Министерство образования Республики Беларусь

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Кафедра «Основы бизнеса»

РАЗРАБОТКА ПРИЛОЖЕНИЙ В СРЕДЕ VISUAL BASIC

Лабораторный практикум

по дисциплине «Информатика» для студентов специальностей

1-36 20 03 «Торговое оборудование и технологии»,

1-52 04 01 «Производство экспозиционно-рекламных объектов»

Учебное электронное издание

Минск 2010

УДК 004.67(076.5)
ББК 32.97

А в т о р :

И.Е. Ругалёва

Р е ц е н з е н т ы :

И.И. Краснова, доцент кафедры «Экономика и управление на транспорте» БНТУ, кандидат экономических наук, доцент;

С.С. Карпович, и. о. заведующего кафедрой «Новые материалы и технологии» ИПК и ПК БНТУ, кандидат технических наук.

Лабораторный практикум содержит краткие теоретические сведения, задания, примеры и рекомендации для выполнения лабораторных работ на языке Visual Basic 6.0, рассмотрена среда программирования Visual Basic 6.0, принципы построения программ и схем алгоритмов, технология разработки приложений в VB6.0. Изложены основные понятия и конструкции языка программирования Visual Basic 6.0.

Белорусский национальный технический университет
пр-т Независимости, 65, г. Минск, Республика Беларусь
тел. (017) 293 91 97
Регистрационный номер № БНТУ/ФММП51-18.2010

ОГЛАВЛЕНИЕ

Лабораторная работа № 1

Работа в среде Visual Basic. Изучение среды разработки проекта

Цель: изучить структуру интегрированной среды разработки приложений Visual Basic, приемы разработки макета формы приложения, порядок установки элементов на форму; приобрести начальные навыки с основными компонентами работы в среде Visual Basic.

1.1. Методические указания

Запуск Visual Basic и создание нового проекта (шаблон Standard EXE)

1 способ:

Запустить программу Visual Basic 6.0 можно: из главного меню Windows; используя ярлык на рабочем столе; выполнив файл VB6.EXE.

2 способ:

1. Нажмите кнопку **Пуск**, расположенную на **панели задач Windows**.
2. В открывшемся **главном меню Windows**:

- выберите подменю **Программы**;
- в нем Microsoft Visual Basic 6.0 (или Microsoft Visual Studio 6.0);
- затем выберите Microsoft Visual Basic 6.0.

При запуске Visual Basic 6.0 (в дальнейшем изложении – Visual Basic, без указания версии) на экране появляется диалоговое окно *New Project (Новый проект),* используя которое можно:

- создать новый проект, используя шаблоны создания проекта (вкладка ***New***);

- создать новый проект, используя мастера создания проекта (вкладка ***New***);

- открыть ранее созданный проект (вкладки ***Existing и Recent***).

Окно создания проекта содержит три вкладки следующего назначения (рис. 1):

1. **New** (*Новый*) — содержит шаблоны и мастера для создания нового проекта;

2. **Existing** (*Существующий*) — позволяет открыть ранее созданный проект или проекты-примеры, поставляемые с Visual Basic. Вкладка имеет раскрывающийся список, с помощью которого можно выбрать любую папку на имеющихся ресурсах компьютера;

3. **Recent** (*Недавно созданный*) — содержит список проектов, открывавшихся.

