С целью обеспечения сохранности значений переменных при выходе из процедуры они могут быть объявлены в данной процедуре как статические оператором ***Static***:

```
Static nPageNumber As Integer
```

Статические переменные являются локальными для процедуры, в которой они используются, но их значения сохраняются до повторного вызова процедуры.

Чтобы объявить статическими все переменные процедуры ключевое слово Static следует записать в заголовке процедуры:

```
Static Sub <имя_процедуры> _<событие> (аргументы)
Static Sub ИзвлечениеКорня_Click ( )
```

3.1.3. Константы

Основное отличие констант от переменных состоит в том, что их значения нельзя изменить в процессе выполнения программы. Они всегда сохраняют значение, присвоенное при разработке. Области видимости для констант определяются также, как и для переменных. Константы бывают локальные, контейнера и глобальные. При объявлении констант используется ключевое слово ***Const***. Глобальная константа объявляется как ***Public Const***.

Глобальные константы объявляются только в модуле.

Одновременно с объявлением константе можно присвоить и значение:

```
Public Const <Имя константы>=<Значение>
Private Const <Имя константы>=<Значение>
```

Например:

```
Public Const  Pi = 3.1415926538897932
Const  G = 9.81
Public Const nName = «Лев Толстой»
Const ПлотностьМатериала = 225
Const Масса = ПлотностьМатериала * Высота* Ширина* Длина.
```

Информацию о константах, их значениях и применении можно получить, обратившись к соответствующим разделам справки или воспользовавшись каталогом объектов: войти в меню ***View \ Object Browser***, щелкнуть кнопку ***Object Browser*** на панели инструментов или нажать клавишу ***F2.***

Константы можно объявлять и с указанием типа данных:

[Public\ Private] Const <Имя константы> As <Тип_данных>=<Значение>

Для указания типа данных используются те же ключевые слова, что и при объявлении переменных.

Visual Basic имеет большое число встроенных констант, буквально “на все случаи жизни”. В виде констант в VB определены коды цветов, коды клавиш, флажки и др. Особенно полезными являются константы, задающие тип пиктограмм и набор кнопок в окне сообщения Message Box.

3.1.4. Упражнение: типы переменных

Задача 3.1. Вычисление определенного интеграла.

Вычислите определенный интеграл на отрезке интегрирования [a,b]. Разработайте форму для вычисления определенного интеграла. Выведите на форму результат вычисления и список переменных и их типов.

Порядок работы:

- Запустите программу Visual Basic и откройте новый проект Standard EXE.
- Установите режим контроля объявления типов переменных: введите команду Tools\Options\Editor, установите флажок Require Variable Declaration.
- Выберите метод и разработайте алгоритм для вычисления определенного интеграла с заданной точностью.
- Разработайте эскиз формы для вычисления определенного интеграла. (рис.3.3.).
- Пример программы вычисления определенного интеграла методом левых прямоугольников. Заданная функция y=x:

```
Option Explicit
Dim a As Single, b As Single, S As Single, N As Integer
Dim x As Single, dx As Single
Dim y As Single, i As Integer
```

```
Sub Form_Load()
      ' Центрирование формы
      Left = (Screen.Width - Me.Width) / 2
      Top = (Screen.Height - Me.Height) / 2
End Sub

Sub Command1_Click()    ‘ Вычисление интеграла
    Dim y As Single, i As Integer
    a = Val(Text1.Text)
    b = Val(Text2.Text)
    N = Val(Text3.Text)
    S = 0
    x = a: dx = (b - a) / N
    For i = 1 To N
          y = x
          S = S + y * dx
          x = x + dx
    Next I
    Text4.Text=Str(S)
End Sub

Private Sub Command2_Click()
     ‘ Выход из программы
     Unload Me
End Sub
```

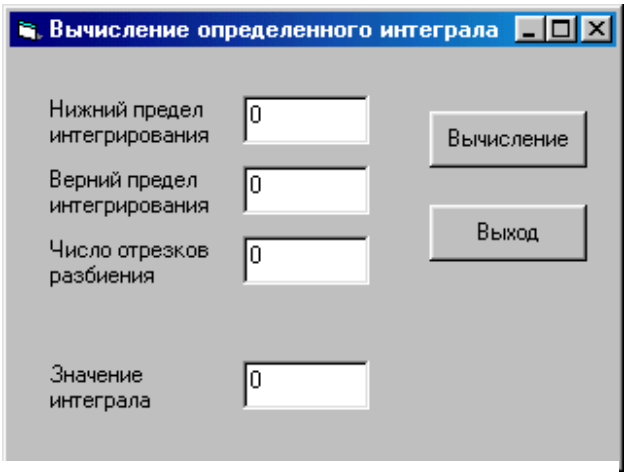


Рис. 3.3. Вычисление определенного интеграла

- Опишите все переменные, используемые в программе, и их типы по следующей форме:

Таблица 3.2

Описание типов переменных

Обозначение переменной	***Имя переменной***	***Тип переменной***	***Значение***	***Комментарий***
a	a	Single	0,5	Нижний предел интегрирования

- Напишите текст программы и запустите программу. Устраните все ошибки, которые могут выявиться при запуске программы.
- Сохраните программу на диске командой File\Save Project As.
- Вычислите определенный интеграл для функции y=x на отрезке [0,1] и сравните полученный результат с контрольным значением – 0,5. Число отрезков разбиения должно быть четным. Увеличьте число отрезков разбиения и сравните полученные результаты. С увеличением числа отрезков разбиения точность вычисления интеграла увеличивается.
- Замените функцию y=x одной из предложенных в задании и вычислите интеграл.
- Сохраните программу на диске командой File\Save Project.

- Выведите на печать текст программы командой File\Print (настройку окна диалога Print , установленную по умолчанию, не изменяйте).
- Создайте исполняемый файл командой Make Project1.exe ...
- Закройте программу Visual Basic.
- Запустите исполняемый файл программы для проверки работы.

3.1.5. Закрепление материала

1. Поясните структуру проекта в Visual Basic.
2. В чем состоит отличие формы от модуля?
3. Расскажите назначение окна кода и его элементов.
4. Какие средства редактирования и ускорения ввода программы в окне программы имеет Visual Basic?
5. Приведите синтаксис присвоения значений переменным и свойствам объектов.
6. Назовите основные типы переменных и констант в VB.
7. Какие способы объявления типов переменных применяются в VB?
8. Как обеспечить контроль типов переменных?
9. Какие операторы используются для объявления типов переменных по умолчанию?
10. Поясните, что такое область видимости и время жизни переменных?
11. Как обеспечить сохранность данных при выходе из процедуры?
12. Как можно получить сведения обо всех константах используемых в проекте и константах VB?

Задачи для самостоятельной работы

№1. $\int_{1,3}^{2,7}\frac{\sqrt{1,3x^2+0,8}dx}{1,7+\sqrt{2x+0,5}}$;	№2. $\int_{0,6}^{1,4}\frac{\sqrt{x^2+0,5}dx}{2x+\sqrt{x^2+2,5}}$;	№3. $\int_{0,4}^{1,2}\frac{\sqrt{2x^2+1}dx}{0,8x+\sqrt{0,5x+2}}$;
№4. $\int_{0,8}^{1,8}\frac{\sqrt{1,5x^2+2}dx}{x+\sqrt{0,8x^2+1}}$;	№5. $\int_{1}^{2,2}\frac{\sqrt{0,8x^2+2}dx}{1,6+\sqrt{1,5x+0,6}}$;	№6. $\int_{1,2}^{2,0}\frac{\sqrt{0,5x^2+3}dx}{2x+\sqrt{2x^2+1,6}}$;
№7. $\int_{1,3}^{2,5}\frac{\sqrt{0,8x^2+1,3}dx}{1,4+\sqrt{x^2+0,6}}$;	№8. $\int_{1,2}^{2,6}\frac{\sqrt{x^2+1,3}dx}{1,5x+\sqrt{0,4x+1,7}}$;	№9. $\int_{0,8}^{1,6}\frac{\sqrt{2x+1,6}dx}{1,8+\sqrt{0,3x^2+2,3}}$;
№10. $\int_{1,2}^{2,0}\frac{\sqrt{0,7x^2+1}dx}{2,1x+\sqrt{0,6x+1,7}}$;	№11. $\int_{0,8}^{2,4}\frac{\sqrt{0,4x^2+1,5}dx}{2,5x+\sqrt{2x+0,8}}$;	№12. $\int_{1,2}^{2,8}\frac{\sqrt{1,2x+0,7}dx}{1,4x+\sqrt{1,3x^2+0,5}}$;
№13. $\int_{0,6}^{2,4}\frac{\sqrt{1,1x^2+0,9}dx}{1,6+\sqrt{0,8x^2+1,4}}$;	№14. $\int_{0,7}^{2,1}\frac{\sqrt{0,6x+1,5}dx}{2x+\sqrt{x^2+3}}$;	№15. $\int_{0,8}^{2,4}\frac{\sqrt{1,5x+2,3}dx}{3+\sqrt{0,3x+1}}$;
№16. $\int_{0,4}^{1,2}\frac{\sin(1,5x+0,3)dx}{2,3+\cos(0,4x^2+1)}$;	№17. $\int_{0,5}^{1,3}\frac{\sin(0,7x+0,4)dx}{2,2+\cos(0,3x^2+0,7)}$;	№18. $\int_{0,4}^{1,4}\frac{\cos(0,8x^2+1)dx}{1,4+\sin(0,3x+0,5)}$;
№19. $\int_{0,3}^{1,1}\frac{\cos(0,3x+0,5)dx}{1,8+\sin(x^2+0,8)}$;	№20. $\int_{0,3}^{1,1}\frac{\sin(0,6x^2+0,3)dx}{2,4+\cos(x+0,5)}$;	

3.2. Операторы и функции языка VB.

3.2.1. Ввод данных

Данные в программу могут быть введены несколькими способами:

- присвоением начальных значений переменным непосредственно в программе с помощью оператора **Let**;
- вводом данных с клавиатуры в режиме диалога с помощью элемента управления **TextBox;**
- с помощью **диалогового окна InputBox**.

Присвоение начальных значений переменным

Присвоение начальных значений переменным осуществляется с помощью оператора ***Let***;

Синтаксис оператора:

```
Let <имя_переменной> = <значение>
```

Например:

```
Let a=3.754
Let b=Sin(x)
Let c$= “Зимние каникулы”
```

Оператор Let может быть опущен, поэтому выражения Let b=Sin(x) и b=Sin(x) эквивалентны.

Ввод данных с помощью элемента управления TextBox

В режиме диалога пользователя с программой данные можно ввести с клавиатуры непосредственно в строку ввода элемента управления TextBox. Текстовое поле используется для ввода самой разнообразной информации. Вся информация вводится как текст. Для преобразования строковых данных в числовые используется функция Val(строковая переменная). Например:

```
<переменная_символьного_типа>= txtText1.Text
<переменная_числового_типа>= Val(txtText1.Text)
```

Ввод данных с помощью окна диалога InputBox

В режиме диалога данные можно ввести в программу с помощью функции ***InputBox*** (рис.3.4.). Функция InputBox используется тогда, когда необходимо ввести только краткую информацию. Она имеет следующий синтаксис:

```
<Переменная> = InputBox (prompt [,title] [,default] [,xpos] [,ypos] [,helpfile], context])
```

Параметр: *Prompt* определяет текст, отображажаемый в диалоговом окне как приглашение («Введите значение переменной X»).

Title – отвечает за надпись заголовка («Ввод данных»). Если этот параметр не указан, то отображается название приложения.

Default – определяет значение по умолчанию, отображаемое в строке ввода («124.786»).

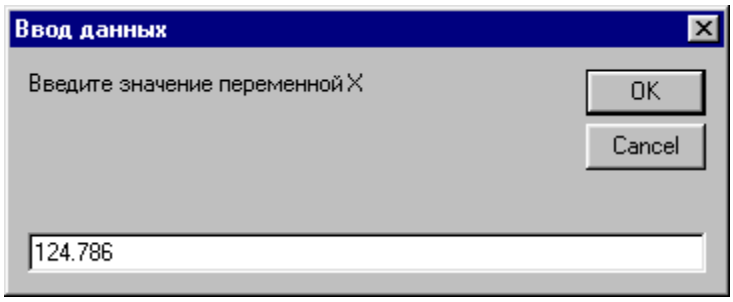


Рис. 3.4. Окно дилога функции InputBox

Параметры *XPos* и *YPos* используются совместно и указывают координаты верхнего левого угла окна. По умолчанию окно отображается посредине экрана.

Функция InputBox имеет еще два необязательных параметра *HelpFile* и *Context*, которые позволяют открывать определенные файлы справочной системы. При нажатии кнопки ОК, функция InputBox_возвращает строку, введенную пользователем. При нажатии кнопки Cancel возвращается пустая строка.

Из-за громоздкости окна ввода, его рекомендуется использовать только при отладке программы. В готовой программе рекомендуется использовать текстовое поле TextBox.

3.2.2. Вывод данных

Для вывода данных также используется несколько способов: с помощью оператора *Print*; с помощью *текстового поля*; с помощью окна сообщений *MessageBox*; с помощью *сетки MSFlexGrid* .

Оператор Print

Оператор Print может выводить информацию непосредственно в форму или графический объект PictureBox.

Синтаксис оператора Print в Visual Basic подобен оператору Print в языке программирования Basic:

Print [“текстовое сообщение”] [;/● /,] <список выражений> [;/,]

Здесь “текстовое сообщение” – тескст, делающий понтным то, что выводится на форму;

[;/● /,] - управляющие символы: если в качестве разделителя используется “;” или пробел “● ”, то очереное значение выводится непосредственно вслед за предыдущим значением. Поэтому, если выводится текст, то он сливается. При выводе положительных чисел оставляется один пробел под знак числа. Если в качестве разделителя используется запятая,то очередное значение выводится в соседнюю зону. Выводная строка оператора Print разбита на пять зон по 14 символов в каждой, что позволяет позиционировать выводимую информацию;

<список выражений> - в качестве выводимых значений могут использоваться константы, переменные, функции, арифметические выражения или текст;

[;/,] - управляющие символы. При наличии этих символов по окончании вывода информации курсор остается в текущей строке и поэтому следующий оператор Print будет продолжать печать с этой позиции. При отсутствии этих управляющих символов по окончании вывода информации курсор переходит на новую строку.

Для позиционирования точки вывода используются свойства ***CurrentX*** и ***CurrentY*** объекта, а также функции ***Tab(N) и Spc(N)***. Например:

```
CurrenX=100: CurrenY=50
Print x,y
```

или

```
Print  Tab(100); x; Spc(10); y
```

Свойства CurrentX и CurrentY перемещают точку вывода в указанные координаты X и Y. Расстояния задаются в твипах.

Функции Tab и Spc используются в операторе Print. Они перемещают точку вывода на заданное число позиций. Отличие в использовании функций Tab и Spc состоит в том, что функция Tab перемещает точку ввода относительно края формы, а функция Spc – относительно текущей позиции.

Для управления представлением чисел, дат и времени в форме используется также функция ***Format***. Синтаксис функции Format:

```
Print Format (значение, “шаблон”)
```

Например: Print Format (1124.75; currency)

Visual Basic 6.0 позволяет использовать ***стандартные, пользовательские и дополнительные*** шаблоны, которые приведены в табл. 3.3, 3.4, 3.5

Таблица 3.3

Стандартные шаблоны функции Format

Имя шаблона	***Описание***	***Пример***
General Number	Выводит числа без специального форматирования.	1124.75
Currency	Выводят знак $, число с разделением тысяч и двумя знаками после запятой (000,000,000.00).	$1,124.75
Fixed	Выводит минимум одну цифру перед запятой и две – после запятой.	1124.75
Standard	То же, что и Fixed, кроме того добавляется разделитель тысяч.	1,1221.75
Percent	Исходные числа умножаются на 100 и добавляется знак процента.	112475.00%
Scientific	Экспоненциальная форма	1,12475E+03
Yes/No	Для ненулевых значений возвращается Yes, а для нулевых – No.	

Имя шаблона	*Описание*	*Пример*
True/False	Для ненулевых значений возвращается True, а для нулевых – False .	
On/Off	Для ненулевых значений возвращается On, а для нулевых – Off.	

Таблица 3.4

Пользовательские шаблоны

Символ	*Назначение*	*Число, шаблон*	*Результат*
0	цифровое знакоместо, отображается цифра или 0	12450023 “000000000”	012450023
#	цифровое знакоместо, отображается цифра или пробел перед значащими цифрами;	12450 **“#########”**	12450
.	место десятичной точки	124500.2375 **“#######.##”**	124500.24
,	разделитель тысяч	124,500.2345 **“#,###,###.##”**	124 500.23
%	знак процентов, число будет умножено на 100	0.25 **“####.##%”**	25%
E-	показательная степень числа, при отрицательном показателе степени перед ним будет отображаться знак «-», знак «+» отображаться не будет	0,0005834 “#.##E-”	5,83E-4
E+	то же, что и предыдущий, только при положительном показателе будет отображаться знак «+».	124500.2375 “#.##E+”	1.25E+5

Таблица 3.5

Дополнительные функции форматирования

Функция	*Тип параметра на входе*	*На входе*	*На выходе*
Format Currency	Денежный	8675.309	$8,675.31
Format Numeric	Числовой	-5000	(5,000.00)
Format Percent	Процентный	0,1234	12.34%
Format DataTime	Дата/Время	“12-31 13:34”	12/31/98 1:34:00 PM
Round	Числовой	123.6	124

Текстовое поле TextBox

Текстовое поле TextBox может использоваться для вывода информации в любом месте формы. Позиция элемента управления может устанавливается как при разработке формы так и в процессе выполнения программы:

```
Text1.Top=Y
Text1.Left=X
```

Text1.Text = “текст”

Окно диалога MessageBox

Окно MessageBox подобно окну InputBox и служит для вывода информации. Этот компонент широко используется для вывода различных сообщений в приложениях Windows.

MessageBox можно вызвать как команду и как функцию.

Синтаксис команды:

MsgBox Prompt [, Buttons] [,Title] [, Helpfile, Context]

Синтаксис функции:

<Переменная >= MsgBox (Prompt [, Buttons] [,Title] [, Helpfile, Context]

Пример 3.1:

Команда (рис.3.5):

MsgBox “Здравствуй, товарищ!”, vbExclamation, “Приветствие”.

Функция:

Text=MsgBox (“Конец игры?”, vbCritical, “End”)

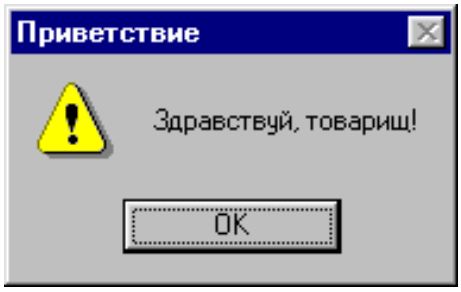


Рис. 3.5. Окно MsgBox

Параметры *Prompt*, *Title*, *Helpfile*, *Context* имеют то же значение, как и в функции InputBox.

Параметр *Buttons* определяет состав кнопок в окне. Он формируется из нескольких частей:

Buttons = Button + Icon + Default + Modal +Extras

где *Button* – количество кнопок и их состав;

Icon – вид пиктограммы;

Default – определяет, какая кнопка активна;

Modal – вид диалогового окна (окно приложения или системы). Модальность означает, что выполнение приложения возможно только после закрытия окна;

Extras – дополнительные свойства.

По умолчанию выводятся кнопки ОК и Отмена.

Значения категорий параметра Buttons приведены в табл. 3.6

Пример 3.2. Формирование параметра Buttons.

Type=4+32+256+1

MsgBox «Сообщение», Type, «Заглавие».

При данном коде в окне отображаются кн. Yes и No (код 4), пиктограмма Warning Query (код 32), активна кнопка No (код 256), окно модальное (код 1).

Таблица 3.6

Константы параметра Buttons

Код	*Константа*	*Значение*
Константы категории Button		
0	(vbOkOnly)	Выводится только кнопка ОК
1	(vbOkCancel)	Выводится кнопка ОК и Cancel
2	(vbAbortRetryIgnore)	Выводится кнопка Abort, Retry, Ignore

Код	*Константа*	*Значение*
3	(vbYesNoCancel)	Выводится кнопка Yes, No, Cancel
4	(vbYesNo)	Выводится кнопка Yes, No
5	(vbRetryCancel)	Выводится кнопка Retry, Cancel
Константы категории Icon		
16	VbCritical	Отображает пиктограмму Critical Message
32	VbQuestion	Отображает пиктограмму Worning Query (запрос)
48	VbExclamation	Отображает пиктограмму Worning Message (предупреждение)
64	vbInformation	Отображает пиктограмму Information Message (информация)
Константы категории Default		
0	vbDefaultButton1	по умолчанию активна первая кнопка
256	vbDefaultButton2	по умолчанию активна вторая кнопка
512	vbDefaultButton3	по умолчанию активна третья кнопка
768	vbDefaultButton4	по умолчанию активна четвертая кнопка
Константы категории Modal		
1	vbModal	модальное диалоговое окно приложения;
4096	vbSystemModal	модальное диалоговое окно системы
Константы категория Extras		
16384	vbMsgBoxHelpButton	дополнительная кнопка для справки
65536	vbMsgBoxSetForeground	отображение диалогового окна в фоновом режиме
524288	vbMsgBoxRight	текст выровнен по правому краю
1048576	vbMsgBoxRtlReading	текст отображается справа налево

Управляющий элемент Сетка (MsFlexsGrid) используется, как правило, для вывода массивов. С принципом использования сетки мы познакомимся в разделе 7.

3.2.3. Процедуры

В Visual Basic весь текст программы разбит на отдельные программные блоки, имеющие один вход и один выход и выполняющие определенные вычислительные или логические операции. Эти программные блоки оформляются в виде ***процедур***.

Процедура начинается оператором ***Sub*** и заканчивается оператором ***End Sub***, между этими операторами и помещается код. Такие процедуры могут вызываться или самим VB (процедуры обработки событий), или другими процедурами.

Различают процедуры *обработки событий* и *пользовательские процедуры*.

Процедуры обработки событий (обработчики событий)

Все обработчики событий реализуются как процедуры. Общий синтаксис заголовка для объявления процедуры обработки события:

[ОбластьОределения] Sub [ИмяФормы.]Объект_Событие(аргументы)

Опции, указанные в квадратных скобках могут быть опущены. Если аргументы отсутствуют, ставятся пустые скобки. Например:

```
Private Sub frmForm1.cmdVvod_KeyUp(KeyCode As Integer, Shift As Integer)
    Тело процедуры
End Sub
```

В данном примере *ОбластьОределения* – область доступности или видимости процедуры: *Private* – локальная, доступна только в данной форме или модуле; *Public* – глобальная, доступна во всем проекте. *frmForm1*– имя формы, *cmdVvod* – имя кнопки, *KeyUp* – событие, связанное с нажатием клавиши, *KeyCode* – возвращаемый код нажатой клавиши , *Shift* – возвращаемый код нажатой дополнительной клавиши;

```
Private Sub Command1_Click ( )
   Тело процедуры
End Sub
```

В данном примере имя формы опущено, по умолчанию принимается имя текущей формы, передаваемые параметры – аргументы отсутствуют.

Процедуры обработки событий для каждого объекта выбираются из правого списка окна программы (Code).

Процедуры пользователя

Visual Basic позволяет создавать и пользовательские процедуры. Основной отличительной чертой пользовательских процедур является то, что они не связаны ни с каким событием и вызов их пользователь осуществляет по своему усмотрению. В виде пользовательской процедуры можно оформить любую подпрограмму и использовать ее в текущем проекте или сохранить на диске и использовать в других программах. Для создания пользовательской процедуры необходимо перейти к секции Главная, в окне программы ввести служебное слово ***Sub*** и ***имя процедуры*** и нажать клавишу Enter. После этого появится новая процедура:

```
Sub <имяПроцедуры>()
   <Тело процедуры>
End Sub
```

Например:

```
Sub HelloOut()
    Print “Здравствуй читатель”
End Sub
```

Процедура, описанная в разделе Главная доступна всем другим процедурам формы.

Синтаксис процедуры пользователя совпадает, в основном, с синтаксисом процедуры обработки событий:

```
[ОбластьОпределения] Sub [ИмяФормы.]Объект_ИмяПроцедуры (аргументы)
    <Тело Процедуры>
End Sub
```

Отличие состоит в том, что вместо имени события указывается имя процедуры пользователя.

Вызов процедуры

Для вызова процедуры используются два способа: по имени процедуры и с помощью оператора Call.

При вызове процедуры ***по имени*** после имени указываются без скобок передаваемые параметры:

```
ИмяПроцедуры аргумент1, аргумент2, ...
```

Передаваемые параметры по типам должны соответствовать типам переменных указанных в описании процедуры. В качестве передаваемых параметров могут использоваться строки символов, имена переменных или функций, возвращающих значения заданного типа.

При вызове процедуры с помощью оператор ***Call*** передаваемые параметры заключаются в скобки:

```
Call ИмяПроцедуры (аргумент1, аргумент2,…)
```

Оператор Call используется, как правило, для вызова внешних процедур. Чтобы отличить вызов функции от вызова процедуры, рекомендуют не использовать оператор Call для вызова процедур (при вызове функций передаваемые параметры также заключаются в .скобки).

Пример 3.3. Процедура вычисления площади поверхности параллелепипеда.

```
‘ Процедура, объявленная в секции главная формы
Sub SPoverch(a As Single, b As Single, h As Single, s As Single)
     s = 2 * (a * b + (a + b) * h)
End Sub
---------------------------------------------------------------------------
Private Sub Form_Click()
    ‘ Вызывающая процедура
    Dim x As Single, y As Single, z As Single, s As Single
    x = Val(InputBox("Длина параллелепипеда”))
    y = Val(InputBox("Ширина параллелепипеда "))
    z = Val(InputBox("высота параллелепипеда "))
   ‘ вызов процедуры
    SPoverch x, y, z, s
    Print Str$(s)
End Sub
```

3.2.4. Функции

Встроенные функции

Visual Basic имеет большое число встроенных функций. Можно выделить следующие группы функций:

- математические;
- работы с массивами;
- преобразования типов данных;
- работы с символьными переменными;
- работы с датами и временем;
- финансово-математические;
- работы с файлами;
- управления компьютером;
- обработки ошибок

Многие из этих функций совпадают с функциями языка Basic, но появилось много и новых, специфических функций, расширяющих возможности языка программирования по работе с файлами, обработке данных.

Все математические функции полностью совпадают с функциями языка Basic (табл. 3.7)

Таблица 3.7

Встроенные функции языка Visual Basic

Математическая функция	***Функция языка Visual Basic***	***Комментарий***
| x |	Abs(x)	Абсолютное значение числа x
e^x	Exp(x)	Возведение в степень x числа e
$\sqrt{x}$	Sqr (x)	Корень квадратный от x
ln(x)	Log (x)	Натуральный логарифм аргумента x
] x [	Fix(x)	Возвращает целое число, меньшее x
] x [	Int(x)	Возвращает целое число, меньшее x
	Round (x, n)	Округляет число x до n знаков после запятой
	Sgn(x)	Определяет знак числа
	Rnd(N)	Генерирует последовательность псевдослучайных чисел. При N<0 генерирует определенное число, зависящее от N, при N=0 возвращает последнее случайное число, выданное Rnd, при N>0 генерируется новое случайное число. Для изменения базы генератора псевдослучайных чисел можно использовать оператор Randomize
Sin(x)	Sin(x)	Синус числа x
Cos(x)	Cos(x)	Косинус числа x
Tg(x)	Tan(x)	Тангенс числа x
Arctg(x)	Atn(x)	Арктангенс x, обратная тригонометрическая функция

В тригонометрических функциях аргумент x должен задаваться в радианах.

Из обратных тригонометрических функций имеется только одна функция arctg(x). Остальные тригонометрические функции вычисляются через arctg(x) или другие тригонометрические и арифметические функции по правилам математики.

Отметим еще две функции:

Val(c) – преобразует символьную переменную в число.

IIf (логическое выражение, V1,V2) – логическая функция, возвращает значение по выражению V1, если логическое выражение истинно и по выражению V2, если логическое выражение ложно.

Функции для обработки строк символов, дат и времени приведены в приложении 1. В некоторых строковых функциях опущены необязательные параметры.

Функции пользователя

Наряду с процедурами имеются и пользовательские функции, синтаксис которых похож на синтаксис пользовательских.

Синтаксис функции пользователя:

```
Function     ИмяФункции ( аргумент1  As тип, [, аргумент2 As тип ] ) As тип
        < операторы >
       ИмяФункции = результат
End  Function.
```

В качестве аргументов могут быть константы, переменные или выражения. В конце процедуры записывается оператор ИмяФункции = результат , здесь ИмяФункции – простая переменная, имя которой совпадает с именем функции. Этим оператором имени функции передается вычисленное значение.

Использование пользовательских функций

Функции пользователя используются так же, как и встроенные функции Visual Basic. То есть, они используются в выражениях справа от знака равно, могут включаться также в оператор Print:

```
< имя переменной > =  ИмяФункции ( аргумент 1 [, аргумент 2, …])
Print ИмяФункции
```

Допускается также использование функции в следующем формате:

```
ИмяФункции (аргумент 1 [, аргумент 2, …])
```

Внутри функции можно объявлять локальные переменные, указывая их тип: статические или динамические. Переменные уровня формы доступны всем функциям, подключенным к данному модулю или форме. Преждевременный выход из функции осуществляется с помощью оператора ***Exit Function***.

Подключение пользовательских функций и процедур к текущей форме осуществляется следующим образом:

- открыть окно Программы (Code);
- выбрать пункт меню ***Tools, Add Procedure***;
- указать в диалоговом окне имя процедуры;

- установить переключатель ***Function***;
- щелкнуть по кн. ***OK***.

Пример 3.4. Вычислить значение Sin(x) с точностью 0.001. Математическая модель решения данной задачи представляется следующей формулой:

$$\mathrm{Sin}(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} + \ldots + \frac{x^{2i-1}}{(2i-1)!} + \ldots = \sum_{i=1}^{\infty} \frac{x^{2i-1}}{(2i-1)!}$$

Вычисление факториала оформим в виде процедуры. Входным параметром функции будет целое число одинарной длины n – факториал. Функция возвращает целое число двойной длины.

```
Public Function Factorial(n As Integer) As Double
    Dim i As Integer, F As Double
    F = 1
    For i = 1 To n
        F = F * i
    Next i
    Factorial = F
End Function
-----------------------------------------------------------
Private Sub Form_Click()
   Dim n As Integer, F As Double, k As Integer
   Dim x As Single, y As Single, s As Single
   x = Val(InputBox("Аргумент X"))
   e = Val(InputBox("Точность вычисления"))
   y = 1 + e: s = 0: k = 1
  While Abs(y) >= e
      y = (-1) ^ (k - 1) * x ^ (2 * k - 1) / Factorial(2 * k - 1)
      s = s + y
      k = k + 1
  Wend
  Print Str$(s)
End Sub
```

В Visual Basic 6.0 функции имеют новую возможность: они могут возвращать массивы. Можно, например, возвратить массив целых чисел без предварительного преобразования его в строку и обратно:

```
Private sub Form_Load()
    Dim b As Byte, I As Integer
    Dim ReturnArray() As Byte
    I=Cbite(54)
    ReturnArray() =  ArrayFunction(b)
    For I=0 To Ubound(ReturnArray)
        Debug.Print ReturnArray(i)
    Next I
End Sub
-------------------------------------------------------------------
Public Function  ArrayFunction(b As Byte) As Byte()
   Dim x(2) As Byte()
   x(0)=b
   x(1)=b+Cbyte(200)
```

```
    x(2)=b+b
    ArrayFunction = x
End Function
```

В данном примере в процедуре Form_Load массиву ReturnArray присваивается массив, возвращаемый функцией ArrayFunction, состоящий из трех элементов.

3.2.5. Операторы для управления вычислительным процессом

Операторы выбора

Оператор выбора позволяет выбрать одну альтернативу из двух (условный оператор IF) или из многих (оператор Select Case).

Условный оператор IF

Различают однострочковые и многострочковые условные операторы If.

Синтаксис однострокового оператора IF:

а) простой однострочковый оператор (рис. 3.6, 3.7):

```
If <условие> Then <операторы>
```

При выполнении оператора *If* проверяется условие и, если оно истинно, то выполняется действие, указанное после оператора *Then.* Если выражение ложно, то управление передается на оператор, следующий за оператором *If*.

Схема алгоритма

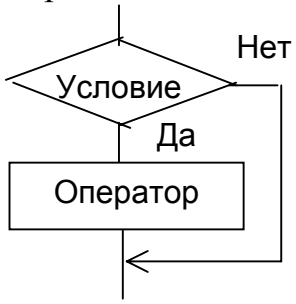


Рис. 3.6. Простой однострочковый условный оператор If

Схема алгоритма

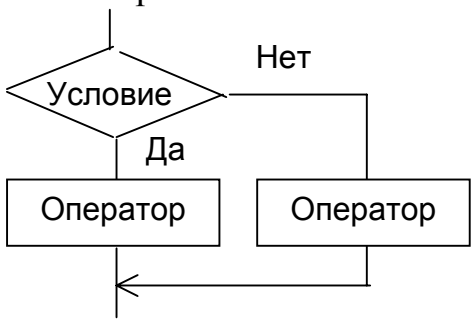


Рис. 3.7. Расширенный однострочковый условный оператор If

Примеры:

```
    If x<e Then Goto m1
IF x<a Then y=Sin(x)*Exp(2^Log(x)): Goto m2
```

б) простой расширенный оператор If

```
If <условие> Then <операторы1> Else <операторы 2
```

При выполнении оператора *If,* если условие истинно, то выполняются операторы, указанные после оператора *Then*, в ином случае выполняются операторы, следующие за оператором *Else*. После выполнения соответствующей груп-

пы операторов управление передается на оператор, следующий за оператором *If*.

После операторов Then и Else может быть указано несколько операторов, разделенных двоеточием. Однако, число операторов ограничено длиной строки. Кроме того, чтение, анализ и поиск ошибок в длинных строках затруднителен

. ***Синтаксис многострокового оператора IF:***

а) простой

```
If <условие> Then
   <перваягруппа операторов>
Else
   <вторая группа операторов>
End If
```

б) расширенный

```
If <условие>  Then
     <первая группа операторов>
ElseIf <условие> Then
     <вторая группа операторов>
Else
     <третья группа операторов>
End If
```

При записи операторов следует обращать внимание на структуру записи. Структура должна соответствовать той, что указана в примере.

Достоинство многострокового оператора If состоит в том, что число операторов в группах не ограничено.

Оператор Select Case

Оператор Select Case, также как и оператор IF является оператором выбора. Он позволяет выполнить одну группу операций из нескольких в зависимости от значения переключающего выражения. Структура данного оператора похожа на структуру многострочного If, но он имеет больше условий выбора, чем этот оператор. Синтаксис оператора:

```
Select Case <переключающее выражение>
    Case  значение1                    ''Значение " может быть
        [операторы ]                       константой
    Case  значение2                        переменной,
        [операторы]                        списком значений (1,2,3,5)
    . . .                                  диапазоном значений (1 To  9 )
    Case Else                              условием (Is > 0)
        [операторы]
End Select
```

Операторы циклов

Циклом называется процедура, в которой вычислительные операции выполняются многократно заданное число раз или до достижения некоторой переменной, вычисляемой в теле цикла, определенного, наперед заданного значения.

Циклы первого типа называются циклами типа "ДО", а циклы второго типа - циклами типа "ПОКА".

В зависимости от времени проверки условия окончания цикла циклы делятся на циклы с предусловием и циклы с постусловием. В циклах с предусло-

вием вначале проверяется условие окончания цикла и, если условие окончания цикла не выполняется, то выполняется тело цикла. В таких циклах значение переменной, проверяемой в теле цикла, должно быть вычислено заранее или ей должно быть присвоено некоторое значение, заведомо большее условия окончания цикла, до входа в цикл. В циклах с постусловием условие окончания цикла проверяется в конце цикла.

Циклы могут быть организованы с использованием оператора ***IF*** или с использованием специальных операторов: ***For/Next, While/Wend, Do/Loop***.

Цикл с постусловием

```
Rem  использование для организации
Rem  цикла  оператора If
M1:
i=iнач:        REM заголовок цикла
<тело цикла, вычисление функции F(i)>
i=i+di
If i<=iкон  Then  Goto M1: Rem конец  цикла
Print "Результат"; F

Rem использование оператора For
For i=iнач To N Step di
    <тело цикла, вычисление функции F(i)>
Next i
Print  "Результат";F
```

Схема алгоритма

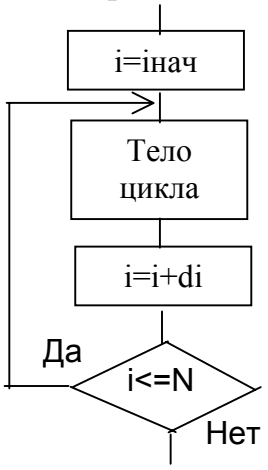


Рис. 3.8

Оператор For/Next.

Оператор For/Next (рис. 3.8) служит для организации циклов с заданным числом повторений. Цикл относится к циклам с постусловием. Схема алгоритма приведена на рис. 3.8. Формат оператора:

```
For i=iнач To iкон Step di
    <тело цикла>
Next I
```

В данном формате *iнач* – начальное значение переменной цикла, *iкон* – конечное значение переменной цикла, а *di* – шаг приращения значения переменной цикла.

Операторы For и Next образуют операторные скобки, между которыми заключено тело цикла.

Оператор While/Wend.

Оператор While/Wend служит для организации циклов с заданным числом повторений, относится к циклам с постусловием. Условием окончания цикла является достижение функцией, вычисляемой в теле цикла, заданного значения. Схема алгоритма приведена на рис. 3.9. Синтаксис оператора:

```
While <условие>
   <тело цикла>
Wend
```

Цикл с предусловием

```
Rem Использование оператора If
F=<выражение>
i=iнач
M1:
If F<=ε  Then Goto M2
    <тело цикла, вычисление F(i)>
    i=i+di
Goto M1
M2:
Print "Результат"; F

Rem  Использование оператора
Rem  While/Wend
F=<выражение>
i=iнач
While F>ε
    <тело цикла, вычисление F(i)>
    i=i+di
Wend
Print  "Результат";F
```

Схема алгоритма

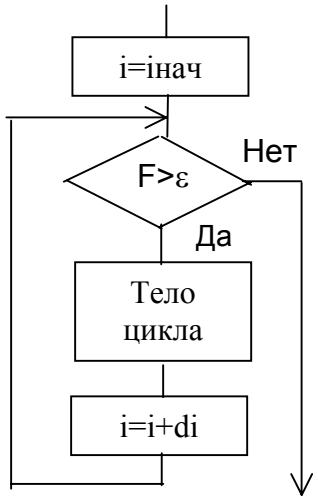


Рис.3.9.

Тело цикла выполняется в том случае, когда условие истинно. Если условие ложно, то программа выходит из цикла. Особенностью использования данного цикла является то, что значение вычисляемой функции должно быть известно перед входом в цикл. Иначе, если начальное значение функции меньше заданной величины ε, программа никогда не войдет в цикл.

Оператор Do/Loop - универсальный оператор, служит для организации циклов типа "ПОКА". Он может быть организован как цикл с предусловием, так и как цикл с постусловием. Когда в теле цикла используется оператор *While*, то тело цикла выполняется в том случае, если условие истинно. Когда используется оператор *Until*, то тело цикла выполняется в том случае, если условие ложно. Синтаксис оператора:

а) цикл с предусловием

```
Do  [ ( While | Until) Условие ]
      [операторы]
      [Exit Do]
      [ операторы]
Loop
```

б) цикл с постусловием

```
Do
      [операторы ]
      [Exit Do]
      [операторы]
Loop [(While | Until) Условие ]
```

Для досрочного выхода из цикла используется оператор **Exit Do.**

В Visual Basic появился новый оператор цикла для работы с целочисленными переменными For/ Each. Оператор **For/Each -** перечислимый тип циклов. Синтаксис оператора:

```
For Each  переменная цикла In объект
    [операторы]
Next  переменная цикла.
```

Пример 3.5:

```
Dim  nNumber As Variant
Dim  MyArray As Variant
MyArray = Array (3,6,9,9,5,2,3,1)
For Each nNumber In MyArray
    If  nNumber > 5 then Debug.Print nNumber
Next nNumber
```

Здесь MyArray – массив, nNumber – текущий индекс элемента массива. Пока nNumber принадлежит массиву цикл выполняется. Пользователю не надо заботиться о контроле числа элементов в массиве. Обязательное требование – переменная цикла должна быть переменной целого типа.

Пример 3.6: Процедура контроля ввода данных

```
Sub ControlVvodaNumber()
   Dim ctl As Control, b As String
   Dim FirstPoint As Boolean
   For Each ctl In Controls
      If TypeOf ctl Is TextBox Then
        FirstPoint = False
        For I = 1 To Len(Trim(ctl.Text))
           b = Mid(Trim(ctl.Text), I, 1)
           Select Case Asc(b)
              Case Asc("0") To Asc("9")
              Case Asc(".") And FirstPoint = False
                FirstPoint = True
              Case Else
                MsgBox "Ошибка ввода данных"
                Exit Sub
           End Select
        Next I
      End If
   Next ctl
End Sub
```

Процедура обеспечивает контроль правильности ввода в текстовые поля числовых данных.

Объявлена переменная ctl как переменная типа Control. К данному типу относятся все элементы управления, размещенные на форме.

Программа проверяет, если тип элемента управления TextBox, тогда проверяется, является ли текущий символ цифрой или точкой. Не допускается в строке более одной точки (разделитель десятичных знаков).

Контроль осуществляется после ввода данных.

3.2.6. Упражнения: использование окон диалога и функций пользователя

Задача 1: Разработать программу для вычисления определенного интеграла с использованием процедуры пользователя и функций пользователя.

Порядок работы.

Доработаем программу из раздела 3.1.4. в соответствии с заданием. На форме оставим только кнопки для вычисления и выхода из программы.

```
Option Explicit
Dim a As Single, b As Single, S As Single, N As Integer

Private Sub Integral(a As Single, b As Single, N As Integer)
‘ процедура для вычисления определенного интеграла
Dim x As Single, dx As Single
Dim y As Single, i As Integer
x = a: dx = (b - a) / N
For i = 1 To N
     y = FNy1(x)
    S = S + y * dx
    x = x + dx
Next i
End Sub

Function FNy1(x)
      ‘ функция пользователя
      Dim y As Single
      y = x
      FNy1 = y
End Function

Private Sub Command1_Click()
  a = Val(InputBox("Введите нижний предел интегрирования", "Ввод данных"))
  b = Val(InputBox("Введите верхний предел интегрирования", "Ввод данных"))
  N = Val(InputBox("Укажите число отрезков разбиения", "Ввод данных"))
  S = 0
  Integral a, b, N
  MsgBox "Интеграл равен" & Str(S), , "Значение интеграла"
End Sub
```

Задача 2: Разработать форму для вычисления определенного интеграла с заданной точностью с использованием глобальной процедуры пользователя и глобальной функции пользователя. Программа должна обеспечивать вычисление площади криволинейной трапеции, ограниченной прямыми x=a, x=b, y=0 и графиками функций y=a*x^2+b или y=1/(a*x)+b.

Порядок работы.

1. Загрузите программу вычисления определенного интеграла (задача1).
2. Добавьте в Ваш проект модуль: Project\Add Module
3. Оформите программу вычисления определенного интеграла в виде процедуры пользователя и поместите ее в модуль.
4. Создайте две функции пользователя для вычисления заданных функции и также поместите их в модуль.