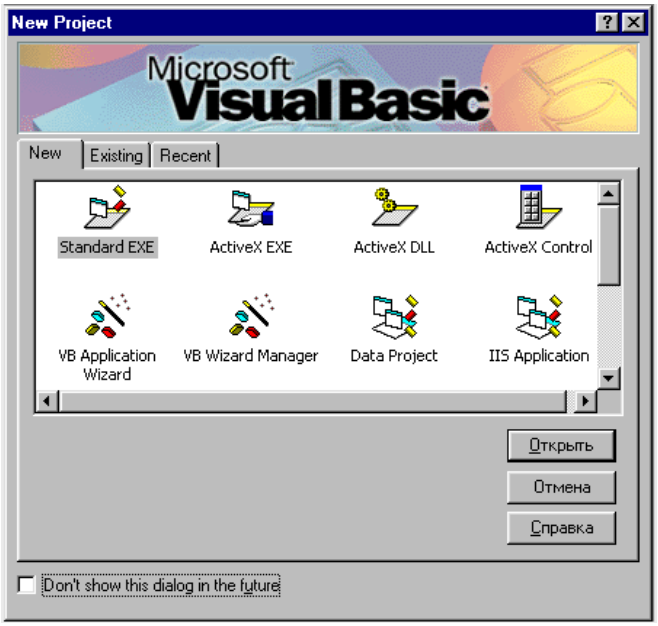


Рис. 1. Окно создания проекта

Выключенный ключ *Don't show dialog in the future* (не показывать этот диалог в будущем) означает, что это окно будет появляться всегда при запуске Visual Basic

Включенный ключ *Don't show dialog in the future* (не показывать этот диалог в будущем) означает, что это окно не будет появляться при запуске Visual Basic. **НЕ включайте этот ключ.**

Типы шаблонов и мастеров проекта

- Standard EXE — стандартное выполняемое приложение;
- ActiveX EXE — выполняемое приложение ActiveX;
- ActiveX DLL — динамическая библиотека ActiveX;
- ActiveX Control — элемент управления ActiveX;
- VB Application Wizard — мастер приложений;
- Wizard Manager — мастер создания пользовательских мастеров;
- Data Project — проект управления базой данных;
- IIS Application — приложение, размещаемот на сервере Web-узла (IIS — Internet Information Server);
- Addin — надстройка, дополнительные утилиты, расширяющие возможности приложений;
- ActiveX Document DLL — динамическая библиотека документов ActiveX;
- ActiveX Document EXE — выполняемое приложение документов ActiveX;
- DHTML Application — приложение, создающее динамические HTML-страницы.

Основные компоненты интерфейса Visual Basic

Рабочее окно представляет собой интегрированную среду разработки – интерфейс языка программирования 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. Большинство элементов рабочего окна, характерны для приложений Windows: заголовок, меню, панели инструментов. Интегрированная среда разработки проекта многооконная (рис. 2).

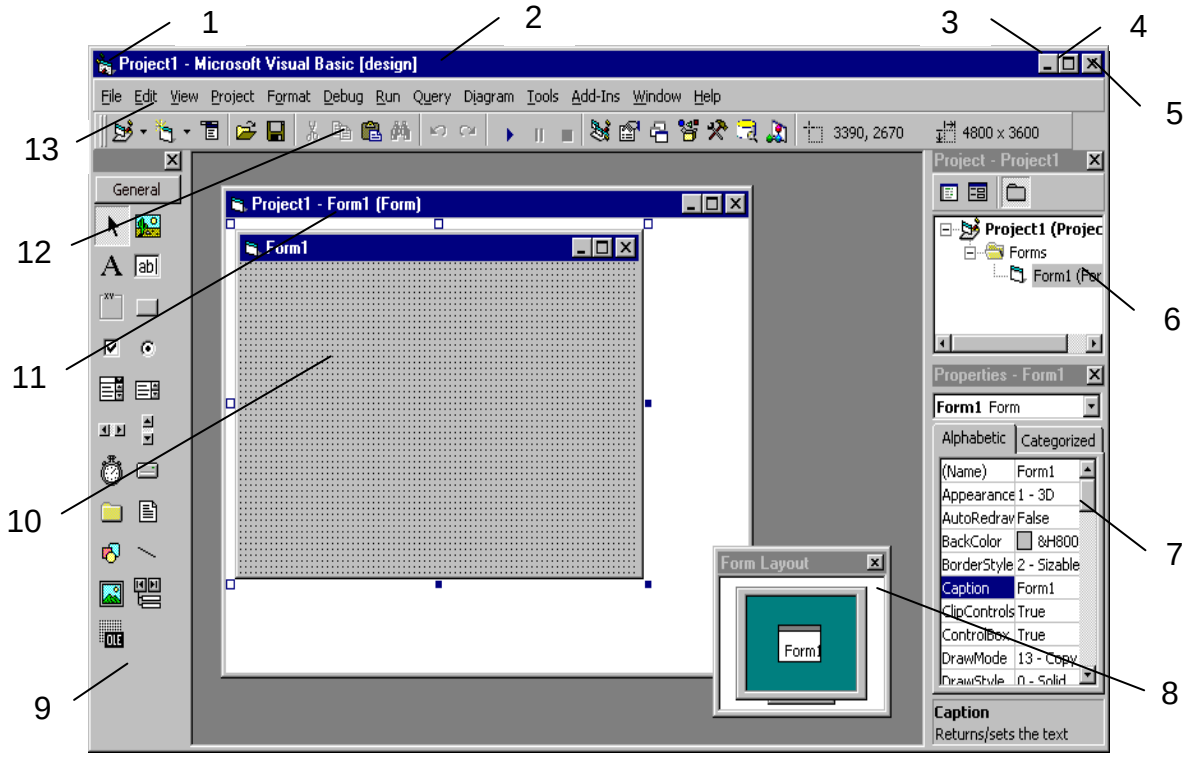


Рис. 2. Окно программы Visual Basic

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 – шаблон формы;
11 – конструктор форм;
12 – стандартная панель управления формы;
13 – главное меню.

Главное меню (текстовое меню), как и во всех приложениях Microsoft, представляет собой линейку раскрывающихся меню, содержащее следующие основные команды: **File** (Файл), **Edit** (Правка), **View** (Вид), **Project** (Проект), **Format** (Формат), **Debug** (Отладка), **Run** (Запуск), **Query** (Запрос), **Diagram** (Диаграмма), **Tools** (Сервис), **Add-Ins** (Надстройки), **Window** (Окно), **Help** (Справка).

Наиболее часто используемые команды меню отображены в виде кнопок со значками на панели инструментов (пиктографическом меню), размещенной ниже меню (рис. 3).

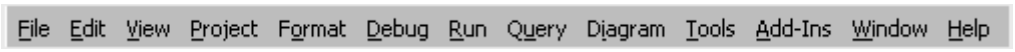


Рис. 3. Текстовое меню

Меню (Menu) содержит список команд, предназначенных для управления разработкой проекта, пункты меню могут иметь 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 6.0 имеет четыре стандартные панели инструментов: *Standard* – стандартная, *Edit* – редактирования, *Debug* – отладки и *Form Editor* – редактор форм.

Стандартная панель инструментов

Команды, которые часто используются при работе, можно разместить в виде кнопок на панели инструментов. Можно также исключить из панели редко применяемые команды. Стандартная панель инструментов, расположенная под главным меню, показана в том виде, в каком она настроена при установке *Visual Basic*.

Стандартная панель инструментов содержит команды, дублирующие основные команды меню (рис. 4).

Рис. 4. Основные команды стандартной панели

Если эта панель отсутствует в главном окне программы, для ее отображения в меню *View* (*Вид*) выберите команду *Toolbars* (*Панели инструментов*), а затем значение *Standard* (*Стандартная*).

На стандартной панели инструментов расположены кнопки для вызова наиболее часто употребляемых команд меню. Вы можете изменить расположение стандартной панели инструментов, разместив ее в нижней части главного окна, справа или слева. Для этого с помощью кнопки мыши захватите любую границу панели и переместите ее в любое удобное для вас место.

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 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*.

Основными рабочими элементами среды Visual Basic, с помощью которых выполняется визуальное проектирование приложения являются окно конструктора форм и панель элементов управления.

Окно конструктора форм

В окне конструктора форм визуально конструируются все формы приложения с использованием инструментария среды разработки. Окно конструктора форм открывается автоматически после загрузки среды Visual Basic (рис. 5).