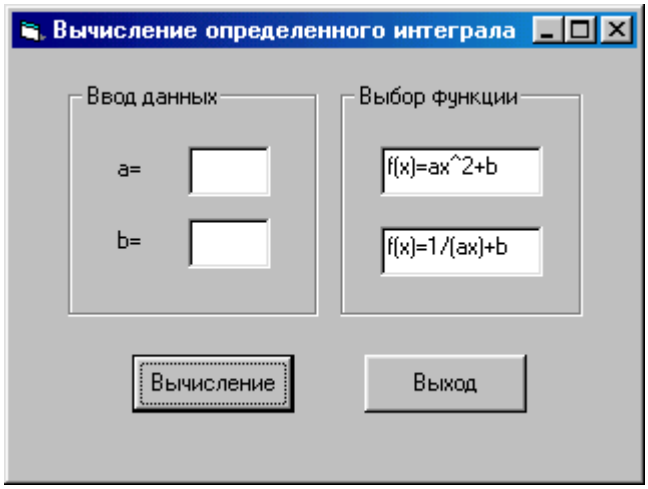


Рис.3.10. Вычисление определенного интеграла с заданной точностью и выбором типа исследуемой функции

5. Разработайте эскиз формы.

Указание: текстовые поля можно применять в качестве командных кнопок, используя событие Click.

6. Выполните отладку программы.
7. Сохраните программу на диске.
8. Создайте исполняемый файл.
9. Распечатайте тексты программ процедуры и функций.

Текст программы модуля

```
Option Explicit
Public a As Single, b As Single, S As Single, N As Integer
Public x As Single, ax As Single, bx As Single, e As Single
Public Flag1 As Boolean

' процедура для вычисления определенного интеграла
' методом двойной прогонки
Public Sub Integral2(a As Single, b As Single, N As Integer)
  Dim y As Single, i As Integer, R As Single
  Dim int0 As Single, int1 As Integer
  Dim dx As Single
  R = 1 + e: int0 = 0
  While Abs(R) > e  'цикл типа "Пока" обеспечивает контроль
                'заданной точности
    int1 = int0:
    x = a: dx = (b - a) / N
    For i = 1 To N  'Цикл типа "ДО" обеспечивает
      'вычисление интеграла при текущем значении N
      If Flag1 Then
        y = FNy2(x, ax, bx)
      Else
        y = FNy1(x, ax, bx)
      End If
```

```
        S = S + y * dx
        x = x + dx
      Next i
      'вычисление точности
      R = (int0 - int1) / 3
      N = N * 2
    Wend
End Sub

Public Function FNy1(x As Single, ax As Single, bx As Single)
    ' функция пользователя
    Dim y As Single
    y = ax * x ^ 2 + bx
    FNy1 = y
End Function

Public Function FNy2(x As Single, ax As Single, bx As Single)
    ' функция пользователя
    Dim y As Single
    y = 1 / (ax * x) + bx
    FNy2 = y
End Function
```

Текст программы формы

```
Sub Form_Load()
  ' Центрирование формы
    Left = (Screen.Width - Me.Width) / 2
    Top = (Screen.Height - Me.Height) / 2
    Flag1 = False
End Sub

Sub Command1_Click()  ' Вычисление интеграла
    ax = Val(Text3.Text)
    bx = Val(Text4.Text)
    ' контроль ввода данных
    If ax = 0 And bx = 0 Then
        MsgBox "Введите данные"
        Exit Sub
    End If
    If Flag1 = True And ax = 0 Then
        MsgBox "Неправильный ввод данных"
        Exit Sub
    End If
    a = Val(InputBox("Введите нижний предел интегрирования", _
        "Ввод данных"))
    b = Val(InputBox("Введите верхний предел интегрирования", _
        "Ввод данных"))
    N = Val(InputBox("Укажите число отрезков разбиения", _
        "Ввод данных"))
    e = Val(InputBox("Укажите требуемую точность", "Ввод данных"))
```

```
    S = 0
    Integral2 a, b, N
    MsgBox "Интеграл равен" & Str(S), , "Значение интеграла"
End Sub

Private Sub Command2_Click()
    ' Выход из программы
    Unload Me
End Sub

Private Sub Text1_Change()
    Flag1 = False
End Sub

Private Sub Text2_Click()
     Flag1 = True
End Sub
```

Чтобы обеспечить выбор нужной формулы введена переменная Flag1 типа Boolean. Переменная Flag1 принимает значение False, если выбрана параболическая функция и True, если выбрана гиперболическая функция.

3.2.7. Закрепление материала

1. Какими способами осуществляется ввод данных?
2. Приведите синтаксис функции InputBox?
3. Как осуществляется вывод данных?
4. Приведите синтаксис оператора Print.
5. Как создать пользовательский формат для управления выводом числовой информации?
6. Приведите синтаксис команды MsgBox.
7. Чем отличается синтаксис функции MsgBox от синтаксиса команды MsgBox?
8. Приведите синтаксис обработчика событий.
9. Чем отличается синтаксис процедуры пользователя от синтаксиса обработчика событий? Приведите пример процедуры пользователя.
10. Какие категории функций используются в VB?
11. Запишите основные арифметические, тригонометрические и обратные тригонометрические функции.
12. Приведите синтаксис функции пользователя.
13. Какие операторы используются в VB для управления вычислительным процессом?
14. Приведите синтаксис условных операторов.
15. Приведите синтаксис операторов цикла For/Next и For/Each.
16. Приведите синтаксис и схему алгоритма оператора While/Wend.
17. Приведите синтаксис оператора Do/Loop

18.Приведите синтаксис и поясните принцип работы оператора Select Case.

3.3. Массивы

3.3.1. Понятие об индексированных переменных. Массивы.

Массивом называется совокупность индексированных элементов ***a(i,j)***:

$$\begin{cases} a_{11}\ a_{12}\ \ldots a_{1j}\ \ldots\ a_{1n} \\ \ldots \\ a_{i1}\ \ a_{i2}\ \ldots\ a_{ij}\ \ \ldots a_{in} \\ \ldots \\ a_{m1}\ \ a_{m2}\ \ldots a_{mj}\ \ \ldots a_{mn} \end{cases} \qquad (3.1.)$$

Здесь a_{ij} – элемент массива. Индекс i означает номер строки, j – номер столбца.

Массивы в VB можно классифицировать по следующим признакам (рис.3.11.): числу измерений, области видимости, способу распределения памяти.

VB позволяет создавать одномерные и многомерные массивы. Число размерностей массива может достигать до 60.

Нумерация элементов массива начинается с нуля. Для изменения индексации с нуля на единицу используется оператор ***Option Base N***, где N может принимать значения 0 и 1. Оператор Option Base записывается в раздел Главная контейнера (формы, модуля, класса). Однако, при объявлении массивов можно задавать произвольные значения верхних и нижних границ массива, используя

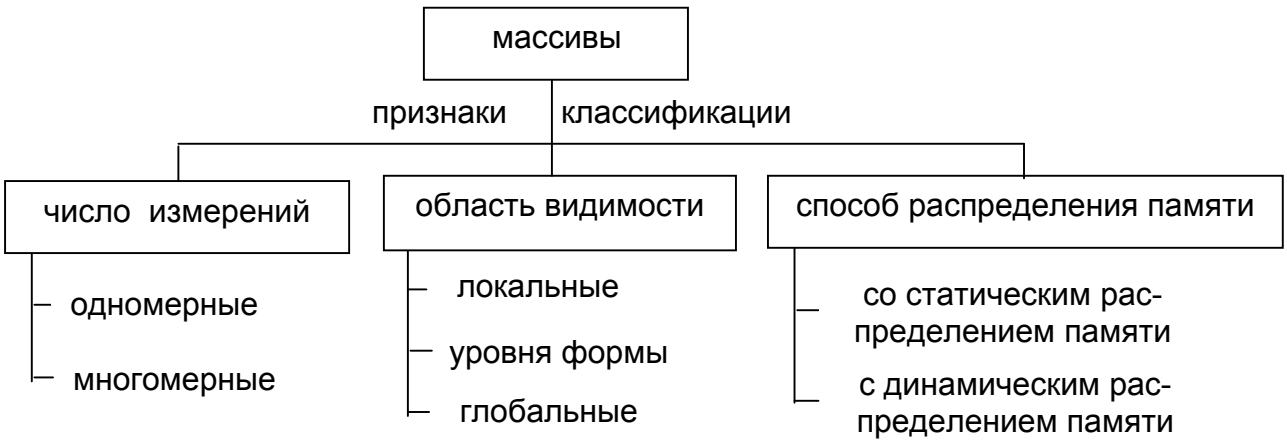


Рис. 3.11. Классификация массивов

следующий синтаксис:

Dim (нижняя_граница TO верхняя_граница).

Примеры объявления массивов:

Dim A(10) As String – одномерный массив, содержит 11 элементов;

Dim B(5 TO 10, 1 TO 20) As Integer – двухмерный массив, имеет 6 строк и 20 столбцов. Нумерация строк начинается с 5, а нумерация столбцов с единицы.

Область видимости массивов, объявление массивов

Так же как и переменные, массивы могут быть ***локальные*** – уровня процедуры. Локальные массивы доступны только в данной процедуре. При выходе из процедуры данные теряются. Такие массивы объявляются оператором ***Dim***: Dim Array1(10, 20). Локальные массивы модуля объявляются оператором ***Private***.

Массивы уровня формы доступны всем процедурам формы. Они объявляются оператором ***Dim*** в разделе Главная.

Глобальные массивы доступны всем процедурам проекта и объявляются оператором ***Public*** или ***Global*** в разделе Главная формы или модуля.

Способы распределения памяти

В зависимости от способа распределения памяти массивы делятся на массивы со статическим и массивы с динамическим распределением памяти.

Массивы со статическим распределением памяти (для краткости – статические массивы) объявляются операторами ***Dim, Private, Public, Static:***

```
Static A (1 TO 5, 1 TO 2).
```

Эти массивы объявляются один раз и не меняют своих размерностей в процессе выполнения программы. Границы размерностей задаются только числами, ***нельзя*** использовать для указания размерности переменные.

Массивы с динамическим распределением памяти (для краткости – динамические массивы) объявляются в два этапа. Сначала они объявляются на уровне контейнера без указания размерности:

```
Dim AMassiv ( ) As Variant.
```

Затем с помощью оператора ***ReDim*** устанавливаются фактические размерности массива. В отличие от оператора Dim оператор ReDim используется только в процедурах. Оператор ReDim допускает использование переменных для указания размерностей массивов:

```
ReDim MassivA(5, 10 ) As Integer
ReDim MassivB(m, n ) As Single
```

Однако, при использовании оператора ReDim нельзя изменять тип данных массива, кроме случая, когда тип массива объявлен как Variant.

Когда переопределяются размерности массива, есть опасность потерять содержимое. Чтобы при переопределении размерности массива не потерять данные, совместно с оператором ReDim используется оператор ***Preserve:***

```
ReDim Preserve MassivA(10, 50)
```

Однако, если используется ключевое слово *Preserve*, то для многомерных массивов можно изменять только последнее измерение, а в последнем измерении можно менять только верхнюю границу индекса, например:

```
ReDim aArray(10,10) ‘ объявленный массив
ReDim Preserve aArray(10,20) ‘ допускается
ReDim Preserve aArray(15,10) ‘ не допускается
```

```
ReDim aArray(5 To 10,10 To 20) ‘объявленный массив
ReDim Preserve aArray(5 To 10,10 To 25) ‘допускается
ReDim Preserve aArray(5 To 10,15 To 20) ‘ не допускается
```

Удаление массива осуществляется командой ***Erase***:

```
Erase < имя_массива>
```

Для статических массивов команда Erase не удаляет массив, а только очищает его.

3.3.2. Функции для работы с массивами

Visual Basic имеет несколько функций для работы с массивами:

Array – создание массива типа Variant.

Lbound(ИмяМассива, Индекс) – возвращает нижнюю границу диапазона индекса массива. Индекс указывается только для многомерных массивов, определяет, к какому измерению массива применяется функция. Например:

```
Dim  aArray(5 To 10,15 To 20)
…
Lbound (aArray, 2) –  определяется нижняя граница второго измерения массива;
```

Ubound (ИмяМассива, Индекс) – возвращает верхнюю границу диапазона индекса массива;

IsArray (ИмяПеременной) – проверка, является ли переменная массивом.

Пример 3.7. Ввод и вывод одномерного массива.

```
Option Explicit
Dim OldMassiv(5 To 10) As Single, NewMassiv() As Single
Sub VvodMasiv1(Massiv As Variant )
    Dim i As Integer
    For i = LBound(Massiv) To UBound(Massiv)
        Massiv(i) = Val(InputBox(“Введите” & Str$(i) & “ элемент массива”))
        Print “Massiv(“; i; “)=”; Massiv(i)
    Next i
End Sub
Private Sub Form_Click()
     Cls
     VvodMassiv1  OldMassiv
End Sub
```

В функции InputBox для отображения номера вводимого элемента массива формируется строка символов *«Введите» & Str$(i) & « элемент массива»*, в которой для объединения фрагментов строки используется символ “***&***” амперсенд (коммерческое И).

Пример 3.8. Ввод и вывод двухмерного массива

```
Option Explicit
Dim A() As Single
```

```
Sub VvodMassiv2()
    Dim i As Integer, j As Integer
    Dim m As Integer, n As Integer
    n = InputBox(«Укажите число строк»)
    m = InputBox(«Укажите число столбцов»)
    ReDim A(n, m) As Single
    For i = 1 To Ubound(A,1)
        For j = 1 To Ubound(A,2)
            A(i, j) = Val(InputBox("Введите A(" & Str$(i) & "," & Str$(j) & ") _
                элемент массива"))
            Print A(i, j);
        Next j
        Print
    Next i
End Sub
```

В данном примере вывод элементов массива в форму осуществляется в виде матрицы. Это обеспечивается за счет использования разделителя ";" в конце оператора Print. Символ ";" в конце оператора Print оставляет курсор в текущей строке. Когда ввод данных в первую строку закончится, оператор Print без параметров возвращает курсор в начало строки.

Пример 3.9. Присвоение значений элементов одного одномерного массива элементам другого массива.

```
Sub CopyMassiv()
    Dim i As Integer
    ReDim NewMassiv(LBound(OldMassiv) To UBound(OldMassiv))
    For i = LBound(OldMassiv) To UBound(OldMassiv)
        NewMassiv(i) = OldMassiv(i)
        Print "NewMassiv("; i; ")="; NewMassiv(i)
    Next i
End Sub
```

```
Private Sub Form_Click()
    Cls
    VvodMassiv1 OldMassiv
    CopyMassiv
End Sub
```

Visual Basic 6 позволяет непосредственно присваивать значения элементов одного массива элементам другого массива:

```
Sub CopyMassiv1()
    Dim i As Integer
    ReDim NewMassiv(LBound(OldMassiv) To UBound(OldMassiv))
NewMassiv = OldMassiv
    For i = LBound(OldMassiv) To UBound(OldMassiv)
        Print "NewMassiv("; i; ")="; NewMassiv(i)
    Next i
End Sub
```

Однако при выполнении данной операции необходимо учитывать типы переменных и размерности массивов. Если новый массив динамический, то

операция обмена всегда успешна. В этом случае число элементов массива в левой части оператора присваивания при необходимости изменяется.

Использование функции Array для объявления массива.

При создании массива с помощью функции *Array* имя создаваемого массива формируется из этого ключевого слова *Array* и индекса, например Array1, Array2 и т.д. Например:

```
Dim Array1 As Variant.
```

Пример 3.10: Обмен данными двух массивов

```
Sub  ОбменДаннымиМассивов()
   Dim i As Long
   ReDim A(1 To 20) As Long
   ReDim B(1 To 20) As Long
   Dim Array1 As Variant, Array2 As Variant, Temp As Variant
   For i= 1 To  20
       A(i) = i           ' Сформированы и заполнены данными
       B(i) = 2 * i       ‘ массивы A и B
    Next i
    Array1 = A(): Erase A()      ‘ массиву  Array1 присвоено значение массива A
    Array2 = B(): Erase B()      ‘ массиву  Array2 присвоено значение массива B
    Temp = Array1
    Array1 = Array2
    Array2 = Temp
    For i = 1 To  20: Print Array1(i), Array2(i): Next i
End Sub
```

3.3.3. Операции с массивами

Массивы широко используются для хранения данных. При работе с файлами данных, вводе и выводе информации удобнее всего записывать данные предварительно в массив. Одной из операций с массивами является сортировка данных.

Известно много различных способов сортировки массивов. Однако, независимо от используемого способа в программе осуществляется сравнение текущего элемента массива с другим элементом этого же массива и, в зависимости от результатов сравнения, осуществляется обмен данными между этими элементами. К сожалению, в VB 6 нет одной полезной функции ***Swap***, которая была в других версиях языка Бейсик. Эта функция была предназначена для обмена значениями двух переменных. Такую функцию можно создать самим. Принцип работы функции обмена двух переменных данными показан на рис. 3.12.

Функция обмена двух переменных данными

```
Sub Swap(A As Variant, B As Variant)
```

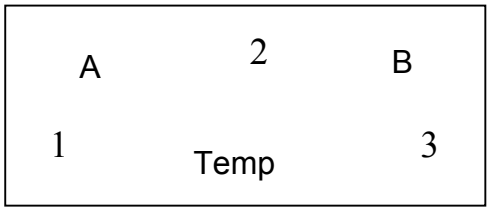


Рис.3.12. Обмен данными

```
    Dim Temp As Variant
    Temp = A
    A = B
    B = Temp
End Sub
```

Сортировка массивов

Самая простая процедура сортировки – перебор.

```
Sub SortMassiv(Massiv() As Variant)
    Dim i As Integer, j As Integer
    For i = LBound(Massiv) To UBound(Massiv) - 1
        For j = i + 1 To UBound(Massiv)
            If Massiv(j) < Massiv(i) Then
                Swap Massiv(i), Massiv(j)
            End If
        Next j
    Next i
End Sub
```

Вызывающую программу можно оформить следующим образом

```
Dim OldMassiv(5 To 10) As Variant, NewMassiv() As Variant
--------------------------------------------------------------------------------
Private Sub Form_Click()
    Cls
    Dim i As Integer
    VvodMassiv1 OldMassiv
    ReDim NewMassiv(LBound(OldMassiv) To UBound(OldMassiv)) As Variant
    NewMassiv = OldMassiv
    SortMassiv NewMassiv()
    For i = LBound(OldMassiv) To UBound(OldMassiv)
        Print OldMassiv(i), NewMassiv(i)
    Next i
End Sub
```

В этой процедуре для сохранения исходного массива введен массив NewMassiv. Сортируется новый массив. На печать выводятся оба массива.

Доработаем процедуру сортировки массива так, чтобы сортировка массива могла происходить в произвольном порядке. Для этого введем три переключателя Option1 – по возрастанию, Option2 – по убыванию, Option3 – не сортировать (рис. 3.13), которые устанавливаются пользователем. Переключатели могут иметь два состояния: включен – True и выключен - False. Причем если один из переключателей включен, то два другие переключателя будут выключены. Начальная установка производится при разработке программы.

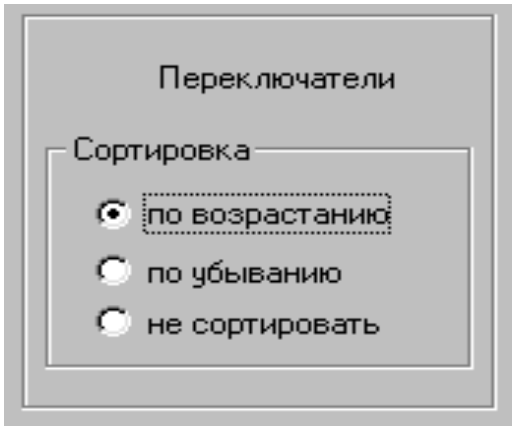


Рис. 3.13. Переключатели

```
Sub SortMassiv1(Massiv() As Variant)
    Dim i As Integer, j As Integer
    For i = LBound(Massiv) To UBound(Massiv) - 1
        For j = i + 1 To UBound(Massiv)
```

```
            If Option1 Then
               If Massiv(j) < Massiv(i) Then
                    Swap Massiv(i), Massiv(j)
               End If
            ElseIf   Option2 Then
               If  Massiv(j) > Massiv(i) Then
                    Swap Massiv(i), Massiv(j)
               End If
            Else
               Exit Sub
            End If
         Next j
      Next i
   End Sub
```

При запуске программы переключатели устанавливаются в состояние, указанное при разработке программы.

В качестве переключателей можно использовать элемент управления OptionsButton.

Этот элемент управления может быть снят или установлен. В группе может быть установлен только один переключатель.

Важнейшим событием для переключателя является событие Click. При щелчке мышью по переключателю меняется значение его свойства Value. Свойство Value может принимать три значения: 0 - не отмечен, 1 – отмечен, 2 – отмечен, но не доступен. Последнее значение может быть установлено только программным путем. Например, Option1.Value = 2. Объединить объекты в группу можно с помощью рамки (Frame), которая обладает свойством контейнера. Свойство Value объекта OptionsButton используется по умолчанию, поэтому при обращении к этому свойству указывать его не обязательно, достаточно указать объект, поэтому выражения If Option1.Value Then … и If Option1 Then … эквивалентны.

3.3.4. Упражнения: работа с массивами

Разработать программу для ввода данных в двухмерный массив с клавиатуры. Преобразовать двухмерный массив в одномерный и выполнить сортировку массива по возрастанию (убыванию) значений элементов массива. Вывести на форму элементы не отсортированного и отсортированного массивов.

3.3.5. Закрепление материала

1. Приведите классификацию массивов.
2. Приведите примеры одномерного и многомерного массивов.

3. Какими операторами объявляются статические массивы?
4. Каков порядок объявления динамических массивов?
5. Какими операторами объявляются динамические массивы?
6. Приведите текст программы для ввода данных в одномерный массив.
7. Напишите основные функции для работы с массивами.
8. Приведите текст программы для ввода данных в двухмерный массив.
9. Напишите программу для сортировки одномерного массива.

Задания для самостоятельных занятий

1. Напишите процедуру для обмена данными между двумя массивами.
2. Напишите программу для проверки значения свойства Value элемента управления OptionButton.
3. Вычислить определитель матрицы.
4. Вычислить сумму элементов одномерного массива.
5. Вычислить сумму элементов двухмерного массива.
6. Вычислить сумму элементов двухмерного массива, расположенных на главной диагонали.
7. Вычислить сумму элементов двухмерного массива, расположенных на обратной диагонали.
8. Вычислить сумму четных элементов двухмерного массива.
9. Вычислить сумму нечетных элементов двухмерного массива.
10. Вычислить сумму элементов двух матриц.
11. Вычислить произведение матрицы на вектор.
12. Вычислить произведение двух матриц.

3.4. Массив элементов управления. Управляющий элемент сетка

3.4.1. Массив элементов управления

Понятие о массиве элементов управления

Microsoft Visual Basic 6.0 предоставляет пользователям несколько дополнительных средств, помогающих создавать эффективные и гибкие приложения. Одно из них – массив элементов управления (control array). Массив элементов управления представляет собой группу элементов управления с одинаковыми именами, типом и обработчиками событий. Однако, элементы такого массива сохраняют и индивидуальные значения свойств. Чтобы различить эти элементы управления используют такие их свойства как *Index*, *Caption* и *Tag*. Каждый элемент массива управления имеет, по крайней мере, один элемент, у которого значение свойства Index равно нулю. Максимальное значение свойст-

ва Index равно 32767, если это не ограничено размером оперативной памяти компьютера.

Обычно такие массивы применяются для элементов управления ”меню”, групп переключателей или флажков, а также для вывода на экран значений элементов массивов.

У массивов элементов управления имеется три существенных преимущества:

- позволяют добавлять новые элементы управления в период выполнения программы. Это особенно важно, когда заранее не известно, сколько новых элементов понадобится в период выполнения. Кроме того, добавление элементов управления только по мере необходимости обеспечивает экономию системных ресурсов. Элементы управления, добавляемые в период выполнения, называются динамическими;
- каждый новый элемент, добавляемый в массив, наследует общие для массива процедуры обработки событий, что облегчает программирование;
- элементы управления в массиве способны разделять код. Например, если на форме имеется несколько текстовых полей, принимающих данные в виде дат, массив этих элементов управления можно настроить так, чтобы текстовые поля использовали общий код для проверки вводимых дат.

Создание массивов элементов управления на этапе разработки

Имеется несколько способов создания массивов элементов управления на этапе разработки:

- копирование и вставка элемента управления;
- присвоение двум существующим элементам управления одного типа одинаковых имен;
- присвоение существующим элементам управления одного типа одинаковых имен.

Алгоритм работы по созданию массива элементов управления путем копирования:

1. Поместите на форму первый элемент управления и задайте начальные значения свойств, общие для всех элементов управления.
2. Установите свойство Name этого элемента.
3. Скопируйте элемент управления и вставьте его в форму. На запрос программы: “Создать массив элементов управления?” ответьте утвердительно. Запрос выдается только при вставке первой копии элемента управления.
4. Повторяйте операцию вставки, пока на форму не будет добавлено нужное количество элементов управления.

Алгоритм работы по созданию массива элементов управления путем присвоения двум однотипным элементам управления одинаковых имен:

1. Поместите на форму два одинаковых элемента управления.
2. Присвойте имя первому элементу управления.
3. Присвойте такое же имя второму элементу управления. Программа выдаст запрос о создании массива элементов управления. Ответьте утвердительно.

Алгоритм работы по созданию массива элементов управления путем присвоения однотипным элементам управления одинаковых имен:

1. Создайте нужное количество элементов управления.
2. Присвойте второму элементу управления такое же имя, как и у первого элемента. Программа выдаст запрос о создании массива элементов управления – ответьте утвердительно.
3. Присвойте такие же имена остальным элементам управления. значения индексов устанавливаются автоматически.

Примечание. Индекс первого элемента управления может быть отличным от нуля.

При программировании массива элементов управления используется следующий синтаксис:

```
ИмяЭлементаУправления (Индекс).Свойство = Значение
```

Например:

```
txtText1(0).Text = “Фамилия”
txtText1(1).Text = ”Имя”
```

Динамическое добавление элементов управления в период выполнения

Для динамического добавления элементов управления на форму используется оператор ***Load*** и метод ***Add***.

Создайте элемент управления и присвойте свойству Index начальное значение. Введите команду

```
Load объект(индекс)
```

Оператор Load копирует значения всех свойств объекта, кроме *Visible*, *Index* и *TabIndex*, из первого элемента массива. Поэтому вновь созданный элемент управления размещается под исходным элементом управления и недоступен. Чтобы сделать этот элемент управления доступным, необходимо сместить его в сторону используя свойства *Top* и *Left*, а также, присвоить свойству ***Visible*** значение ***True***:

```
объект(индекс).Top = объект(индекс-1).Top + объект(индекс-1). Height
объект(индекс).Left = объект(индекс-1).Left + объект(индекс-1). Height
 объект(индекс).Visible = True
```

Попытка загрузки уже существующего элемента управления вызывает ошибку периода выполнения. Если свойство Index у объекта не установлено, то также выдается сообщение об ошибке.

Удаление элементов массива элементов управления осуществляется оператором ***Unload***. Нельзя удалять исходный (нулевой) элемент массива, так как после этого его нельзя будет восстановить программным путем:

```
Unload  объект(Index)
```

.

Пример 3.11. Добавление элементов управления по щелчку мыши по кнопке cmdAddTextbox:

```
Private Sub cmdAddTextBox_Click().
    Dim i As Integer.
    ‘ищем следующий доступный индекс (индексация от 0)
     i = txtMassiv().Count
    ‘помещаем новый элемент управления в форму
     Load txtMassiv(i)
     ‘позиционируем новый элемент  массива
    txtMassiv(i).Top = txtMassiv(i-1).Top + txtMassiv(i).Height
’делаем новый элемент массива видимым
    txtMassiv(i).Visible=True.
End Sub
```

3.4.2. Управляющий элемент сетка

Управляющий элемент ***MSFlexGrid*** – сетка предназначен для вывода данных на экран. Вводить данные в ячейки сетки непосредственно нельзя.

Сетки нет среди стандартных элементов панели ToolBox. Для ее загрузки необходимо ввести команду ***Project\Components*** и выбрать в диалоговой панели элемент ***Microsoft Flex Grid Control 5.0.***

Основные свойства сетки

Сетка имеет более 80 свойств, 20 событий и 10 методов.

Heigt – высота; **Width** - ширина; **Enabled** – доступность.

ScrolBar - линейка прокрутки, имеет 4 значения (0- выводится автоматически, 1- горизонтальная, 2- вертикальная, 3 – обе).

Свойства сетки легко настраивать с помощью диалоговой панели: щелкните по полю ***Custom*** окна ***Properties*** – появится кнопка *троеточие,* щелкните мышью по этой кнопке – появится диалоговая панель. В закладках диалоговой панели можно установить необходимые параметры.

Cols, **Rows** – устанавливает число колонок и столбцов.

Col, **Row** - возвращают/ устанавливают номер колонки и строки.

ColPosition, RowPosition – позволяют перемещать целые колонки и столбцы по сетке. Синтаксис использования свойства:

```
ИмяСетки.ColPosition(N)=значение%
```

ИмяСетки.RowPosition(N)=значение%

здесь N номер колонки или строки.

ColWidth, RowHeight - ширина и высота столбца. Синтаксис использования этих свойств аналогичен предыдущему примеру.

Text, TextMatrix - возвращает или устанавливает текст, хранящийся в текущей ячейке. Свойство TextMatrix имеет синтаксис:

TextMatrix (номер строки, номер столбца) = строка

Это свойство позволяет читать текст в произвольной ячейке без изменения свойств Row и Col.

ColAlignment - выравнивание текста в ячейках:

0 - выравнивание по левому краю;

1 - выравнивание по правому краю;

2 - центрирование текста.

Синтаксис: ИмяСетки. ColAlignment (индекс) = Значение%

Можно использовать 10 возможных значений свойства Alignment для управления выравниванием информации в ячейках.

FixedCols, FixedRows, FixedAlignment – фиксация строк, колонок или значений:

Имя Сетки. FixedRows = ЧислоФиксированныхСтрок%

LeftCol, TopRow – номер самого левого столбца и самой верхней строки, которые будут отображаться в сетке. Эти свойства используют, когда таблица не помещается в форме:

Имя Сетки. LeftCols = ЛеваяКолонка%

Имя Сетки. TopRows = ПерваяСтрока%

GridsLines – контролирует отображение разделительных линий.

ScrollBars – контролирует отображение линеек прокрутки.

Пример 3.12. Выделить ячейку в 4 –й строке, 5 - м столбце, записать в нее текст и вывести в форму. Очистить ячейку в нулевой строке и нулевой колонке:

```
MSFlexGrid1.Col = 3
MSFlexGrid1.Row =4
MSFlexGrid1.Text = “Лев Толстой “
MSFlexGrid1.Col =0
MSFlexGrid1.Row=0
MSFlexGrid1.Text = ”“
```

или

```
MSFlexGrid1.TextMatrix(3,4)="Лев Толстой"
```

Свойства для выделения ячеек внутри таблицы

ColSel, RowSel – выделение ячеек.

Сначала устанавливается начальная ячейка, а затем указывается область:

```
MSFlexGrid1:Row=0: MSFlexGrid1.Col=0
MSFlexGrid1:ColSel=4: MSFlexGrid1.RowSel=3
```

Выделяются колонки с 0 до 4 и строки с 0 до 3

Clip - используется для считывания и установки содержимого выделенной части таблицы:

```
Имя сетки.Clip=Строка
```

В строке содержатся данные, предназначенные для разных столбцов. Эти данные должны отделяться друг от друга символами табуляции, а строки должны отделяться символами возврата каретки.

```
MSFlexGrid1.Col =0; MSFlexGrid1.Row =0
MSFlexGrid1.ColSel=2; MSFlexGrid1.RowSel=1
S$=Str$(1)+vbTab+Str$(2)+vbTab+Str$(4)
S$=$+vbCr+Str$(4)+vbTab+Str$(5)+vbTab+Str$(6)
MSFlexGrid1.Clip=S$
```

Здесь *vbTab* – константа табуляции;

VbCr - константа возврат каретки.

FillStyle - автозаполнение. Заполнение выделенной области данными, внесенными в одну ячейку. По умолчанию значение равно 0.

HighLight – сообщение о выделении ячейки: 0 - никогда, 1 – подсвечивание ячейки; 2 – ячейка подсвечена даже если потеряла фокус.

AllowBigSelection – разрешает или запрещает выделять столбцы щелчком мыши по заголовку. Имеет два значения: True и False.

AllowUserResizing – устанавливает возможность изменения размеров строк и столбцов с помощью мыши: 0 - нельзя, 1 – можно менять размеры колонок; 2 – можно менять размеры строк; 3 – можно менять и то и другое.

Sort – сортировка сетки: 0 – нет сортировки; 1 – по возрастанию; 2 – по убыванию ; 3 – по возрастанию, но не конвертировать строки в числа; 4 – по убыванию, но не конвертировать строки в числа; 5 – по возрастанию без учета регистра; 6 – по убыванию без учета регистра; 7 – по возрастанию с учетом регистра; 8 – по убыванию с учетом регистра; 9 – для сравнения используется событие ***Compare***.

События и методы сетки

EnterCell, LeaveCell – выделение и отмена выделения ячейки при щелчке мыши.

RowColChange - сохранение информации из глобальной переменной в текущей ячейке.

AddItem – вставка строк.

```
MsFlexGrid1.AddItem Item$[,номер строки]
```

Строка в переменной *Item$* помещается в первую колонку новой записи. Дополнительный параметр *номер строки* дает возможность указывать, перед какой строкой следует добавить новую запись. По умолчанию строка вставляется после последней строки таблицы.

RemoveItem – удаление строки записи.

```
MsFlexGrid1. RemoveItem <номер строки>
```

3.4.3. Упражнения: использование массивов управляющих элементов и сетки

Задача 1. Разработать программу для добавления и удаления элементов массива управляющих элементов.

Форма для реализации этой задачи приведена на рис. 3.14. Поместим на форму один элемент управления TextBox – Text1 и присвоим свойству Index этого элемента значение 0. Поместим на форму также две кнопки для добавления и удаления элементов массива.

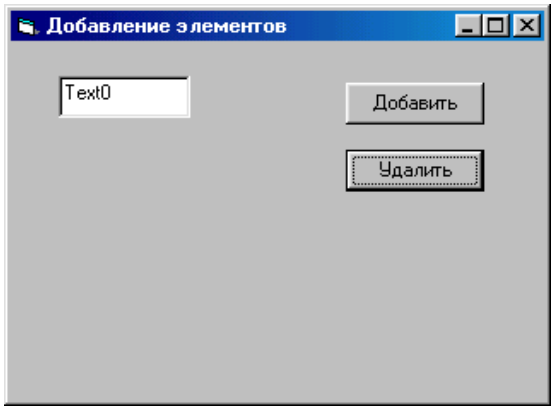


Рис.3.14. Форма для добавления и удаления элементов массива управляющих элементов

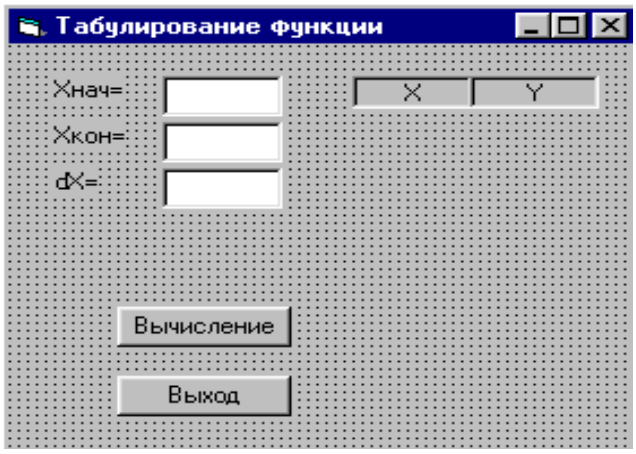


Рис.3.15.Использование массива элементов управления

Текст программы.

```
Private Sub Command1_Click()
 Dim i As Single
 i = Text1().Count
   Load Text1(i)
   Text1(i).Text = "Текст" & Str$(i)
   Text1(i).Top = Text1(i - 1).Top +  Text1(0).Height
   Text1(i).Visible = True
End Sub

Private Sub Command2_Click()
        Dim i As Single
        i = Text1().Count - 1
        If i > 0 Then
        Unload Text1(i)
  End If
End Sub
```

Задача 2. Протабулировать функцию *y=sin(x)* с использованием массива элементов управления.

Порядок работы.

1. Запустите программу Visual Basic.
2. Откройте новый проект.
3.Разработайте эскиз формы (рис.3.15).
4. Опишите свойства элементов управления (табл. 3.8).
5. Создайте на форме массив элементов управления типа Label для вывода надписей Хнач, Хкон, dX:

- поместите на форму Надпись. Установите ее свойства. Свойству Index присвойте значение 0;
- скопируйте Надпись и вставьте ее на форму два раза. Просмотрите значения свойства Index у вставленных элементов. (1,2);

Таблица3.8

Описание свойств элементов формы

Тип элемента	*Элемент*	*Свойство*	*Значение*
Форма	Form1	Name	frmMassivUpravl
		Caption	Табулирование функций
Надпись	Label(0)	Name	lblLabel1
		Caption	Хнач=
		Index	0
	Label(1)	Name	lblLabel1
		Caption	Yнач=
		Index	1
	Label(2)	Name	lblLabel1
		Caption	dX=
		Index	2
Текстовое поле	Text1(0)	Name	txtText1
		Caption	0
		Index	0
	Text1(1)	Name	txtText1
		Caption	0
		Index	1
	Text1(2)	Name	txtText1
		Caption	0
		Index	2
	Label2(0)	Name	lblArgument
		Caption	X
		Aligment	2
		Index	2
		Enabled	False
	Label3(0)	Name	lblFunction
		Caption	Y
		Aligment	3
		Index	2
		Enabled	False
Командная	Command1	Name	cmdCalc

Тип элемента	*Элемент*	*Свойство*	*Значение*
кнопка			
		Caption	Вычисление
	Command2	Name	cmdExit
		Caption	Выход

6. Создайте массив элементов управления типа TextBox для ввода значений Хнач, Хкон, dX:

- поместите на форму Текстовое поле. Установить его свойства. Свойству Index присвойте значение 0;
- скопируйте Текстовое поле и вставьте его на форму два раза. Просмотрите значения свойства Index у вставленных элементов. (1,2);

7. Установите на форму две Надписи, присвойте им имена, установите значение свойства Index у обоих элементов равным нулю, измените цвет полей и присвойте свойству Caption этих полей значение “X” и “Y” соответственно.

8. Создайте две кнопки: “Вычисление” и “Выход”.

9. Напишите тексты программы.

```
Dim x As Single, y As Single, dx As Single
-------------------------------------------------------------
Private Sub cmdCalc_Click()
  Dim i As Integer
  Xnach = Val(txtText1(0).Text)
  Xkon = Val(txtText1(1).Text)
  dx = Val(txtText1(2).Text)
  n = Int((Xkon - Xnach) / dx) + 1
  x = Xnach
  For i = 1 To n
      ' Загружаем элементы управления
      Load lblArgument(i):    Load lblFunction(i)
      ' Вычисляем значение функции
      y = Sin(x)
      ' Настраиваем параметры добавленных элементов управления
      lblLabel2(i).Top = lblLabel2(i - 1).Top + lblLabel2(0).Height
      lblLabel3(i).Top = lblLabel3(i - 1).Top + lblLabel3(0).Height
      ‘ при использовании функции Format значение аргумента не должно
      ‘ быть равно нулю, иначе возникает ошибка периода выполнения
      If x <> 0 Then
          lblArgument(i).Caption = Str$(Format(x, "####.##"))
      Else
          lblArgument(i).Caption = Str$(x)
      End If
      If y <> 0 Then
          lblFunction(i).Caption = Str$(Format(y, "####.##"))
      Else
         lblFunction(i).Caption = Str$(y)
      End If
         lblArgument Visible = True: lblFunction(i).Visible = True
      x = x + dx
  Next i
```

```
End Sub
-------------------------------------------------------------------------------------------
Private Sub cmdExit_Click()
    Unload Me
End Sub
```

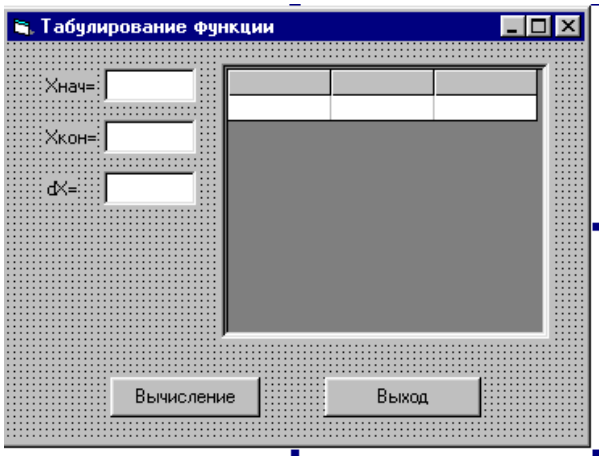


Рис. 3.16. Использование сетки

10.Сохраните программу на диске.

11. Проведите отладку программы.

12. Создайте исполняемый файл и запустите его для проверки работы.

Задача3. Выполнить табулирование функции с использованием сетки (рис.3.16).

Порядок работы.

1. Выполните пункты 1-6 задания1.

2. Добавьте на панель инструментов сетку ***MSFlexGrid*** командой ***Project\Components\ Microsoft Flex Grid Control 5.0.***

3. Поместите на форму сетку MSFlexGrid. Установите начальное число строк равное 2 и начальное число колонок равное 3.

4. Напишите текст программы

```
Dim Xn As Single, Xk As Single, Dx As Single
Dim X As Single, Y As Single, N As Integer
-------------------------------------------------------------
Private Sub Command1_Click()
   Xn = Val(Text1(0).Text)
   Xk = Val(Text1(1).Text)
   Dx = Val(Text1(2).Text)
   Cls
   N = Int((Xk - Xn) / Dx) + 1
   Grid1.Rows = N + 1
   For I = 1 To N
       Y = Sin(X)
       Grid1.Row = I
       Grid1.Col = 0
       Grid1.ColAlignment(0) = 2
       Grid1.Text = Str$(I)
       Grid1.Col = 1
       If X <> 0 Then
           Grid1.Text = Str$(Format(X, "#.##"))
       Else
           Grid1.Text = Str$(X)
       End If
       Grid1.ColAlignment(1) = 2
       Grid1.Col = 2
       If X <> 0 Then
```

```
            Grid1.Text = Str$(Format(Y, "#.##"))
        Else
            Grid1.Text = Str$(Y)
        End If
        Grid1.ColAlignment(2) = 2
        X = X + Dx
    Next I
End Sub
---------------------------------------------------------------------
Private Sub Command2_Click()
    Unload Me
End Sub
---------------------------------------------------------------------
Private Sub Form_Load()
    Grid1.Row = 0
    Grid1.Col = 1
    Grid1.Text = "X"
    Grid1.ColAlignment(1) = 2
    Grid1.Col = 2
    Grid1.Text = "Y"
    Grid1.ColAlignment(2) = 2
End Sub
```

6. Выполните работу согласно п. 10 - 12 задания 1.

3.4.4. Закрепление материала

1. Что такое массив элементов управления?
2. В чем заключается преимущество в использовании массивов элементов управления по сравнению с отдельными элементами управления?
3. Как создать массив элементов управления на этапе разработки программы?
4. Как добавить элементы управления во время выполнения программы?
5. Для чего предназначен элемент MSFlexGrid?
6. Как поместить элемент MSFlexGrid на панель элементов управления ToolBox?
7. Как установить число строк и столбцов на сетке?
8. Как вызвать окно диалога для настройки параметров сетки?
9. Напишите фрагмент программы для присвоения значения ячейке на пересечении пятой строки и третьей колонки.
10. Как зафиксировать первую строку?
11. Напишите фрагмент программы для установки начальных параметров сетки.
12. Напишите фрагмент программы для добавления строки.

Задания для самостоятельной работы

1. Разработать программу для размещения элементов управления типа Label или TextBox по периметру прямоугольника произвольных размеров без наложения объектов друг на друга.

2. Разработать программу для размещения элементов управления типа Label в вершинах прямоугольного треугольника со сторонами a и b.

3. Разработать программу для размещения элементов управления типа TextBox в вершинах прямоугольника со сторонами a и b.

4. Разработать программу для размещения элементов управления типа CommandBox в вершинах равностороннего треугольника со стороной a.

5. Разработать программу для размещения элементов управления типа Label по окружности радиусом R через n градусов.

6. Разработать программу для размещения элементов управления типа TextBox по дуге M градусов с шагом D градусов и пронумеровать их.

7. Разработать программу для размещения элементов управления типа Label по главной диагонали прямоугольника произвольных размеров так, чтобы они не перекрывали друг друга.

8. Разработать программу для размещения элементов управления типа TextBox по обратной диагонали прямоугольника произвольных размеров так, чтобы они не перекрывали друг друга.

9. Разработать таблицу размером 10X10 и заполнить ее целыми числами натурального ряда с использованием сетки.

10. Разработать таблицу размером 5X10 и заполнить ее нечетными числами натурального ряда с использованием сетки.

11. Разработать таблицу размером 10X5 и заполнить ее четными числами натурального ряда с использованием сетки.

12. Разработать таблицу размером 10X10 и заполнить ее простыми целыми числами с использованием сетки.

13. Разработать таблицу размером 8X10 и заполнить ее рядом целых чисел, изменяющихся по арифметической прогрессии от 1 до 1000 с шагом D с использованием сетки.

14. Разработать таблицу размером 10X10 и заполнить ее рядом чисел изменяющихся по геометрической прогрессии от 1 до 1000 с шагом D с использованием сетки.

15. Разработать таблицу размером 5X10 и заполнить ее целыми числами, каждое из которых равно сумме номера строки и номера столбца на пересечении которых находится ячейка с использованием сетки.

4. Графические средства Visual Basic.

4.1. Графические объекты Visual Basic.

4.1.1. Экран. Метод Scale

Экран

В графическом режиме экран видеомонитора представляет собой набор точек, расположенных по строкам. Каждая точка на экране называется пикселем, число точек по горизонтали и вертикали определяет разрешающую способность экрана. Для работы в среде Windows разрешающая способность должна быть не менее 800x600 пикселей.

Для работы с графикой Visual Basic 6.0 имеет графические объекты, графические элементы управления и графические методы.

К ***графическим объектам*** относятся **форма** (Form) и **графическое окно** (Picture Box). К этим объектам могут быть применены графические методы.

Графические элементы управления **–** позволяют помещать на графические объекты линии и геометрические фигуры. К ним относятся элементы управления *Line* и *Shape*. Особо следует выделить элемент управления *Image*. Он не является ни графическим объектом, ни графическим элементом управления, так как не позволяет применять графические методы, но может использоваться для вставки рисунков.

Графический метод – это метод, который позволяет изображать на объекте данного класса какой-нибудь геометрический элемент, например точку, линию, окружность и т.д. Графический метод ориентирован на абсолютную или относительную систему координат экрана дисплея.

Абсолютная система координат ориентирована на верхний левый угол экрана со значениями x=0; y=0, то есть представляет собой IV квадрант прямоугольной декартовой системы координат (рис.4.1).

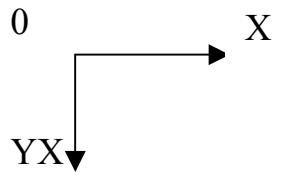


Рис. 4.1. Экранная система координат

Основной единицей измерения в VB является твип. Твип = 1/1440 логического дюйма. Логический дюйм – это расстояние на форме, которое при печати на принтере будет равно 1 дюйму (1 дюйм =2,5 см). Используя свойство *ScaleMode*, можно перейти к другим единицам измерения:

1. Твип (по умолчанию);
2. Точка (72 на дюйм);
3. Пиксель (пиксель – одна точка на экране монитора, число пикселей определяется установленным разрешением экрана Windows);
4. Символ (12 точек в высоту и 20 в ширину);
5. Дюйм;

6. Миллиметр;
7. Сантиметр.

Form1.ScaleMode = 3 – установлена единица измерения *пиксель*.
Form1.ScaleMode = 7 - установлена единица измерения *сантиметр*.

Метод Scale

Для установки другого масштаба, пользовательского, используется метод *Scale*. Синтаксис метода:

[имяОбъекта]. Scale (x1,y1) – (x2,y2)

где x1,y1 – координаты верхнего левого угла экрана; x2,y2- координаты правого нижнего угла экрана.

При отрицательных значениях координат меняется ориентация объекта.

```
Scale(-5,-10)-(5,10)  ‘ исходная, экранная система координат
Scale(-5,10)-(5,-10)  ‘ изменена ориентация оси Y
```

Можно использовать и такой способ установки координат:

```
Object. ScaleLeft =xxxx   -  левый угол (верхний)
Object. ScaleTop = xxxx  - верхний угол (левый)
```

Например:

```
Picture1. ScaleLeft =100   -  левый угол (верхний)
Picture1. ScaleTop = 50  - верхний угол (левый)
```

Можно также использовать для установки пользовательской системы измерений значение свойств ScaleWidth, ScaleHeight:

ScaleWidth = 3200 - масштаб по ширине
ScaleHeight = 2000 - масштаб по высоте.

Если в качестве объекта используется форма, то имя объекта в графических методах можно не указывать.

Объект Screen

Экран в VB представляет собой системный объект ***Screen***. Наряду с объектом Screen существует и свойство ***Screen*** объекта ***Global***.

Поскольку имеется только один единственный экран Windows, переменная типа Screen не объявляется, а используется системный объект Screen.

Объект Screen имеет ряд свойств:

ActiveControl - определяет активный элемент управления;
ActiveForm - определяет активную форму;
FontCount - количество доступных шрифтов;
Fonts( ) - возвращает имена всех доступных шрифтов;
Height - высота экрана;
MouseIcon - позволяет установить пользовательскую пиктограмму для курсора мыши;
TwipsPerPixelX - количество твипов в пикселе (разрешение по горизонтали);

TwipsPerPixelY - количество твипов в пикселе (разрешение по вертикали);

Width - ширина экрана.

Параметры экрана по умолчанию измеряются в твипах:

```
HeightInTwips = Screen.Height
WidthInTwips = Screen.Width
```

Эти параметры можно пересчитать и в пиксели:

```
HeigtInPixel = Screen.Height / Screen.TwipsPerPixelY
WidthInPixel = Screen.Width / Screen.TwipsPerPixelX
```

Свойство ***ActiveControl*** позволяет обращаться к объектам без указания конкретного объекта. Чтобы обратиться к активному элементу, нужно написать текст программы следующего вида

```
Screen.ActiveControl.Свойство
```

Например:

```
Private Sub mnuDelete_Click ( )
      ‘Удаление выделенного текста.
      Screen. ActiveControl.SelText =””
End Sub.
```

Аналогично обращаются и к активной форме:

```
Screen.ActiveForm.ActiveControl.SelText = “”
```

Ключевое слово ***TypeOf*** позволяет проверить тип активного элемента управления:

```
Private Sub mnuDelete_Click()
    If TypeOf Screen.ActiveControl Is TextBox Then
        'Удаляется текст.
        Screen.ActiveControl.SelText = ""
    ElseIf TypeOf Screen.ActiveControl Is PictureBox Then
        'Удаляется рисунок
        Screen.ActiveControl.Picture = LoadPicture()
    End If
End Sub
```

В данной процедуре делается проверка: если активный элемент *TextBox*, то удаляется выделенный текст, а если активным является элемент *PictureBox*, то удаляется рисунок (удаление рисунка осуществляется путем загрузки “пустого” рисунка – LoadPicture ()).

В приведенном примере текст программы включен в обработчик события пункта меню. Пункт меню не имеет фокуса. Нельзя использовать для удаления теста кнопку, так как она имеет фокус. Поэтому при щелчке мышью по кнопке она получает фокус и программа будет пытаться удалить текст с кнопки, но так как кнопка не имеет свойства SelText, то будет выдано сообщение об ошибке.

4.1.2. Элементы управления Line и Shape

Элемент управления Line

Элемент управления Line позволяет рисовать линии различной толщины и стиля (Рис. 4.2).

Этот элемент обладает 15 свойствами. Основными являются *X1, Y1, X2, Y2, BorderStyle , BorderWidth и BorderColor.*

X1, Y1, X2, Y2 – координаты концов линии, X1, Y1 – координаты левого конца линии; X2, Y2 - координаты правого конца линии.

BorderStyle - определяет стиль линии:

0 - невидимая; 1 – сплошная; 2 – пунктирная; 3 – пунктирная с коротким штрихом; 4 – штрих пунктирная; 5 – штрих штрих пунктирная; 6- InsideSolid. Данное свойство работает только при значении свойства BorderWidth=1

BorderWidth - определяет толщину линии и может принимать любые значения кроме нуля.

BorderColor - определяет цвет объекта. Существует четыре способа задания цвета:

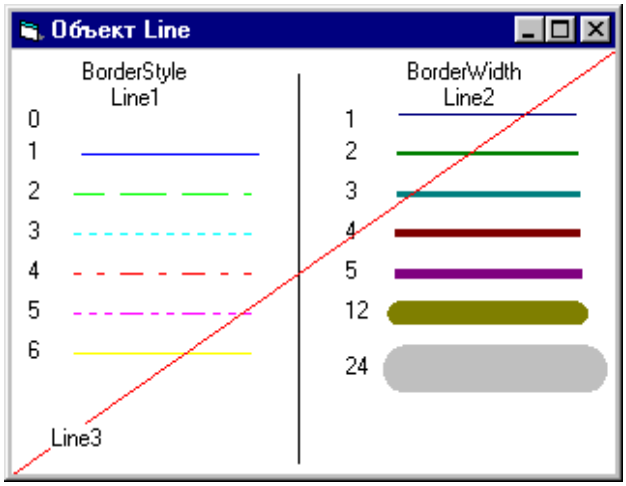


Рис.4.2. Свойства элемента управления Line

- непосредственное задание ***16-ричной константой***. Например: &H00000000& - черный цвет; &H000080FF& - красный цвет. Такой способ задания цвета в программе достаточно неудобен;

- использование ***RGB*** – функции: RGB (Red, Green, Blue).

RGB – функция формируется из трех цветов: красного, зеленого и синего. Каждый цвет задается числовой константой от 0 до 255. Эта функция позволяет формировать различные цвета и оттенки. Всего можно сформировать 255*255*255 различных оттенков. Например:

```
R=100: G=150: B=75
Line.BorderColor=RGB(R,G,B) 'темно зеленый цвет
```

- использование ***констант Visual Basic.*** Имеется 8 констант: *vbBlack* - черный; *vbBlue* - синий; *vbCyan* - голубой; *vbGreen* - зеленый; *vbMagenta* - сиреневый; *vbRed* - красный; *vbWhite* - белый; *vbYellow* – желтый;

- использование функции ***QBColor (C)***, где C - цвета от 0 до 15:

Черный	- 0	Коричневый	- 6
Темно-синий	- 1	Светло-серый	- 7
Темно-зеленый	- 2	Темно – серый	- 8
Темно-голубой	- 3	Синий	- 9
Темно-красный	- 4	Зеленый	- 10
Темно-сиреневый	- 5	Голубой	- 11
Красный	- 12	Желтый	-14
Сиреневый	- 13	Белый	-15

Объект *Line* устанавливается на форму во время разработки программы, как и другие объекты управления. Положение объекта Line на форме можно

изменить программным путем. Для управления объектом Line необходимо поместить объектный код в обработчик события ***Resize***.

Пример 4.1. Свойства прямой линии (рис.4.2)

```
Option Explicit
Dim x1 As Single, x2 As Single, y1 As Single, y2 As Single
Dim i As Integer
---------------------------------------------------------------------------
Private Sub Form_Resize()
    ' горизонтальная черта по диагонали.
    Line3.x1 = 0
    Line3.y2 = 0
    Line3.x2 = Form1.ScaleWidth
    Line3.y1 = Form1.ScaleHeight
    Line3.BorderColor = vbRed
End Sub
---------------------------------------------------------------------------
Private Sub Form_Click()
    For i = 0 To 6
        Line1(i).BorderStyle = i
        Line1(i).BorderColor = QBColor(i + 8)
        Line2(i).BorderColor = QBColor(i + 1)
     Next i
End Sub
```

Объекты Line1 и Line2 представляют собой массивы элементов управления.

Элемент управления Shape

Элемент управления Shape служит для изображения геометрических фигур: квадратов, прямоугольников, эллипсов, окружностей.

Элемент Shape обладает практически теми же свойствами, что и элемент Line, но имеет и ряд специфических свойств. Основные свойства *Top, Left, Height, Width, Shape, BorderStyle, BorderWidth, FillStyle, FillColor.*

Top, Left, Height, Width – эти свойства аналогичны свойствам других элементов управления. Они определяют положение объекта на форме и его размеры.

Shape – определяет форму объекта и может принимать следующие значения: 0 - прямоугольник; 1 – квадрат; 2 – эллипс; 3 – круг; 4 – прямоугольник с закругленными углами; 5 – квадрат с закругленными углами (рис. 4.3.).

FillStille - обеспечивает автоматическое заполнение фигур, построенных с помощью графических методов. Это свойство имеет 8 значений: 0 - однотонное заполнение; 1 – пусто; 2- горизонтальные линии; 3- вертикальные линии; 4- диагонали верхние; 5- диагонали нижние; 6- сетка; 7- сетка диагональная.

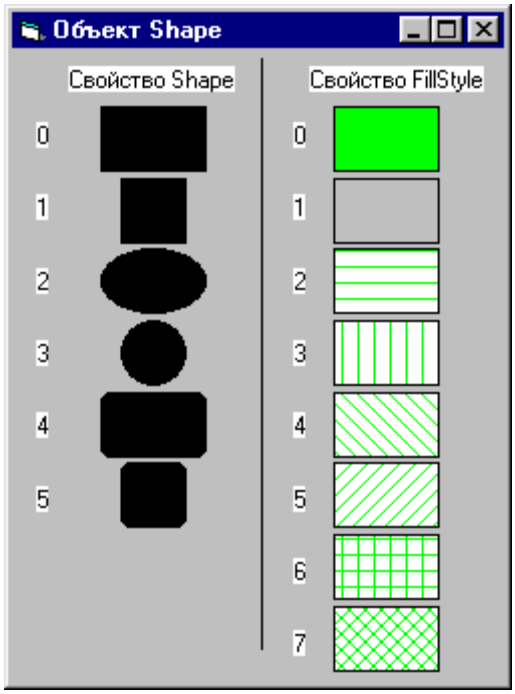


Рис.4.3. Свойства элемента управления Shape

BorderStyle, BorderWidth – определяют стиль контура и толщину линии соответственно. Эти свойства аналогичны соответсвующим свойствам объекта Line.

FillColor – определяет цвет заполнения объекта. Аналогичен свойству ***BorderColor*** объекта Line.

Пример 4.2. Демонстрация свойств элемента Shape.

Поместите на форму пять кнопок: Shape, FillStyle, BorderStile, BorderWidth, FillColor и один элемент управления Shape, напишите в обработчиках событий Click кнопок объектные коды:

```
Option Explicit
Dim i As Integer, j As Integer
-----------------------------------------
Private Sub Command1_Click()
      If i > 5 Then i = 0
      Shape1.Shape = i
      i = i + 1
End Sub
-----------------------------------------
Private Sub Command2_Click()
      If i > 6 Then i = 0
     Shape1.BorderWidth = 1
     Shape1.BorderStyle = i
     i = i + 1
End Sub
-----------------------------------------
```

```
Private Sub Command3_Click()
     If i = 0 Or i > 24 Then i = 1
     Shape1.BorderWidth = i
      i = i + 1
End Sub
-----------------------------------------
Private Sub Command4_Click()
     If i > 15 Then i = 0
     Shape1.FillColor = QBColor(i)
     i = i + 1
End Sub
-----------------------------------------
Private Sub Command5_Click()
      If i > 7 Then i = 0
     Shape1.FillStyle = i
     i = i + 1
 End Sub
```

Пример 4.3. Использование массива элементов управления Shape для демонстрации свойств элемента управления (рис. 4.3).

```
Option Explicit
Dim i As Integer
Const Pi = 3.14
-----------------------------------------------------------------------------------------
Private Sub Form_Click()
    For i = 1 To 5
        Load Shape1(i)
            Shape1(i).Shape = i
            Shape1(i).Top = Shape1(i - 1).Top + Shape1(i - 1).Height + 40
```

```
            Shape1(i).Visible = True
        Load Label1(i)
            Label1(i).Caption = i
            Label1(i).Top = Label1(i - 1).Top + Label1(i - 1).Height + 350
            Label1(i).Visible = True
    Next i
    For i = 1 To 7
        Load Shape2(i)
            Shape2(i).FillStyle = i
            Shape2(i).Top = Shape2(i - 1).Top + Shape2(i - 1).Height + 40
            Shape2(i).Visible = True
         Load Label2(i)
            Label2(i).Caption = i
            Label2(i).Top = Label2(i - 1).Top + Label2(i - 1).Height + 350
            Label2(i).Visible = True
    Next i
End Sub
-----------------------------------------------------------------------------------------------
Private Sub Form_Load()
    Me.Height = 5150
    Me.Width = 3800
End Sub
```

4.1.3. Управление пикселем

Чтобы использовать режим управления пикселем необходимо установить свойство ScaleMode =3

Число твипов, приходящихся на один пиксель возвращают функции TwipsPerPixelX и TwipsPerPixelY.

Для управления цветом точки используется метод ***Pset***. Синтаксис метода:

```
Pset(x,y) [, C].
```

Метод Pset можно использовать для изображения графиков функций, а также для закраски фигур произвольной формы. К сожалению, в VB нет полезной функции для закраски фигур произвольной формы.

Для определения цвета точки используется метод ***Point***.

Этот метод возвращает длинное целое, используемое в кодировании (шестнадцатеричную константу). Синтаксис метода:

```
Object. Point(x,y)
```

Пример 4.4. Построение графика функции на заданном интервале.

Построить график функции $e^x \sin(x)$ на отрезке от –2 до 2.

Порядок работы.

Для решения данной задачи необходимо протабулировать функцию на заданном отрезке с некоторым шагом и найти приближенное значение экстремумов функции. Затем провести масштабирование графического объекта (формы или элемента PictureBox) и построить график функции. Для построения

графика функции повторите операцию табулирования функции, и на каждом шаге стройте точку. Чтобы график функции был плотным, шаг табулирования следует выбирать достаточно малым. С целью сокращения времени построения графика при некотором ухудшении качества для построения графика можно использовать метод Line.

```
Option Explicit
Dim x As Single, y As Single
Dim xn As Single, ymin As Single
Dim xk As Single, ymax As Single
Dim dx As Single
-----------------------------------------
Function FNy(x As Single)
      y = Exp(x) * Sin(x)
      FNy = y
End Function
-----------------------------------------
Private Sub Form_Click()
     Picture1.Cls
     Picture1.ScaleMode = 3
    xn = Val(Text1(0).Text)
xk = Val(Text1(1).Text)
dx = Val(Text1(2).Text)
          ymax = FNy(xn): ymin = ymax
    For x = xn To xk + dx / 2 Step dx
        y = FNy(x)
        If y > ymax Then ymax = y
        If y < ymin Then ymin = y
     Next x
     Picture1.Scale (xn, ymin)-(xk, ymax)
     For x = xn To xk + dx / 2 Step dx
        y = FNy(x)
        Picture1.PSet (x, y), vbRed
    Next x
End  Sub
```

4.1.4. Упражнения: графические объекты

Задача 1. Разработать формы, написать и отладить программы в соответствии с примерами 4.1- 4.4.

Задача 2. Разработать форму для исследования свойств элемента Shape в соответствии с вариантами заданий, приведенных в 4.1.5.

Задача 3. Построить график функции e^x на отрезке [-2,2].

Указание к задаче 2. Протабулировать функцию на заданном отрезке, найти максимальное и минимальное значение функции на отрезке табулирования, выполнить масштабирование формы в соответствии с вычисленными параметрами. Построить график функции, используя метод Pset.

4.1.5. Закрепление материала

1. Что такое графический объект? Назовите графические объекты.
2. Что такое графический метод? Назовите графические методы.
3. Какие графические элементы управления Вам известны?
4. Какие единицы измерения применяются в графических объектах?
5. Как пересчитать размеры экрана, заданные в твипах, в пиксели?
6. Расскажите назначение метода Scale. Приведите синтаксис метода.
7. Что такое объект Screen и каковы его основные свойства?
8. Для каких целей используется свойство объекта Screen ActiveControl?

9. Перечислите основные свойства элемента управления Line.

10. Перечислите основные свойства элемента управления Shape.

11. Какие способы используются в VB для задания цвета?

12. Приведите формат RGB – функции.

13. Какие графические методы используются для управления пикселем?

14. Приведите синтаксис метода PSet.

15. Напишите алгоритм построения графика функции с масштабированием.

Варианты заданий для самостоятельной работы.

Номер варианта	Свойство объекта Line	Свойство объекта Shape
1	BorderStyle	Shape
2	BorderWidth	FillColor (RGB)
3	BorderColor (RGB)	BorderStyle
4	BorderColor (VB –конст.)	BorderWidth
5	BorderStyle	FillStyle
6	BorderWidth	Shape
7	BorderStyle	FillColor (VB –конст.)
8	BorderColor (QBColor)	BorderStyle
9	BorderStyle	BorderWidth
10	BorderWidth	FillStile
11	BorderColor (RGB)	Shape
12	BorderStyle	FillColor (QBColor)
13	BorderWidth	BorderStyle
14	BorderStyle	BorderWidth
15	BorderColor (VB –конст.)	FillStile

4.2. Графические методы Visual Basic

Visual Basic поддерживает несколько методов, которые можно использовать для рисования изображений в форме или элементе управления PictureBox. Эти методы можно использовать вместе с объектом Printer, чтобы направить вывод на принтер. Если метод вызывается без указания объекта, то его вывод направляется в форму, которая находится в фокусе в текущий момент времени.

Для создания графических образов можно применять методы *Line, Circle, PSet, Point, Point Picture, Cls.*

Line – рисует линию или прямоугольник;

Circle - рисует окружность или овал;

PSet – размещает в указанном объекте точку;

Point – возвращает цвет определённой точки;

PointPicture – рисует в указанном объекте изображение, которое хранится в другом элементе управления;

Cls- очищает область вывода указанного объекта;

Print – выводит текст в указанный объект.

Часть из этих методов можно использовать для изображения графических элементов в объекте. Некоторые свойства графических объектов влияют на работу этих методов. Эти свойства приведены в таблице 4.1.

Таблица 4.1

Свойства графических объектов,

Свойство	***Назначение***
DrawMode	Определяет, как рисующий цвет взаимодействует с уже находящимся в объекте цветами.
DrawStyle	Определяет узор для рисования.
DrawWidth	Устанавливает толщину линий и границу объектов, например окружности.
FillColor	Определяет цвет, которым заполняется прямоугольник или круг.
FillStyle	Определяет узор, используемый для заполнения прямоугольника или круга.
FillColor	Определяет цвет рисуемых объектов графическими методами, если цвет не был указан при их вызове.
AutoRedraw	Определяет, будет ли результат работы графических методов автоматически обновляться в окне, если оно скрыто (только для Form и PictureBox)

влияющие на работу графических методов

4.2.1. Графический метод Line

Метод *Line* используется для рисования линий и прямоугольников.

Синтаксис метода:

```
[имя объекта.]Line [(x1,y1)] – (x2,y2), C, B[F]
```

Здесь также как и в объекте Line *x1, y1*- координаты левого верхнего угла, *x2, y2*- координаты правого нижнего угла прямоугольника; *C*- цвет; *B* и *BF*- флаг. Флаг *B* определяет, что изображается прямоугольник, а не линия; флаг *BF* определяет, что изображается закрашенный прямоугольник. Если первый параметр опущен, то в качестве начальной координаты используется текущая точка.

```
Dim x1 As Single, x2 As Single, y1 As Single, y2 As Single.
    x2= Me.ScaleWidth:  y2= Me.ScaleHeight
    Line (0,0)- (x2,y2)
```

рисуется линия из верхнего левого угла формы в правый нижний угол формы. Цвет принимается по умолчанию.

```
Current x= 1500
Current y= 750
Line – (x2,y2)  ‘рисуется линия из точки с координатами 1500, 750 _
                ‘в точку с координатами x2, y2 ( правый нижний угол)
Line (100, 850) – (500,1800), vbBlue, B  ‘синий прямоугольник
Line (200,150) – (300,1000), vbRed, BF  'красный закрашенный
```

```
                                                ‘ прямоугольник
Me.BorderStyle = 2:   Me.DrawWidth = 2
Line (50,100) – (1200,500), vbGreen, B  'зелёный прямоугольник,  _
                                        ’ штриховая линия толщиной 2 твипа
```

4.2.2. Метод Circle

Метод Circle используется для изображения эллипсов и окружностей.

Синтаксис метода:

[Объект.] Circle [Step] (x, y), R, C, [-] старт (start), [-] стоп (end), сжатие

Здесь *x, y* – координаты центра окружности; *R* – радиус, *C*- цвет; *старт* – начальный угол дуги, по умолчанию равен 0; *стоп* – конец дуги, по умолчанию равен 2π. Углы измеряются в радианах. Для указания угла в градусах необходимо использовать преобразование: $\alpha*\pi/180$, где α- угол в градусах. Если перед значением угла стоит знак минус, то конец дуги соединяется с центром окружности.

```
Const PI As Double = 3.14159
Circle (200,200), 150, vbRed   ‘- красная окружность’
Circle (300,100), 100, vbGreen, 30/180*PI, 120/180*PI, 1.5
‘дуга эллипса в интервале от 30 до 120 градусов’
Circle (500,300), 200, vbBlue, -30/180*PI, -120/180*PI  ‘сектор синего цвета’
```

Пример 4.5. Построение графиков функций.

Построить на одном графике две функции и оси координат. Установить контроль правильности и полноты ввода данных. Для построения графиков используется метод Line.

Решение.

```
Const Pi = 3.14159
Option Explicit
---------------------------------------------------------------------------------
Private Sub Form_Click()
        ‘ Объявление переменных
    Dim x As Single, y As Single, dx As Single
    Dim xn As Single, xk As Single
    Dim y1 As Single
        ‘ Контроль ввода данных
    If Text1.Text = "0" Or Text2.Text = "0" Or Text3.Text = "0" Then
        MsgBox "Нет данных"
        Exit Sub
    End If
        ‘ Присвоение значений переменным
    xn = Val(Text1.Text)
    xk = Val(Text2.Text)
    dx = Val(Text3.Text)
        ‘ Масштабирование формы и построение осей координат
    Scale (xn, 10)-(xk, -10)
    Line (xn, 0)-(xk, 0)
    Line (0, -5)-(0, 10)
```

```
        ‘ Построение графика функций
      y = Exp(xn):  PSet (xn, y), vbRed
      For x = xn To xk Step dx
          If y <= 10 Then
              y = Exp(x)
              Line -(x, y), vbRed
          End If
      Next x
      y1 = Cos(xn):   PSet (xn, y1), vbBlue
      For x = xn To xk Step dx
          y1 = Cos(x)
          Line -(x, y1), vbBlue
      Next x
  End Sub
---------------------------------------------------------------------------------
  Private Sub Text1_LostFocus()
      ‘ процедура контроля правильности ввода данных (только числовые)
      If Not IsNumeric(Text1.Text) Then
          Beep
          MsgBox "Неправильный ввод данных"
          Text1.SetFocus
      End If
  End Sub
```

Пример 4.6. Построение столбиковой диаграммы (рис. 4.4.)

Дано N параметров и их значения. Требуется построить столбиковую диаграмму. Диаграмма должна занимать часть формы и быть расположена в центре нее. Данные вводить с клавиатуры в режиме диалога.

Порядок работы.

1. *Ввод данных*. Для ввода данных в режиме диалога воспользуемся оператором InputBox. Чтобы данные можно было удобно анализировать и использовать запишем их в массив.

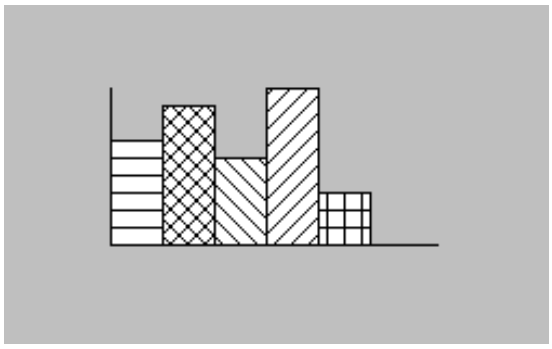


Рис.4.4. Столбиковая диаграмма

2. *Масштабирование.* Выберем ширину диаграммы произвольно, например *Bd=100*, а масштаб по оси X примем равным удвоенному значению ширины диаграммы *Mx= 2*Bd =200*. Высоту диаграммы примем равной максимальному значению параметров *Hd= max*. Масштаб по оси Y примем равным удвоенному значению максимального параметра – *My=2*Hd*. Тогда масштаб формы определится оператором:

```
Scale (0, My)-(Mx, 0)
```

3. *Построение осей координат*. Начальное смещение по оси X примем равным половине разности ширины формы и ширины диаграммы – *cx = (Mx-Bd)/2*, Начальное смещение по оси Y примем равным половине разности высоты формы и высоты диаграммы – *cy = (My – Hd)/2*.

4. *Определение ширины столбца*. Чтобы все столбики диаграммы были одинаковой ширины, определим шаг *dx= bd/(N+1)*, где *N* – число параметров. Число отрезков увеличено на единицу, чтобы ось X не сливалась с диаграммой и в то же время вся диаграмма располагалась в центре формы.

Для построения осей координат используем метод Line:

```
Line (cx, cy)-(cx, cy+Hd), vbBlue   'вертикальная ось
Line (cx, cy)-(cx + Bd, cy), vbBlue  'горизонтальная ось
```

5. *Построение диаграммы*. Для построения диаграммы применим цикл, так как значения параметров хранятся в массиве. Строить столбики диаграммы будем с помощью метода Line

```
Line (cx + (i - 1) * dx, cy)-(cx + i * dx, cy + A(i)), QBColor(i), BF
```

На форме не предусмотрено кнопок. Поэтому для запуска программы будем использовать событие Click формы. При запуске программы перед пользователем будет пустая форма. Чтобы пользователь не гадал, что надо делать дальше, поместим на форму надпись с текстом “Для начала работы щелкните мышью по форме”. Когда пользователь щелкнет мышью по форме, уберем надпись, сделав ее не видимой.

Программа содержит значительное число переменных, без которых можно было бы обойтись при построении простой диаграммы. Однако, чтобы программа была универсальной, все переменные необходимы. Теперь, при необходимости можно дополнительно разработать форму для настройки параметров диаграммы. Ниже приведен текст программы.

6.*Текст программы:*

```
Option Explicit
Dim A() As Single
---------------------------------------------------------------------------------
Private Sub Form_Click()
    Dim i As Integer, N As Integer, S As Single, Bd As Single, Hd  As Single
    Dim max  As Single, dx As Single, Mx  As Single, My As Single
    Dim cx  As Single, cy As Single
    Label1.Visible = False
    max = -1E+38
    N = Val(InputBox("Укажите число параметров"))
    ReDim A(N) As Single
    max = -1E+38
    Bd = 100: Mx = 2 * Bd
    For i = 1 To N
        A(i) = Val(InputBox("Введите значение " & Str$(i) & "параметра"))
        If A(i) > max Then max = A(i)
    Next i
    Hd = max: My = 2 * Hd
    Me.Scale (0, My)-(Mx, 0)
    cx = Bd / 2: cy = Hd / 2
    dx = Bd / (N + 1)
    'Рисуем оси координат'
    Line (cx, cy)-(cx, cy + Hd), vbBlue ' вертикальная ось
    Line (cx, cy)-(cx + Bd, cy), vbBlue 'горизонтальная ось
    For i = 1 To N
```

```
            Line (cx + (i - 1) * dx, cy)-(cx + i * dx, cy + A(i)), QBColor(i), BF
        Next I
    End Sub
```

Пример 4.7. Построение сектора.

Разбить круг на N секторов и обозначить их разным цветом.

```
    Dim i As Integer, k As Integer, ttime As Single
    Const Pi = 3.14
    ---------------------------------------------------------------------------
    Private Sub Form_Click()
    Dim n As Integer, dx As Single
        Me.Scale (0, 0)-(200, 200)
        n = Val(InputBox("укажите число секторов"))
        k = 0
        dx = 360 / n
        For i = 0 To 360 - dx Step dx
            If k > 15 Then k = 0: 'MsgBox "Недопустимое значение"
                Circle (100, 100), 50, QBColor(k), -i / 180 * Pi, -(i + dx) / 180 * Pi
            ttime = Timer
        tretri:
            If Timer - ttime < 0.5 Then GoTo tretri          ' пауза
            DoEvents
            k = k + 1
        Next i
    End Sub
```

Пример 4.8. Разбить круг на N секторов и закрасить их разным цветом.

Эта задача подобна задаче 4.3, но дополнительно надо закрашивать эти сектора. Чтобы закрасить сектора, достаточно прочертить радиусы в пределах сектора на достаточно малом расстоянии друг от друга.

Решение.

```
    Private Sub Form_Click()
        Dim n As Integer, dx As Single
        Me.Scale (0, 0)-(200, 200)
        n = Val(InputBox("укажите число секторов"))
          k = 0
          dx = 360 / n
        For i = 0 To 360 - dx Step dx
            a = i: a1 = i + dx
            If k > 15 Then k = 0: 'MsgBox "Недопустимое значение"
            For j = a To a1 - 0.5 Step 0.5
                Circle (100, 100), 50, QBColor(k),-j / 180 * Pi, -(j + 0.5) / 180 * Pi
            Next j
            k = k + 1
        Next i
    End Sub
```

Пример 4.9. Построение секторной диаграммы

Дано число параметров и их значения. Требуется построить секторную диаграмму. Ввод данных обеспечить в режиме диалога.

Решение.

1. *Ввод данных*. Для ввода данных в режиме диалога воспользуемся оператором InputBox. Чтобы данные можно было удобно анализировать и использовать запишем их в массив B(). При вводе данных подсчитаем и сумму значений параметров *S*.

2. Масштабирование. Определим угол, приходящийся на единицу значения параметра *da =360/S.*

3. Построение секторов. Построение секторов будем осуществлять с помощью цикла. Присвоим начальному углу *f1* значение нуль. Угол текущего сектора определим по формуле *f2= B(i)*da.*. Конечное значение угла сектора определим по формуле f3=f1+f2. Цвет секторов будем задавать с помощью функции QBColor(k). Чтобы программа работала корректно при любом числе секторов, необходимо контролировать параметр *k.* Значение параметра K не должно быть больше 15.

4. Текст программы процедуры.

```
Private Sub Form_Click()
    Dim n As Integer, dx As Single, s As Single, a As Single, a1 As Single
    Dim da As Single, fi1 As Single, fi2 As Single, fi3 As Single
    Dim i As Integer, j As Single, k As Integer, ttime As Single
    Me.Scale (0, 0)-(200, 200)
    n = Val(InputBox("укажите число секторов"))
    ReDim b(n)
    Cls
    s = 0
    For i = 1 To n
        b(i) = Val(InputBox("Введите значение " & Str$(i) & "параметра"))
        s = s + b(i)
    Next i
    da = 360 / s
      k = 0: fi1 = 0
     For i = 1 To n
        fi2 = b(i) * da
        fi3 = fi1 + fi2
        If k > 15 Then k = 0
        For j = fi1 To fi3 - 1 Step 1
            Circle (100, 100), 50, QBColor(k), -j / 180 * Pi, -(j + 1) / 180 * Pi
        Next j
        k = k + 1
        fi1 = fi3
     Next i
End Sub
```

4.2.3. Метод Print

Метод Print “рисует” текст на объекте, как и другие графические методы. Его можно использовать в сочетании с другими методами для создания диаграмм или для анкетирования существующих растровых изображений. Синтаксис метода

```
Print “текст”
```

Работа метода Print зависит от значений некоторых свойств объекта, в который выводится текст:

CurrentX – горизонтальная координата точки вывода;

CurrentY – вертикальная координата точки вывода;

Font - тип и размер шрифта;

ForeColor - цвет текста;

FontTransparent – определяет, будет ли просвечиваться фон формы сквозь незаполненные части текста.

Например:

```
CurrentX=100
CurrentY=100
‘Тип шрифта, начертание, высота и некоторые атрибуты
‘ выбирается в режиме диалога, параметры объекта Font
‘ могут устанавливаться программным путем
Me.Font.Italic = True   ‘курсив
Me.Font.Size = 12
Me.ForeColor=vbRed
Print “Ввод текста красным цветом”
```

4.2..4. Упражнения: Графические методы

1. Разработайте формы, напишите и отладьте программы в соответствии с примерами 4.5 – 4.9.

2. Разработайте форму для исследования свойств метода Line с учетом влияния свойства DrawWidth формы.

3. Разработайте форму для исследования свойств метода Circle с учетом влияния свойств формы FillColor формы.

4.2.5. Закрепление материала

1. Назовите основные графические методы.
2. Приведите синтаксис метода Line. Дайте пояснения.
3. Приведите синтаксис метода Circle. Поясните назначение опций.
4. Напишите алгоритм построения графика функции.
5. Напишите алгоритм построения столбиковой диаграммы.
6. Напишите алгоритм построения круговой диаграммы.
7. Какие свойства графических объектов влияют на работу графических методов?

8. Приведите синтаксис метода Print. Какие свойства графического объекта используются совместно с методом Print?

Задания для самостоятельной работы

1. Разработайте форму для построения графиков функций с масштабированием. На графике отобразите оси координат и выведите наименование осей и разметку по осям X и Y.
2. Разработайте форму для построения столбиковой диаграммы (рис.4.4).
3. Разработайте форму для построения столбиковой диаграммы. На диаграмму выводить разметку по осям X и Y, наименование диаграммы и название осей.
4. Разработайте форму для построения круговой диаграммы.
5. Разработайте форму для построения круговой диаграммы с выделением указанного сектора.
6. Разработать форму для построения круговой диаграммы с выделением указанного сектора и выводом размеров секторов в процентах.

4.3. Объекты PictureBox, Image

4.3.1. Понятие векторной и растровой графики

Растровые изображения представляют собой рисунки, состоящие из пикселей - точек на экране компьютера, формирующих картинку.

В векторной графике, фигуры представляются последовательностью точек, соединенных отрезками линий и кривыми.

Эти особенности определяют достоинства и недостатки каждого из этих методов представления рисунков. Растровые изображения позволяют получать рисунки высокого качества при постоянном разрешении. При изменении разрешения экрана или изменении размеров рисунка качество рисунка ухудшается. При увеличении размера растрового изображения появляется ступенчатость за счет увеличения размеров точки на экране, при уменьшении размеров рисунка качество рисунка также ухудшается за счет исключения отдельных пикселей. Растровые изображения требуют для своего хранения большего объема памяти, так как для каждого пикселя необходимо хранить значение цвета.

Графические элементы в векторном файле называются объектами. Каждый объект является самостоятельной единицей с такими свойствами, как цвет, форма, толщина линии (абрис), координаты начальной и конечной точек отрезка и размер. Поэтому размеры векторных графических файлов существенно меньше, чем размеры растровых файлов. Размеры векторных изображений легко изменяются без потери качества.

Visual Basic 6.0 имеет возможность работать с различными типами графических файлов (табл. 4.2).

Таблица 4.2

Типы файлов, загружаемых в объект PictureBox

Расширение имени файла	***Тип файла***
.bmp	побитовый (то есть несжатый) растровый файл[6]
.dib	устройство-независимый растровый файл (Device Independent Bitmap)
.ICO, .CUR	пиктограммы, значки объектов и формы курсора, соответственно
.wmf	метафайл Windows – векторное изображение
.emf	расширенный метафайл
.gif	Graphics Interchange Format - растровый формат широко используемый на Web - страницах в Internet[7]
.jpg, .peg	Joint Photographic Experts Group – растровый формат, для уменьшения размера файла используется метод сжатия с потерей качества, широко используется на Web-страницах в Internet[8]

4.3.2. Окно с рисунком (PictureBox)

Элемент управления PictureBox предназначен для отображения рисунков и других графических объектов. Этот элемент управления является элементом-контейнером, поэтому его можно использовать для объединения других элементов.

События объекта PictureBox обычно не обрабатываются.

Основные свойства объекта: *Align, Autosize, Picture*.

Align - определяет положение PictureBox в форме: 0 - None, 1 - Top, 2 - Bottom, 3 - Left, 4 - Right. В зависимости от значения этого свойства объект будет закрепляться у одного из краев формы или будет сохранять положение, заданное разработчиком (None). Если элемент управления закрепляется у одного из краев формы, то его размер (ширина и высота) всегда будут устанавливаться в соответствии с размерами формы.

Autosize - определяет, будут ли размеры элемента управления автоматически изменяться для отображения рисунков различного размера. Если свойство *Autosize* установлено в *False*, то в объект помещается весь рисунок, но в зависимости от его размера будет видна либо часть рисунка, либо будут оставать-

[6] В стандарте Window для файлов .bmp предусмотрена возможность их сжатия, но только в рамках алгоритма RunLength, когда повторяющиеся байты (или биты) заменяются числом повторений и самим повторяющимся байтом.

[7] В файлах формата .gif используется сжатие по алгоритму Лемпела-Зива (LZW) без потери информации, эти файлы допускают использование простейших вариантов анимации в виде нескольких сменяющих друг друга картинок

[8] В файлах формата .jpg для сжатия используется быстрое преобразование Фурье для трех составляющих картинки, яркостной и двух цветоразностных составляющих, как в телевидении. При этом отбрасываются высшие составляющие разложений Фурье, что обеспечивает сжатие файла в 10 – 20 раз.

(Примечания по п. 6, 7 и 8 Колосова С. В.)

ся пустые места на экране. Если свойство *Autosize* установлено в *True*, то размеры элемента управления автоматически изменяются до размеров рисунка.

Picture - позволяет загружать графические объекты и сохранять их, содержит отображаемый графический объект.

Для загрузки графического объекта щелкните мышью по кнопке *троеточие* в поле свойства *Picture* – появится диалоговое окно *Load Picture*, которое позволяет осуществить поиск и загрузку нужного файла с диска. Напимер: C:\Windows\облака.bmp

В процессе работы программы загрузку изображений можно осуществлять с помощью функции ***LoadPicture***:

```
Объект.Picture = LoadPicture("спецификация_файла")
```

Загружаемые во время разработки файлы хранятся вместе с формой, что увеличивает размер программы и время ее загрузки.

Сохранить изображение можно функцией ***SavePicture***. Файлы сохраняются в формате .BMP:

```
SavePicture(Picture1.Picture, “BUILD.BMP”)
```

Методы объекта PictureBox позволяют нарисовать точку, линию и окружность, а также вывести текст (метод Print).

Выгрузка рисунков

Рисунки, помещенные в PictureBox, не удаляются, а выгружаются. Для выгрузки рисунка в процессе разработки программы необходимо выделить в строке свойства *Picture* слово *bitmap* и нажать клавишу ***Del***.