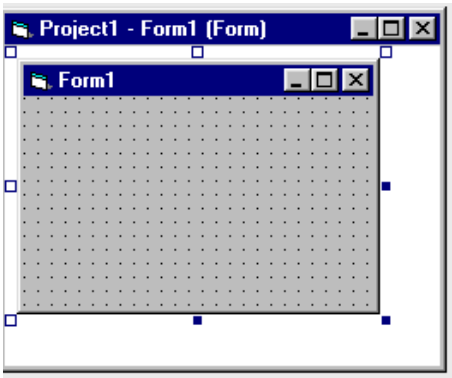


Рис. 5. Окно конструктора форм

Если окно закрыто, вызвать его можно из главного меню командой *Object (Объект)* меню *View (Вид)* или командой *View Object* контекстного меню объекта, находящегося в группе Forms в проводнике проекта.

Для точного позиционирования объектов в форме в окне имеется сетка. Размер ячеек сетки можно менять. При необходимости сетку можно отключать, воспользовавшись параметрами диалогового окна *Options*, открываемого командой *Options (Параметры*) из меню *View (Вид)*.

Размер формы в окне можно изменять, используя маркеры выделения формы и мышь. Для изменения размера формы необходимо установить указатель мыши на маркер и, когда он примет вид двунаправленной стрелки, перемещать до получения требуемого размера.

Панель элементов управления

Панель элементов управления открывается автоматически после загрузки среды *Visual Basic*. Если панель элементов закрыта, вызывать ее можно из меню *View (Вид)* командой *Toolbox (Панель элементов управления*) или воспользоваться кнопкой *Toolbox* на стандартной панели инструментов.

В составе панели элементов управления содержатся основные элементы (компоненты), размещаемые на форме при ее проектировании — метки, текстовые поля, кнопки, списки и другие элементы для быстрого визуального проектирования макета формы (рис. 6).

Рис. 6. Панель элементов управления

Для размещения элементов управления в форме с помощью панели элементов выполните следующие действия:

1. Выделите требуемый элемент управления с помощью мыши.

2. Перейдите в окно конструктора форм. Указатель мыши при этом превратится в крестик, при помощи которого можно установить местоположение размещаемого объекта. Левой кнопкой мыши зафиксируйте позицию нового объекта и, удерживая кнопку, задайте размеры объекта.

Поместить на панель элементов управления компоненты можно используя пункт *Components...(Компоненты*) (рис. 7) контекстного меню, вызываемого правой кнопкой мыши при нахождении курсора на свободной области панели элементов управления. При этом откроется окно выбора компонентов (рис. 8).

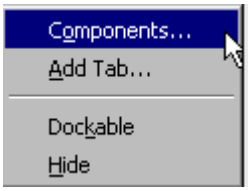


Рис. 7. Пункт Components...(Компоненты)

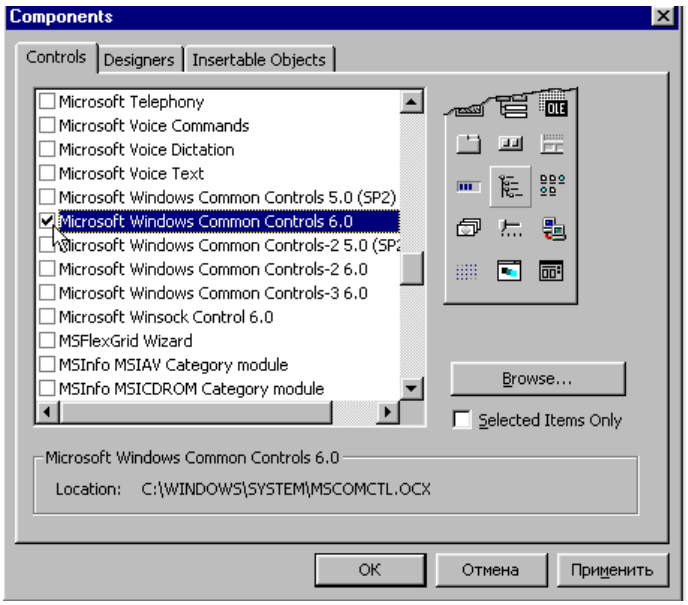


Рис. 8. Окно выбора компонентов

Окно выбора компонентов содержит три вкладки:

Controls (Средства управления) – позволяет выбрать компоненты, размещаемые на панели элементов управления. Компоненты хранятся в файлах, имеющих расширение .OCX или в файлах динамических библиотек .DLL. Выбрать файлы можно также вручную, используя кнопку *Browse... (Просмотр)*

Designers (Проектировщики) – позволяет подключить динамические библиотеки, необходимые для работы создаваемого в среде Visual Basic приложения.