Выгрузка рисунка в процессе работы осуществляется этой же функцией без указания имени файла:

```
Объект.Picture = LoadPicture()
```

Пример 4.10. Вывод текста и рисунка

```
Private Sub Form_Click()
    Cls
    Dim X As Single, Y As Single
    picPicture1.CurrentX = 100
    picPicture1.CurrentY = 100
    picPicture1.Print "Ввод текста красным цветом"
    'Тип шрифта, начертание, высота и некоторые атрибуты
    ' можно выбирать и  в режиме диалога
    picPicture1.Font.Italic = False
    picPicture1.Font.Size = 12
    picPicture1.ForeColor = vbRed
    picPicture1.CurrentX = 400
    picPicture1.Print "Ввод текста красным цветом"
    picPicture1.CurrentX = 700
    picPicture1.CurrentY = 600
    picPicture1.Font.Size = 8
    picPicture1.ForeColor = vbBlack
    picPicture1.Print "Ввод текста черным цветом, при X=700 твипов "
    ' Управление рисунком
```

```
        X = 2000: Y = 1000
        picPicture1.DrawWidth = 15
        picPicture1.PSet (X, Y), vbRed
        picPicture1.CurrentX = 900: picPicture1.CurrentY = 900
        picPicture1.Print "Рисунок"
        picPicture1.DrawWidth = 2
        picPicture1.Line (0, 0)-(3000, 1500), vbBlue
        picPicture1.Circle (2000, 1000), 500, vbGreen
        picPicture1.CurrentX = 1100: picPicture1.CurrentY = 1200
        picPicture1.Print "Рисунок1"
    End Sub
```

Элемент PictureBox позволяет преобразовывать одни форматы изображений в другие. Например, пиктограмму (.ICO) можно преобразовать в растровый рисунок (.bmp).). Для этого надо загрузить пиктограмму, а затем сохранить ее с расширением .bmp. (Обратное преобразование не возможно

4.3.3. Элемент управления Image

Элемент управления *Image* также создан для отображения рисунков. Но в отличие от *PictureBox*, он не является элементом контейнером. Элемент управления *Image* не позволяет ни рисовать, ни группировать объекты. Однако Image использует меньше ресурсов и перерисовывает быстрее, чем PictureBox. Поэтому для отображения рисунков Image является более предпочтительным.

Основными свойствами элемента Image является свойство *Stretch* и *Picture*.

Свойство ***Picture*** аналогично соответствующему свойству элемента PictureBox.

Свойство ***Stretch*** позволяет устанавливать соответствие между размерами элемента управления и размером рисунка. Если свойству Stretch присвоено значение *True*, то размеры рисунка изменяются до размеров элемента управления Image, в противном случае элемент управления изменяется до размеров рисунка.

4.3.4. Загрузка изображений в форму

Изображения можно помещать непосредственно в форму. При этом изображение помещается от верхнего левого угла формы, никак не масштабируется и не растягивается.

Изображения не просвечиваются сквозь элементы управления за исключением Label и Shape. Чтобы элементы управления Label и Shape были прозрачны, значение их свойств *BackStyle* должно быть равно *Transparente*.

Основным преимуществом помещения изображения непосредственно в форму является то, что при этом используется меньше ресурсов системы, чем

при предварительном размещении его в элементах управления PictureBox или Image.

Основные недостатки размещения изображений в форме:

- нельзя скрыть изображение, его можно только загрузить или выгрузить;
- нельзя управлять расположением изображения в форме;
- в форму одновременно можно поместить только одно изображение;
- размер изображения нельзя изменить. Оно помещается в форму в своем оригинальном размере;
- время на перерисовку формы требуется больше.

Метод Picture формы MDI позволяет поместить фоновое изображение на задний план.

Пример 4.11. Загрузка и выгрузка рисунков.

Установим на форму элементы управления Image и PictureBox. Свойство AutoSize объекта PictureBox и свойство Stretch объекта Image установим в True.

Загрузим в форму и элементы управления Image и PictureBox рисунки. Для этого необходимо выделить объект, щелкнуть по строке свойства Picture и выбрать в диалоговом окне нужный файл, например: C:\Windows\Лес.BMP, C:\Windows\Волны.BMP, C:\Windows\Облака.BMP. Напишем текст программы.

```
Private Sub Form_Click()
    Dim i As Double
    ‘выгрузка рисунков
    Form1.Picture = LoadPicture()
    Image1.Picture = LoadPicture()
    picPicture1.Picture = LoadPicture()
    For i = 1 To 50000: DoEvents: Next I  ‘ пауза
    ‘ Загрузка рисунка
    Form1.Picture = LoadPicture("C:\Windows\Наждак.BMP")
    Image1.Picture = LoadPicture("C:\Windows\Облака.BMP")
    picPicture1.Picture = LoadPicture("C:\Windows\Лес.BMP")
End Sub
```

4.3.5. Управление графическими объектами

Visual Basic имеет ряд свойств позволяющих управлять перерисовкой графических объектов.

Свойство AutoRedraw

Свойство *AutoRedraw* графических объектов служит для их перерисовки. Если значение свойства равно False, то VB сохраняет копию объекта в памяти и перерисовку графического объекта надо будет производить самостоятельно. Для ускорения выполнения программы при использовании графики свойство AutoRedraw следует установить в True.

Существуют некоторые различия, как свойство AutoRedraw, будучи установлено в True, работает с формами и графическими окнами.

При уменьшении форм VB сохраняет копию всего экрана. Для увеличения формы этого не требуется, так как графическая информация не теряется. Опция True для сохранения всего экрана и быстрого выполнения требует большого объема памяти. Но если графический объект некорректно подогнан к форме, надо использовать эту опцию.

Для графических окон VB сохраняет изображение только по наибольшему размеру текущего окна, ничего нового не появляется, даже если окно было раньше увеличено. Поэтому рисование в графических окнах требует меньше памяти, чем рисование в форме, даже если графическое окно заполняет всю форму.

В процедуре Paint нельзя размещать никаких инструкций по перемещению или изменению размеров объектов.

Свойство AutoRedraw определяет также, будут ли результат работы графических методов автоматически обновляться в окне, если оно скрыто (только для Form и PictureBox). При установке в True результаты работы графических методов будут автоматически обновляться в окне, даже если оно скрыто.

Событие ***Paint*** также служит для перерисовки объекта. События *Paint,* по сравнению с методом *AutuRedraw,* требует меньше памяти, но более медленно перерисовывает изображение.

Метод Refresh

Этот метод имеется у форм и элементов управления. Он приводит к немедленному обновлению формы или элемента управления и позволяет наблюдать подготовку изображения даже если свойство AutoRedraw равняется True. При использовании данного метода VB будет вызывать процедуру обработки событий Paint, созданную для данного объекта. Наиболее часто данный метод используется в процедуре ***Form_Resize,*** чтобы вывести повторно на экран любые графические изображения, подготовленные в Paint. Кроме того, поскольку VB управляет обновлением экрана в течение времени простоя, иногда требуется брать управление данными операциями на себя. В любом месте при использовании метода Refresh происходит немедленная перерисовка изображения объекта и генерация событий Paint, если данный объект имеет его.

Синтаксис использования метода:

```
Private Sub Form_Resize ( )
    Form1.Refresh
End Sub
```

При использовании метода Refresh изображение выводится на экран за несколько этапов. Однако каждый раз при этом VB рисует изображение по точкам. Такой способ занимает много времени.

Для перерисовки объекта необходимо периодически использовать метод Refresh.

Метод Refresh обычно используется в процедуре Form_Resize для отображения заново на экране монитора любой графики, которая обрабатывается в процедуре Paint.

Свойство ClipControls

Свойство ClipControls влияет на скорость выполнения программы.

Если свойство ClipControls установлено в True (по умолчанию), а AutoRedraw в False, то VB перерисовывает весь объект. Если свойство ClipControls установлено в False, тогда Visual Basic перерисовывает только заново открываемую область.

При установке ClipControls в True вокруг неграфических элементов управления создается усеченная область. Это означает, что VB создает контур формы и элементов управления и хранит их в памяти. Установка этого свойства в False может уменьшить время, необходимое для рисования и перерисовки объекта, вследствие того, что усекаемая область создается в памяти. Чем сложнее объект, тем больше требуется времени на его обработку. Усеченные области исключают такие элементы управления, как Image, Label, Line и Shape.

Обобщенные сведения о влиянии свойства ClipControls и метода AutoRedraw на работу программы приведены в табл. 4.3.

Таблица 4.3

Влияние свойства ClipControls и метода AutoRedraw на работу программы

ClipControl	*Autoredraw*	*Результат*
True (по умолчанию)	False	VB перерисовывает весь объект
False	False	VB перерисовывает заново только вновь открываемую форму
False	True	Увеличивает скорость работы программы

Для увеличения скорости работы программы свойство AutoRedraw следует установить в True, а ClipControls в False.

Метод PaintPicture

Метод PaintPicture перерисовывает изображение, находящееся в одном (исходном) объекте в другой. Задавая соответствующие значения аргументов *height* и *width* исходного и результирующего объектов, можно увеличить или уменьшить размер исходного изображения. Синтаксис данного метода:

ОбъектНазначения.PaintPicture ИсходныйОбъект.Paint X, Y, B, H, X1, Y1, [B1,H1]

Здесь *ОбъектНазначения* – имя объекта, куда будет помещаться изображение, *ИсходныйОбъект* – объект, откуда берется часть изображения; X, Y –

координаты верхнего левого угла рисунка, В, Н – ширина и высота результирующего рисунка; X1, Y1 – координаты рисунка в исходном объекте; В1, Н1 – размеры исходного рисунка.

```
Picture1.PaintPicture Form1.Picture, 0, 0, 1500, 1000, 0, 0, 7000, 5000
```

В данном примере копируется рисунок из формы в элемент управления PictureBox с уменьшением.

Если размеры исходного объекта (В1, Н1) не указаны, то копируется все изображение исходного рисунка.

Метод PaintPicture может применяться в следующих случаях:

- для выполнения масштабирования рисунка, чтобы можно было подробнее рассмотреть некоторую область изображения, например при предварительном просмотре;
- для копирования или затирания определенной области изображения;
- для перемещения содержимого элемента управления PictureBox в объект Printer, например, если нужно поместить рисунок в отчет.

Метод Point

Метод Point возвращает RGB – цвет определенной точки,

синтаксис: Point (x, y)

Пример 4.12. Определить принадлежность точки прямоугольнику.

```
‘ Рисуем красный прямоугольник
 Private Sub Form_Load ( )
     Me. AutoRedraw = True
     Line (0,0) – (1500,1500), VbRed, BF
 End Sub
---------------------------------------------------------------------------------------------
   Private Sub Form_MouseDown (Button As Integer,  _
       Shift As Integer, X As Single, Y As Single)
       X = Val(InputBox("Введите значение X"))
       Y = Val(InputBox("Введите значение Y"))
       If Point (x,y) = VbRed Then
           MsgBox “Точка (“ & x &” , “& y &”) находится в прямоугольнике”
       Else
           MsgBox “ Точка находится вне прямоугольника”
       End If
   End Sub.
```

Visual Basic позволяет сохранить изображения, нарисованные в форме или графическом окне, например:

```
SavePicture    ИмяОбъекта.Image.ИмяФайла
```

Операционная система использует свойство *Image* для определения того, как нарисовано изображение: в форме или графическом окне. Если работа в графическом окне прекращается, то VB использует для сохранения текущую форму:

```
SavePicture    Image.Имя_файла
```

Если изображение загружено первоначально из файла, определяющего свойство Picture, то VB сохраняет изображение в том же формате, что и оригинальный файл, иначе VB сохраняет файлы в формате побитового изображения - bitmap – файлы (.bmp).

Функция DoEvents

При организации вычислительных процедур с циклами Visual Basic по умолчанию не реагирует на события. Негативные последствия этого наглядно проявляются при работе с движущимися графическими объектами: объект не меняет положение или не изменяется предусмотренный цвет и т. д. Чтобы избежать таких последствий необходим механизм отслеживания состояния операционной системы и реагирования на различного рода события.

Задачи такого рода выполняет функция ***DoEvents***. В каком бы месте программы ни стоял данный оператор, он сигнализирует Visual Basic о том, что управление передано операционной системе для обработки всех событий.

*Функцию **DoEvents** нельзя использовать в процедуре обработки событий, которая вызывается несколько раз. Иначе можно легко организовать в программе бесконечный цикл.*

4.3.6. Упражнения: графические объекты

1. Поместите в форму элемент управления PictureBox. Исследуйте графические методы Line, Circle, Point и Print и способы управления представлением текста и рисунков (ClipControls и AutoRedraw) (табл. 4.3, примеры 4.10).

2. Поместите в форму элемент управления PictureBox. Загрузите в него рисунок. Исследуйте влияние свойств AutoSize и Align на положение и размеры объекта. Результаты исследования представьте в виде таблицы.

3. Поместите в форму элементы управления Image. Загрузите в него рисунок. Исследуйте влияние свойства Stretch элемента управления на взаимодействие объекта с рисунком.

4. Поместите на форму элементы управления PictureBox и Image. Поместить в объект PictureBox рисунок и скопируйте его с уменьшением в элемент управления Image.

5. Исследуйте метод Point (пример 4.12).

4.3.7. Закрепление материала

1. Для чего предназначен графический объект PictureBox?

2. Какими свойствами обладает объект PictureBox?

3. Как влияет свойство AutoSize на размер рисунка, помещаемого в объект PictureBox?

4. Какие типы файлов можно загружать в VB?
5. Чем отличается растровое изображение от векторного?
6. Для чего предназначен элемент управления Image?
7. Поясните основные свойства элемента управления Image?
8. Какую функцию выполняет свойство AutoRedraw?
9. Для чего предназначен и как используется метод Refresh?
10. Как влияет свойство ClipControl на скорость выполнения программы?
11. Поясните назначение метода PaintPicture.
12. Поясните назначение метода Point.

4.4. Анимация

4.4.1. Элемент управления Animation

Элемент управления Animation позволяет легко включать в программу различные анимационные эффекты. Он используется для воспроизведения не озвученных видео роликов в формате AVI. Видео ролик в данном формате фактически представляет собой набор растровых изображений (кадров). Кадры выводятся последовательно через равные промежутки времени, за счет чего и создается анимационный эффект, почти как в обычных мультфильмах. Обычно элемент управления Animation применяется там, где нужно "создать видимость работы", например в процедуре копирования файлов в Windows.

Другим способом создания анимационных эффектов является использование таймера. Через определенные промежутки времени нужно просто изменять положение на экране элемента управления Image или другого рисованного объекта.

Настройка элемента управления Animation

Чтобы настроить элемент управления, его нужно сначала поместить в Форму. При этом создается контейнер для воспроизведения последовательности анимационных изображений. Чтобы теперь просмотреть фильм, нужно открыть конкретный файл и запустить его на воспроизведение. Для этой цели в форму надо поместить кнопку и в процедуру Click этой кнопки написать текст программы, который открывает и проигрывает видеофильм.

Элемент управления Animation три метода: *Open, Play, Stop.*

Метод ***Open*** используется для открытия файла**.**

```
Animation1.Open "Спецификация файла"
```

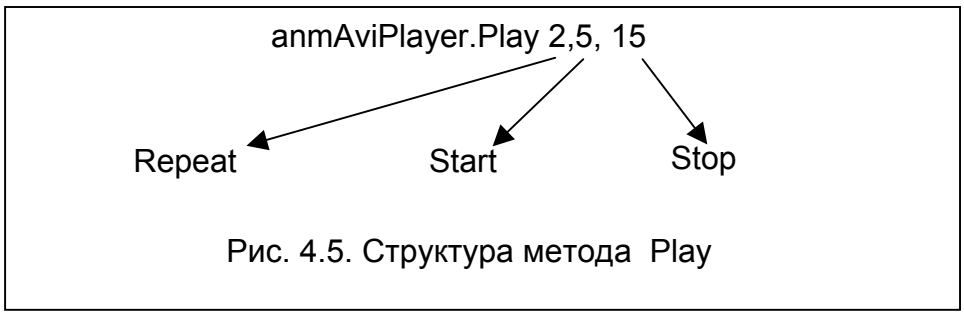


Рис. 4.5. Структура метода Play

Метод ***Play*** (или свойство ***AutoPlay*** элемента управления Animation) обеспечивает управление демонстрацией ролика. Этот метод имеет три параметра:

Repeat – определяет число повторных воспроизведений;

Start – определяет номер кадра, с которого должно начаться воспроизведение;

Stop – определяет номер кадра, на котором должно заканчиваться воспроизведение.

```
Animation1.Play   Repeat, Start, Stop
```

Метод ***Stop*** используется для остановки видеофильма.

```
Animation1.Open "C:\Progra~1\Micros~1\Common\Graphics\Video\ FileNuke.AVI"
Animation1.Play  2,5, 15
```

В приведенном примере, и на рис. 4.5., дважды воспроизводятся кадры с №5 по №15.

4.4.2. Создание анимации пользователем

Каждый пользователь при небольшом навыке может создать собственные анимационные эффекты. При этом можно использовать различные способы и возможности языка программирования:

- пересчет координат объекта и использование свойств *Top* и *Left* объекта или операторов *CurrentX*, *CurrentY* для переопределения координат объекта;
- использование метода *Move*;
- использование буфера обмена;
- прямое присвоение значений свойств одного графического объекта другому.

Общий алгоритм работы программы при создании анимации следующий:

- воспроизвести изображение в начальной точке;
- сделать паузу на некоторое время, достаточное для фиксации изображения (десятые доли секунды). Длительность паузы, когда изображение на экране неподвижно определяет и скорость перемещения объекта;
- стереть изображение;
- пересчитать координаты объекта и воспроизвести его в новой позиции.

Самая большая проблема в анимации для VB состоит в необходимости перерисовки объекта каждый раз, когда он передвигается. Этот процесс занимает много времени. Рисование цветом фона тоже не работает, поскольку затирает все, что было до этого на экране монитора.

Ключом к решению данной проблемы является режим DrawMode, определяющий как рисующий цвет взаимодействует с уже находящимися в объекте цветами.

Режим DrawMode

Существует 15 возможных установок DrawMode. Во всех случаях VB сравнивает значение цвета пикселя на экране со значением цвета пикселя объекта, который рисуется на экране.

Если DrawMode = 7, то результатом его работы будет оператор ***Xor***. DrawMode = 6 – соответствует оператору ***Not***, а DrawMode = 4 определяет работу VB с оператором Not над значениями цветов переднего плана и использует значения этих цветов для рисования.

Если оператор Xor применяется дважды, то происходит восстановление первоначального цвета.

Повторное воспроизведение графического объекта при установленном режим DrawMode = 7 для формы или окна позволяет стереть построенное ранее изображение без потери другой информации.

Для перемещения изображения можно использовать метод ***Move***.
Например:

```
Command1.Move x, y
```

Организация пауз

Для наблюдения процесса движения, особенно на быстродействующих компьютерах, необходимо позаботиться о замедлении движения объекта. Это можно сделать двумя способами: уменьшением шага переноса и организацией пауз.

Для организации пауз можно использовать различные приемы: использование пустых циклов, использование системных часов, использование таймера.

Использование пустых циклов:

```
For i=1 To 1000: DoEvents: Next i
```

Использование системных часов:

```
T=Time() или T=Timere()
While   Time() – T < 2: Wend
```

Использование таймера

Visual Basic позволяет устанавливать до 36 таймеров. Основные свойство таймера: *Intrval* и *Enabled*. Основное событие – *Timer*.

Interval - позволяет установить интервал выдачи сигнала от 0 до 10000, что соответствует примерно одной минуте. Для получения больших интервалов времени необходимо применять счетчики.

Enabled – позволяет запускать и останавливать таймер. Если Enabled равно True, то таймер запускается. При установке значения свойства Enabled в False таймер останавливается.

```
Private Sub Timer1_Timer()
    Static TimerTimes As Integer
    TimerTimes = TimerTimes + 1
    If TimerTimes = 2 Then
      TT = TimerTimes
        TimerTimes = 0
      Else
    Exit Sub
  End If
End Sub
```

Событие ***Timer*** используется для размещения текста программы, которая должна выполняться через заданный интервал времени.

В приведенном примере таймер выдает сигнал через две минуты.

Примеры анимации

Простая анимация

Пример 4.13. Движение кнопки по случайному закону - “Броуновское движение”.

```
Dim TT As Integer
Option Explicit
-------------------------------------------------------
Private Sub Form_Click()
  Dim x As Single, y As Single, a As Single
  Dim xmove As Single, ymove As Single
  Dim i As Integer, j As Integer, ttek As Integer
  Dim TimerTimes As Integer
  Randomize (1)
  Me.ScaleMode = 3 ' шкала в пикселях
  Me.WindowState = 2
     x = Me.ScaleWidth / 2
     y = Me.ScaleHeight / 2
     PSet (x, y)
     For i = 0 To 50
        If j > 15 Then j = 0
        xmove = 100 * Rnd
        ymove = 50 * Rnd
        If Rnd < 0.5 Then
           x = x + xmove
        Else
           x = x - xmove
        End If
```

```
        If Rnd < 0.5 Then
              y = y + ymove
         Else
             y = y - ymove
         End If
         If x < 0 Or x > ScaleWidth Or y < 0 Or y > ScaleHeight Then
               ' ничего не делать
         Else
              Command1.Move x, y
          Line -(x, y), QBColor(j)
          End If
       ttek = 0
       While ttek < 20000
          Timer1_Timer   'вызов процедуры таймера
          ttek = ttek + TT
          DoEvents
       Wend
       TimerTimes = 0
       j = j + 1
   Next i
End Sub
```

Пример 4.14. Вспышка звезды

```
Dim TT As Integer
Option Explicit
-------------------------------------------------------
Private Sub Form_Click()
  Dim x As Single, y As Single, a As Single
  Dim xmove As Single, ymove As Single
  Dim i As Integer, j As Integer, ttek As Integer
  Dim TimerTimes As Integer
  Randomize (1)
  Me.ScaleMode = 3 ' шкала в пикселях
  Me.WindowState = 2
     x = Me.ScaleWidth / 2
     y = Me.ScaleHeight / 2
     PSet (x, y)
     For i = 0 To 50
        If j > 15 Then j = 0
        xmove = 100 * Rnd
        ymove = 50 * Rnd
        If Rnd < 0.5 Then
           x = x + xmove
        Else
          x = x - xmove
       End If
       If Rnd < 0.5 Then
          y = y + ymove
       Else
          y = y - ymove
       End If
```

```
        If x < 0 Or x > ScaleWidth Or y < 0 Or y > ScaleHeight Then
          ' ничего не делать
        Else
          Command1.Move x, y
          Line -(x, y), QBColor(j)
        End If
        ttek = 0
        While ttek < 20000
           Timer1_Timer
           ttek = ttek + TT
           'Print ttek
           DoEvents
        Wend
        TimerTimes = 0
        j = j + 1
    Next i
End Sub
```

Анимация посредством переноса изображений через буфер обмена

Независимо от способа создания рисунка в VB предусмотрена возможность перенести его в другой элемент управления или другие приложения Windows. Для этого используются методы *SetData, GetData( ), GetForm, Clear*.

Метод *SetData* – перемещает данные в объект.

Синтаксис:

Объект. SetData [данные],[формат]

Здесь:

Объект - буфер обмена, его идентификатор ***ClipBoard****;*

Данные – указывают, откуда переносятся данные (ImageBox, PictureBox и др.);

Формат – задает формат исходных данных.

Допустимые значения опции Формат приведены в табл. 4.4.

Таблица 4.4.

Опции функции Формат

Константа	***Описание***
VbCFText	текст
VbCFBitmap	растровое изображение (*.bmp)
VbCFMetaFile	метафайл(*.Wmf)
VbCFFMetaFile	расширенный метафайл(*.EMF)
VbCFDIB	независимое от устройства растровое изображение(*.dib)
VbCFPalette	цветовая палитра
VbCFRtf	файл в формате*.rtf

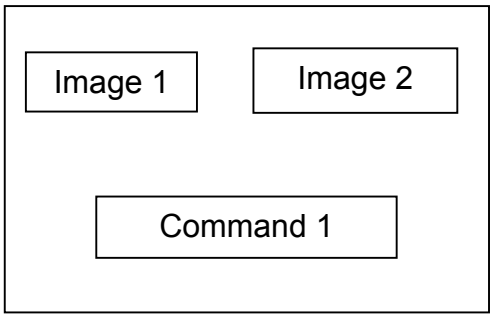


Рис. 4.6. Форма для анимационного перемещения объекта

Метод ***GetData()*** – восстанавливает данные из объекта.

Синтаксис: Объект.GetData [,формат]

Метод ***GetFormat*** – возвращает логическое значение, подтверждающее, хранятся ли в объекте данные указанного формата.

Синтаксис: Объект.GetFormat[,формат]

Пример 4.15. Анимационное перемещение изображения, содержащегося в элементе управления Image1.Picture в элемент управления Image2.Picture (рис.4.6.):

```
Private Sub Command1_Click()
    Dim i As Integer, j As Long
    Clipboard.Clear
    Clipboard.SetData Image1.Picture, vbCFBitmap
    For i = 1 To 300
       Image2.Left = 1000 + i * 20
       Image2.Top = 480 + i * 20
       For j = 1 To 100000: Next j
       Image2.Picture = Clipboard.GetData()
    Next i
End Sub
```

Анимация посредством присвоения значения свойства одного графического объекта другому

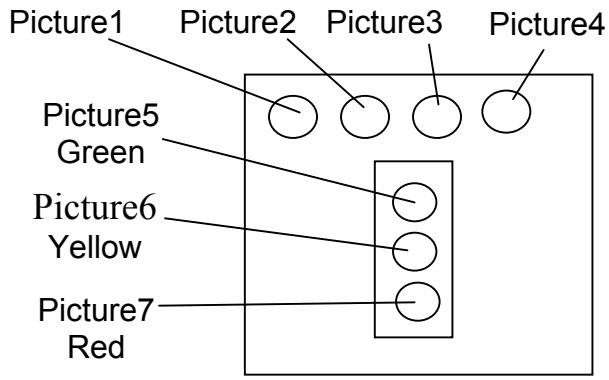


Рис. 4.7. Форма “Светофор”

Пример 4.16. “Светофор”

Другой способ анимации состоит в присваивании значения свойства Picture одного графического элемента управления другому.

На форме (рис. 4.7.) Расположен. “светофор”. На форму помещены семь элементов управления PictureBox или Image. Первый, второй и третий элементы содержат круги зеленого, желтого и красного цвета, соответственно (круги соответствующего цвета можно нарисовать в программе Paint и сохранить в отдельных файлах). Четвертый элемент пустой. Через установленные интервалы времени рисунок из элементов Picture1, Picture2, Picture3, Picture4 перемещается в элементы Picture5, Picture6, Picture7. Элементы 1- 4 можно сделать невидимыми.

```
Private Sub Form_Click()
    Dim i As Integer, ltime As Single
    For i = 1 To 10
```

```
            ltime = Timer()
             ' пауза
          While Timer() - ltime < 1: Wend
            Picture6.Picture = Picture4.Picture
            Picture7.Picture = Picture4.Picture
            Picture5.Picture = Picture1.Picture
            ltime = Timer()
            'пауза
          While Timer() - ltime < 0.5: Wend
             Picture5.Picture = Picture4.Picture
             Picture6.Picture = Picture2.Picture
          ltime = Timer()
             'пауза
            While Timer - ltime < 1: Wend
             Picture6.Picture = Picture4.Picture
             Picture7.Picture = Picture3.Picture
          Next i
    End Sub
```

Пример 4.17.. Радар

```
    Private Sub Form_Click()
        Const Pi = 3.14159
        Const Alfa = 50
        Dim x As Integer, y As Integer, r As Integer
        Dim xNach As Single, xKon As Single, nCount As Integer
        Dim i As Integer, j As Integer, A As Integer
            ‘Рисуем окружность в центре формы
        x = Me.ScaleWidth / 2
        y = Me.ScaleHeight / 2
        r = x / 2
        Me.FillStyle = vbSolid                  'без окантовки
        Me.DrawMode = vbCopyPen                 ' взаимодействие с цветом
        Me.FillStyle = vbFSTransparent          'прозрачный
        Me.Circle (x, y), r
           'рисуем сектор радара
        Me.DrawMode = vbXorPen ' исключающее или
        Me.FillColor = vbGreen 'QBColor(5)
        Me.FillStyle = vbSolid 'без окантовки
    nach:
        For i = 0 To 360
          xNach = i
          xKon = i + Alfa
          If xKon > 360 Then xKon = xKon - 360
          Me.Caption = "Дуга" & Alfa & "градусов с началом в" & i
          Me.Circle (x, y), r, , -xNach * Pi / 180, -xKon * Pi / 180
          For j = 1 To 15000: A = A: Next j
           DoEvents
          Me.Circle (x, y), r, , -xNach * Pi / 180, -xKon * Pi / 180
        Next i
        GoTo nach
        Me.Caption = "Конец программы"
```

```
End Sub
```

4.4.3. Создание форм, независимых от используемого разрешения экрана

При переносе программы с одного компьютера на другой изображение на экране может меняться, если эти компьютеры имеют различное разрешение экрана.

Необходимо разработать программу, чтобы пропорции формы и элементов управления не изменялись. В этой программе можно использовать метод Move в событии FormResize для упорядочения размещения элементов управления. При вызове метода Move необходимо использовать следующий синтаксис:

```
Объект.Move   Left, Top, Width, Height
```

Пример 4.18. Программа пересчета размеров формы

```
Private Sub Form_Resize()
    Dim TheHeight As Single, TheWidth As Single
    TheHeight = (495 / 4140) * ScaleHeight
    TheWidth = (1215 / 6690) * ScaleWidth
    Command1.Move ScaleWidth / 2 – TheWidth / 2, ScaleHeight – TheHeight, _
        TheWidth, TheHeight
End Sub
```

Здесь параметры объекта

Объект	Высота	Ширина
Командная кнопка	495	1215
Форма	4140	6600

Этот простой пример для одного элемента. При большом числе элементов такой метод не эффективен. Надо разработать другую программу.[9]

4.4.4. Упражнения: анимация

Разработайте формы, напишите и отладьте программы из примеров 4.13 – 4.17.

[9] При несовпадении разрешения экрана, использованного при разработке программы с разрешением экрана компьютера на котором используется программа, у пользователя есть простой способ обеспечить совместимость программы с разрешением экрана компьютера - изменить разрешение экрана, если характеристики видеомонитора позволяют это сделать.

Введите команду Пуск\Настройка\Панель управления щелкните по значку Экран выберите заставку Настройка и установите требуемое разрешение экрана. Если на панели Индикация Панели задач есть значок Экран, то для изменения разрешения монитора щелкните правой кнопкой мыши по значку и выберите требуемое разрешение из списка. При необходимости сделайте перезагрузку компьютера.

4.4.5. Закрепление материала

1. Поясните назначение и принцип работы элемента управления Animation.
2. Для чего используется режим DrawMode?
3. Расскажите общий алгоритм создания анимационных эффектов.
4. Перечислите способы создания анимационных эффектов.
5. Как можно создать паузу?
6. Как создать простую анимацию?
7. Поясните способы создания анимационных эффектов с помощью буфера обмена.
8. Поясните, как создать анимационные эффекты путем обмена значениями между графическими объектами.
9. Объясните принцип работы программы “Броуновское движение”.
10. Объясните принцип работы программы “Вспышка звезды”.
11. Объясните принцип работы программы “Радар”.
12. Объясните принцип работы программы “Светофор”.

Задания для самостоятельной работы

1. Разработайте программу движения графического объекта по заданной траектории:
- по эллипсу;
- по параболе;
- по периметру прямоугольника;
- по произвольной функции.
2. Разработайте программу запуска ракеты и выхода ее на орбиту.
3. Разработайте программу полета снаряда с имитацией взрыва при достижении цели, используя программу “Вспышка”.

5. Дополнительные средства для разработки интерфейса.

5.1. Стандартные элементы управления VB

При разработке интерфейса программы пользователя до настоящего момента нами использовались только Надписи, Текстовые поля, Командные кнопки и рамки. Однако в большинстве случаев при разработке пользовательского интерфейса этих элементов управления может оказаться недостаточно. На панели инструментов *Toolbox* имеется ряд элементов управления, которые позволяют улучшить интерфейс. Это такие элементы управления, как *флажки, переключатели, списки, таймер, линейки прокрутки, список устройств, список каталогов,строка состояния, список файлов и др.*

5.1.1. Флажки и переключатели

Флажки (CheckBox) – это элементы управления, которые можно отмечать «галочками», выбирая из ряда опций одну или несколько.

Переключатели (OptionButton) – это элементы управления, которые можно отмечать точкой, выбирая один элемент из группы.

Важнейшим событием для этих элементов является событие ***Click***, а основным свойством – его значение ***Value***. С помощью этого свойства можно определить состояние объекта. Свойство Value может иметь три значения: 0 – не отмечен; 1- отмечен; 2 – отмечен, но недоступен. Последнее значение может быть установлено только программно.

Создание элементов управления

Флажки можно применять как самостоятельно, так и группами, а Переключатели только группами, поэтому их следует помещать в рамку. Для удобства управления эти элементы управления целесообразно объявлять как массив управляющих элементов.

Рассмотрим примеры применения этих элементов.

Пример 5.1. Написать текст программы для управления выводом графиков на экран с использованием элементов CheckBox и OptionButton.

Решение.

Создадим форму согласно рисунку (рис.5.1.).

В рамку “Схема ИЛИ” поместим массив элементов управления Check1: Check1(0), Check1(1), Check1(2).

В рамку “Схема И” поместим элемент управления Check2 и массив элементов управления Option1: Option1(0), Option1(1), Option1(2).

В обработчике события Click кнопки “Показать” запишем текст программы.

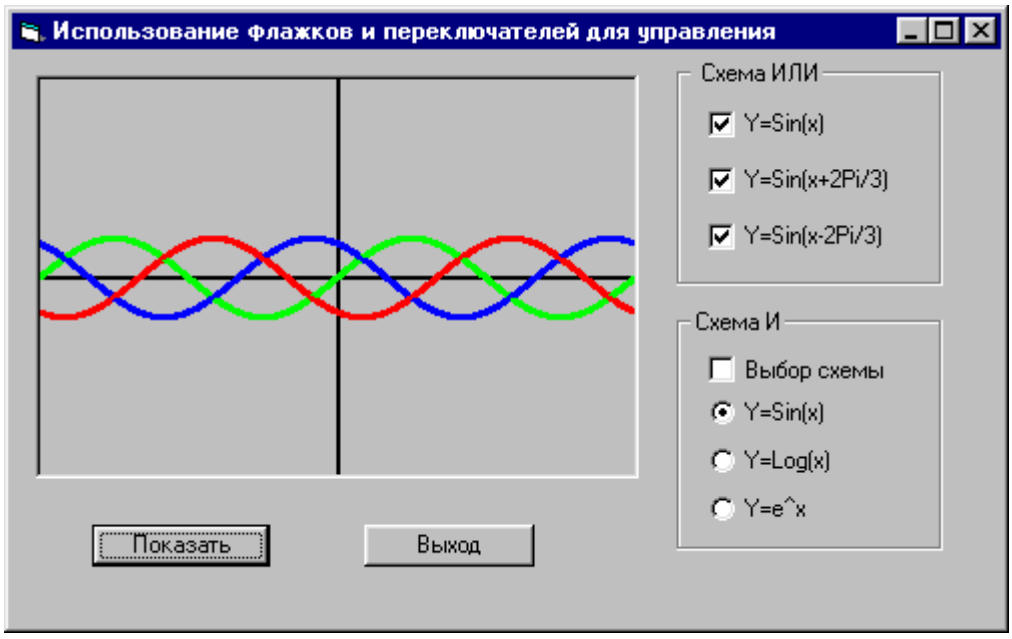


Рис. 5.1. Графики функций

```
Const pi As Single = 3.14159
Dim x As Single, y1 As Single, y2 As Single, y3 As Single
-------------------------------------------------------------------------------------------------
Private Sub Command1_Click()
  Picture1.Cls
  Picture1.Scale (-2 * pi, 5)-(2 * pi, -5)
  Picture1.Line (-2 * pi, 0)-(2 * pi, 0): Picture1.Line (0, -5)-(0, 5)
  For x = -2 * pi To 2 * pi Step 0.01
     If Check1(0).Value Then y1 = Sin(x): Picture1.PSet (x, y1), vbGreen
     If Check1(1).Value Then y2 = Sin(x + 2 * pi / 3): Picture1.PSet (x, y2), vbBlue
     If Check1(2).Value Then y3 = Sin(x + 4 * pi / 3): Picture1.PSet (x, y3), vbRed
  Next x
  If Check2.Value Then
      For x = -2 * pi To 2 * pi Step 0.1
           If Option1(0).Value Then y1 = Cos(x): Picture1.PSet (x, y1), vbMagenta
           If Option1(1).Value Then y2 = Tan(x): Picture1.PSet (x, y2), vbCyan
           If Option1(2).Value Then y3 = Exp(x): Picture1.PSet (x, y3), vbYellow
        Next x
  End If
End Sub
```

Пример 5.2. Использование индекса элемента OptionButton (рис.5.2.).

В обработчике события Click элемента управления Option1 параметром является индекс (Index) активного элемента управления. Значение параметра обрабатывается с помощью оператора Select Case.

Рис. 5.2. Переключатели

```
Private Sub Option1_Click (Index As Integer)
    Select Case Index
        Case 0
            MsgBox “Вы выбрали чай”
        Case 1
            MsgBox “Вы выбрали кофе”
        Case 2
            MsgBox “Вы выбрали какао”
     End Select
End Sub
```

5.1.2. Списки и поля со списками

Имеется несколько типов полей со списками: простые списки, (*ListBox*), раскрывающиеся списки (*ComboBox*) или поле со списком, а также элемент *ImageCombo*.

Списки позволяют выбирать значения из списка, вносить записи в процессе работы программы и на этапе разработки. Если данные не умещаются в окне, то автоматически появляется линейка прокрутки.

Основные свойства и методы списков

Свойства списков:

Text, List, ListIndex, ListCount, Columns, Sorted, ItemData, MultiSelect.

Text – хранить значение выбранного элемента списка;

List – это свойство хранит все значения списка. Все записи в списке имеют индекс (как массивы), нумерация элементов списка начинается с нуля. Зная индекс элемента можно выбрать его из списка. Синтаксис команды:

```
<переменная>=lstBox.List(i)
```

Для добавления элемента в список на этапе разработки введите его в строке свойства List и нажмите Ctrl-Enter для перехода на новую строку.

ListIndex - возвращает индекс элемента списка:

```
< номер_элемента >= lstBox.ListIndex
```

Можно комбинировать свойства List и ListIndex:

```
<Элемент_списка>=lstBox.List(lstBox. ListIndex)
```

Если в списке не выбран ни один элемент, то значение свойства ListIndex равно –1.

ListCount – сохраняет текущее значение числа элементов списка.

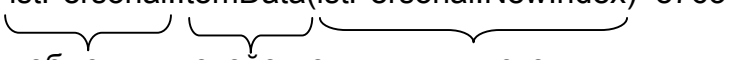
Columns – это свойство позволяет в процессе разработки отображать данные в несколько столбцов. Заполнение столбцов в этом случае осуществля-

ется последовательно – сначала заполняется первый столбец, затем второй и т.д.

Sorted – определяет способ расположения элементов в списке. Если это свойство установлено в *True*, то все элементы будут сортироваться по алфавиту, даже если они добавлены с указанием индекса. Индекс последнего добавленного элемента имеет свойство **NewIndex**. Значение свойства сортировка устанавливается только на этапе разработки.

Вновь добавленный элемент имеет и другое интересное свойство списка – **ItemDate**(). С помощью этого свойства каждому элементу списка можно поставить в соответствие число типа *Long* (целое двойной длины). Используя это свойство, можно составить список сотрудников, сохранив их индивидуальные номера в свойстве ItemData:

```
lstPersonal. AddItem  “Vogel”
lstPersonal.ItemData(lstPersonal.NewIndex)=8763
```



или

```
List1.ItemData (List1.ListIndex)=10986
```

MultiSelect – это свойство позволяет выбирать одновременно несколько элементов. Данное свойство имеет три значения:

0 – множественный выбор невозможен; 1 – простой множественный выбор; 2 – расширенный выбор. Можно использовать при выборе клавиши Shift и Ctrl в сочетании с мышью или клавишами управления курсором.

Множественный выбор элементов списка

- при выделении непрерывной группы элементов списка необходимо установить курсор на первый выделяемый элемент списка, нажать и удерживать клавишу Shift, щелкнуть мышью последний выделяемый элемент списка;
- при выделении группы разрозненных элементов списка выделить первый элемент списка, нажать и удерживать клавишу Ctrl, щелкнуть мышью по другим выделяемым элементам списка.

При множественном выборе свойство Text содержит текст последнего выбранного элемента списка или нет.

Selected – это свойство позволяет определить выделен данный элемент списка или нет:

```
<Логическая_переменная>= List1.Selected
```

Методы списков

AddItem - добавление элементов в список:

```
List1.AddItem  <Элемент [ , Индекс ]>
List1.AddItem “Персональная выставка”,5
```

RemoveItem - удаление элементов из списка:

```
List1. RemoveItem.ИндексЭлемента
```

ListBox Clear - удаление всех элементов списка.

Поле со списком (ComboBox)

Поле со списком обладает свойствами как списка, так и поля ввода.

Для выбора элемента списка используется событие ***Click***, а для изменения записи в поле ввода текста событие – ***Change***.

Для поля со списком важное значение имеет свойство **Style**, определяющее внешний вид и функционирование поля со списком. Оно может принимать три значения:

0 (значение по умолчанию) - текстовое поле и раскрывающийся список. В текстовое поле можно вносить данные;

1 – текстовое поле и постоянно открытый список;

2 - отличается от значения 0 тем, что пользователь не может вводить текст в текстовое поле.

Пример 5.3. Разработать форму для исследования свойств элемента ComboBox.

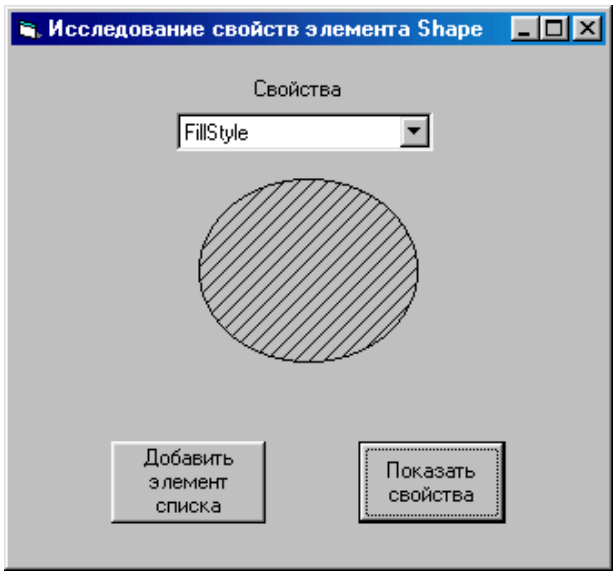


Рис. 5.3. Использование элемента управления ComboBox

Порядок работы

Разработаем форму для исследования свойства объекта Shape с использованием элемента управления ComboBox (рис.5.3).

Поместим на форму объект ComboBox, Shape и две кнопки “Добавить элемент в список” и “Показать свойства”.

Элементу ComboBox присвоим имя Combo1, элементу Shape – Shape1. Внеесем в свойство List объекта Combo1 наименование свойств в следующем порядке: 0 – Shape; 1 – BorderStyle; 2 – BorderWidth. Индексы элементов проставляются автоматически.

Присвоим свойству Style элемента ComboBox значение 2, чтобы запретить добавление элементов в список в процессе работы с клавиатуры, а свойства Sorted присвоим значение True.

Пример 5.4. Напишите программу для добавления элементов в список в процессе выполнения программы.

В обработчик события Click кнопки “Добавить элемент в список” введем текст программы:

```
Private Sub Append_Click()
    Dim A As String, A1 As String, n As Integer
    ' Ввод элементов списка
    A = InputBox("Введите элемент списка и его индекс через запятую")
    ' поиск, выделение номера элемента списка и индекса
    n = InStr(a, ",") ‘ поиск в строке текста запятой
```

```
        A1 = Left(a, n - 1)  ‘ выделение из строки A элемента списка
        n = Val(RTrim(Right(a, Len(A)-n))) ‘ выделение индекса и
                                              ‘ преобразование его в число
        Combo1.AddItem A1, n  ‘ добавление элемента в список
    End Sub
```

Пример 5.5. Демонстрация свойств элемента Shape.

В обработчик события Click кнопки “Показать свойства” введем текст программы:

```
Private Sub Demo_Click()
    Dim a As String, n As Integer
    Static I As Integer
    Dim Index As Integer
    Index = Combo1.ListIndex
    ‘ установка  числа циклов в зависимости от исследуемого свойства
    Select Case Index
        Case 0, 1, 2
            n = 5
        Case 3
            n = 7
        Case 4
            n = 15
    End Select
    I = I + 1
    If I > n Then I = 0
        If Trim(Combo1.Text) = "Shape" Then Shape1.Shape = I
        If Trim(Combo1.Text) = "BorderStyle" Then Shape1.BorderStyle = I
        If Trim(Combo1.Text) = "FillStyle" Then Shape1.FillStyle = I
        If Trim(Combo1.Text) = "FillColor" Then
            Shape1.FillColor = QBColor(I)
    End If
    If I > 0 Then
        If Trim(Combo1.Text) = "BorderWidth" Then Shape1.BorderWidth = I
    End If
End Sub
```

Установка начального значения

При загрузке списка в поле ввода обычно появляется какое-то значение. Это значение можно установить при разработке программы, присвоив свойству *Text* нужное значение, или установить его программно, используя свойство ListIndex:

```
Combo1.ListIndex = 2
```

Если вы хотите, чтобы при загрузке формы поле ввода осталось пустым, присвойте свойству *Text* на этапе разработки пустую строку.

5.1.3. Полоса прокрутки (ScrollBar)

Полосы прокрутки используются во всех списках, формах, но могут выполнять и некоторые специфические свойства, например, роль регуляторов.

С точки зрения программирования, полосы прокрутки являются одними из самых простых элементов управления.

Основными свойствами полосы прокрутки *ScrollBar* являются *Max*, *Min* и *Value, SmallChange*.

Свойства ***Max*** и ***Min*** определяют диапазон измеряемых величин, который может изменяться от – 32768 до + 32767. Значение свойства ***Value*** напрямую зависит от установленного диапазона и определяется текущим положением ползунка.

Свойство ***SmallChange*** определяет, на какую величину будет изменяться значение свойства Value при щелчке мышью по правой или левой кнопке полосы прокрутки.

Для полосы прокрутки важное значение имеют события ***Change*** и ***Scroll.*** При изменении положения ползунка автоматически возникает событие *Change* для описываемого элемента управления. Событие *Scroll* возникает перед событием Change.

Пример 5.6. Использование свойств полосы прокрутки (рис.5.4.).

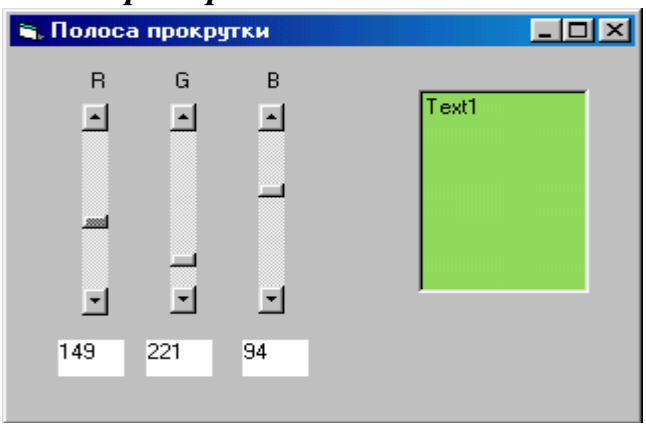


Рис. 5.4. Полоса прокрутки

Поместим на форму три полосы прокрутки, элемент TextBox и шесть Надписей. Текстовое поле будем использовать для отображения цвета. Три Надписи задействуем для обозначения полос прокрутки, а остальные Надписи используем для отображения числового значения свойства Value полос прокрутки. Свойству Max элемента ScrollBar присвоим значение 255, а свойству Min – ноль, а свойству SmallChange присвоим значение 1.

Текст программы для полос прокрутки поместим в обработчик события Scroll, чтобы одновременно с перемещением ползунка менялось и числовое значение. При перетаскивании ползунка полосы прокрутки цвет текстового поля на экране будет синхронно изменяться, а в Надписях будет показываться значение свойства Value.

```
Private Sub VScroll1_Scroll()
    Label4(0).Caption = VScroll1.Value
    Text1.BackColor = RGB(VScroll1.Value, VScroll2.Value, VScroll3.Value)
End Sub
Private Sub VScroll2_Scroll()
    Label4(1).Caption = VScroll2.Value
    Text1.BackColor = RGB(VScroll1.Value, VScroll2.Value, VScroll3.Value)
End Sub
Private Sub VScroll3_Scroll()
```

```
    Label4(2).Caption = VScroll3.Value
    Text1.BackColor = RGB(VScroll1.Value, VScroll2.Value, VScroll3.Value)
End Sub
```

5.1.4. Элемент управления Slider.

Другим элементом управления, похожим на полосу прокрутки и встречающимся в приложениях Windows является элемент ***Slider***. Он позволяет выбирать дискретное значение или набор значений из определенного диапазона.

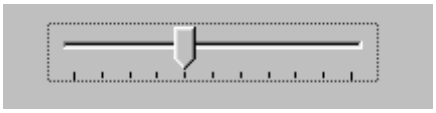


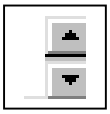
Рис.5.5. Элемент управления Slider

Этого элемента управления нет среди стандартных элементов управления панели ToolBox. Для загрузки его на панель элементов управления выберите команду ***Project\Components***, установите флажок ***Microsoft Windows Common Controls 6.0 (SP3)*** или ***Common Controls 5.0 (SP2*)**.

Элемент Slider имеет свойства **Min**, **Max** и **Value** как и полоса прокрутки. Параметры изменения значений при перемещении ползунка в области значений определяют свойства **SmallChange** – минимальное изменение и **LargeChange** – максимальное изменение.

В отличие от полосы прокрутки для этого элемента можно определить не только одно значение, но и некоторый диапазон значений. Для этого служат свойства **SelStart** и **SelLength**, но само выделение диапазона должно выполняться программно. Новое свойство **Text** позволяет задавать текст надписи, которая будет отображаться при перемещении ползунка. Позиция отображения этой надписи определяется значением свойства **TextPosition**. Новым является также и событие ***Validate***. В обработчике этого события помещается код проверки правильности введенных данных.

5.1.5. Счетчик (UpDown)



Счетчик (элемент управления UpDown) используется для установки различных числовых значений. Данный элемент можно использовать без написания кода, если правильно определить свойства *AutoBuddy, SyncBuddy, BuddyControl и BuddyProperty* элемента управления Buddy.

Buddy – это элемент управления, значение которого меняет UpDown (рис.5.6.). Buddy можно установить с помощью свойства *Buddy-Control* или свойства *AutoBuddy*. В первом случае надо вручную записать в строку ввода имя элемента управления, значение которого будет менять элемент UpDown, во втором случае программа автоматически внесет в строку ввода имя элемента управления, у которого значение свойства TabIndex на единицу меньше, чем у UpDown.

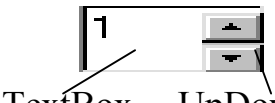


Рис.5.6. Композиция TextBox +UpDown

Свойство ***BuddyProperty*** указыавет, какое свойство элемента Buddy должно синхронизи-роваться со свойством *Value* элемента UpDown. Для элемента управления TextBox таким свойством будет свойство свойство Text. Если свойству *BuddyProperty* присвоить значение Default, то автоматически будет обновляться стандартное свойство, элемента управления, связанного с UpDown.

Область значений устанавливается с помощью свойств Min, Max, Value.

Другие свойства элемента управления UpDown:

SyncBuddy – предписывает элементу управления UpDown обновлять значения связанного с ним элемента управления.

Wrap – определяет, примет ли элемент управления свое минимальное значение, если щелкать мышью по кнопке со стрелкой вверх до тех пор, пока будет превышено максимально допустимое значение (и наоборот).

Aligument – определяет положение элемента (слева, справа);

Orintation – определяет положение элемента по горизонтали или по вертикали.

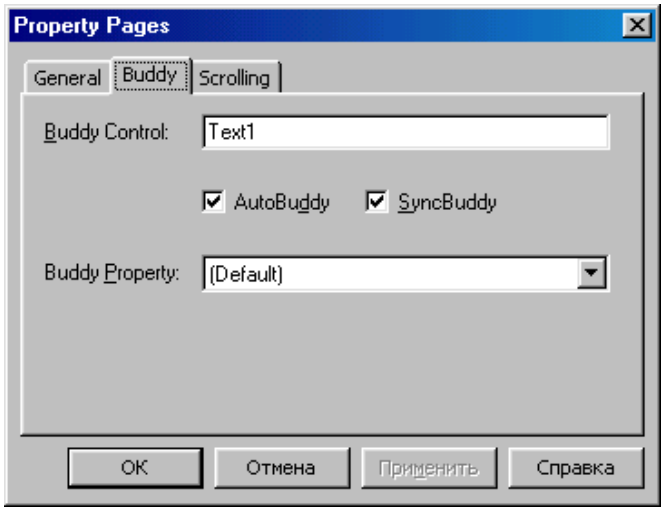


Рис.5.7. Настройка элемента UpDown

Порядок работы по настройке элемента UpDown.

1. Создать элемент TextBox и присвоить ему имя Text1.

2. Создать элемент UpDown и присвоить ему имя UpDown1. Свойства ***TabIndex*** этих элементов будут смежными.

3. Щелкнуть правой кнопкой мыши по элементу UpDown и выбрать в контекстном меню опцию Properties – откроется диалоговое окно (Property Pages) настройки свойств элемента UpDown.

4. Открыть закладку Buddy (рис. 5.7) диалогового окна и установить флажок AutoBuddy. В строке ввода автоматически устанавливается имя элемента управления Text1.

5. Установить свойство BuddyProperty равным Default. При этом автоматически устанавливается флажок SyncBuddy - обновлять стандартное свойство Text элемента Text1.

6. Открыть закладку Scroling и установить значения Min, Max и Increment. Последнее свойство позволяет установить шаг изменения значения свойства Text элемента Text1 при щелке мышью по кнопкам элемента UpDown1.

7. Настроить, при необходимости, ориентацию элемента UpDown с помощью закладки General диалогового окна.

8. Завершить работу, щелкнув по кнопке OK.

5.1.6. Упражнения: Основные элементы интерфейса

1. Разработайте и отладьте программы примеров 5.1, 5.3, 5.4.

2. В программу “Броуновское движение“ (пример 4.13.) добавить элементы управления UpDown для установки числа циклов и множителей для вычисления приращения координат по осям X и Y.

5.1.7. Закрепление материала

1. Поясните назначение элементов управления CheckBox и OptionButton. В чем состоит отличие в использовании названных элементов управления?

2. Назовите основные свойства элементов управления CheckBox и OptionButton.

3. Приведите фрагмент программы для использования свойства Value элементов управления CheckBox и OptionButton.

4. Как создать массивы элементов управления CheckBox и OptionButton. Приведите фрагмент программы для использования свойства Index этих элементов управления.

5. Для каких целей применяются списки List и ComboBox? В чем отличие списка и поля со списком?

6. Перечислите основные свойства поля со списком ComboBox. Приведите синтаксис использования свойств элемента ComboBox.

7. Перечислите основные методы поля со списком ComboBox. Приведите синтаксис использования методов элемента ComboBox.

8. Как занести данные в список на этапе разработки формы?

9. Как дополнить список элементов управления ComboBox или List в процессе работы?

10. Для чего предназначена полоса прокрутки ScrollBar?

11. Назовите основные свойства полосы прокрутки.

12. В чем отличие элемента управления ScrollBar от элемента управления Slider?

13. Каково назначение элемента управления UpDown?

14. Как настроить элемент UpDown без написания программы?

Задание для самостоятельной работы

Разработать форму для построения графиков элементарных функций (синуса, косинуса, тангенса, котангенса, эллипса, окружности, параболы, гиперболы) с использованием элементов управления CheckBox и OptionButton.

Одновременно на графике должно быть не более трех функций. Список элементарных функций хранить в поле со списком.

5.2. Дополнительные элементы управления

Одним из важных аспектов проектирования приложений пользователя является информирование пользователя о состоянии приложения и ходе выполнения программы. Например, в текстовых редакторах необходимо указывать номер текущей страницы и общее число страниц в документе, при работе с файлами данных пользователя могут интересовать сведения о размере файла, числе записей, номере текущей записи и т. д. В процессе работы пользователю может быть интересно знать, как идет загрузка программы, копирование файла или как долго будет идти расчет параметров. Для предоставления информации об интересующих пользователя данных и ходе протекания процессов могут использоваться такие средства Visual Basic, как

- строка состояния – StatusBar;
- графическое представление хода выполнения программы – ProgressBar;
- анимационные видеоролики – Animation.

5.2.1. Строка состояния

Создание строки состояния

Строки состояния нет среди стандартных элементов панели элементов управления, поэтому ее надо загрузить командой ***Project\Components*** и установить флажок ***Microsoft Windows Common Controls 6.0 (SP3).*** После загрузки строки состояния на панель элементов управления установите ее на форму. Панель состояния может содержать до 16 отдельных панелей, в которых отображается различная информация, например, дата, время, имя файла и др.

Настро окна Свойства среды разраб ty Pages) (рис. 5.8).

Прежде е панелей. Оно определяется

Свойст состояния может быт ели или составной (sbrNorm елей со своими свойствами. енты массивов управляющих няется с указанием индекса

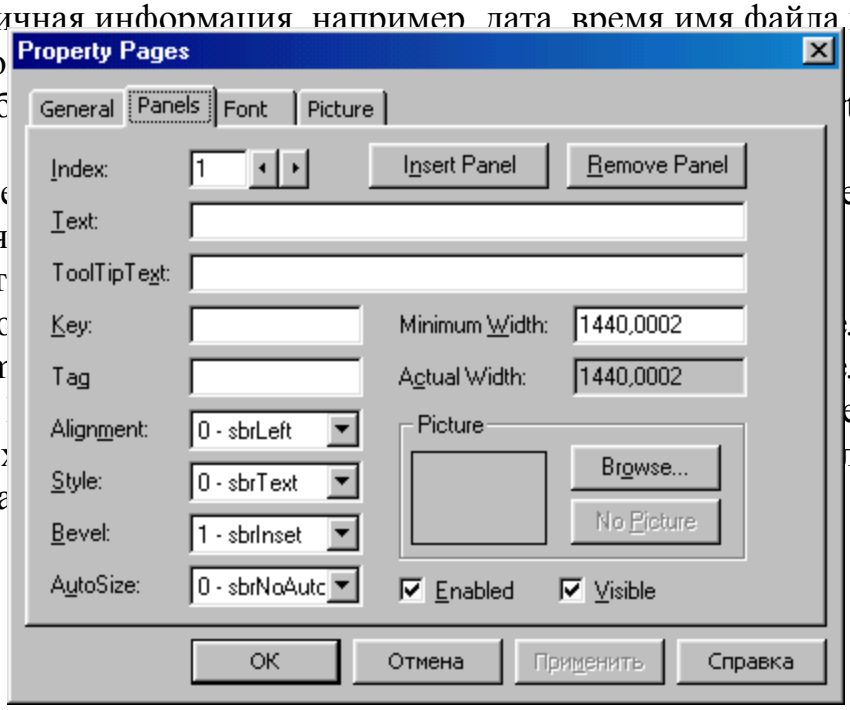


Рис. 5.8. Строка состояния

Свойство **Align** определяет, к какой стороне формы должна “прикрепиться” строка состояния. Это свойство может принимать пять значений: vbAlignTop – к верхнему краю формы; vbAlignBottom – к нижнему краю формы; vbAlignLeft – к левому краю формы; vbAlignRight – к правому краю формы; vbAlignNone – строка состояния может располагаться в любой части формы.

Настройку свойств панелей строки состояния удобно проводить с помощью панели свойств. Щелкните правой кнопкой мыши по строке состояния и выберите в меню пункт Properties. На экране появится панель свойств (рис. 5.8). Закладка General позволяет установить общие свойства строки состояния, а закладка Panels – свойства каждой из 16 панелей строки состояния.

Основные свойства панелей строки состояния

Имеется восемь основных свойств панелей строки состояния, которые определяют их внешний вид и функции.

Text – определяет текст, который будет появляться в текстовой панели. Этот текст выводится обычно в процессе выполнения программы;

ToolTipText – текст подсказки, который появляется при зависании указателя мыши над панелью;

Alignment – Управляет выравниванием текста в панели: слева, справа или по центру;

Style – определяет тип создаваемой панели. Можно установить семь типов панелей: ***sbrText***- позволяет отобразить текст или растровое изображение, указанные соответственно в свойствах Text или Picture панели строки состояния; ***sbrCaps*** – отображает состояние клавиши CapsLock; ***sbrNum*** - отображает состояние клавиши NumLock; ***sbrIns*** - отображает состояние клавиши Insert; ***sbrScrl*** - отображает состояние клавиши ScrolLock; ***sbrTime*** – отображает текущее время; ***sbrDate*** – отображает текущую дату.

Bevel – Определяет тип затенения для имитации объемности панели;

AutoSize – определяет принцип управления размером панели из прогрпаммы;

MinWidth – устанавливает минимальные размеры панели;

Picture – позволяет поместить изображение в строку состояния.

Управление панелью состояния

На этапе разработки добавление и удаление панелей осуществляется с помощью кнопок: **Insert Panel** – добавление панели и **Remove Panel** – удаление панелей.

Для управления панелями строки состояния в процессе работы программы используются методы *Add, Remove* и *Clear.*

Add **–** добавить панель. Синтаксис метода:

```
Add ([Index] [,клавиша] [,текст] [, стиль] [,картинка]) As Panel
```

Remove – удаление панели. Синтаксис метода:

Remove [Index]

Clear – удаляет все панели из строки состояния. Синтаксис метода: Clear

Пример 5.7. Вывод текст в строку состояния динамически:

StatusBar1.Panels(1).Text =”Ждите. Идет решение задачи”

Пример 5.8. Управление строкой состояния

Создать строку состояния с одной текстовой панелью. Вывести в нее текст “Моя панель”. Добавить в строку состояния одну текстовую панель, в которую поместите текст “Для продолжения нажмите любую клавишу” и панель для отображения текущего времени (рис. 5.9). Удалять первую текстовую панель по нажатию любой клавиши.

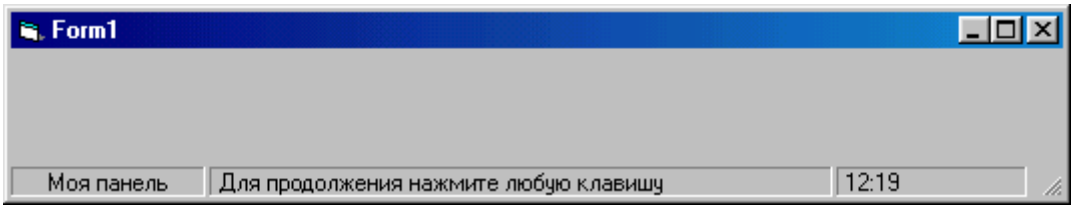


Рис. 5.9. Пример строки состояния

Порядок работы.

- Добавьте на панель элементов управления элемент StatusBar командой Project\Components, установите флажок Microsoft Windows Common Controls 6.0 (SP3) и нажмите Ok.
- Установите свойства Align строки состояния равным 2 – прицеплятся к нижнему краю формы, а свойство Style оставьте без изменения (sbrNormal).
- Вызовите панель Property Pages и откройте закладку Panels.
- Введите в поле Text текущей панели текст “Моя панель”.
- Напишите в обработчики событий Load и KeyPress формы текст программы:

```
Private Sub Form_Load()
      Me.Width = 8000
      Me.Height = 1500
      Dim S As String
      With StatusBar1
          'добавляем текстовую панель в левую часть строки состояния
          .Panels.Add 2, , "Для продолжения нажмите любую клавишу", sbrText
          .Panels(2).AutoSize = sbrSpring
          ' добавляем панель с текущим временем
         .Panels.Add , , , sbrTime
       End With
End Sub
Private Sub Form_KeyPress(KeyAscii As Integer)
      ' удаляем панель созданную по умолчанию
      With StatusBar1
```

```
            .Panels.Remove 1
        End With
End Sub
```

- Запустите программу и проверьте ее работу.

5.2.2. Индикатор процесса

Индикатор процесса (ProgressBar) предназначен для отображения медленно протекающих процессов. Этот элемент управления загружается на панель элементов управления в одной группе со строкой состояния.

Основными свойствами этого элемента управления являются:

Max и ***Min*** – дла установки диапазона изменения контролируемого параметра. Если необходимо выводить результат в процентах, то свойству Min надо присвоить значение 0, а свойству Max – 100;

Value – возвращает текущее значение. Определение значения этого свойства предоставляется разработчику, так как сам элемент управления не имеет возможности отслеживать продвижение процесса:

```
‘пример для загрузки нескольких файлов
Sub LoadFiles()
      Progress1.Min=0
      Progress1.Max=nFiles
      For i=1 To nFiles
          Call LoadFile(i)
          Progress1.Value=i
    Next i
End Sub
```

5.2.3. Списки устройств, каталогов и файлов

Visual Basic 6 имеет три элемента управления, предназначенные для работы с файлами и каталогами: ***DriveListBox*** (Список устройств), ***DirectoryListBox*** (Список каталогов), ***FileListBox*** (Список файлов). Все эти элементы используются, как правило, совместно. Взаимодействие между ними осуществляется через событие *Change*. Большинство их свойств совпадают со свойствами поля со списком.

Список устройств

Этот элемент управления предназначен для отображения списка всех доступных дисков и устройств системы и обеспечивает возможность их выбора.

Основным свойством элемента *DriveBox* является свойство ***Drive***, которое возвращает выбранный диск или устройство, например, “C:\”.

Список каталогов

Данный элемент управления предназначен для выбора каталогов. Он отображает структуру каталогов выбранного диска и позволяет осуществлять выбор и смену каталогов. Основным свойством списка каталогов является свойство ***Path***. Оно возвращает полный путь к выбранному каталогу, включая имя диска (например, C:\Windows\Word).

Список файлов

Список файлов отображает файлы текущего каталога и обеспечивает их выбор.

Основными свойствами данного элемента управления являются свойства *FileName* и *Pattern*.

Свойство ***FileName*** содержит имя выбранного файла (например, Bock.doc).

Свойство ***Pattern*** позволяет установить типы файлов, которые должны отображаться в списке. Для установки типов файлов допускается использование маски: *.ICO, *.BMP и т.д.

Например:

```
File. Pattern="*.ICO;*.BMP
```

Здесь *File* – имя элемента управления *FileListBox*. Для организации совместного использования списков необходимо написать следующие тексты программ:

а) для списка устройств:

```
Private Sub Drive1_Change( )
   Dir1.Path=Drive1.Drive
End Sub
```

б) для списка каталогов:

```
Private Sub Dir1_Change( )
   File1.Path=Dir1.Path
End Sub
```

в) для списка файлов:

- для отображение полного маршрута выбранного файла напишем команду:

```
lblPath.Caption=File1.Path & "\" & File1.FileName.
```

Чтобы избежать отображения в маршруте излишнего количества символов "\" рекомендуется применять следующий код:

```
Private Sub File1_Click ( )
    If Right (File1.Path,1)= "\" Then
       lblPath.Caption=File1.Path & File1.FileName
    Else
       lblPath.Caption=File1.Path & "\" & File1. FileName.
    End If
End Sub
```

Пример использования данных списков приведен на рис. 5.10.

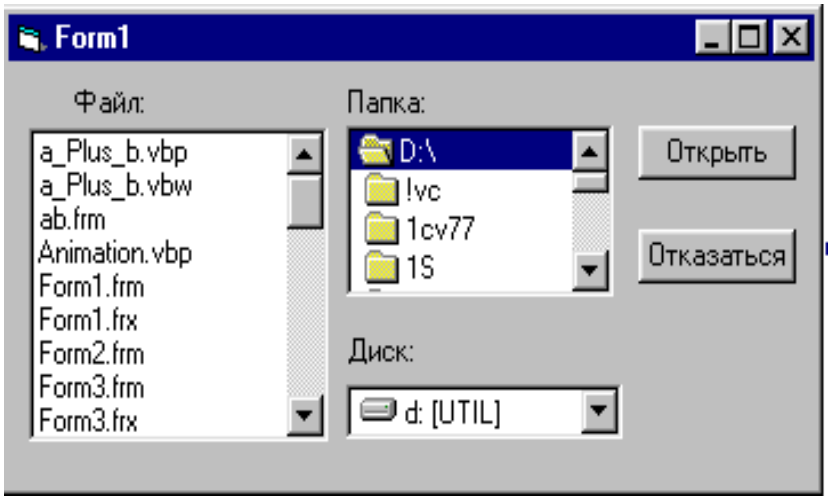


Рис. 5.10. Окно диалога для открытия файла

Рассмотренные элементы управления позволяют создать, при необходимости, средства интерфейса для открытия файлов и их сохранения. Однако для этих целей VB 6 имеет более мощные средства. Это стандартные диалоговые окна Windows.

5.2.4. Стандартные окна диалога Windows

Стандартные диалоговые окна представлены элементом управления **CommonDialog**. Загрузка этого элемента управления осуществляется командой ***Project\Components.*** После входа в окно диалога установите флажок ***Microsoft Common Dialog. Control 6.0*** и щелкните клавишу ***OK.***

Элемент управления Common Dialog, помещенный на форму, позволяет получить доступ к нескольким диалоговым окнам:

Open (Открыть) (рис.5.11.). Это окно позволяет реализовать функцию открытия файла;

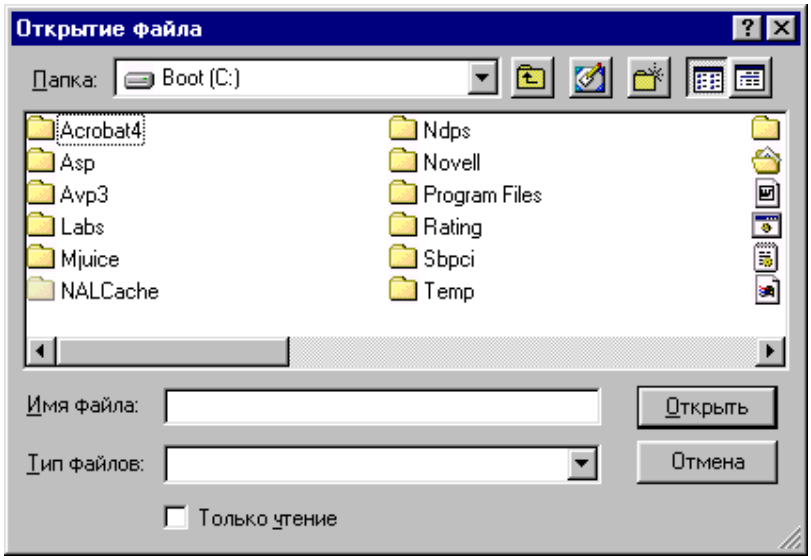


Рис.5.11. Стандартное окно диалога Open

Save As (Сохранить как…), реализует функцию сохранения файлов на диске;

Print (Печать), реализует стандартную функцию Windows по настройке принтера печати документов;

Font (Выбор шрифта), позволяет выбрать тип шрифта и все его атрибуты;

Color (Цвет), предназначено для выбора одного из цветов стандартной палитры либо для создания нового цвета;

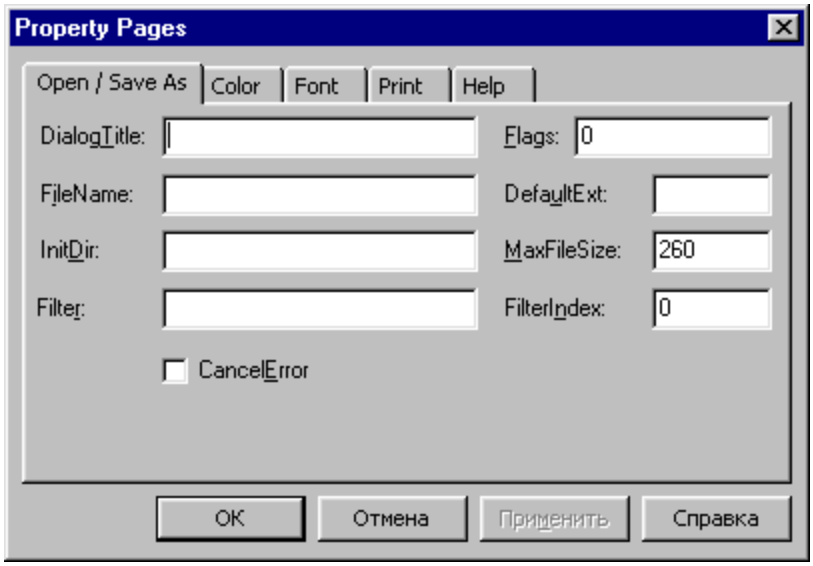


Рис. 5.12. Окно настройки свойств окон диалога

Help (Справочная система), позволяет создать интерфейс для работы пользователя со справочной системой Windows.

Диалоговые окна *Open* и *Save As* выглядят совершенно одинаково (рис.5.11) и похожи на стандартные окна диалога Windows.

Настройку окна диалога удобно вести с помощью окна свойств (рис. 5.12.), которое вызывается с помощью контекстного меню объекта CommonDialog. Щелкните правой кнопкой мыши по окну объекта CommonDialog и выберите команду свойства - ***Properties***, откроется окно свойств *Property Pages*. Откройте закладку ***Open/***

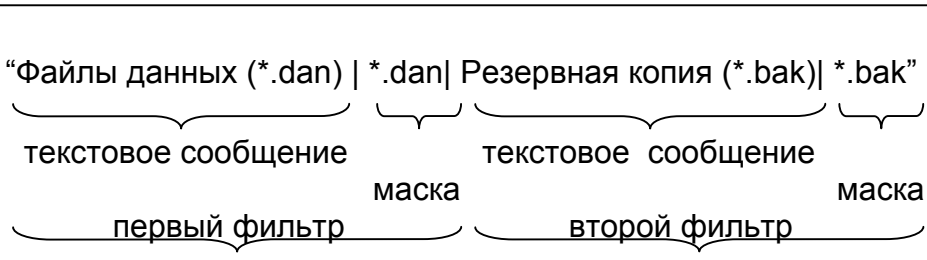


Рис. 5.13. Фильтр, вводимый в строке Filter окна диалога Открытие файла

Save As.оле ввода *DialogTitle* внесите название окна диалога “Открытие файла”; в поле ввода *InitDir* – внесите имя каталога, используемого по умолчанию; в поле ввода *Filter* – маску для чтения файлов с нужным расширением имени файла. Фильтр состоит из двух частей. Первая часть – текст, выводимый в строку ввода “тип файлов” (рис.5.11.), например, “Файлы данных(*.dan)”. Вторая часть – собственно фильтр – “*.dan”, отделенный от первой части вертикальной чертой. Если в фильтре используется несколько масок, то они имеют одинаковую структуру и отделяются друг от друга также вертикальной чертой (рис.5.13).

Флажок *Flags* позволяет ввести в диалоговое окно флажок “Только для чтения”.

Диалоговые окна Open и Save As позволяют только возвратить в программу полный путь (спецификацию) открываемого или сохраняемого файла. Спецификация выбранного файла сохраняется в свойстве ***FileName*** окна диалога. Задача использования этих данных ложится на программиста.

Для вызова диалогового окна Open используется метод ***ShowOpen***. Синтаксис команды:

```
ИмяФормы.ShowOpen.
<переменная>= ИмяФормы.FileName.
```

Для вызова диалогового окна сохранения файла используется метод ***ShowSave***:

```
ИмяФормы. ShowSave.
<переменная>=ИмяФормы.FileName.
```

Здесь <переменная> – имя переменной для хранения и последующего использования имени открываемого или сохраняемого файла.

5.2.5. Печать документов.

Для управления выводом информации на печать используются метод ***PrintForm*** и объект ***Printer***.

Метод PrintForm

С помощью метода PrintForm на принтер выводится форма в виде растрового изображения с установленным в системе разрешением (чаще всего 96 dpi). Метод PrintForm может использоваться только для печати форм. При этом на принтер, установленный по умолчанию, выводится содержимое формы без строки заголовка и рамки.

Все элементы управления выводятся на печать так, как отображаются на экране, т.е. с соответствующими надписями, границами, видами шрифтов и т.д. Невидимые во время выполнения элементы управления на печать не выводятся. Содержимое элемента управления PictureBox выводится на печать только в том случае, если значение свойства AutoRedraw равно True.

Фрагмент кода для печати формы.

```
Load frmForm1
frmForm1.lblOutput.Caption=<выводимый текст>
frmForm1.picOutput.Picture=LoadPicture("c:/bibl.bmp")
frmForm1.PrintForm
Unload frmForm1
```

Объект Printer

Объект Printer предназначен для вывода на печать текста и графики.

В отличие от метода PrintForm, объект Printer позволяет выводить документ на печать с разрешением, установленным для принтера, а не для экрана, благодаря чему можно достичь лучшего качества печати. Кроме того этот объект можно использовать для печати многостраничных документов. Однако, весь процесс печати необходимо программировать.

Основные свойства и методы объекта Printer

Наиболее важным методом объекта Printer является метод ***Print***, с помощью которого текст передается на принтер:

```
Printer. Print “ Здравствуйте “
Printer. Print “ Печать документа “
```

Вывод осуществляется с верхнего левого угла печатной страницы, с использованием текущих параметров объекта Printer. Для изменения вида шрифта используется свойство объекта Font:

```
Printer. Font. Name = “ Times New Roman “
Printer. Font. Size = 12
Printer. Font. Underline = True
Printer. Print = “Здравствуй, читатель “
```

Для изменения единицы измерения служит свойство **ScaleMode**. По умолчанию в качестве единицы измерения используется твип. При установки значения свойства ScaleMode можно использовать константы ***vbCentimeters, vbMillimeters*** или ***vbPixels*** – сантиметры, миллиметры, пиксели – соответственно.

Для позиционирования точки вывода используются свойства **CurrentX** и **CurrentY**. CurrentY устанавливает расстояние от верхнего края печатаемой области, CurrentX – от левого края печатаемой области. Ширина и высота печатаемой области в условных единицах измерения устанавливается свойствами **ScaleWidth** и **ScaleHeight**.

Ширина и высота строки устанавливается с помощью свойств **TextHeight** и **TextWidth**.

При выводе текста можно использовать функцию ***Tab***, а также управляющие символы (;) и (,) также как в операторе Print. Для печати графических объектов используются методы ***Pset***, ***Line***, и ***Circle*** объекта Printer

```
Printer. Line (1,1) – (10,5)
```

Готовые графические изображения различных форматов можно выводить с помощью метода ***PaintPicture*** (см. 4.3.5):

```
Printer.PaintPicture Form1.Picture, 0, 0, 1500, 1000, 0, 0, 7000, 5000
```

После направления всех данных на печать с помощью объекта Printer готовая страница пока еще находится в оперативной памяти. Для запуска процесса печати этот объект должен получить сообщение о том, что формирование этой страницы завершено. Для этой цели предназначен метод ***NewPage***. После того, как все страницы сформируются, вызывается метод ***EndDoc***, который направляет сформированный документ на принтер.

Для прерывания печати используется метод ***KillDoc***.

5.2.6. Упражнения: дополнительные элементы управления для разработки интерфейса

Задача 1. Разработать форму для табулирования функции двух переменных. Данные выводить в сетку и сохранять в массиве для последующего вывода на печать и в файл на диске. Поместить на форму строку состояния. В строку состояния выводить динамически число строк и столбцов в мвссиве и текущее время.

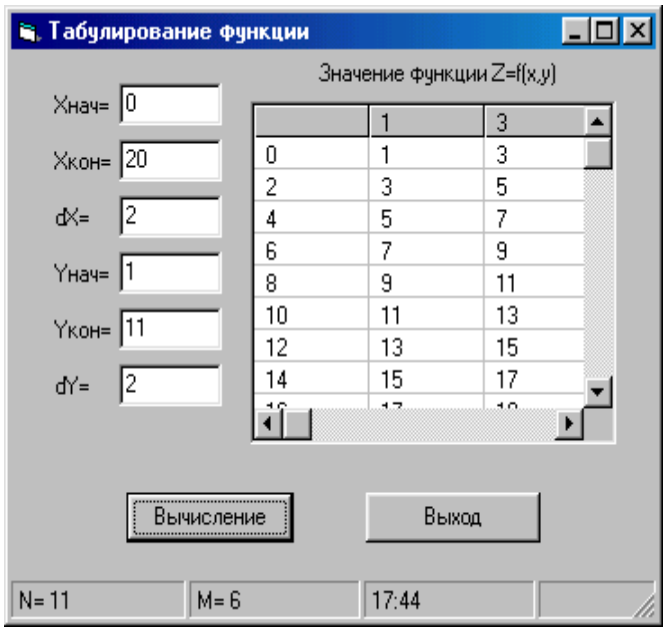


Рис.5.14. Табулирование функции двух переменных

Порядок работы.

Разработайте программу табулирования функции двух переменных.

Разработайте эскиз формы (рис.5.14).

Поместить на форму строку состояния и установить на ней четыре поля: первое и второе – текстовые; третье – текущее время, четвертое – текстовое (резервное).

Напишите текст программы:

```
Option Explicit
Dim Xn As Single, Xk As Single, Dx As Single
Dim Yn As Single, Yk As Single, Dy As Single
Dim X As Single, Y As Single, Z() As Single
Dim M As Integer, N As Integer
Dim NameFile As String, NameFile1 As String
Dim Nkan As Integer
Private Sub Command1_Click()
    Dim i As Integer, j As Integer
    Xn = Val(Text1(0).Text)
    Xk = Val(Text1(1).Text)
    Dx = Val(Text1(2).Text)
    Yn = Val(Text1(3).Text)
    Yk = Val(Text1(4).Text)
    Dy = Val(Text1(5).Text)
    Cls
   ' определение размерности массива и сетки
    N = Int((Xk - Xn) / Dx) + 1
    M = Int((Yk - Yn) / Dy) + 1
    ReDim Z(N, M)
    Grid1.Rows = N + 1
    Grid1.Cols = M + 1
    ' нумерация строк
    X = Xn
    Grid1.Col = 0
    Grid1.ColAlignment(0) = 2
    For i = 1 To N
        Grid1.Row = i
        Grid1.Text = Str$(X)
```

```
            X = X + Dx
        Next i
        ' нумерация столбцов
        Y = Yn
        Grid1.Row = 0
        For j = 1 To M
            Grid1.Col = j
            Grid1.ColAlignment(j) = 2
            Grid1.Text = Str$(Y)
            Y = Y + Dy
        Next j
        ' вычисление значения Z
        X = Xn
        For i = 1 To N
            Y = Yn
            Grid1.Row = i
            For j = 1 To M
                Z(i, j) = X + Y
                Y = Y + Dy
                Grid1.Col = j
                If X <> 0 Then
                    Grid1.Text = Str$(Format(Z(i, j), "#.##"))
                Else
                    Grid1.Text = Str$(Z(i, j))
                End If
            Next j
            X = X + Dx
        Next i
        StatusBar1.Panels(1) = "N=" & Str(N)
        StatusBar1.Panels(2) = "M=" & Str(M)
    End Sub

    Private Sub Command2_Click()
        Unload Me
    End Sub
```

Для контроля правильности работы программы составьте исходные данные для тестирования: Хнач=0, Хкон=10, Dx=2, Yнач=1, Yкон=11, Dy=2. Функция Z=x+y.

Запустите программу и проверьте результаты работы: Z(0,0)=1; Z(0,1)=3; ... Z(6,6)=21

Задача 2. Разработайте интерфейс для открытия файлов, сохранения их на диске, без изменения имени файла и с изменением имени файла, а также вывода данных на печать. При разработке формы использовать:

а) элементы управления ***DriveListBox***, ***DirectoryListBox***, ***FileListBox***;

б) элемент управления ***CommonDialog***.

Указания к выполнению п.2. Создайте меню пользователя по аналогии с пунктом меню File Visual Basic. Для пунктов меню Открыть, Сохранить, Сохранить как... и Печать создайте формы с использованием элементов управле-

ния рассмотренных в настоящем разделе. Вопрос об использовании этих форм будет рассматриваться в шестом разделе.

5.2.7. Закрепление материала

1. Для чего предназначена строка состояния?
2. Как создать строку состояния?
3. Каким образом осуществляется настройка панелей строки состояния?
4. Для чего предназначен индикатор процесса.
5. Поясните назначение элементов управления DriveListBox, DirectoryListBox, FileListBox.
6. Назовите основные свойства и события элементов управления DriveListBox, DirectoryListBox, FileListBox.
7. Как организовать совместное использование элементов управления DriveListBox, DirectoryListBox, FileListBox?
8. Каково назначение элемента управления CommonDialog?
9. Как используются диалоговые окна элемента управления CommonDialog?
10. Как осуществляется настройка окна диалога Save или Open?
11. Для чего предназначен метод PrintForm?
12. Для чего предназначен объект Printer?
13. Как осуществляется настройка параметров шрифта для вывода данных на печать?
14. Как осуществляется управление позиционированием точки вывода при печати?

6. Работа с файлами данных

6.1. Файлы последовательного доступа

6.1.1. Понятие о файлах данных

В процессе разработки программ часто возникает необходимость в хранении и обработке сохраненной информации. Эта информация может быть самой разнообразной: исходные данные для решения задач, результаты вычислений, списки и так далее. Для хранения такой информации могут использоваться файлы баз данных. Visual Basic 6 имеет достаточно средств для работы с данными, но они требуют больших ресурсов вычислительной системы. Однако, класс решаемых задач, порой, не оправдывает использования полноценного механизма баз данных, так как это может привести к усложнению программы,

увеличению ее размера и замедлению работы. В таком случае целесообразно использовать текстовые файлы. Visual Basic 6.0, так же как и предыдущие версии языка Basic, имеет достаточно средств для работы с текстовыми файлами.

В зависимости от организации данных на дисках или других машинных носителях текстовые файлы делятся на файлы с *последовательным доступом, файлы с прямым доступом* и *двоичные файлы*.

Текстовые файлы с последовательным доступом (файлы последовательного доступа) не имеют какой-либо структуры. Структура этих файлов определяется самой считывающей программой. В текстовых файлах с последовательным доступом каждая строка заканчивается двумя специальными символами: конец строки и возврат каретки, которые вводятся в текст программы при нажатии клавиши Enter (Ввод) на клавитауре. Поэтому один из самых легких способов обработки текстового файла с последовательным доступом состоит в чтении его строка за строкой. Создание текстовых файлов с последовательным доступом также не представляет большого труда. Его можно создать любым текстовым редактором. Данные в файл последовательного доступа записываются последовательно байт за байтом. Чтобы проанализировать и выбрать нужную информацию файл должен быть полностью прочитан. Это повышает требования к объему оперативной памяти и снижает скорость выполнения программы.

Текстовые файлы с прямым доступом (файлы прямого доступа) предназначены для чтения и записи текста или структурированных двоичных файлов с записями фиксированной длины. Они позволяют записывать и извлекать данные из файла по номеру записи. Это сокращает время на поиск и извлечение данных. Однако при этом имеет место неэффективное использование дискового пространства, так как длина каждого поля в записи должна быть заранее оговорена.

Двоичные файлы (бинарные) используются для чтения и записи произвольно структурированных данных. Бинарные файлы это, строго говоря, не новый тип файлов, а новый способ управления файлами любого типа. Методы работы с бинарными файлами позволяют считывать и изменять любой байт файла.

Для работы с файлами данных используются команды открытия файла, закрытия файла, записи и чтения данных из файла, а также ряд функций, облегчающих работу с файлами. Все эти команды традиционны для всех версий языка Basic.

Открытие файлов

Для открытия файлов служит команда ***Open***.

```
Open “спецификация_файла” For { тип файла}[Access{доступ}]
    [Lock{блокировка}] As [#] N [Len=длина]
```

Опция "***Спецификация_файла***", как известно, позволяет указать диск, маршрут, имя и расширение имени файла. Например: R:/Prognoz/Ucheb/prognoz1.dan. Чтобы файл мог использоваться на компьютерах и с операционной системой MS DOS, имя файла и его расширение должны формироваться по правилам операционной системы MS DOS. То есть в имени файла и расширении имени файла используются только латинские символы и цифры, имя файла начинается с буквы, длина имени файла не должна превышать 8 символов, а расширение имени файла – четырех символов, включая точку. В имени файла не допускается использование точек и пробелов. Спецификация файла заключается в кавычки.