Insertable Objects (Встраиваемые объекты) – позволяет выбрать компоненты, представляющие собой OLE – объекты или программные приложения. Компоненты хранятся в файлах, имеющих расширение .OCX, в файлах динамических библиотек .DLL или в исполняемых файлах .EXE.

Ключ *Select Items Only (Только выбранные пункты)* позволяет исключить из списка компонентов пункты, не отмеченные ключами выбора.

Форма (Form)

Создаваемые в *Visual Basic* окна называются *формами*. ***Форма*** – это основное окно интерфейса разрабатываемой программы, форма – это также основа для создания окон диалога.

Окно Проект (Project)

В окне проекта (броузер проектов) (рис. 9.) отображаются все элементы приложения: формы, модули, классы и т.п., сгруппированные по категориям.

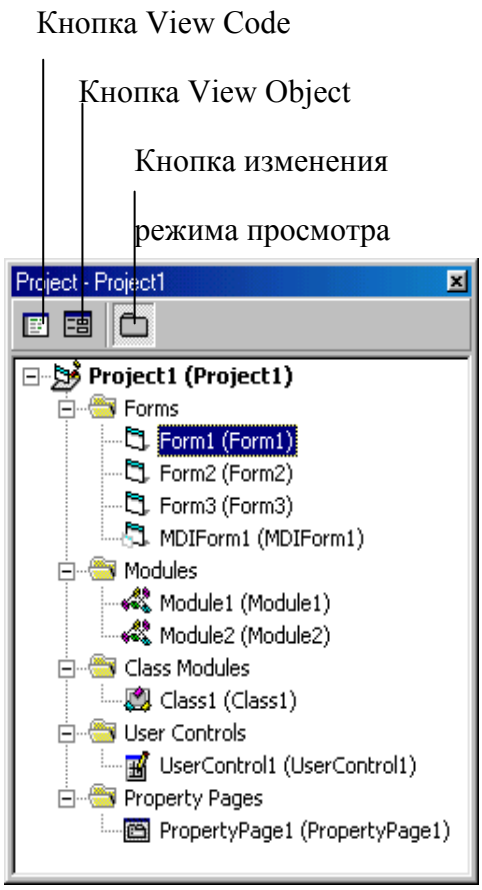


Рис. 9. Окно Проект

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; кнопка изменения режима просмотра: в виде списка компонентов или в виде дерева групп компонентов. В качестве объектов могут быть формы, MDI-формы, модули, классы, управляющие элементы, страницы свойств. Модули и классы не имеют визуальных компонентов.

Окно Свойства (Properties)

В окне свойств (рис. 10) задаются свойства выбранного элемента управления.

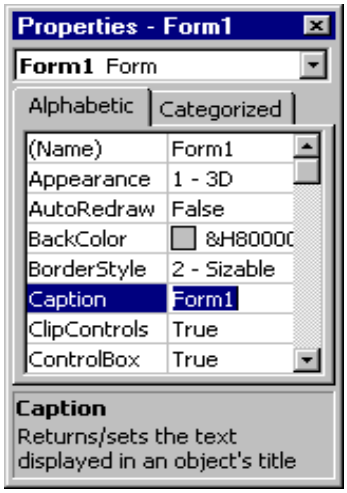


Рис. 10. Окно

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.

Для установки свойств объекта имеется три способа:

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

Окно Программа (Code)

Сразу после запуска окно Программа не отображается. Текст программы в VB разделяется на части – подпрограммы и записывается в процедуры обработки событий или процедуры пользователя. Поэтому текст программы, 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. 11) 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).

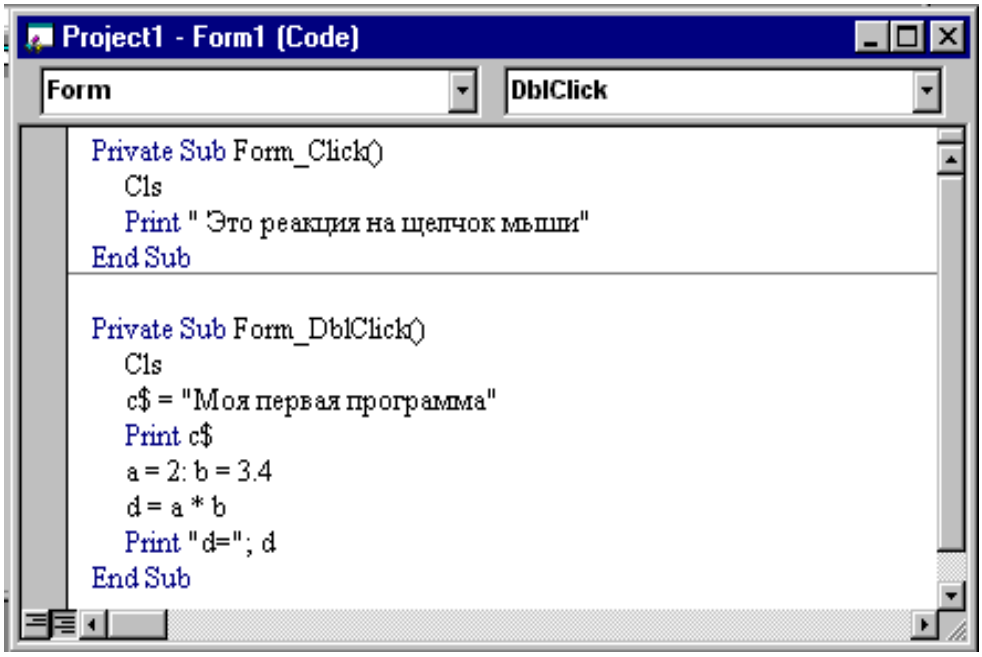


Рис. 11. Окно Программы (Code)

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.

Запуск созданного приложения

Для запуска (выполнения) созданного в Visual Basic приложения можно:

– выполнить команду **Start**, расположенную в пункте **Run** (Выполнить) главного меню;

– нажать кнопку **Start** на стандартной панели инструментов Visual Basic;

– нажать клавишу <**F5**> на клавиатуре ПК.

После этого приложение начнет выполнять запрограммированные в нем действия.

1.2. Задания

1. Запустите Visual Basic, используя один из приемов запуска приложений в Windows.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 и *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. Создайте новый проект приложения, выбрав в окне создания проекта вкладку *New* шаблон *Standard EXE*. Изучите среду разработки проекта. 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 (среда разработки программы) и команды меню. Измените размеры и положение формы в окне, перетаскивая его мышью. Используя пиктографическое меню и пункт *View* (Вид) главного меню, откройте основные рабочие окна среды *Visual Basic* и расположите их на экране так, чтобы они не перекрывали друг друга.

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  и  .

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. Откройте новую форму. Присвойте имя, наименование, установите размеры формы и ее положение в проекте.

9. Запустите программу командой *Run/ Start*. Появится пустая форма.

10. Закройте проект. Установите положение формы в проекте с помощью окна *Form Layout* или  . Снова запустите проект и проверьте положение формы в окне проекта.

11. Разместите на панели элементов управления компоненты *Microsoft Windows Commom Controls 6.0.* Установите на форму пять меток и пять окон ввода (*Label, TextBox*). Опишите в таблице их свойства и присвойте значения этих свойств элементам управления (табл. 1).

Таблица 1

Тип объекта	**Имя (Name)**	**Наименование (Caption)**	**Положение верхнего левого угла**		**Размеры**	
			сверху (Top)	**Слева (Left)**	**ширина (Width)**	**высота (Height)**
Метка (*Label*)	1	Label1				
Окно ввода (*TextBox*)	1					

Можно также добавить такие свойства как цвет, шрифт и т.п.

12. Изучите приемы установки элементов управления на форму и управления размещением элементов управления на форме.