Опция ***For*** определяет тип файла. Тип файла указывает на его структуру и способ использования и может принимать следующие значения:

Input – файл последовательного доступа, открыт для чтения;

Output – файл последовательного доступа, открыт для записи;

Append - файл последовательного доступа, открыт для добавления данных;

Bynary – двоичный файл открыт для записи и чтения данных:

Random – файл прямого доступа открыт для записи и чтения данных.

Опция ***Access*** определяет права доступа к данным при работе в сетях ЭВМ. Она может иметь три значения:

Read – разрешено чтение данных из файла;

Write – разрешена запись данных в файл;

Read Write - разрешено чтение и запись данных. Этот режим доступа используется по умолчанию.

Опция ***Lock***. Так как режим чтения-записи, обычно, предназначен для работы с файлами, которые могут использоваться многими пользователями или приложениями, необходимо обеспечить целостность данных при коллективном использовании. Для этой цели используется параметр блокировка, который может принимать следующие значения:

Shared – файл может использоваться всеми процессами для считывания и записи данных;

LockRead – запрет чтения. Никакой другой процесс не может считывать данные из файла. Этот параметр можно установить, если в данный момент никакой другой процесс не выполняет операцию чтения.

LockWrite – запрет записи. Никакой другой процесс не может записывать данные в файл. Данный параметр можно установить, если в текущий момент никакой другой процесс не выполняет операцию записи.

LockReadWrite – запрет записи, чтения данных. Этот параметр можно установить, если в данный момент никакой другой процесс не выполняет операцию записи, чтения.

Опция ***As*** – определяет номер канала. Знак # можно опустить. Номер канала может принимать значения от 1 до 255. Число одновременно открытых ка-

налов определяется ограничениями операционной системы, указанными в файле Config.sys.

Так как пользователь при написании программы в принципе не может знать, сколько каналов занято и каков номер свободного канала, то для определения номера свободного канала следует использовать функцию ***FreeFile***. Функция FreeFile возвращает номер свободного канала.

Опция ***Len*** – используется только в файлах прямого доступа. Она устанавливает длину записи в байтах.

При *открытии* или, иными словами, *инициализации* файлов выполняются следующие операции:

- устанавливается связь между спецификацией файла и его программным номером. Поэтому во всех последующих операциях с данным файлом дается ссылка на номер канала, а не на спецификацию файла;
- закрепляется системный или программный буфер, используемый для реализации операторов ввода-вывода. Использование буфера уменьшает число обращений программы к диску, а следовательно, повышается скорость записи-чтения данных;
- формируются начальные значения параметров, расположенных в так называемом блоке управления файлом.

Закрытие файлов

Для закрытия файлов используется команда ***Close***. Синтаксис команды:

Close [# <номер канала>]

Команда Close с параметром номера канала закрывает указанный канал. Команда Close без параметров закрывает все открытые файлы. Команда Close очищает буфер и дает указание операционной системе обновить таблицу размещения файлов [FAT]. Но этого может не произойти из-за собственных методов буферизации Windows. По этой причине внезапная потеря напряжения в то время, когда файл открыт, почти неизбежно ведет к потере информации и иногда даже повреждает диск.

С целью надежного сохранения информации рекомендуется использовать вместо команды Close команду **Reset**. Эта команда, в отличие от команды Close, дает указание операционной системе сбросить содержимое буфера на диск.

*С целью надежного сохранения информации на диске рекомендуется использовать вместо команды Close команду **Reset.***

Команды записи данных в файл и чтения информации из файлов данных зависят от типа файла.

6.1.2. Файлы последовательного доступа

Файлы последовательного доступа отличаются не только простотой организации данных, но и простотой управления ими.

Файл последовательного доступа используется, обычно, для работы с текстовой информацией, хотя ничто не мешает использовать их для работы с числами.

Работа с файлами последовательного доступа состоит из двух самостоятельных операций: создания файла и использование файла.

Создание файла последовательного доступа

Создание файла последовательного доступа можно представить следующей схемой:

```
Открытие файла ‘ (команда Open или Append с опцией Output)
    Запись данных в файл.
Закрытие файла ‘ (команда Close)
```

При необходимости, файл последовательного доступа может быть создан или отредактирован любым текстовым редактором.

Использование файла последовательного доступа

При ***использовании файла*** последовательного доступа также реализуется простая схема:

```
Открытие файла ‘ (команда Open с опцией Input)
    Чтение данных из файла.
Закрытие файла ‘ (команда Close)
```

Запись данных в файл последовательного доступа

Для записи данных в файл последовательного доступа используются команды **Print # и Write #.**

Формат команды Print # полностью соответствует команде Print, используемой для вывода данных в форму. Числовые данные, записываемые в файл, в этом случае необходимо преобразовывать в строку символов, особенно это касается вещественных чисел, так как десятичную точку программа воспринимает как разделитель данных.

Синтаксис оператора Write # такой же, как и у оператора Print #, но если оператор Print # сохраняет данные в виде обычного текста, то оператор Write # заключает текстовые строки в кавычки, а цифры выводятся без кавычек:

```
Print #1, “Анна”,  “Минск”, 17 , 3.75
```

В файле будет: Анна Минск 17 3 75

```
Write #1, “Анна”,  “Минск”, 17 , 3.75
```

В файле будет: “Анна” , “Минск”, 17, 3.75

Поэтому при работе с числами предпочтительнее использовать оператор Write #.

Чтение данных из файла последовательного доступа

Чтение данных из файла последовательного доступа осуществляется операторами *Input #, Line Input #* и функцией *Input$.*

Оператор ***Line Input #*** считывает из файла строку данных. Разделителем данных в файле в этом случае должен быть символ возврата каретки (вводится

в строку текста автоматически при нажатии клавиши Enter). Строка данных не должна превышать 255 символов.

Оператор ***Input #*** имеет следующий синтаксис:

```
Input #, “текстовое сообщение”,<список переменных>
```

Переменные в списке разделяются запятыми.

Функция ***Input$*** служит для вывода из файла на экран указанного числа символов, не отображаемых на экране. Синтаксис функции Input$:

```
<символьная_переменная>= Input$ (n, #N),
```

где n – число символов, выделяемых из файла, # N – номер открытого канала файла последовательного доступа.

Пример 6.1. Создание файла последовательного доступа.

```
Open “R:Test.dan” For Output As #1
    A$ = “Минск – столица Республики Беларусь”
    B%=13875
    C!=7.58
    Print#1, A$, B%, Str$ (C!)
Close #1
```

Пример 6.2. Использование файла последовательного доступа

```
Open “R:Test.dan” For Input As #1
    Input Line A$
    Print A$
Close
```

На форме будет следующая строка:

```
Минск – столица Республики Беларусь, 13875, 7.58
```

Здесь 13875 и 7.58 текст.

```
Open “R:Test.dan” For Input As #1
   Input #1, A$, B%, C$
   Print A$, B%, Val (C8)
Close #1
```

На форме будет строка следующего вида:

```
Минск – столица Республики Беларусь 13875  7.58
```

В данном примере 13875 и 7.58 – числа

Оператор Input # целесообразно использовать в сочетании с оператором Write #.

6.1.3. Создание базы данных с использованием файла последовательного доступа

Базы данных предназначены для хранения структурированных данных. На экране монитора и на бумаге базу данных можно представить в виде двухмерной таблицы. Каждая строка такой таблицы представляет собой запись. Первая строка таблицы – строка заголовков. Каждая запись состоит из полей. Каждое поле имеет имя. Поля в таблице образуют колонки. В каждой колонке хранятся данные одного типа.

Под ***структурой базы*** данных понимают состав полей, их имена, типы, размеры в символах.

Пример 6.3.

В качестве примера создадим базу данных для учета успеваемости студентов. База данных должна содержать поля Номер по порядку, Фамилия и инициалы, оценки по предметам обучения (Физика, Математика, Информатика) и Средний балл успеваемости за сессию. База данных должна обеспечивать ввод данных с их визуализацией, сохранение данных на диске, чтение данных с диска и вывод результатов на печать.

Порядок работы.

1. Изобразим структуру базы данных (рис 6.1):

Для хранения базы данных в ОЗУ будем использовать двухмерный массив BD(n,5). Где *n* – число записей в базе данных, а 5 – число полей. Номер записи нужен только на экране или на бумаге, в программе хранить его не требуется. Для отображения базы данных на экране воспользуемся сеткой MSFlexGrid. Для ввода данных создадим линейку из массива элементов управления.

N	Фамилия и инициалы	Оценки			Средний балл
		Физика	Математика	Информатика	

Рис. 6.1. Структура базы данных

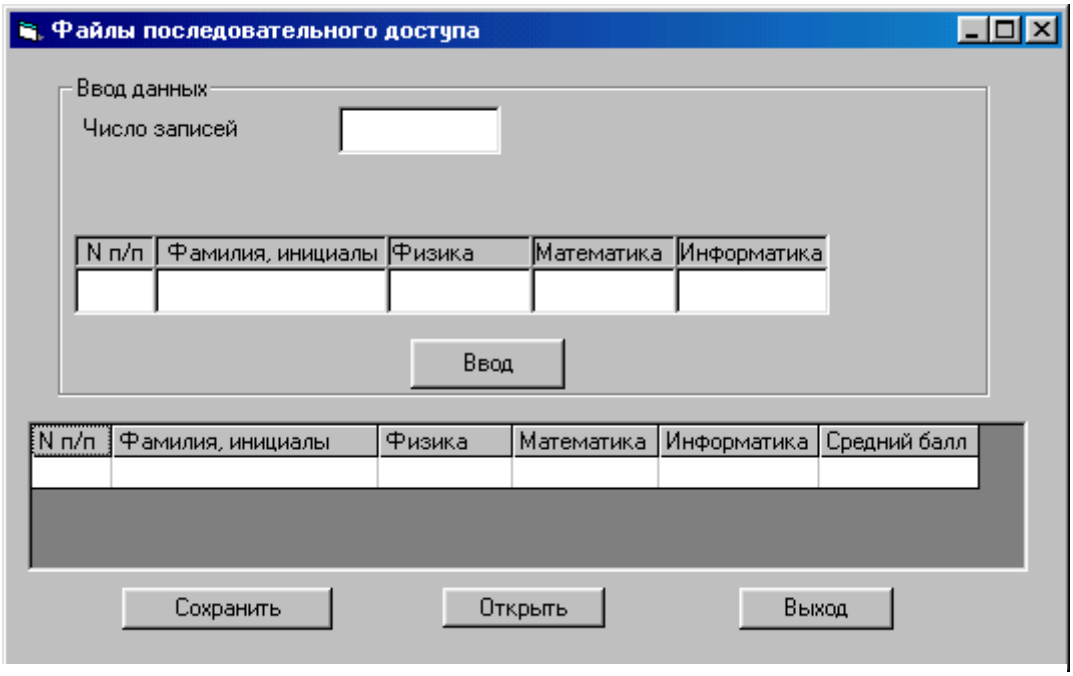


Рис. 6.2. Форма базы переданных “Успеваемость”

2. Разработаем эскиз формы базы данных “Успеваемость” (рис. 6.2).
3. Опишем состав элементов управления на форме (табл. 6.1).

4. Опишем переменные, используемые в программе (табл.6.2)
5. Напишем текст программы.

Объявление переменных уровня формы.

```
Dim i As Integer, j As Integer, bd() As String
Dim n As Integer, Sb As Single, nKanal As Integer
```

Установка начальных параметров при загрузке формы.

```
Private Sub Form_Load()
    Me.Height = 5000
    Me.Width = 8000
    For i = 0 To 4
        Grid1.Row = 0
        Grid1.Col = i
```

Тип	*Имя*	*Назначение*
Label	lblLabel1 lblLabel2()	текст “Число записей” массив элементов управления. Текст – заголовки шапки таблицы ввода данных
TextBox	txtText1 txtText2( )	ввод числа записей массив элементов управления. Поля для ввода данных
MSFlexGrid	Grid1	таблица для вывода результатов
Command	cmdVvod cmdSave cmdOpen cmdExit	ввод данных в массив сохранение данных чтение данных с диска (открытие файла) выход

```
        Grid1.Text = Label2(i).Caption
        Grid1.Visible = True
    Next i
    Grid1.Col = 5
    Grid1.Text = "Средний балл"
    Grid1.Visible = True
    Grid1.Row = 0
    Grid1.Col = 0
    Grid1.ColWidth(0) = 600
    Grid1.ColWidth(1) = 2000
    Grid1.ColWidth(2) = 1000
    Grid1.ColWidth(3) = 1100
    Grid1.ColWidth(4) = 1200
    Grid1.ColWidth(5) = 1200
End Sub
```

Таблица 6.1.

Описание элементов управления

Таблица 6.2.

Описание переменных

Имя переменной	*Тип переменной*	*Видимость переменной*	*Комментарий*

i, j, k	Integer	Локальная	Используются в качестве переменных цикла
n	Integer	Глобальная	Число студентов в группе
Sb	Single	Глобальная	Средний балл
bd(5,n)	Single	Глобальная	Массив для хранения данных
nKanal	Integer	Глобальная	Имя файла

Первой операцией при вводе данных необходимо указать число записей. Текст программы запишем в обработчик события *Change* элемента управления Text1:

```
Private Sub txtText1_Change()
    n = Val(txtText1.Text)
    Grid1.Rows = n + 1
End Sub
```

Программы ввода данных, сохранения и чтения данных с диска запишем в обработчики событий соответствующих кнопок.

Процедура ввода данных.

```
Private Sub cmdVvod_Click()
    Dim i As Integer, j As Integer
    If n = 0 Then
        MsgBox "Укажите число записей"
        Exit Sub
    End If
    ReDim Preserve bd(5, n) As String
    If n < Val(txtText2(0).Text) Then
        n = Val(txtText2(0).Text)
        Grid1.Rows = n + 1
        ReDim Preserve bd(5, n) As String
    End If
    i = Val(txtText2(0).Text)
    Grid1.Row = i:
    For j = 0 To 4
        bd(j, i) = txtText2(j).Text
        Grid1.Col = j
        Grid1.Text = bd(j, i)
        Grid1.Visible = True
    Next j
  Sb = (Val(txtText2(2).Text) + Val(txtText2(3).Text) + Val(txtText2(4).Text)) / 3
    Grid1.Col = j
    Grid1.Text = Str$(Format(Sb, "#.##"))
    Grid1.Visible = True
    bd(5, i) = Str$(Sb)
    For j = 1 To 4
      txtText2(j).Text = ""
    Next j
    If i < n Then txtText2(0).Text = i + 1
End Sub
```

Процедура сохранения данных на диске

```
Private Sub cmdSave_Click()
```

```
        nKanal = FreeFile
        Open "r:\Laborat\VisualBasic\file.dan" For Output As #nKanal
        Write #nKanal, n
        For i = 1 To n
          For j = 0 To 5
            Write #nKanal, Format(bd(j, i), "#.##")
          Next j
        Next i
        Close #nKanal
    End Sub
```

Процедура чтения данных (открытия файла)

```
    Private Sub cmdOpen_Click()
      nKanal = FreeFile
      Open "r:\Laborat\VisualBasic\file.dan" For Input As #nKanal
        Input #nKanal, n
        ReDim bd(5, n)
        Grid1.Rows = n + 1
        For i = 1 To n
          Grid1.Row = i:
          For j = 0 To 5
            Input #nKanal, bd(j, i)
            Grid1.Col = j
            Grid1.Text = bd(j, i)
            Grid1.Visible = True
          Next j
        Next i
      Close #nKanal
    End Sub
```

Процедура завершения работы с программой

```
    Private Sub cmdExit_Click()
      Unload Me
    End Sub
```

6.1.4.Упражнение: создание базы данных

Задача 1. Разработайте программу для сохранения и загрузки исходных данных и вывода на печать исходных данных и результатов для задачи 2 задания 5.2.6.

Порядок работы.

При табулировании функции двух переменных исходные данные содержат шесть значений Хнач, Хкон, dX, Yнач, Yкон, dY. При открытии файла данных эти значения должны быть считаны и присвоены текстовым полям.

- Откройте пункт меню Файл\Открыть и запишите текст программы:

```
Private Sub mnuOpen_Click()
  Dim Z1 As String
  Dim i As Integer
    cldDialog1.ShowOpen
    Nkan = FreeFile  'Nkan - номер свободного канала для обмена данных
    NameFile = cldDialog1.FileName
```

```
        NameFile1 = NameFile   'NameFile1 хранит имя  открытого файла
        If NameFile = "" Then Exit Sub
        Open NameFile For Input As #Nkan
        For i = 0 To 5
            Input #Nkan, Z1
            Text1(i).Text = Z1
        Next i
        Close #Nkan
    End Sub
```

- Откройте пункт меню Файл\Сохранить и запишите текст программы:

```
    Private Sub mnuSave_Click()
        Dim i As Integer, Nkan As Integer
        NameFile = NameFile1
        Nkan = FreeFile
        If NameFile = "" Then
            cldDialog1.ShowSave
            NameFile = cldDialog1.FileName
        End If
        Open NameFile For Output As #Nkan
        For i = 0 To 5
            Write #Nkan, Text1(i).Text
        Next i
        Close #Nkan
    End Sub
```

- Откройте пункт меню Файл\Сохранить как ... и запишите текст программы:

```
    Private Sub mnuSaveAs_Click()
        Dim i As Integer, Nkan As Integer
        NameFile = NameFile1
        Nkan = FreeFile
        cldDialog1.ShowSave
        NameFile = cldDialog1.FileName
        Open NameFile For Output As #Nkan
        For i = 0 To 5
            Write #Nkan, Text1(i).Text
        Next i
        Close #Nkan
    End Sub
```

- Откройте пункт меню Файл\Печать и запишите текст программы:

```
    Private Sub mnuPrint_Click()
        Dim i As Integer
        Dim j As Integer
        cldDialog1.ShowPrinter

        Printer.Font.Name = "Times New Roman"
        Printer.Font.Size = 16
        Printer.Font.Bold = True
```

```
        CurrentX = 20: CurrentY = 20
        Printer.Print Tab(25); "Исходные данные"
        Printer.Print ""
        Printer.Font.Size = 12
        Printer.Font.Bold = False
        For i = 0 To N
          For j = 0 To M
            Printer.Print Z(i, j);
          Next j
          Printer.Print
        Next i

    Printer.NewPage
    Printer.EndDoc
End Sub
```

Задача 2. Создайте базу данных “Успеваемость” (раздел 6.1.3.)

Задача 3. Добавьте на форму рис.6.2. меню для организации связи с внешними устройствами (см. задача 1 данного раздела, задача 2 раздела 5.2.6)

6.1.5. Закрепление материала

1. Какие типы файлов данных Вам известны и чем они отличаются?
2. Приведите синтаксис команды Open.
3. Приведите синтаксис команды Close.
4. Какие команды используются для записи данных в файл последовательного доступа?
5. Какие команды используются для чтения данных из файла последовательного доступа?
6. Какая последовательность команд необходима для создания файла последовательного доступа?
7. Какая последовательность команд необходима для использования файла последовательного доступа?
8. Расскажите алгоритм разработки базы данных на основе файла последовательного доступа.

Задание для самостоятельной работы

Разработать и отладить базу данных “Успеваемость” с использованием файла последовательного доступа.

6.2. Файлы прямого доступа

6.2.1. Создание файлов прямого доступа

В файле прямого доступа данные организованы в виде записей фиксированной длины. Каждая запись состоит из полей. Для каждого поля указывается имя и размер в байтах. Поэтому для символьных полей тип поля должен быть указан с фиксированной длиной, например, Familij As String*20. Длина числовых полей определяется по типу данных.

Файлы прямого доступа обеспечивают доступ к каждой записи файла по его номеру. Если номер записи не используется, то считывание данных производится последовательно начиная с первой записи. В этом режиме файл прямого доступа превращается в файл с последовательным доступом.

Открытие файла прямого доступа.

```
Open “спецификация_файла” For Random  [Access доступ] [Lock блокировка]
        As[ # ] N  [ Len = длина]
```

Определение длины записи довольно трудоемкая задача. Однако эту задачу можно облегчить, если использовать функцию ***Len (переменная).***

Запись данных в файл прямого доступа осуществляется командой

```
Put # НомерФайла, НомерЗаписи, Переменная.
```

Для чтения данных из файла служит оператор Get:

```
Get # НомерФайла, НомерЗаписи, Переменная.
```

Для чтения и записи данных в файл прямого доступа используется переменная ***пользовательского типа***. Объявление переменной пользовательского типа осуществляется оператором ***Type / End Type***.

Пример 6.4. Объявим пользовательскую переменную ***Каталог***.

```
Type Каталог
        Фамилия_Автора As String * 20
        Соавтор  1   As String * 20
        Соавтор  2   As String * 20
        Наименование As String * 100
        Издательство As String * 15
        ГодИздания As Integer
As Integer
End Type
```

А теперь объявим переменную ***Библиотека***.

```
Dim  Библиотека As Каталог
```

Обращение к полям переменной осуществляется также как и к свойствам элементов управления.

```
Библиотека. Фамилия_Автора = “Чингиз Айтматов”
Библиотека. Наименование = “Буранный полустанок”
Библиотека. Издательство = “М.: Нева”
Библиотека. ГодИздания = 1975
```

Запись данных в файл

```
Private Sub mnuSave_Click ( )
      Put # 1, 1, Библиотека
 End Sub
```

Чтение данных из файла

```
Private Sub mnuOpen_Click ( )
    Dim Автор As String, Наименование As String,
    Dim Издательство As String, ГодИздания As Integer
    Get #1, 1, Библиотека
    Автор = Библиотека. Фамилия_Автора
    Наименование = Библиотека. Наименование
    Издательство = Библиотека. Издательство
    ГодИздания  = Библиотека. ГодИздания
End Sub
```

6.2.2. Команды и функции для работы с файлами

Команды для работы с файлами

Visual Basic 6 имеет значительное число команд для работы с файлами, которые непосредственно взаимодействуют с операционной системой на низком уровне. Эти команды имитируют команды ОС для работы с файлами и накопителями на компьютере:

MkDir – создание каталога: MkDir “C:\PROBA”

Kill - удаление одного или нескольких файлов: Kill “ *.bak “

Name - переименование, а также перемещение файлов:

Перемещение файла:

```
Name “C:\VB\TEST.BAS” As “C: \ ARCHIV\ TEST.BAS”
```

Переименование файла:

```
Name “C:\VB\TEST.BAS” As “TEST.BAK”
```

RmDir – удаление каталога файла: RmDir “C:\PROBA”

ChDir – замена текущего каталога: ChDir <маршрут>

ChDrive – замена текущего диска: ChDrive <диск>

Функции для работы с файлами

Dir$(Шаблон [, Атрибуты]) - поиск и получение списка файлов.

```
If Dir$(“C:\VYFILE.TXT”) = “” Then
    MsgBox “Файл не найден”
End If
```

CurDir$(Drive) – возвращает строку, содержащую текущий каталог. Первый символ параметра Drive возвращает текущий путь для заданного диска.

FileCopy – копирует файл из одного места в другое:

```
FileCopy исходный_файл, место_назначения.
```

FileDataTime – возвращает дату и время создания файла или его последней модификации:

```
FileDataTime (маршрут)
```

GetAttr – возвращает атрибуты файла:

```
GetAttr (маршрут) As Integer
```

0 - обычный;	8 - метка тома;
1 - только для чтения;	16 - каталог;
2 - скрытый;	32 - архивный.
4 - системный;	

SetAttr – устанавливает атрибуты файла

```
SetAttr  маршрут, атрибут
```

Shell(путь, стиль_окна) – используется для запуска любого файла в форматах .com, .exe, .bat или .pif:

```
X= Shell "C:\ WINDOWS\ COMMAND \ Format.com A:"
```

Когда VB 6 запускает программу, он создает новое окно и передает ему фокус ввода. Параметр "стиль окна" может принимать 6 значений:

0 - окно скрыто, но имеет фокус;
1 - обычное окно, с фокусом;
2 - окно, представленное значком, с фокусом;
3 - развернутое окно, с фокусом;
4 - обычное окно, без фокуса;
6 - окно, представленное значком, без фокуса.

6.2.3. Упражнение: создание базы данных с использованием файла прямого доступа

Создания базы данных с использованием файла прямого доступа рассмотрим на примере задачи, рассмотренной в разделе 6.1.3.

1. Создадим модуль и в разделе Главная объявим переменную пользовательского типа Ekzamen:

```
Type Ekzamen
    Familij As String * 20
    Fizika As Integer
    Matematika As Integer
    Informatika  As Integer
    SBall As Single
End Type
```

2. Объявим там же переменную Usp – успеваемость как переменную типа Ekzamen:

```
Dim Usp As Ekzamen
```

Добавим на форму две кнопки: *Сохранить запись* и *Открыть запись* (рис. 6.3.).

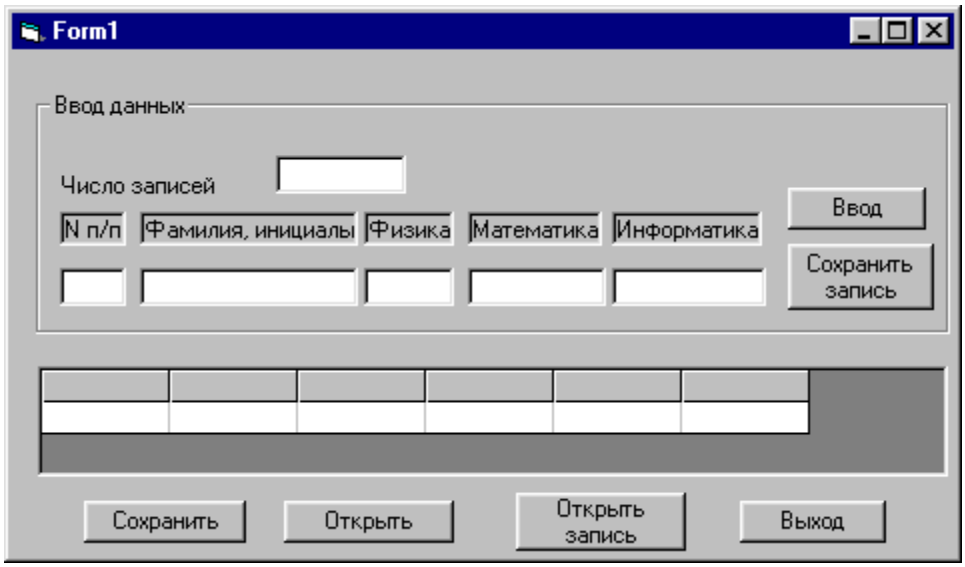


Рис. 6.3. База данных успеваемость

Запишем текст программы.

```
‘ Запись данных в файл
    Private Sub cmdSave1_Click()
        Dim i As Integer
        nKanal = FreeFile
        Open "r:\Laborat\VisualBasic\file1.dan" For Random As #nKanal
         ‘ длина записи принимается по умолчанию
        i = Val(txtText2(0).Text) - 1
        Usp.Familij = bd(1, i)
        Usp.Fizika = Val(bd(2, i))
        Usp.Matematika = Val(bd(3, i))
        Usp.Informatika = Val(bd(4, i))
        Usp.SBall = Val(bd(5, i))
        Put #nKanal, i, Usp
    End Sub

 ‘ Чтение данных из файла
    Private Sub mnuOpen1_Click ( )
        nKanal = FreeFile
        Open "r:\Laborat\VisualBasic\file1.dan" For Random As #nKanal
        i = Val(InputBox("Укажите номер записи"))
        If n = 0 Then
            MsgBox "укажите число записей"
             Exit Sub
        End If
        ReDim bd(5, n)
        Get #nKanal, i, Usp
        bd(0, i) = Str(i)
        bd(1, i) = Usp.Familij
        bd(2, i) = Str(Usp.Fizika)
        bd(3, i) = Str(Usp.Matematika)
```

```
        bd(4, i) = Str(Usp.Informatika)
        bd(5, i) = Str(Usp.SBall)
            Grid1.Rows = 2:
            Grid1.Row = 1:
            For j = 0 To 5
                Grid1.Col = j
                Grid1.Text = bd(j, i)
                Print bd(j, i)
                Grid1.Visible = True
            Next j
         Close #nKanal
    End Sub
```

Тексты программ для кнопок *Сохранить* и *Открыть* будут иметь следующий вид:

```
    Private Sub cmdSave_Click()
        ‘ Сохранение всех данных
        nKanal = FreeFile
        Open "r:\Laborat\VisualBasic\file1.dan" For Random As #nKanal
        For i = 1 To n
            Usp.Familij = bd(1, i)
            Usp.Fizika = Val(bd(2, i))
            Usp.Matematika = Val(bd(3, i))
            Usp.Informatika = Val(bd(4, i))
            Usp.SBall = Val(bd(5, i))
            Put #nKanal, i, Usp
        Next i
        Close #nKanal
    End Sub

    Private Sub mnuOpen1_Click ( )
        ‘ Открыть весь файл
        nKanal = FreeFile
        Open "r:\Laborat\VisualBasic\file1.dan" For Random As #nKanal
        If n = 0 Then
            MsgBox "укажите число записей"
            Exit Sub
        End If
            Grid1.Rows = n + 1:
            ReDim bd(5, n)
        For i = 1 To n
            Get #nKanal, i, Usp
            bd(0, i) = Str(i)
            bd(1, i) = Usp.Familij
            bd(2, i) = Str(Usp.Fizika)
            bd(3, i) = Str(Usp.Matematika)
            bd(4, i) = Str(Usp.Informatika)
            bd(5, i) = Str(Usp.SBall)
            Grid1.Row = i:
            For j = 0 To 5
                Grid1.Col = j
                Grid1.Text = bd(j, i)
```

```
                'Print bd(j, i)
                Grid1.Visible = True
            Next j
        Next i
        Close #nKanal
    End Sub
```

6.2.4. Закрепление материала

1. В чем состоит отличие файла прямого доступа от файлов последовательного доступа?
2. Приведите синтаксис команды для открытия файла прямого доступа.
3. Как записать данные в файл прямого доступа?
4. Как прочитать данные из файла прямого доступа?
5. Как объявить переменную пользовательского типа?
6. Назовите команды для работы с файлами?
7. Перечислите функции, используемые для работы с файлами.

Задание для самостоятельной работы

Разработайте форму и создайте базу данных для учета успеваемости студентов с использованием файла прямого доступа.

7. Проверка и обработка пользовательского ввода. Обработка ошибок

7.1. Проверка и обработка пользовательского ввода

7.1.1. Контроль ввода

При вводе числовых данных, дат и времени целесообразно обеспечить контроль ввода данных, чтобы избежать получения некорректных результатов или появления неустранимых ошибок выполнения программы, приводящих к прерыванию выполнения программы.

Существует два способа контроля вводимых данных:

- на уровне формы - данные проверяются после того, как заполнены все поля на форме;
- на уровне полей - данные проверяются после заполнения каждого поля.

При контроле данных на уровне полей рекомендуется сообщать пользователю об ошибке звуковым сигналом и текстовой информацией в строке состояния или окне MsgBox.

```
If Not IsNumeric(txtZipCode.Text) Then
    Beep      ′ звуковой сигнал
    StatusBar1. Panels("ErrorDescription").Text= " код  _
                должен быть числовой "
End If.
```

При контроле на уровне формы на экран желательно выводить форму с сообщением о всех обнаруженных ошибках.

После обнаружения ошибки фокус должен быть установлен на поле, содержащем ошибку. Если несколько полей формы содержит ошибку, то фокус надо установить на первое поле по порядку ввода данных (меньшее значение свойства TabIndex).

Установку фокуса можно выполнить программно с помощью метода SetFocus:

```
txtZipCode.SetFocus
```

Для контроля ввода на уровне полей может использоваться событие потери фокуса LostFocus:

```
Private Sub txtZipCode_LostFocus ( )
    If Not IsNumeric (txtZipCode.Text) Then
        MsgBox  " код должен быть числовой "
        txtZipCode.SetFocus
     End If.
End Sub.
```

Недостатком этого метода может быть опасность войти в бесконечный цикл.

7.1.2. Реализация проверки данных на уровне формы

Обработчик клавиатуры на уровне формы

Обработчик клавиатуры - это более совершенный вариант проверки данных на уровне формы. При наличии единого обработчика клавиатуры можно управлять вводом данных в любые поля на форме.

Для реализации обработчика необходимо использовать события, связанные с клавиатурой. Можно задействовать три основных события: *KeyPress, KeyDown* и *KeyUp*. (См. 1.2.2).

Использование события KeyPress

Это событие возникает, когда пользователь нажимает клавишу, у которой имеется свой ASCII-код. К таким клавишам относятся все алфавитно-цифровые клавиши и ряд управляющих, например *Enter* (код 13), *Backspace* (код 8). Собы-

тие KeyPress не генерируется при нажатии функциональных и других специальных клавиш.

Событие KeyPress удобно для перехвата клавиш, нажимаемых при вводе в поля TextBox и ComboBox. Событие KeyPress предоставляет аргумент KeyASCII, в котором содержится код нажатой клавиши. Если нужно запретить ввод любых данных, кроме числовых, можно проверять этот аргумент в обработчике события KeyPress.

Использование события KeyDown и KeyUp

Обработчики этих событий могут использоваться для анализа нажатия функциональных клавиш, навигационных клавиш, клавиш редактирования. Коды этих клавиш не относятся к стандартным ASCII-кодам. Эти события перехватывают нажатие комбинации клавиш, вроде Ctrl + Del, Shift + Ctrl + Del и т.д.

Использование события KeyPreview

Если свойство формы KeyPreview установлено в True, то все события, связанные с клавиатурой будут анализироваться сначала в обработчиках событий формы и лишь потом в обработчиках событий элементов управления, расположенных на данной форме. Это свойство позволяет проанализировать код нажатой клавиши прежде, чем включится в работу обработчик события активного элемента формы.

```
Private Sub Form1_Load()
   ′события формы будут генерироваться до событий
   ‘элементов управления.
    KeyPreview=True
End Sub
Private Sub Form_KeyDown (KeyCode As Integer, Shift As Integer)
    Select Case KeyCode  ′сообщаем какая клавиша нажата
        Case vbKeyF1
            MsgBox "нажата клавиша F1"
            [ операторы ]
        Case vbKeyF2
            MsgBox "нажата клавиша F2"
            [ операторы ]
            …
        Case  Else
     End Select
End Sub
```

VB имеет класс элементов управления ***Control***, к которому относятся все элементы управления на форме. Тип элемента управления анализируется с помощью функции ***TypeOf***. Данный класс элементов управления в сочетании с функцией TypeOf можно успешно использовать для контроля правильности ввода данных на уровне формы.

Пример 7.1. Отключение элемента управления (кнопка cmdOK), пока не будут заполнены все поля ввода.

```
Private Sub Form_KeyUp(KeyCode As Integer, Shift As Integer)
   Dim ctl As Control
   For Each ctl In Controls
       If TypeOf ctl Is TextBox Then
           If ctl.Text="" Then
               cmdOK.Enabled=False
               Exit Sub
            End If
        End If
   Next ctl
       cmdOk. Enabled = True
End Sub
```

В примере 7.1. с помощью цикла For Each просматриваются все элементы управления формы и проверяется состояние текстовых полей. Если хотя бы одно поле TextBox пусто, то свойству Enebled кнопки cmdOk присваивается значение False. Когда все поля TextBox будут заполнены, то свойству Enabled кнопки cmdOk присваивается значение True и она становится доступной.

7.1.3. Реализация проверки данных на уровне полей формы

Конроль на уровне полей форм позволяет обнаруживать и устранять ошибки в момент ввода. Для этой цели можно использовать и некоторые свойства элемента TextBox: MaxLength; PassWordChar, Locked.

Свойство ***MaxLength*** задает максимальное количество символов, вводимых в текстовое поле. Когда пользователь пытается ввести в поле больше символов, чем указано в свойстве MaxLength, система подает звуковой сигнал.

Свойство ***PassWordChar*** позволяет скрывать вводимые символы, заменяя их звездочками. В качестве маскирующего можно использовать любой символ, но чаще применяют именно звездочки. Такой прием часто используется для скрытия паролей в диалоговых окнах регистрации, а также для ограничения доступа.

Свойство ***Locked*** запрещает редактирование поля ввода вводом данных с клавиатуры, но оставляет возможность программного изменения свойства Text.

Использование событий клавиатуры для контроля на уровне полей

Событие ***KeyPress*** распознает регистр нажатой клавиши, поэтому при контроле вводимых символов их целесообразно приводить к одному регистру и лишь потом анализировать.

```
Sub txtText1_KeyPress(KeyAscii As Integer)
     KeyAscii=Asc(VCase(Chr(KeyAscii)))
```

```
End Sub
```

Функция *Chr* преобразует код нажатой клавиши в символ, функция *VCase* переводит данный символ в верхний регистр, а функция *Asc* снова преобразует символ в числовой код.

Например, нажата клавиша с символом “a” - код 97. Код 97 преобразуется в символ “a” функцией Chr, функция VCase преобразует символ ”a” в символ “A”, функция Asc возвращает код 65.

События ***KeyUp*** и ***KeyDown*** распознают статус клавиатуры: первое событие генерируется в момент нажатия клавиши, а второе в момент отпускания. Кроме того имеется возможность отслеживать нажатие функциональных клавиш *Shift*, *Ctrl*, *Alt* одновременно с нажатием символьной клавиши, например Ctrl + C. Факт нажатия и отпускания клавиши возвращается через параметр *KeyCode*, а нажатие функциональной клавиши – через параметр *Shift*.

Пример 7.2. Контроль нажатой клавиши

```
Private Sub txtText1_KeyDown(KeyCode As Integer, Shift As Integer)
    Select Case Shift
        Case 1
            Print “Нажата клавиша Shift”
        Case 2
            Print “Нажата клавиша Ctrl”
        Case 3
            Print “Нажаты клавиши Shift и Ctrl”
        Case 4
            Print “Нажата клавиша Alt”
        Case 5
            Print “Нажаты клавиши Shift и Alt”
        Case 6
            Print “Нажаты клавиши Ctrl и Alt”
        Case 7
            Print “Нажата клавиша Ctrl, Shift  и Alt”
     End Select
End Sub.
```

Следующий код контролирует нажатие клавиш “A” и Shift:

```
Private Sub txtText1_KeyDown(KeyCode As Integer, Shift As Integer)
    If KeyCode=65  And Shift=1 Then
          Print “Нажаты клавиши A и Shift”
    End If
End Sub.
```

Проверка данных с использованием события Change

При небольшом числе полей ввода для контроля ввода данных можно использовать событие *Change* элемента управления TextBox.

Например, если форма имеет два поля ввода txtText1 и txtText2, то код контроля будет иметь следующий вид:

```
Private Sub cmd OKEnabled ( )
   If txtText1.Text < > “ “ And txtText2.Text < > “ “ Then
```

```
            cmdOK.Enabled=True
        Else
            cmdOK.Enabled=False
        End If
    End Sub.
    Private Sub txtText1_Change ( )
        Call cmdOKEnabled
    End Sub
    Private Sub txtText2_Change ( )
        Call cmdOKEnabled
    End Sub
```

В данном примере процедура контроля cmdOKEnabled вызывается в обработчике события Change каждого поля ввода. Если хотя бы одно из полей ввода пустое, кнопка ОК остается недоступной.

Функции проверки данных

Visual Basic 6 имеет несколько функций, позволяющих проверять типы данных: *IsNumeric*, *IsDate* и другие.

Функция IsNumeric возвращает True, если аргумент является числовым и False в ином случае. Функция IsDate возвращает True, если в аргументе содержится допустимая дата. Эти функции можно применять совместно с функцией InputBox:

```
StrValue=InputBox(“Введите число”)
    If IsNumeric (strValue) Then
        Print “Введено число”
    Else “Не корректный ввод данных”
End If
```

Пример 7.3. Проверка ввода числовых данных на уровне полей формы.

```
Public Sub txtTextNumeral(KeyAscii As Integer)
   Select Case KeyAscii
     Case Asc("0") To Asc("9"), Asc("."), 8
          ‘ если нажаты клавиши от 0 до 9, точка или клавиша BackSpace
          Numeral = Numeral + Chr(KeyAscii)
     Case Else
          ‘ если иначе, то код KeyAscii сбрасывается и никаких
          ‘  изменений не происходит
          KeyAscii = 0
          Beep
     End Select
End Sub
```

Событие Validate.

Перед тем, как элемент управления потеряет фокус ввода, в нем возникает событие Validate – но только если свойство CausesValidation элемента управления, который вскоре получит фокус ввода, установлено как True. Событие Validation в сочетании со свойством CausesValidation позволяет проверять дан-

ные, когда пользователь пытается переключиться на другой элемент управления. Это событие предоставляет аргумент Cancel: установив его, как True, можно запретить переключение на другой элемент.

7.1.4. Упражнение: Контроль ввода

1. Напишите процедуру для контроля правильности ввода данных на уровне формы для формы вычисления площади поверхности геометрических фигур.
2. Напишите процедуру для контроля правильности ввода данных на уровне полей ввода для программы табулирования функции одной переменной.

7.1.5. Закрепление материала

1. Какие способы контроля ввода данных Вам известны?
2. Чем отличаются известные Вам способы контроля ввода данных?
3. Как реализовать проверку правильности ввода данных на уровне формы?
4. Для какой цели может использоваться событие KeyPress?
5. Для какой цели могут использоваться события KeyUp и KeyDown?
6. Для чего используется событие KeyPreview?
7. Как обеспечить контроль ввода данных на уровне полей ввода?
8. Как обеспечить использование события Change для контроля ввода данных?
9. Какие функции VB6 можно использовать для контроля правильности ввода данных?

7.2. Обработка ошибок

7.2.1. Общие сведения об обработке ошибок

При разработке программы неизбежно возникновение ошибок. Ошибки могут быть вызваны невнимательностью при написании текста программы или из-за логических ошибок. Такие ошибки устраняются при отладке программы и ее тестировании. Но при использовании программы все равно могут возникать ошибки либо по вине пользователя, например, неправильно указано имя файла, не вставлен диск в дисковод и др., а также ошибки, не предусмотренные разработчиком. Такие ошибки принято называть ***ошибками периода выполнения.***

В идеальном случае разработанная программа должна работать без ошибок, однако все возможные ситуации предусмотреть невозможно. В VB предусмотрены стандартные средства обработки ошибок. При обнаружении ошибок

периода выполнения программа выводит на экран сообщение о коде ошибки и рекомендации по ее устранению. Однако, после нажатия кнопки ОК в окне сообщения программа завершает работу. При этом обработчик события *Unload* игнорируется и пользователь теряет все не сохраненные данные. Чтобы при таких ситуациях программа не завершилась аварийно, необходимо в программу включать *пользовательские обработчики ошибок*. Обработчик ошибок перехватывает сообщение об ошибках периода выполнения и передает управление специальной процедуре обработки ошибок, которая, как правило, предлагает пользователю в режиме диалога устранить причину, вызвавшую ошибку .

Для обработки ошибок в VB6 предусмотрены специальные средства: операторы *On Error* и *Resume,* а также объект *Err*.

Оператор **On Error** активизирует режим обработки ошибок. Это означает, что при возникновении ошибки периода выполнения после ввода в программу этой строки выполняется предусмотренная в программе процедура обработки ошибки. Выполнение программы не прерывается и стандартное сообщение об ошибке не выводится.

Синтаксис оператора:

```
On Error {[GoTo <метка> \ Resume Next \ GoTo  0}
```

Опция ***GoTo <метка>*** – передает управление на метку, <метка> должна находиться в той же процедуре, что и оператор On Error. Если в качестве метки указан 0, то предусмотренная разработчиком процедура обработки ошибок отключается и включается стандартный механизм обработки ошибок.

Опция ***Resume Next*** позволяет продолжить выполнение программы со следующего оператора, то есть ошибка в этом случае игнорируется.

Например:

On Error GoTo metka1 – управление передается на процедуру обработки ошибки по метке metka1;

On Error GoTo 0 - процедура обработки ошибки отключается;

On Error Resume Next – выполнение программы начинается со следующего оператора. Ошибка периода выполнения игнорируется.

Оператор **Resume [{Label\Next}]** – позволяет продолжить выполнение программы. Если параметры опущены, то управление передается на строку, вызвавшую ошибку, для ее повторного выполнения, опция ***Next*** возвращает управление на строку, следующую за строкой, вызвавшей ошибку, а опция ***Label*** передает управление на указанную метку. **Объект Err** содержит код ошибки и системное сообщение об ошибке, а также некоторую другую информацию.

Объект Err, как и все объекты, имеет несколько свойств и методов. Свойство ***Number*** возвращает код последней возникшей ошибки. Для совместимости с предыдущими версиями Visual Basic это свойство принимается по умолчанию, поэтому коды

```
ErrNum1=Err
ErrNum2=Err.Number
```

будут давать одинаковые результаты.

Свойство ***Description*** выводит сообщение об ошибке.

Для очистки объекта Err используется метод ***Clear***:

```
Err Clear
```

Если не очистить объект Err после возникновения ошибки, то последующие обработчики будут давать сообщения об одной и той же ошибке.

7.2.2. Реализация локального обработчика ошибок

Порядок обработки ошибок

Обработка ошибок производится в три этапа:

- подготовка перехвата;
- проверка и устранение ошибки;
- продолжение выполнения программы.

Структура процедуры с обработчиком ошибок приведена на рис. 7.1.

```
Private Sub Primer ( )
      ‘подготовка перехвата
      On Error GoTo metka1
          [ код программы]
      Exit Sub
metka1:       ' обработчик ошибок

          [ код обработчика ошибок]
      Resume
      ` конец процедуры обработки ошибки
End Sub
```

Рис. 7.1. Структура процедуры с обработчиком ошибок.

Первым шагом является расстановка “ловушек”. Для этого предназначен оператор On Error. Оператор On Error должен располагаться в начале процедуры. Он не выполняет никаких действий, а только передает управление обработчику ошибок по указанной метке при их возникновении.

Проверку кода ошибки и устранение ошибок обеспечивают обработчики ошибок.

Продолжение выполнения программы осуществляется с помощью оператора Resume.

Обработчик ошибок должен располагаться в конце процедуры. Перед обработчиком ошибок вставляется оператор Exit Sub – выход из процедуры. Это необходимо для того, чтобы при нормальном выполнении программы обработчик ошибок не выполнялся.

Реализация обработчика ошибок

Для реализации обработчика ошибок необходимо:

- определить код ошибки;

- определить действия, которые должны быть выполнены при соответствующей ошибке;

- обеспечить продолжение выполнения программы после устранения ошибки.

Рассмотрим пример локального обработчика ошибок при открытии файла.

Пример 7.4. Локальный обработчик ошибок.

```
Private Sub cmdVvod_(Click)
    On Error GoTo OpenError
    ‘ Вызов процедуры ввода данных
FileNameVvod:  ‘ метка возврата при выходе из обработчика ошибок
    Nkanal=FreeFile ( )
    Name=Input “Введите имя файла”
    Open Name For Input As Nkanal
    Exit Sub
OpenError:     ‘ метка процедуры обработчика ошибок
    Select Case Err.Number  `поверяется номер ошибки
        Case53  ` ошибка файл не найден
            MsgBoxErr. Deskription
            ‘ выводится текст системного описания ошибки
            MsgBox “Введите правильно имя файла”
        Case Else ` на случай появления непредвиденных ошибок
            MsgBox “Неустановленная ошибка:” & Err.Description
             Exit Sub
    End Select
    Resume FileNameVvod
End Sub.
```

Алгоритм работы программы при обработке ошибок приведен на рис. 7.2.

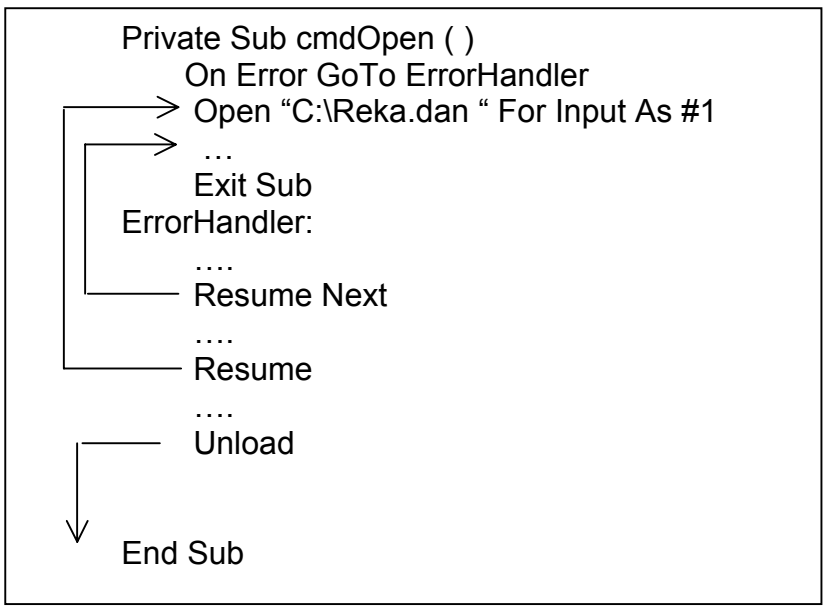


Рис. 7. 2. Передача управления оператором Resume

Пример реализации обработчика ошибок ErrorHandler:

```
Dim Answer As Integer
```

```
---------------------------------------------------------------------------------------------
Error Handler:
    Answer =MsgBox("ошибка выполнения",vbAbortRetryIgnore)
    'окно MsgBox содержит три кнопки: "Отказаться",  _
        "Повторить" и "Игнорировать"
Select Case   Answer
    Case vbAbort
        Unload Me  ' отказаться, выйти из программы с сохранением  _
                   данных
    Case vbRetry
        Resume  ' повторить операцию
    Case vbIgnore
        Resume  Next   ' игнорировать
    End Select
```

7.2.3. Централизованная обработка ошибок

Для обеспечения надежной работы программы обработчик ошибок должен включаться в каждую процедуру. Некоторые процедуры, выполняющие сходные функции, могут иметь одинаковые обработчики ошибок. В целях сокращения числа обработчиков ошибок целесообразно создавать универсальные процедуры для выполнения той или иной операции и включать в нее обработчик ошибок, реагирующий на все возможные ситуации.

Пример 7.5. Применение централизованного обработчика ошибок.

```
Private Sub cmdGetData_Click ( )
    Dim FileName As String
    FileName = InputBox ("Введите имя файла")
    If OpenFile(FileName)=True Then
      ' Open File (FileName) - функция, централизованный обработчик ошибок
       MsgBox "Файл успешно открыт"
    Else
        MsgBox "Файл не открыт. Проверьте пожалуйста имя файла"
    End If
End Sub
---------------------------------------------------------------------------------------------
Function OpenFile (FileName As String) As  Boolean
    ' активизируем локальный обработчик ошибок
    On Error CoTo OpenError
    ' открываем файл
    Open FileName For Input As # 1
    ' все в порядке, возвращаем True
    OpenFile=True
    ' выходим из функции, чтобы не попасть в локальный
    ' обработчик ошибок
     Exit Function

Open Error:     ' локальный обработчик ошибок
    Select Case Err.Number
        Case 53
```

```
            ‘ Файл не найден
        Case 55
            ‘ Файл уже открыт
        Case Else
            ` неизвестная данной программе ошибка
    End Select
        ` Операция не удалась, возвращаем False
        OpenFile=False
End Function
```

В данном примере в обработчике ошибок не указаны конкретные действия, которые необходимо выполнить в той или иной ситуации, а даны лишь комментарии.

Обработка ошибок при вложенных вызовах процедур

При возникновении ошибки периода выполнения программа ищет обработчик ошибок в текущей процедуре, если в ней обработчик ошибок отсутствует, то программа ищет обработчик ошибок в процедуре, вызвавшей текущую процедуру и т.д. до главной процедуры. Если и в главной процедуре нет обработчика ошибок, то программа вызывает стандартный обработчик ошибок и выводит сообщение о коде ошибки и системную информацию об ошибке. Алгоритм работы программы при обработке ошибок вложенных процедур приведен на рис. 7.3.

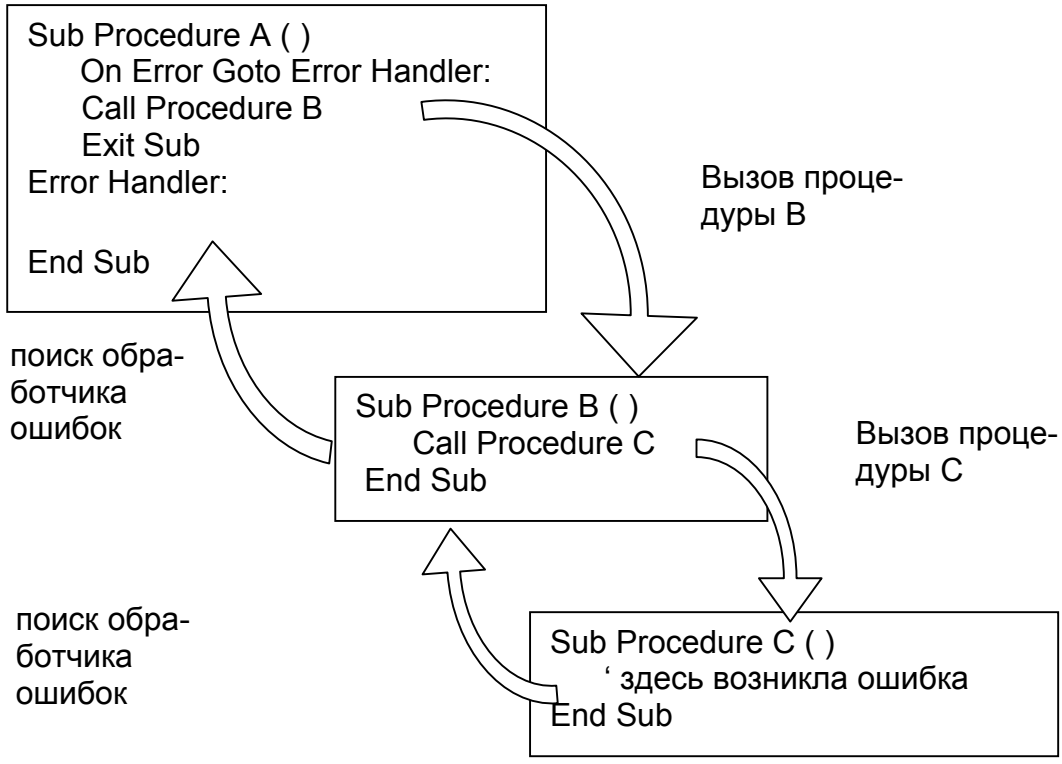


Рис. 7.3. Обработка ошибок при вложенных процедурах

Обработка ошибок объекта CommonDialog

Типичным примером использования обработчика ошибок может быть обработка свойства ***CancelError*** элемента управления ***CommonDialog***. Если значение свойства CancelError равно True, то при щелчке на кнопке Cancel в одном из диалоговых окон элемента управления CommonDialog возникнет ошибка. Если значение этого свойства равно False, ошибка не возникает, но различить нажатие кнопок OK и Cancel в диалоговых окнах будет затруднительно.

```
Function OpenFile() As String
    On Error GoTo Cancel
    CommonDialog1.CancelError = True
    CommonDialog1.ShowOpen
    OpenFile = CommonDialog1.FileName
    Exit  Function
Cancel:
    If Err.Number = cdlCancel  Then
        OpenFile = “”
        MsgBox Err.Description
        Exit Function
    Else
         MsgBox Err.Description
         Stop
     End If
End Function
```

Функция работает следующим образом. Если программа нашла имя файла, введенное в строке ввода диалогового окна открытия файла, то ошибки не возникает. Если при поиске файла возникнет ошибка, то вызывается обработчик ошибки по метке Cancel. Если пользователь нажал кнопку Cancel окна сообщения, тогда код, возвращаемый объектом Err, будет равен коду нажатой клавиши. В этом случае функция OpenFile получает значение “пусто” и процедура функция работу. В ином случае происходит выход из программы по команде Stop.

7.2.4. Упражнение: Контроль ввода

Разработайте обработчик ошибок для программы “Успеваемость”.

7.2.5. Закрепление материала

1. Чем вызвана необходимость обработки ошибок периода выполнения?
2. Какими средствами для обработки ошибок периода выполнения располагает Visual Basic?
3. Приведите синтаксис оператора On Error и дайте пояснение его опциям.

4. Поясните принцип работы оператора Resume.

5. Приведите структуру программы с обработчиком ошибок.

6. Как можно обеспечить централизацию обработки ошибок?

7. Поясните, как работает программа при возникновении ошибки в вызываемой процедуре.

8. Как обеспечить обработку ошибок объекта Common Dialog?

Приложение 1

Основные приемы работы в среде Visual Basic

Основные приемы работы с мышью

Программа VB6 работает под управлением операционной системы Windows, поэтому основным устройством для управления объектами, редактирования является графический манипулятор мышь (в дальнейшем изложении - мышь). Начинающим пользователям полезно знать основные приемы работы с мышью.

Мышь имеет две или три клавиши, но для управления используются только две клавиши: левая и правая. По умолчанию всегда подразумевается левая клавиша. Если для управления применяется правая клавиша, это специально оговаривается. В Windows имеется возможность поменять настройку клавиш мыши для левшей.

Мышь имеет указатель. Форма указателя может изменяться в зависимости от того, какой объект выделен: заголовок формы, граница объекта, разделительная линия и тому подобное.

При работе с мышью пользуются определенными понятиями.

Зависание мыши – установить указатель мыши на объект и задержать его на некоторое время.

Щелкнуть мышью – кратковременно нажать и отпустить клавишу мыши.

Дважды щелкнуть мышью – два раза кратковременно нажать и отпустить клавишу мыши. Интервал времени между нажатиями клавиш, необходимый для того, чтобы программа распознала нажатия клавиш как двойной щелчок, может настраиваться средствами Windows.

Выделить объект - это значит щелкнуть по нему мышью. Выделенный объект выделяется, обычно, синим цветом.

Зацепить объект – установить указатель мыши на объект, нажать и удерживать клавишу мыши.

Протащить мышью – зацепить объект и протянуть указатель мыши по другим объектам.

Перетащить мышью – выделить объект, зацепить его, убедиться, что возле указателя появился маленький прямоугольник с крестиком или без него и перенести объект.

Перетаскивание объектов левой или правой клавишами мыши используют для перемещения или копирования объектов. При перетаскивании объектов осуществляется перемещение объекта, а при перетаскивании при нажатой клавише ***Ctrl*** осуществляется копирование. При перетаскивании объекта с нажатой клавишей Ctrl к указателю мыши прицепляется маленький прямоугольник с крестиком. Перетаскивание файлов правой клавишей отличается от перетаскивания левой клавишей мыши тем, что в первом случае при отпускании клавиши мыши в новом положении объекта появляется меню, которое предлагает выбрать операцию копирования или перемещения.

Управление окнами

Для перемещения окна зацепите его мышью за заголовок и переместите в требуемое положение.

Для изменения размеров окон зацепите мышью за одну из сторон (курсор принимает вид двухнаправленной стрелки) и протяните мышь в нужном направлении. Если зацепить мышью за угол окна, то можно изменять одновременно оба размера.

Размещение элементов управления в форме

Установка элемента на форму

Выделите на Панели элементов управления (Toolbox) элемент, который вы хотите поместить на форму. Выделенный элемент изменяет серый цвет на белый и становится объемным. Переместите указатель мыши на форму. Обратите внимание, что указатель мыши изменил форму и принял вид тонкого черного крестика. Установите указатель мыши в точку, где должен находиться верхний левый край объекта, нажмите левую клавишу мыши и, протаскивая мышь по диагонали к нижнему правому углу объекта, установите требуемые размеры элемента управления.

Имеется и другой, более простой способ установки объекта на форму: щелкните дважды по элементу управления, - элемент управления будет вставлен в центре формы. После этого можно переместить объект в требуемое положение и изменить его размеры.

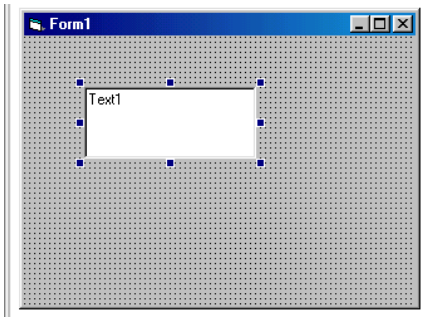


Рис. П1.1. Выделенный объект

Установка размеров элемента управления и его положения на форме

Выделите элемент управления на форме. Выделенный объект выделяется восемью черными точками (маркеры выделения) по периметру (рис.

П1.1.). Если зацепить мышью за маркеры выделения расположенные на сторонах объекта, то можно изменить его ширину или высоту, а если зацепить мышью за маркеры выделения, расположенные в углах, то можно изменять одновременно оба параметра.
Для перемещения объекта в другое положение установите на него указатель мыши и нажмите левую клавишу и удерживая ее в нажатом состоянии переместите объект в требуемое положение. При перемещении указателя мыши вместе с ним будет перемещаться и контур объекта.

Размеры элемента управления и его положение на форме можно установить также с помощью панели Свойств (Properties) – свойства Height – высота объекта, Width – ширина объекта, Top – расстояние объекта от верхнего края формы, Left – расстояние объекта от левого края формы.

Копирование объекта

С помощью копирования можно создать объект с такими же свойствами, как и у исходного объекта. Наследуются все свойства исходного объекта, кроме свойств Index и Name. Для копирования объекта необходимо воспользоваться командами Копировать (Copy) и Вставить (Paste) команды Редактирование (Edit) главного меню, контекстного меню или соответствующими кнопками на панели инструментов. Контекстное (или всплывающее) меню появляется при щелчке правой клавишей мыши по объекту:

- выделите объект и выберите команду ***Edit\Copy***. Копия объекта помещается в буфер (специальный участок оперативной памяти компьютера). (Информация будет сохраняться в буфере до тех пор, пока в нее не будет помещена новая информация или не перезагрузят компьютер);
- выберите команду ***Edit\Paste*** – копия объекта помещается в левый верхний угол формы или другого объекта – контейнера. Свойствами контейнера обладают также Рамка (Frame) и Окно с рисунком (PictureBox). При вставке объекта программа выдаст запрос: Создать массив элементов управления или нет? Ответьте Нет.

Управление группой объектов

В некоторых случаях возникает необходимость изменить свойства одновременно у нескольких объектов или переместить их на форме. Нужную операцию можно выполнить для каждого элемента управления по очереди, но для этого требуется много времени. Операцию можно ускорить, если выделить все эти элементы. Чтобы выделить группу элементов, выделите первый элемент, нажмите и удерживайте клавишу ***Ctrl,*** щелкните мышью по другим выделяемым объектам.

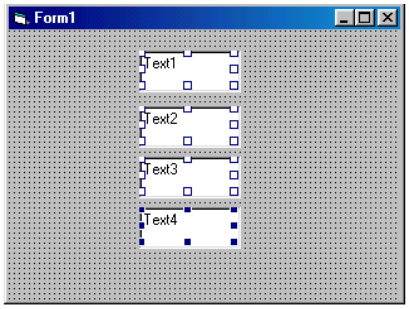


Рис. П1.2. Выделение группы объектов

Другой способ выделения группы объектов – ***обводка.*** В верхнем левом углу панели элементов управления расположена кнопка с черной стрелкой – кнопка выделения. Эта кнопка по умолчанию ак-

тивна. Если эта кнопка не выделена, щелкните по ней мышью. Для выделения группы элементов обводкой установите указатель мыши выше и левее верхнего левого угла крайнего левого элемента группы, нажмите левую клавишу мыши и протягивайте мышь по направлению к правому нижнему углу последнего элемента группы. Вслед за указателем мыши будет формироваться прямоугольник, выделенный пунктирной линией. Убедитесь, что пунктирный прямоугольник полностью охватывает все выделяемые фигуры и отпустите клавишу мыши. Все объекты будут выделены так, как показано на рис. П1.2. Выделенную группу элементов можно перемещать, копировать, удалять или изменять общие свойства: цвет, шрифт, выравнивание текста и т. д. Для отмены выделения щелкните мышью по форме.

Научившись выделять группы элементов, Вы сможете управлять их размещением на форме: выравнивать по горизонтали, по вертикали или выравнивать расстояние между элементами. Эти операции выполняются с помощью команды главного меню Формат (Format). Команда Формат содержит ряд команд (опций):

Align – выравнивание. Позволяет выровнять расположение элементов: по левому краю, правому краю или по центру выделенной группы; по верхнему краю, нижнему краю или по центру выделенной группы; по сетке формы.

Size to Grid – выравнивание по сетке. Выравнивает размеры элементов по сетке (это хорошо заметно при крупной сетке). Изменить расстояние между горизонтальными и вертикальными линиями сетки можно с помощью команды ***Tools\Options\General***. Далее следует в группе ***Form Grid Settings*** установить нужные размеры сетки или убрать ее, сняв флажок ***Show Grid***. По умолчанию установлен флажок автоматического выравнивания элементов управления по сетке ***Align Controls to Grid***. При необходимости его можно снять.

Make Same Size – выравнивание размеров. Позволяет выровнять размеры элементов выделенной группы по ширине, высоте или оба параметра.

Horizontal Spacing и ***Vertical Spacing*** – горизонтальные и вертикальные промежутки. Устанавливает равные расстояния между элементами по горизонтали и по вертикали.

Center in Form – центрирование в форме. Выравнивает положение элементов управления в самой форме по горизонтали и по вертикали.

Order – порядок. Позволяет изменить взаимное расположение элементов управления по слоям. Если элементы управления перекрывают или полностью закрывают один другой, то их можно поменять местами. Видимый сделать невидимым и наоборот.

Lock Controls – блокировка элементов управления. Позволяет запретить изменение размещения элементов на форме.

Приложение 2
Основные функции и типы данных

Арифметические операторы

Оператор	*Описание*
+	Вычисление суммы двух чисел
-	Вычисление разности двух чисел или изменение знака числового выражения
*	Вычисление произведения двух чисел
/	Вычисление частного от деления двух чисел
\	Вычисление частного от деления двух целых чисел (целочисленное деление).
^	Возведение числа в степень
Mod	Возвращает остаток при целом делении двух чисел

Логические операторы

Оператор	*Описание*	*Оператор*	*Описание*
<	меньше	<=	меньше или равно
>	больше	>=	больше или равно
=	равно	<>	не равно

Арифметические функции

Функция	*Описание*
Abs(N)[10]	Вычисляет абсолютное значение числа N
Atn(N)	Вычисляет арктангенс числа N. Значение аргумента должно находиться в интервале от –1 до +1.[11]
Cos(N)	Вычисляет косинус аргумента.
Exp(N)	Возвращает число e, возведенное в указанную степень
Fix(N)	Возвращает целое число, меньшее или равное N для положительных чисел и большее или равное N для отрицательных чисел
Int(N)	Возвращает целое число, меньшее или равное N как для положительных, так и для отрицательных чисел
Iif(усл., V1,V2)	Условная функция, возвращает результат согласно выражения V1, если условие истинно и согласно выражения V2, если условие ложно
Log(N)	Вычисляет натуральный логарифм аргумента
Rnd(N)	Возвращает случайное число в интервале от 0 до 1. При N<0 возвращает определенное число, зависящее от N; при N=0 – псевдослучайное число; при N>0 – новое случайное число
Round(N[, дес])	Возвращает число, округленное к заданному числу десятичных знаков

[10] N – переменная числового типа, C – строковая переменная

[11] В тригонометрических функциях аргумент измеряется в радианах. Для пересчета радианов в градусы зна надо умножить значение функции на 180/• . Для пересчета градусов в радианы надо умножить значение функции на • /180. Приближенное значение числа • равно 3,141593, а более точное значение можно вычислить по формуле • = 4*Atn(1)

Функция	*Описание*
Sgn(N)	Возвращает знак числа: 1 , если N>0; 0 , если N=0; -1, если N<0
Sin(N)	Вычисляет синус угла
Sqr(N)	Вычисляет корень квадратный из аргумента
Tan(N)	Вычисляет тангенс угла
Val(C)	Преобразует аргумент строкового типа в число

Логические функции

Функция	*Описание*
And	Вычисление логического И (умножения) для двух выражений
Eqv	Проверка логической эквивалентности двух выражений
Imp	Логическая импликация для двух выражений
Is	Сравнение двух переменных, содержащих ссылки на объекты
Like	Сравнение двух строковых выражений
Not	Операция логического отрицания над выражением
Or	Операция логического ИЛИ (сложения) для двух выражений
Xor	Операция исключающего ИЛИ для двух выражений
&	Операция слияния двух строковых выражений

Функции и операторы работы с массивами

Оператор	*Действие*
Option Base	Установка нижней границы индекса массива по умолчанию
Dim	Описание переменных и выделение для них областей памяти
Private	Описание личных переменных и выделение для них областей памяти. Применяется на уровне модуля
Public	Описание общих переменных и выделение для них областей памяти. Применяется на уровне модуля
ReDim	Перераспределение памяти для переменных динамического массива. Применяется на уровне процедуры.
Static	Описание переменных и выделение памяти. Применяется на уровне процедуры. Переменные, описанные с помощью оператора Static, не изменяются , пока выполняется программа
Lbound	Определение нижней границы диапазона индексов массива
Ubound	Определение верхней границы диапазона индексов массива
Erase	Повторная инициализация элементов массивов фиксированной длины и освобождение памяти, отведенной для динамического массива
IsArray	Проверка, является ли переменная массивом
Array	Создание массива типа Variant

Функции работы с датами и временем

Date	Возвращает текущую дату в формате мм.дд.гг, где мм – месяц, дд – день, гг – год. Позволяет установить системную дату
DateAdd(интенвал, число интервалов, дата)	Добавляет указанный интервал к дате или вычитает из нее. Параметр “интервал” – строковая переменная и мо-

	жет принимать следующие значения: Yyyy – год, Q – квартал, M – месяц, Y - день в году, D – число месяца, W – день недели, Ww – неделя, H – Час, N – минута, S – секунда. Параметр “число интервалов” означает число интервалов, которые должны быть добавлены или отняты, положительные значения прибавляются, а отрицательные отнимаются. Параметр “дата” содержит допустимую дату.
DateDiff(интервал, дата1, дата2, первый день недели, первая неделя года)	Вычисляет, сколько временных интервалов проходит между двумя указанными датами. Параметр “интервал” может принимать те же значения, что и в функции DateAdd. Параметр “первый день недели” по умолчанию имеет значение “воскресенье”. Параметр “первая неделя “ принимает значения: 0 – установка приложения , 1 – неделя с первым январем, 2 – неделя с не менее чем четырьмя днями нового года, 3 - первая полная неделя в новом году, по умолчанию принимает значение недели, на которую приходится 1 января:
DatePart(интервал, дата1, дата2, первый день недели, первая неделя года)	Вычисляет, к какой части указанного интервала относится дата Значения параметров аналогичны ранее описанным функциям
DateSerial(год, месяц, день)	Возвращает дату, состоящую из указанного года, месяца и дня
DateValue(C)	Возвращает дату, указанную в аргументе – строке.
Day(N)	Возвращает целое значение от 1 до 31, которое означает число месяца указанной даты.
Month	Возвращает целое число от 1 до 12, обозначающее месяц указанной даты
MonthName(N)	Возвращает строку с именем месяца по его номеру
Now	Возвращает системную дату и системное время компьютера
Weekday(дата [, первый день недели])	Возвращает целое число от 1 до 7, соответствующее дню недели указанной даты
WeekdayName(N)	Возвращает строку, указывающую определенный день недели, соответствующее числовому аргументу
Year(дата)	Возвращает целое число от 100 до 9999, соответствующее году указанной даты
Hour Month	Возвращает целое число от 0 до 23, обозначающее час указанного времени
Minute(N)	Возвращает целое число от 0 до 59, обозначающее минуту текущего часа указанного времени
Second(N)	Возвращает целое число от 0 до 59, обозначающее секунду текущей минуты указанного времени
Time	Возвращает текущее системное время
Timer	Возвращает число прошедших с полуночи секунд. Функция может служить в качестве секундомера для определения временных интервалов.
TimeSerial	Возвращает значение времени при заданных часе, минуте и секунде
TimeValue	Преобразует строку, представляющую дату, в значение

	времени

Функции обработки строк

Функция	*Комментарий*
Asc(C)	возвращает ASCII код первого символа строки символов C
Chr(N)	возвращает символ, соответствующий коду ASCII (числа от 32 до 255)
Space(N)	генерирует строку символов из N пробелов
String(N,Chr)	генерирует строку из символов, повторяющую N раз символ, соответствующий ASCII – коду. Код может принимать числовые значения от 0 до 255. Числа больше 255 преобразуются по формуле (код Mod 256), т. е. целочисленное деление кода числа на основание 256
LCase(C)	преобразует прописные буквы в строчные
UCase(C)	преобразует строчные буквы в прописные
LTrim(C), RTrm(C), Trim(C)	удаляет левые, правые или и левые и правые пробелы из строки символов
Len(C)	возвращает длину строки
Left(C, N)	возвращает N символов с левого конца строки C
Rignt(C, N)	возвращает N символов с правого конца строки C
Mid(C, N1, N2)	возвращает N2 символов из строки C начиная с N1 символа
InStr ([I,] C, C1, Compare)	осуществляет поиск вхождения строки C1 в строку C. Поиск начинается с позиции i, если i отсутствует, то поиск осуществляется с начала строки. Опция Compare указывает способ сравнения: -1 – использует метод сравнения, заданный оператором Option Compare, 0 – двоичное сравнение выражений; 1 – посимвольное сравнение; 2 – только в Microsoft Access, сравнение осуществляется на основе параметров, установленных в базе данных
InStr (C, C1, I, Compare)	то же, что и InStr, но возвращает позицию первого вхождения последовательности символов C1 в C, начиная с конца строки
Str (N)	преобразует числовое выражение в строку
Cint(N), CLng(N), CSng(N), Dbl(N), CStr(N) и др.	преобразуют числовой аргумент в целое число одинарной или двойной точности, вещественное число одинарной или двойной точности или в строку символов, соответственно.

Операторы и функции работы с файлами

Оператор	*Описание*
Open	Открытие файла
Close	Закрытие файла
Reset	Закрытие всех файлов, открытых с помощью оператора Open
EOF	Возвращает значение, показывающее, расположен ли указатель текущей записи в позиции, после последней записи объекта Recordset
FreeFile	Возвращает номер свободного канала, доступного для использования оператором Open

Loc	Возвращает значение, определяющее текущее положение указателя чтения/записи в открытом файле
LOF	Возвращает размер в байтах файла, открытого с помощью оператора Open
Get	Чтение данных из открытого файла прямого доступа на диске
Input	Чтение данных из файла, открытого в режиме Input или Binary
Input#	Чтение данных из открытого последовательного файла и присвоение их переменным
Input$	Возвращает значение типа String , содержащее символы из файла, открытого в режиме Input или Binary
Line Input#	Чтение строки из открытого последовательного файла и присвоение ее переменной типа String
Print#	Запись в файл
Put	Запись содержимого переменной в файл прямого доступа на диске
Write#	Запись не форматированных данных в файл последовательного доступа
Spc	Установка позиции вывода на экран или печать при использовании вместе с оператором Print# или методом Print
Tab	Используется совместно с оператором Print# или методом Print для задания позиции вывода на экран или печать
Width#	Задание ширины строки для файла, открытого с помощью оператора Open
FileCopy	Копирование файлов. В качестве аргументов могут указываться имя каталога или папки на диске
Dir	Получение имени файла, каталога или папки, удовлетворяющего шаблону имени файла, набору атрибутов файла или метке тома на диске
Kill	Удаление файлов с диска
Name	Изменение имени файла, каталога или папки
ChDir	Изменение текущего каталога или папки
ChDrive	Изменение текущего диска
MkDir	Создание нового каталога или папки
RmDir	Удаление существующего каталога или папки
GetAttr	Считывание атрибутов файла
SetAttr	Задание атрибутов файла
LoadPicture	Загрузка графического изображения
SavePicture	Сохранение графического изображения

Функции проверки значений выражений

Функция	*Описание*
IsArray	Возвращает значение типа Boolean, показывающее, является ли переменная массивом
IsDate	Возвращает значение типа Boolean, которое показывает, может ли значение выражения быть преобразовано в дату
IsEmpty	Возвращает значение типа Boolean, показывающее, была ли переменная инициализирована
IsMissing	Возвращает значение типа Boolean, которое показывает, был ли передан в процедуру необязательный параметр
IsNull	Возвращает значение типа Boolean, которое показывает, являет-

	ся ли результатом вычисления выражения значение Null
IsNumeric	Возвращает значение типа Boolean, которое показывает, является ли результат числовым значением
IsObject	Возвращает значение типа Boolean, которое показывает, является ли переменная объектной

Средства обработки ошибок

Элемент	*Описание*
Err (объект)	Содержит информацию, однозначно определяющую ошибку периода выполнения
Clear (метод)	Сбрасывает до нулей или пустых строк (“”) все значения свойств объекта Err
Error (функция)	Генерирует ошибку периода выполнения
Raise (метод)	Создает ошибку, которая возникает на этапе выполнения программы
On Error (оператор)	Перехватывает ошибки периода выполнения
IsError (функция)	Возвращает значение типа Boolean, показывающее, представляет ли аргумент значение ошибки
CVErr (функция)	Возвращает значение типа Variant с подтипом Error, содержащее код ошибки, указанный пользователем

Типы данных

Тип переменной	*Описание*
Boolean	16-разрядные (двухбайтовые) числа, которые могут иметь только два значения : True или False
Byte	8-разрядные (1 байт) числа без знака в диапазоне от 0 до 255
Currency	64-разрядные (8-байтовые) целые числа, которые после деления на 10000 дают число с фиксированной десятичной точкой с 15 разрядами в целой части и 4 разрядами в дробной (Денежный тип, используется для финансовых расчетов. Диапазон изменения +/- $9*10^{14}$
Date	64-разрядные (8-байтовые) числа с плавающей точкой стандарта IEEE , представляющие даты из диапазона от 1 января 100 года до 31 декабря 9999; и значения времени от 0:00:00 до23:59:59
Double	64-разрядные (8-байтовые) числа с плавающей точкой стандарта IEEE в диапазоне от –1,797 693 134 862 32 E308 до -4,940 656 458 412 47E-324 для отрицательных значений и от 4,940 656 458 412 47E-324 до 1,797 693 134 862 32E308 для положительных значений
Integer	16-разрядное (2-байтовые) числа в диапазоне от –32768 до 32767
Long	32-разрядное (4-байтовые) числа со знаком в диапазоне от

Тип переменной	*Описание*
	- 2 147 483 648 до 2 147 483 647
Object	32- разрядные (4-байтовые) адреса, которые содержат ссылки на объекты
Single	32-разрядные (4-байтовые) числа с плавающей точкой стандарта IEEE в диапазоне от –3,402823E38 до –1, 401298E-45 для отрицательных значений и от 1, 401298E-45 до 3,402823E38 для положительных значений
String	Кодами для символов, образующих значения типа String, служат целые числа от 0 до 255. Строки переменной длины могут содержать 2^{31} символов; Строки постоянной длины могут содержать от 1 до 2^{16} символов
Variant	К данным типа Variant относятся все переменные, не описанные явно с другим типом данных (с помощью операторов Dim, Private, Public или Static)

Приложение 3
Классы и операторы Visual Basic[12]

Класс	*Наименование класса*
CheckBox	Флажок
ComboBox	Поле со списком
CommandButton	Командная кнопка
Common Dialog	Окно диалога Windows, используется для организации открытия файлов и сохранения их на диске, настройки принтера и печати документов, настройки параметров шрифтов и установки цвета
Control	Обобщенное имя объектов, помещенных на форму
DirListBox	Список каталогов
DriveListBox	Список дисков
ErrObject	Содержит код ошибки и системное сообщение об ошибке, а также некоторую другую информацию
FileListBox	Список файлов
Form	Форма
Frame	Рамка
Global	Абстрактный объект, обозначающий компьютер
HScrollBar	Горизонтальная линейка прокрутки
Image	Изображение
Label	Метка
Line	Линия
ListBox	Список
MDI-Form	Форма – контейнер для всех дочерних форм приложения
Menu	Меню

[12] Приведен перечень классов и операторов, упоминаемых в учебном пособии

MSFlexGrid	Сетка. Служит для вывода данных
OptionButton	Переключатель
PictureBox	Окно с рисунком
Printer	Печатающее устройство
Screen	Экран Windows
Shape	Абстрактный объект, предназначенный для изображения геометрических фигур
Slider	Регулятор
StatusBar	Строка состояния
TextBox	Текстовое поле
Timer	Часы
VScrollBar	Вертикальная линейка прокрутки

Операторы и методы	***Назначенине***
Call	Вызов процедуры
Circle	Рисование окржностей
Cls	Очистка формы
Debug.Printer	Вывод информации в окно отладки
DefBool	Объявление переменной типа Boolean
DefByte	Объявление переменной типа Byte
DefCur	Объявление переменной типа Currency
DefDate	Объявление переменной типа Date
DefDbl	Объявление переменной типа Double
DefInt	Объявление переменной типа Integer
DefLng	Объявление переменной типа Long
DefSng	Объявление переменной типа Single
DefStr	Объявление переменной типа String
DefObj	Объявление переменной типа Object
DefVar	Объявление переменной типа Variant
Do Loop	Оператор цикла
For/Each	Оператор цикла
For/Next	Оператор цикла
Hide	Скрытие формы
If/Then/Else/End If	Условный оператор
Let	Оператор присвоения
Line	Рисование линии
Load	Загрузка формы
Load Picture	Загрузка рисунка
Move	Перемещение
OLEDrag	Перемещение объекта с помощью мыши
PaintPicture	Перерисовка изображения
Point	Определение цвета точки графического объекта

PopupMenu	Вывод контекстного меню
Print	Вывод информации на форму или на принтер
Print Form	Печать формы
Pset	Рисование точки
Refresh	Обновление изображения
Select Case	Оператор множественного выбора
Scale	Установка масштаба графического объекта
ScaleX	Установка масштаба графического объекта по оси X
ScaleY	Установка масштаба графического объекта по оси Y
SetFocus	Установка фокуса
Show	Показать объект
Unload	Выгрузить объект
While/Wend	Оператор цикла типа “Пока”
ZOrder	Изменение порядка элементов управления

Приложение 4
Моделирование движения механизма

Разработка программы “Моделирование движения механизма”

Задание

Разработать программу для моделирования движения механизма. Программа должна обеспечивать воспроизведение механизма в исходном состоянии, в динамике (несколько положений), строить траекторию заданной точки, определять область допустимых перемещений звеньев механизма и область, занимаемую механизмом при движении.

П4.1. Пояснительная записка

П4.1.1. Краткие теоретические сведения

Под движением механизма понимается воспроизведение положения механизма в последовательные моменты времени. При моделировании движения механизма рассматривают следующие вопросы:

1. Воспроизведение положения механизма в исходном состоянии.
2. Воспроизведение положения механизма в нескольких промежуточных положениях.
3. Определение зон движения звеньев механизма.
4. Определение области, занимаемой механизмом в процессе движения.
5. Построение траектории движения заданной точки.

Основными элементами механизма являются (рис.П.1): кривошип (1), шатун (2, 3), ролик (4) с радиусом (5), ползун (6).

.

П4.1.2. Исходные данные

К исходным данным относятся координаты опор, ползуна, ролика, длины звеньев механизма. Исходные данные удобно задавать в пикселях, тем более что координаты опор на экране иначе задать невозможно. Масштаб преобразования для линейных величин примем равным 100:1, то есть 100 пикселей равны 1 м.

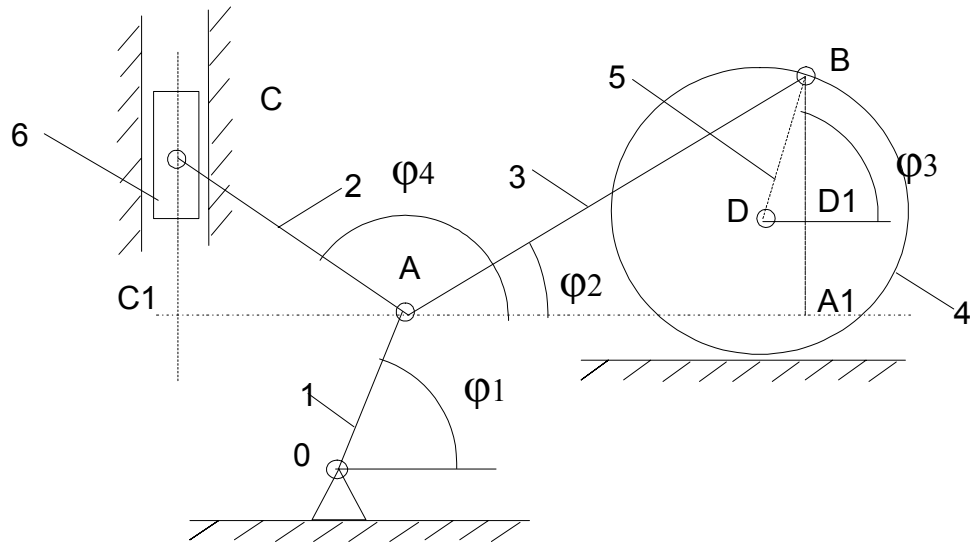


Рис.П4.1. Механизм

Координаты опоры О : X=100, Y=50, угол φ_1 =45 градусов, длина звена ОА=0,40 м. Длина звена АВ=1,5 м. Координата X точки С равна 100, длина звена СА=0,8 м. Высота расположения опоры Е для ролика равна 0,60 м. Радиус ролика DB равен 0,4 м.

Исходные данные сведем для удобства использования в таблицу (табл. П4.1). При записи данных в таблицу проведем преобразование линейных размеров звеньев механизма с учетом принятого масштаба.

В табл.П4.1 номера строк обозначены номерами точек звеньев механизма О, А, В, С, D, E, номера столбцов обозначают координаты точек X, Y, угол положения звена механизма - F и длину звена механизма - L. Значения неизвестных параметров обозначены в таблице знаком “-”.

Таблица П4.1

Исходные данные

	X	Y	F	L
0	100	50	45	40
A	-	-	-	150
B	-	-	-	-

C	100	-	-	80
D	-	60+40	-	40
E	-	60	-	-

Состояние механизма в статике определяется положением кривошипа (угол φ_1).

Расчету в данном механизме подлежат значения координат точек A, B, координаты Yc для точки C и Xd для точки D. В качестве промежуточных результатов необходимо иметь значения углов φ_2 и φ_3.

П4.1.3. Разработка математической модели

Координаты точки A определяются по длине кривошипа OA и значению угла φ_1:

$Xa=Xo+L_1 \cos\varphi_1$ (1.1)

$Ya=Xo+L_1 \sin\varphi_1$. (1.2)

Координата Y точки C определяется из прямоугольного треугольника CAC_1. Длина CA известна, а $C_1A=Xa-Xc$, тогда

$$Y_c = Y_a + \sqrt{CA^2 - C_1A^2} \quad (1.3)$$

В данном механизме ролик имеет две степени свободы. Он может вращаться вокруг своей оси и перемещаться в горизонтальном направлении. В исходных данных высота расположения ролика и радиус ролика заданы, а следовательно и вертикальная координата центра ролика Yd известна. Горизонтальное положение ролика Xd - неизвестно. Наибольшие трудности возникают при определении значения угла φ_2.

Для определения значения угла φ_2 необходимо составить и решить систему из пяти уравнений:

$$x_b = x_a + l_2 \cos\varphi_2 \quad (1.4)$$

$$x_b = x_d + l_3 \cos\varphi_3 \quad (1.5)$$

$$y_b = y_a + l_2 \sin\varphi_2 \quad (1.6)$$

$$y_b = y_d + l_3 \sin\varphi_3 \quad (1.7)$$

$$(x_b - x_d)^2 + (y_b - y_d)^2 = l_3^2 \quad (1.8)$$

Здесь l_2 - длина шатуна AB, l_3 - радиус ролика. Первые четыре уравнения получают из рассмотрения прямоугольников ABB_1 и DBD_1. Пятое уравнение составляется из следующих соображений: координаты точки B лежат на поверхности окружности, поэтому можно записать уравнение окружности с центром в точке D и радиусом, равным радиусу ролика l_3.

Из уравнений 1.4 и 1.5 находим значение Xd:

$$x_d = x_a + l_2 \cos\varphi_2 - l_3 \cos\varphi_3 \quad (1.9)$$

Из уравнений 1.6 и 1.7 находим значение угла φ_3:

$$\varphi_3 = \arcsin((y_a + l_2 \mathrm{Sin}\varphi_2 - y_d)/l_3) \qquad (1.10)$$

Заменив в выражении (1.9) угол φ_3 на его значение из (1.10) и подставив значения Xb (1.4), Xd и Yb (1.9) в (1.6) получим уравнение

$$((x_a + l_2 \mathrm{Cos}\varphi_2 - (x_a + l_2 \mathrm{Cos}\varphi_2 - l_3 \mathrm{Cos}(\arcsin((y_a + l_2 \mathrm{Sin}\varphi_2 - y_d)/l_3)))^2 +$$
$$+(y_a + l_2 \mathrm{Sin}\varphi_2 - y_d)^2) - l_3^2 = 0. \qquad (1.11)$$

В выражении 1.11 один неизвестный параметр - угол φ_2. Найти значение угла φ_2 из данного уравнения можно с помощью одного из методов поиска значения корня, например методом итераций.

П4.1.4. Схема алгоритма программы

Список задач, выполняемых программой, полностью вытекает из задания. В соответствии с этим в программе необходимо предусмотреть следующие функциональные блоки:

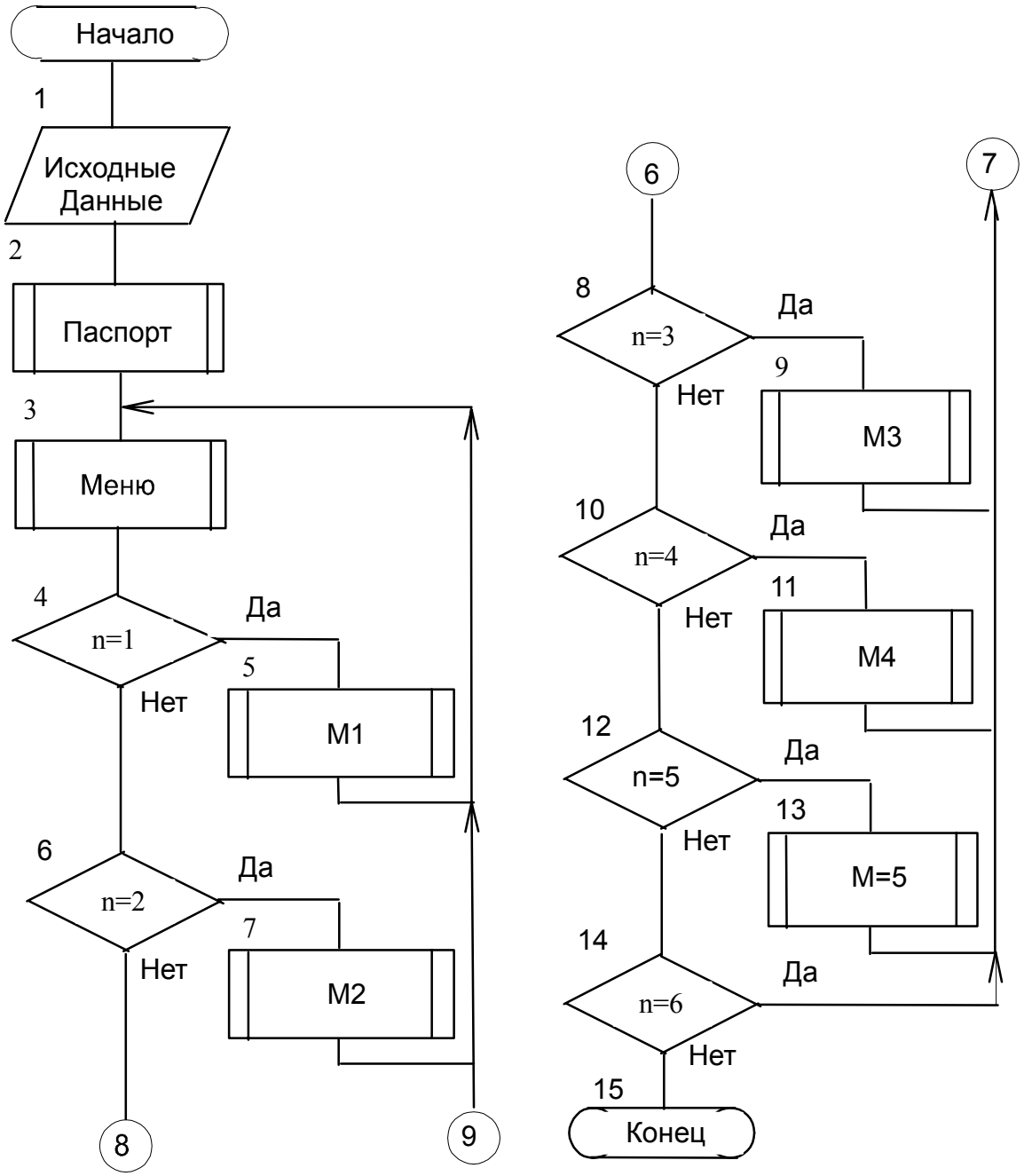


Рис. П4.2 . Схема алгоритма построения механизма (окончание)

- Паспорт программы;
- Меню;
- Блок моделирования механизма в статике;
- Блок моделирования механизма в динамике;
- Блок моделирования движения заданной точки или произвольной точки механизма;
- Блок определения зон движения звеньев механизма и механизма в целом.

Все блоки независимы друг от друга. Вызов блоков программы осуществляется из меню пользователя и после окончания работы блока управление передается меню программы. Указанная особенность программы полностью согласуется с принципами структурного программирования, характерными для программирования в среде Visual Basic.

Схема алгоритма приведена на рис. П4.2, где обозначено:

n=1 - переход на метку M1 - построение механизма в статике;

n=2 - переход на метку M2 - построение механизма в динамическом режиме;

n=3 - переход на метку M3 - построение зон действия звеньев механизма;

n=4 - переход на метку M4 - построение зоны занимаемой механизмом;

n=5 - переход на метку M5 - построение траектории движения точки;

n=6 - выход из программы.

П4.1.5. Разработка проекта программы

Программа должна состоять из нескольких форм в соответствии с определенными задачами. Совместную работу форм удобно организовать с помощью MDI-формы. В эту форму можно поместить меню пользователя и фоновый рисунок.

Эскизы форм представлены на рис. П4.2 – П4.5. Так как в задании не оговорены требования к интерфейсу пользователя при моделировании движения механизма, то состав элементов управления на форме разработчик определяет в соответствии со своим представлением об удобстве управления демонстрацией свойств механизма. В простейшем случае можно ограничиться двумя кнопками: старт и выход. В учебных целях при разработке интерфейса целесообразно использовать как можно больше различных элементов управления.

Описание переменных

Опишем глобальные переменные, используемые в программе (табл. П4.1)

Аналогично описываются переменные процедур и функций.

Таблица П4.1

Глобальные переменные

Наименование программы *Mechanizm*	**Тема проекта:** *Моделирование движения механизма*		**Лист** *1* **Листов** __
Обозначение переменной	***Имя переменной***	***Тип***	***Примечание***
	Obj	Object	Глобальная переменная типа Object, заменяет имя объекта при вызове процедур, осуществля-

			ющих графические построения. В каждой форме, в которой рисуется механизм введен оператор: ***Set Obj=<имя формы>***
Xo,Xa,Xb,Xc,Xd	X(5)	Single	Координаты X узлов механизма
Yo,Ya,Yb,Yc,Yd	Y(5)	Single	Координаты Y узлов механизма
φ_1, φ_2,φ_3,φ_4	f(5)	Single	Значения углов
l_1, l_2, l_3 , l_4, l_5	L(5)	Single	Длины звеньев механизма
	fi	Single	Текущий угол поворота
	pi	Double	число
	pi2	Double	число 2π
	dx	Single	шаг
	Ft	Single	Вспомогательная переменная, хранит вычисленное значение угла φ_2
	i1	Double	Максимальный угол поворота кривошипа i1=2*pi
	fl	Byte	Флаг,
	fl2	Byte	Флаг,
	xo	Integer	Вспомогательная переменная, для хранения целых чисел, используется при построении опор.
	yo	Integer	то же
	xr	Integer	то же
	yr	Integer	то же
	n	Integer	то же

Описание элементов управления

Опишем основные свойства элементов управления формы Статика (табл. П4.2). Аналогично описываются элементы управления и других форм.

Таблица П4.2.

Свойства элементов управления формы Статика

Наименование программы ***Mechanizm***	**Тема проекта:** ***Моделирование движения механизма***	**Лист _*2*_**
		Листов __

Объект,	***Свойство***	***Значение***
Форма	Name	FormStatica
	Caption	Статика
	AutoRedraw	True
	MDIChild	2 ‘ Maximize
Командная кнопка 1	Name Caption	Command1Statica Старт
	Height	375
	Width	1335
Командная кнопка 2	Name Caption	Command2Statica Выход
	Height	375

Наименование программы *Mechanizm*	Тема проекта: *Моделирование движения механизма*	Лист _2_
		Листов __

Объект,	*Свойство*	*Значение*
	Width	1335
Сетка	MSFlexGrid	MSFlexGridStatic
	Rows	6
	Cols	3
Надпись	Name	LabelStatica
	Caption	“Укажите угол поворота в градусах”
	Alignment	2 ‘ Center
Текстовое поле	Name	TextStatica
	Text	0

П4.2. Текст программы

Текст программы MDI-формы

```
Private Sub MenuEnd_Click()
  End
End Sub
Private Sub MenuDinamika_Click()
   FormDinamic.Show
End Sub
Private Sub MenuPasport_Click()
   FormPasport.Show
End Sub
Private Sub MenuSonaDvigenij_Click()
   FormZona.Show
End Sub
Private Sub MenuStatika_Click()
```

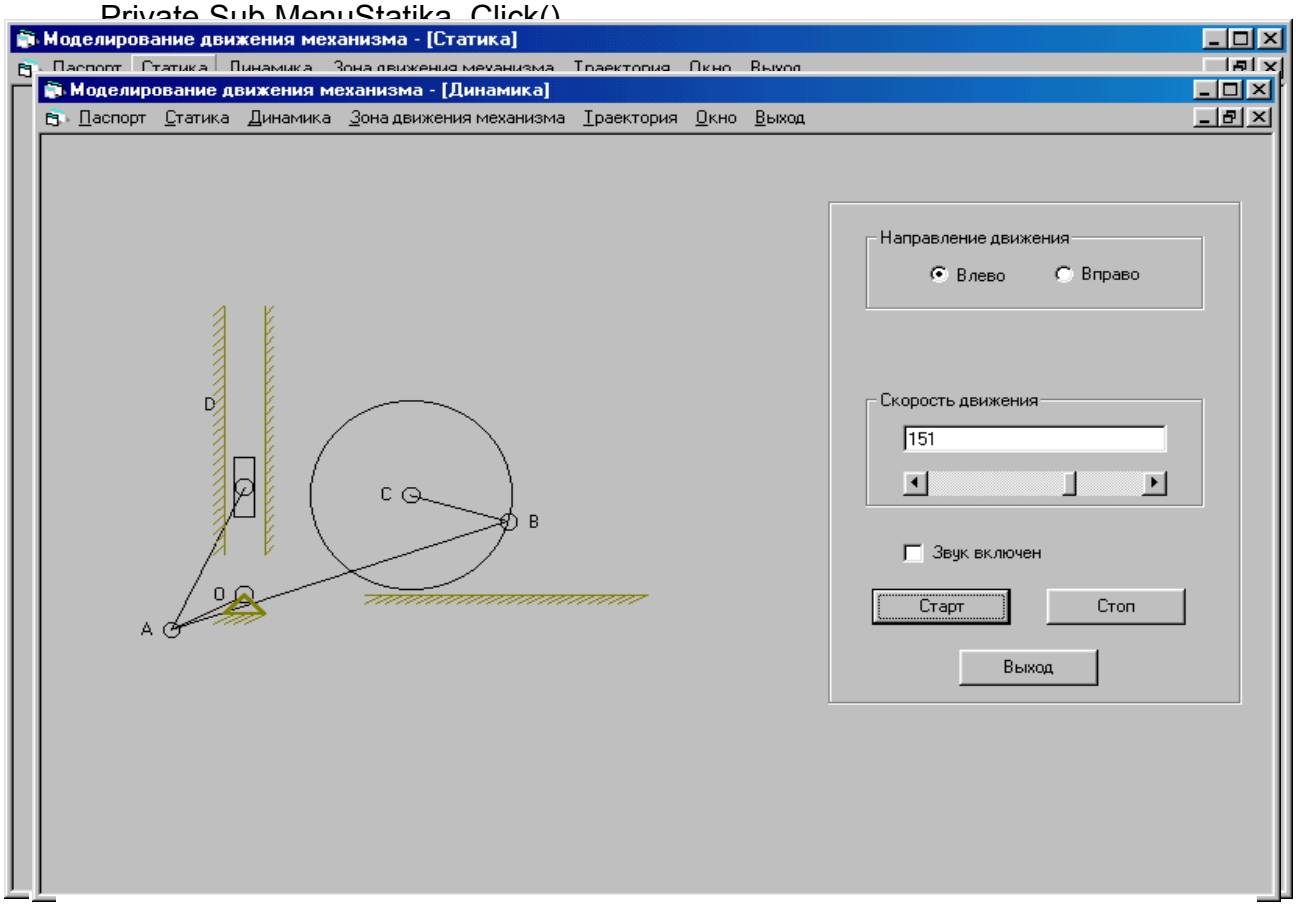


Рис.П4.3. Форма для демонстрации движения механизма

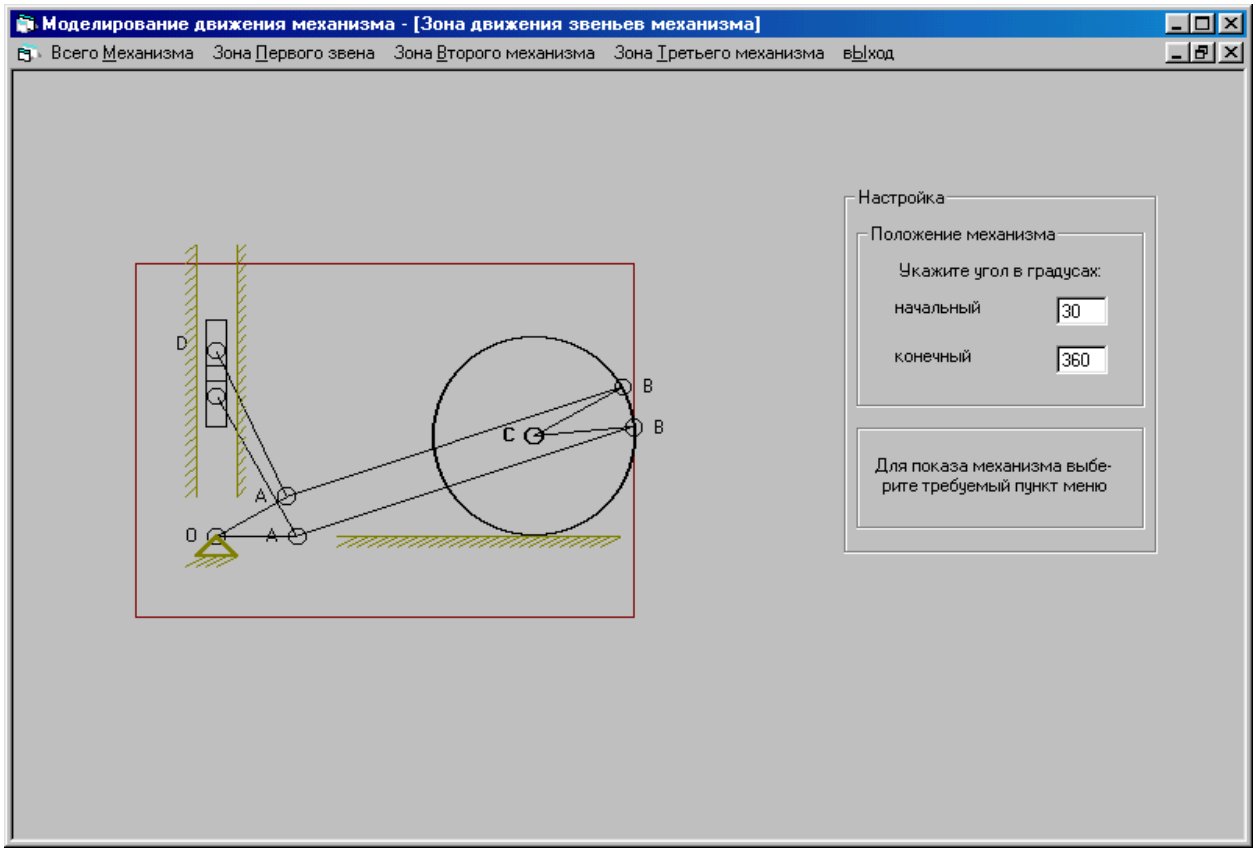


Рис.П4.4. Форма для демонстрации зон действия механизма и его звеньев

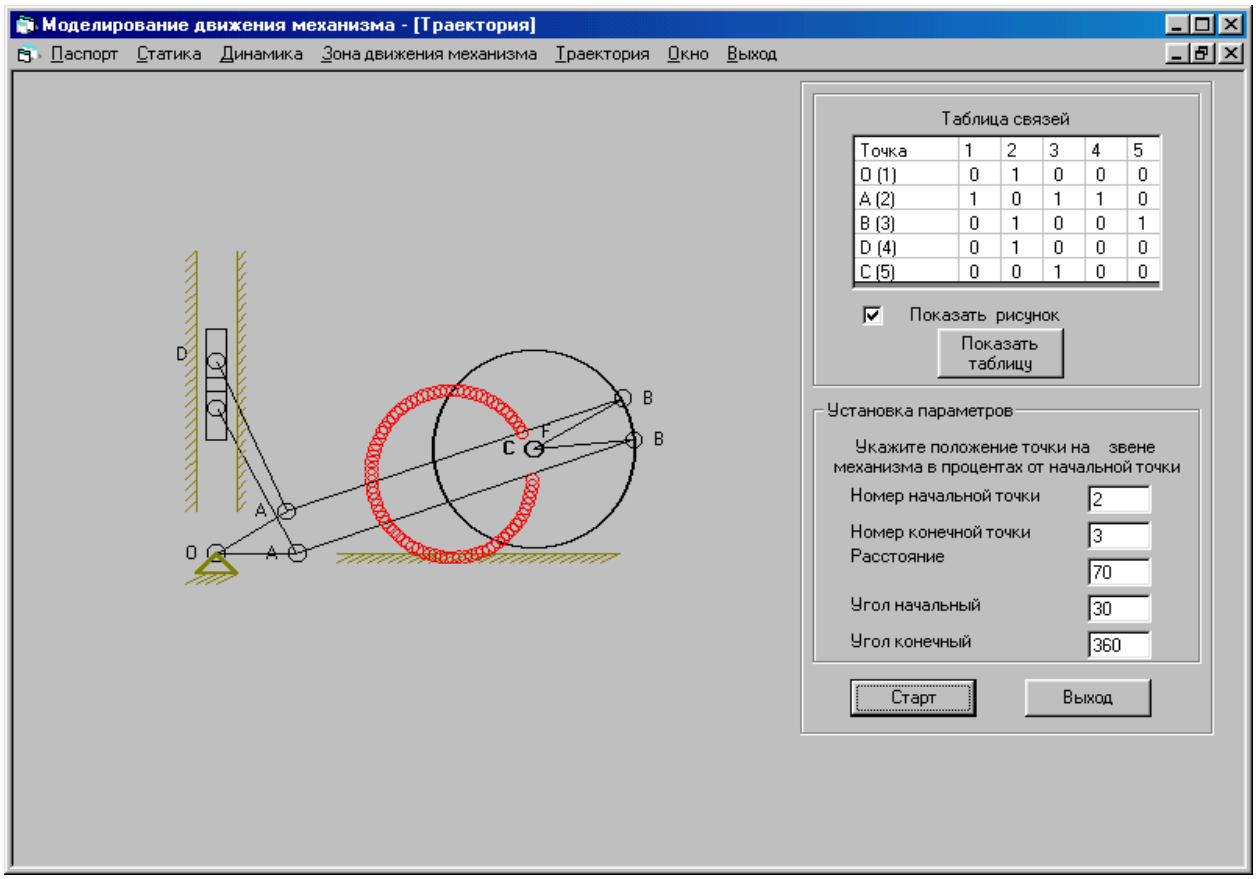


Рис. П4.5. Форма для демонстрации траектории движения заданной точки

```
  frmMDIForm.Arrange vbCascade
End Sub
Private Sub mnuHorizont_Click()
  frmMDIForm.Arrange vbTileHorizontal
End Sub
Private Sub mnuVertical_Click()
  frmMDIForm.Arrange vbTileVertical
End Sub
```

Текст программы Модуля

```
Rem Объявление переменных и массивов
Option Base 1
Global Obj As Object
Global X(5) As Single, Y(5) As Single
Global f(5) As Single, l(5) As Single
Global dlina1 As Single, dlina2 As Single
Global dlina3 As Single, dlina4 As Single
Global msg0 As String, msg1 As String
Global msg2 As String, msg3 As String
Global msg4 As String, msg11 As String
Global msg12 As String, msg13 As String
Global msg14 As String, msg15 As String
Global fi As Single ' Текущий угол поворота
Global pi As Double  ' Константа
Global i1 As Double  ' Константа
Global fl As Byte
Global fl2 As Byte
Global xo As Integer, yo As Integer
Global xr As Integer, yr As Integer, n As Integer
Global pi2 As Single, dx As Single, Ft As Single

Rem  процедура ввода данных
Public Sub VvodDannych()
    y11 = 500: y12 = 4 / 3 * y11
    X(1) = 100: Y(1) = 150: f(1) = 45: l(1) = 40
    l(2) = 175
    X(4) = 100: l(4) = 80
    Y(5) = 200: l(5) = 50
    Ft = 30
    pi = Atn(1) * 4
    pi2 = 2 * pi
    i1 = pi2:  'максимальный угол поворота кривошипа
    dx = 0.3: 'шаг построения изображения в динамике
End Sub

Rem процедура вычисления координат
Public Sub Raschet()
    Dim x1 As Single, y1 As Single
    Dim b1 As Single, f1 As Single, e As Single
    Dim s1 As Single, ss1 As Single, df As Single
    Dim ss2 As Single, f2 As Single
```

```
    Dim a As Single, b As Single
    f1 = fi
    X(2) = X(1) + l(1) * Cos(f1)
    Y(2) = Y(1) + l(1) * Sin(f1)
    b1 = l(4) ^ 2 - (X(2) - X(4)) ^ 2
    If b1 < 0 Then GoTo m1
    Y(4) = Sqr(b1) + Y(2)
    Rem  вычисление  значения угла f2
    e = 1: Rem Точность поиска
    Rem цикл изменения значения угла f1
    s1 = e + 1: df = 0.01: f2 = pi / 10
      Rem цикл изменения значения угла f2
      While Abs(s1) > e And f2 < pi / 2
         a = (Y(2) - Y(5) + l(2) * Sin(f2)) / l(5)
         B = a * l(5)
         If (1 - a ^ 2) <= 0 Then GoTo m1
         ss1 = Cos(Atn(a / Sqr(1 - a ^ 2)))
         ss2 = (X(2) + l(2) * Cos(f2) - (X(2) + l(2) * Cos(f2) - l(5) * ss1))
         s1 = l(5) ^ 2 - (ss2 ^ 2 + B ^ 2)
         GoTo m2
    m1:
    m2:
         Ft = f2
         f2 = f2 + df
      Wend
         Rem Расчет координат
         f2 = Ft: f(2) = f2
         a = (Y(2) - Y(5) + l(2) * Sin(f2)) / l(5)
         If (1 - a ^ 2) <= 0 Then GoTo m3
         f(3) = Atn(a / Sqr(1 - a ^ 2))
         X(3) = X(2) + l(2) * Cos(f2)
         Y(3) = Y(2) + l(2) * Sin(f2)
         X(5) = X(3) - l(5) * Cos(f(3))
    m3:
End Sub

Rem построение рисунка
Public Sub Postroenie()
     Dim a1 As Single, a2 As Single, b1 As Single, b2 As Single
    Obj.Circle (X(1), Y(1)), 4
    Obj.Line (X(1), Y(1))-(X(2), Y(2))
    Obj.Circle (X(2), Y(2)), 4
    Obj.Line (X(2), Y(2))-(X(3), Y(3))
    Obj.Circle (X(3), Y(3)), 4
    Obj.Circle (X(5), Y(5)), l(5)
    Obj.Circle (X(5), Y(5)), 4
    Obj.PSet (X(5), Y(5))
    Obj.Circle (X(4), Y(4)), 4
    Obj.Line (X(2), Y(2))-(X(4), Y(4))
    Obj.Line (X(5), Y(5))-(X(3), Y(3))
    a1 = X(4) - 5: a2 = X(4) + 5: b1 = Y(4) - 15: b2 = Y(4) + 15
```

```
      Obj.Line (a1, b1)-(a2, b2), , B
End Sub

******* ******************************************
' Rem построение опорных точек
 Public Sub Psopor()
    ' опора кривошипа
    Obj.CurrentX = X(1) - 15: Obj.CurrentY = Y(1) + 5
    Obj.Print "O"
    Obj.CurrentX = X(4) - 20: Obj.CurrentY = Y(1) + 100
    Obj.Print "D"
    Obj.CurrentX = X(2) - 15: Obj.CurrentY = Y(2) + 5
    Obj.Print "A"
    Obj.CurrentX = X(3) + 10: Obj.CurrentY = Y(3) + 5
    Obj.Print "B"
    Obj.CurrentX = X(5) - 15: Obj.CurrentY = Y(5) + 5
    Obj.Print "C"
    xo = X(1): yo = Y(1)
    xr = xo - 15: yr = yo - 15: n = 4
    Form_opora
    Form_StrichG
    ‘ левая направляющая ползуна
    xo = X(1) - 10: yo = Y(1) + 20
    xr = xo: yr = yo: n = 25
    Obj.Line (xo, yo)-(xo, yo + 125), QBColor(6)
    xr = xo - 5
    Form_StrichV
    ' правая направляющая ползуна
    xo = X(1) + 10: yo = Y(1) + 20
    xr = xo: yr = yo: n = 25
    Obj.Line (xo, yo)-(xo, yo + 125), QBColor(6)
    Form_StrichV
    ' Каток
    xo = X(1) + 60: yo = Y(5) - l(5)
    xr = xo: yr = yo - 5: n = 27
    Obj.Line (xo, yo)-(xo + 140, yo), QBColor(6)
    Form_StrichG
End Sub
'*************************************************************************
Rem построение опоры
Public Sub Form_opora()
   Obj.PSet (xo, yo), QBColor(6) 'коричневый
   Obj.DrawWidth = 2
   Obj.Line -(xo - 10, yo - 10), QBColor(6)
   Obj.Line -(xo + 10, yo - 10), QBColor(6)
   Obj.Line -(xo, yo), QBColor(6)
End Sub
Rem горизонтальная штриховка
Public Sub Form_StrichG()
    Dim ii As Integer
    For ii = 1 To n
```

```
        Obj.DrawWidth = 1
        Obj.Line (xr, yr)-(xr + 10, yr + 5), QBColor(6)
        xr = xr + 5
    Next ii
End Sub
Rem вертикальная штриховка
Public Sub Form_StrichV()
    Dim ii As Integer
    For ii = 1 To n
        Obj.DrawWidth = 1
        Obj.Line (xr, yr)-(xr + 5, yr + 5), QBColor(6)
        yr = yr + 5
    Next ii
End Sub
```

Текст программы формы Статика

```
Dim tc1(6, 3) As String, Ft As Integer
Private Sub Form_Load()
    Set Obj = FormStatica
     VvodDannych
     tc1(1, 1) = "Точка": tc1(1, 2) = "X": tc1(1, 3) = "Y"
     tc1(2, 1) = "O": tc1(3, 1) = "A": tc1(4, 1) = "B":
     tc1(5, 1) = "D": tc1(6, 1) = "C"
End Sub
Private Sub Command1Statica_Click()
    Dim rx As Integer, ry As Integer, i As Integer
    Dim ax As Single, ay As Single
    ry = 380: rx = 1.6 * ry
    Scale (0, ry)-(rx, 0)
    Statica
        MSFlexGrid1Static.ColAlignment(0) = 2
        MSFlexGrid1Static.ColWidth(0) = 1000
        MSFlexGrid1Static.ColAlignment(0) = 2
    For i = 1 To 2
         MSFlexGrid1Static.ColAlignment(i) = 2
    Next i
    For i = 1 To 3
        MSFlexGrid1Static.Col = i - 1
        MSFlexGrid1Static.Row = 0
        MSFlexGrid1Static.Text = tc1(1, i)
     Next i
     For i = 2 To 6
        MSFlexGrid1Static.Col = 0
        MSFlexGrid1Static.Row = i - 1
        MSFlexGrid1Static.Text = tc1(i, 1)
     Next i
     For i = 1 To 5
         MSFlexGrid1Static.Col = 1
         MSFlexGrid1Static.Row = i
         ax = Format(X(i), "###")
         MSFlexGrid1Static.Text = Str$(ax)
         MSFlexGrid1Static.Col = 2
```

```
            ay = Format(Y(i), "###")
            MSFlexGrid1Static.Text = Str$(ay)
        Next i
End Sub
'*********************************************************
' Rem построение механизма в статике
Public Sub Statica()
        Cls
        Ft = Val(TextStatic.Text)
        fi = Ft / 180 * pi
        Raschet
        Postroenie
        Psopor
End Sub
'*************************************************
Private Sub Command2Statica_Click()
      Unload Me
End Sub
```

Текст программы формы Динамика

```
Dim TT As Integer
Dim TT1 As Integer, Scor As Double, FlagDin As Byte
Dim inf1(200) As Integer, inf2(200) As Integer

Private Sub ComDinEnd_Click()
   If FlagDin = 1 Then
      MsgBox "Остановите механизм", , "Динамика"
      Exit Sub
   Else
      Unload Me
    End If
End Sub

Private Sub Form_Click()
    Unload Me
End Sub

Private Sub Form_Load()
    Dim beep1 As Integer, i As Integer
    For i = 1 To 200: inf1(i) = i: inf2(i) = 201 - i: Next i
    Set Obj = FormDinamic
    VvodDannych
    HScroll1Dinamic.Value = 50
    Scor = HScroll1Dinamic.Value * 100
    ComDinStop.Cancel = False
    FlagDin = 1
End Sub
'*********************************************************
Rem построение механизма в динамике
Private Sub Dinamica()
     Dim beep1 As Integer, i As Integer
    Cls
```

```
dx = 0.3: beep1 = 0
dinam2:
Scor = HScroll1Dinamic.Value * 100
If Option1Dinamic = True Then
For fi = 0 To pi2 Step dx
Cls
beep1 = beep1 + 1
Scor = HScroll1Dinamic.Value * 100
If Scor <= 2 Or Scor / 100 > 148 Then
MsgBox "Предельная скорость", , "Динамика"
GoTo dinam1
End If
If ComDinStop.Cancel = True Then FlagDin = 0: Exit Sub
   TT = 0: TT1 = 0:
     Raschet
     Postroenie
     Psopor
     If CheckDinam = Checked Then
       If beep1 = 5 Then
         Beep
       End If
     End If
    While TT < Scor
       Timer1_Timer
       TT = TT + TT1
       DoEvents
    Wend
    For i = 1 To 200
      If Int(TT / 100) = inf1(i) Then
        TextDinamic1.Text = inf2(i)
      End If
    Next i
    If beep1 = 5 Then beep1 = 0
Next fi
End If
If Option2Dinamic = True Then
For fi = 2 * pi To Ft / 180 * pi Step (-dx)
Cls
beep1 = beep1 + 1
Scor = HScroll1Dinamic.Value * 100
If Scor <= 2 Or Scor / 100 > 148 Then
MsgBox "Предельная скорость", , "Динамика"
GoTo dinam1
End If
If ComDinStop.Cancel = True Then FlagDin = 0: Exit Sub
     TT = 0: TT1 = 0
     Raschet
     Postroenie
     Psopor
     If CheckDinam = Checked Then
       If beep1 = 5 Then
```

```
            Beep
          End If
        End If
      While TT < Scor
        Timer1_Timer
        TT = TT + TT1
      DoEvents
      Wend

      For i = 1 To 200
      If Int(TT / 100) = inf1(i) Then
        TextDinamic1.Text = inf2(i)
      End If
      Next i
      If beep1 = 5 Then beep1 = 0
  Next fi
  End If
    GoTo dinam2
  dinam1:
  HScroll1Dinamic.Value = 50
  TextDinamic1.Text = HScroll1Dinamic.Value
End Sub

'*************************************************************************
Private Sub ComDinStart_Click()
   Dim rx As Integer, ry As Integer
   ry = 380: rx = 1.6 * ry
   Scale (0, ry)-(rx, 0)
   Dinamica
End Sub

Private Sub ComDinStop_Click()
  ComDinStop.Cancel = True
End Sub

Rem Таймр
Private Sub Timer1_Timer()
  Static TimerTimes As Integer
  TimerTimes = TimerTimes + 1
  If TimerTimes = 2 Then
    TimerTimes = 0
  Else
    TT1 = TimerTimes
    Exit Sub
  End If
End Sub
```

Текст программы формы определения зон действия механизма

```
Dim k As Byte
```

```
Private Sub Form_Click()
   Unload Me
End Sub

Private Sub Form_Load()
   FormZona.Height = 7095
   Set Obj = FormZona
End Sub

Private Sub MenuZona1_Click()
   Cls
   fl = 0
   fl2 = 1
   Zonazv
End Sub

Private Sub MenuZona2_Click()
   Cls
   fl = 0
   fl2 = 2
   Zonazv
End Sub

Private Sub MenuZona3_Click()
   Cls
   fl = 0
   fl2 = 3
   Zonazv
End Sub

Private Sub MenuZonaEnd_Click()
    Unload Me
End Sub

Private Sub MenuZonaVsego_Click()
    Cls
    fl = 1
    Zonazv
End Sub

Rem определение зоны действия звеньев механизма
Private Sub Zonazv()
    Dim rx As Integer, ry As Integer
    Dim i As Integer, j As Integer, DrawStile As Integer
    Dim maxx0 As Single, minx0 As Single
    Dim maxy0 As Single, miny0 As Single
    VvodDannych
    ry = 380: rx = 1.6 * ry
    Scale (0, ry)-(rx, 0)
    Dim xa(100, 5) As Single, ya(100, 5) As Single
    Dim maxx(5) As Single, maxy(5) As Single
```

```
Dim minx(5) As Single, miny(5) As Single

fi = 0
Raschet
For i = 1 To 5
      If i = 5 Then X(i) = X(i) + l(5)
      maxx(i) = X(i): maxy(i) = Y(i)
      minx(i) = X(i): miny(i) = Y(i)
Next i
i = 0: dx = 0.3
For fi = 0 To pi2 Step dx
      Raschet
      i = i + 1
      For j = 1 To 5
        xa(i, j) = X(j)
        ya(i, j) = Y(j)
        If xa(i, j) > maxx(j) Then maxx(j) = xa(i, j)
        If xa(i, j) < minx(j) Then minx(j) = xa(i, j)
        If ya(i, j) > maxy(j) Then maxy(j) = ya(i, j)
        If ya(i, j) < miny(j) Then miny(j) = ya(i, j)
 Next j
Next fi
If fl = 1 Then GoTo Zona2
Select Case fl2
     Case 1
          DrawStile = 2
          Line (minx(2), maxy(2))-(maxx(2), miny(2)), QBColor(2), B
     Case 2
          DrawStile = 4
          Line (minx(2), maxy(4) + 15)-(maxx(2), miny(2)), QBColor(3), B
     Case 3
          DrawStile = 5
          Line (minx(2), Y(5) + l(5))-(maxx(3), miny(2)), QBColor(5), B
     End Select
     GoTo Zona1
Zona2:
   maxx0 = maxx(1): maxy0 = maxy(1)
   minx0 = minx(1): miny0 = miny(1)
   For j = 1 To 5
        If maxx(j) > maxx0 Then maxx0 = maxx(j)
        If maxy(j) > maxy0 Then maxy0 = maxy(j)
        If minx(j) < minx0 Then minx0 = minx(j)
        If miny(j) < miny0 Then miny0 = miny(j)
   Next j
   DrawStile = 3
   Line (minx0, maxy0 + 15)-(maxx0, miny0), QBColor(4), B
Zona1:
   k = 0
   Ft = Val(TextZonaNach.Text)
   fi = Ft / 180 * pi
   Raschet
```

```
        Postroenie
        Psopor
        k = 1
        Ft = Val(TextZonaCon.Text)
        fi = Ft / 180 * pi
        Raschet
        Postroenie
        Psopor
        Erase xa, ya
        fl = 0
End Sub
```

Текст программы формы демонстрации траектории заданной точки

```
Dim tc(5, 5) As Integer, XT(10) As Integer
Dim tc1(6, 6) As String, Ft As Integer

Private Sub Form_Load()
    Rem таблица связей
    Set Obj = FormTrec
    VvodDannych
     tc1(1, 1) = "Точка": tc1(1, 2) = "1": tc1(1, 3) = "2"
     tc1(1, 4) = "3": tc1(1, 5) = "4": tc1(1, 6) = "5"
     tc1(2, 1) = "O (1)": tc1(3, 1) = "A (2)": tc1(4, 1) = "B (3)":
     tc1(5, 1) = "D (4)": tc1(6, 1) = "C (5)"
     tc(1, 2) = 1: tc(2, 3) = 1: tc(2, 4) = 1: tc(3, 5) = 1
     tc(2, 1) = 1: tc(3, 2) = 1: tc(4, 2) = 1: tc(5, 3) = 1
    Ft = 30
End Sub

Private Sub ComTrecRis_Click()
    Dim rx As Integer, ry As Integer
    Traectorij
    If CheckTrec.Value = Checked Then
            ry = 380: rx = 1.6 * ry
           Scale (0, ry)-(rx, 0)
           fi = Ft / 180 * pi
          Raschet
          Postroenie
          Psopor
     Else
         Exit Sub
    End If
End Sub

Private Sub ComTrec2_Click()
     Unload Me
End Sub
```

```
Private Sub ComTrec1_Click()
    Dim rx As Integer, ry As Integer
    ry = 380: rx = 1.6 * ry
    Scale (0, ry)-(rx, 0)
       Traectorij
       Traectorij1
End Sub

Rem построение траектории движения точки
Private Sub Traectorij()
    Dim i As Integer, j As Integer, k As Integer
    Cls
     MSFlexGridTrec.ColWidth(0) = 1000
     k = (MSFlexGridTrec.Width - 1000) / 5
     MSFlexGridTrec.ColAlignment(0) = 2
     For i = 1 To 5
          MSFlexGridTrec.ColAlignment(i) = 2
          MSFlexGridTrec.ColWidth(i) = k
     Next i
     ' CurrentX = 50: CurrentY = 10
     For i = 1 To 6
           MSFlexGridTrec.Col = i - 1
           MSFlexGridTrec.Row = 0
           MSFlexGridTrec.Text = tc1(1, i)
           'Print MSFlexGridTrec.Text
      Next i
      For i = 2 To 6
           MSFlexGridTrec.Col = 0
           MSFlexGridTrec.Row = i - 1
           MSFlexGridTrec.Text = tc1(i, 1)
           'Print MSFlexGridTrec.Text
       Next i
       For i = 1 To 5
         For j = 1 To 5
            MSFlexGridTrec.Col = i
            MSFlexGridTrec.Row = j
            MSFlexGridTrec.Text = Str$(tc(i, j))
            'Print MSFlexGridTrec.Text
         Next j
      Next i
End Sub

Rem построение траектории движения точки
Private Sub Traectorij1()
    Dim f1 As Single, f2 As Single, i As Integer, j As Integer
    Dim n2 As Integer, n3 As Integer, n1 As Integer
    Dim x1 As Single, y1 As Single
    dx = 0.3
     fi = Val(TextTrecUgol1.Text) / 180 * pi
     f1 = fi
     Raschet
```

```
 Postroenie
 Psopor
traekt1:
If TextTrec1.Text = "" Or TextTrec2.Text = "" _
   Or TextTrec3.Text = "" Then
   DoEvents
   GoTo traekt1
End If
n1 = Val(TextTrec1.Text)
n2 = Val(TextTrec2.Text)
n3 = Val(TextTrec3.Text)
If n1 > 5 Or n2 > 5 Or n1 = 0 Or n2 = 0 Then
   MsgBox "НЕТ ТАКОЙ ТОЧКИ", , "Траектория"
     TextTrec1.Text = ""
     TextTrec2.Text = ""
     TextTrec3.Text = ""
     GoTo traekt1
End If

For i = 1 To 5
  For j = 1 To 5
    If tc(n1, n2) = 0 Then
      MsgBox "НЕТ СВЯЗИ МЕЖДУ ТОЧКАМИ", , "Траектория"
      TextTrec1.Text = ""
      TextTrec2.Text = ""
      TextTrec3.Text = ""
      GoTo traekt1
    End If
  Next j
Next i
   fi = Val(TextTrecUgol2.Text) / 180 * pi
   f2 = fi
   Raschet
   Postroenie
   Psopor
   'DoEvents
For fi = f1 To f2 Step dx / 4
  Raschet
  x1 = X(n1) + n3 / 100 * (X(n2) - X(n1))
  y1 = Y(n1) + n3 / 100 * (Y(n2) - Y(n1))
  If fi = f1 Then
     CurrentX = x1 + 10: CurrentY = y1 + 5
     Print "F"
  End If
Circle (x1, y1), 3, QBColor(12)

   Next fi
End Sub
```

Паспорт программ

Private Sub Form_Load()

```
    Picture1.AutoRedraw = True
    Picture1.Height = 7800
End Sub

Private Sub Picture1_Click()
  Unload Me
End Sub

Private Sub Picture1_KeyPress(KeyAscii As Integer)
  If Str(KeyAscii) <> "" Then
    Unload Me
  End If
End Sub
```

П4.3. Список использованной литературы

1. Новожилов И. В. , Зацепин М. Ф. Типовые расчеты по теоретической механике на базе ЭВМ. - М.: Высшая школа, 1986. - 136 с.: ил.

2. Дьяконов В. П. Применение персональных ЭВМ и программирование на языке Бейсик. - М.: Радио и связь, 1989. - 288 с.: ил.

3. Вычислительная техника, программирование и математическое моделирование: методические указания - Брест, БрПИ, 1991. - 34 с.

Литература

1. Волчёнков Н. Г. Программирование на Visual Basic 6. – М.: ИНФРА, 2000.

2. Гарри Корнель . Программирование в среде VB5, - Мн.:ООО “Папури”, 1998.-608 с.:ил.

3. Михаель Рейтингу, Геральд Муч. Visual Basic 6.0-К.:Издательская группа BHV, 2000-288 с.: ил.

4. Microsoft Corporation. разработка приложений на VB 6.0 - М.: Издательско-торговый дом “Русская Редакция”, 2000 - 400 с.ил.

5. Брайн Сайлер и Джефф Скоттс. Использование VB 6.0 - М.: СПб.; К.: Издательский дом “Вильямс”, 2001.- 832 с.: ил.