13. Упорядочите элементы управления на форме с помощью опций команды главного меню *Format*.

14. Установите на форму рамку (*Frame*). Скопируйте в нее несколько элементов управления с формы, используя контекстное меню, и выровняйте их по вертикали.

15. 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.

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

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

18. Откройте новый проект командой *File\New Project*.

19. Откройте сохраненный ранее проект командой *File\Open Project.*

20. 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).

21. Выйдите из программы *Visual Basic*.

1.3. Порядок выполнения работы

Выполните последовательно все пункты лабораторной работы. По ходу выполнения задания оформите отчет, отражая в нем порядок выполнения задания, вводимые команды и результаты выполнения команд. Вводимые команды описывать, например, следующим образом: *Файл\Сохранить,* указать диск, маршрут и имя файла.

1.4. Контрольные вопросы

1. Назовите основные элементы рабочего окна VB.Как установить элементы управления на форму?
2. Какая команда используется для управления размещением элементов управления на форме?
3. Как добавить новые элементы управления на панель инструментов *Toolbox*?
4. Каково назначение окон *Проект, Свойства, Программа*?
5. Как вызвать окно Программа?
6. Где записывается текст программы объекта?
7. Как вывести на печать сведения об установленных свойствах формы?

Лабораторная работа № 2

Организация ввода-вывода данных

Цель: приобрести первоначальные навыки в разработке программ в среде Visual Basic; изучить основные компоненты интегрированной среды разработки приложений Visual Basic; и приобрести начальные навыки работы в среде при создании простейших приложений.

2.1. Методические указания

В языке Visual Basic для ввода и вывода данных используются *Окно ввода* (*TextBox*), функции *InputBox* и *MsgBox*.

Окно ввода используется как для ввода, так и для вывода данных:

писатель = Text1.Text – переменной “писатель” присваивается значение свойства Text объекта Text1;

физика = Val(Text1.Text) – числовой переменной “физика” присваивается значение свойства *Text* объекта *Text1*.

Свойство *Text* объекта *TextBox* хранит значение в символьном виде, поэтому при присвоении значение свойства *Text* объекта *Text1* переменной оно должно быть преобразовано в переменную числового типа с помощью функции *Val*;

Text1.Text = “Лев Толстой как зеркало русской революции” –

свойству *Text* объекта *Text1* присваивается текстовая константа;

Text1.Text = Str(физика) – свойству *Text* объекта *Text1* присваивается числовая переменная. Для перевода числовой переменной из числовой формы в текстовую используется функция *Str*.

Функция *InputBox* используется для ввода данных в режиме диалога. Простейший формат функции:

<имя переменной> = InputBox(“<текстовое сообщение>”)

писатель = InputBox(“Лев Толстой как зеркало русской революции ”)

физика = Val(InputBox(“Введите оценку по предмету”))

Функция *MsgBox* используется для вывода данных. Простейший формат функции:

MsgBox ”<Текстовое сообщение>”

MsgBox ”Лев Толстой”

MsgBox ”Средний балл успеваемости =” & Str(SB)

В последнем примере к текстовому сообщению, заключенному в кавычки, прибавляется числовая переменная, которая переводится из числового вида в символьный. В качестве оператора сложения символьных переменных используется символ амперсанд – ***“&”***.

2.2. Задания

1. Изучение способов ввода данных в Visual Basic. Создайте новый проект приложения и реализуйте ввод данных с помощью функции *InputBox* и элемента управления *TextBox*. Сохраните проект на диске.

2. Изучение способов вывода результатов в Visual Basic. Создайте форму, разместив на ней компоненты *PictureBox* и *TextBox***.** Выполните вывод значений переменной *на форму*, в окно *PictureBox* и в окно *TextBox* и на метку *Label*.

3. Создание простого приложения. Создайте проект и напишите программу вычисления площади сектора при известных внешнем диаметре *D*, внутреннем диаметре *d* и центральном угле α. Спроектируйте свой, отличающийся от приведенного в примере, программный интерфейс. Используйте различные варианты организации ввода-вывода данных. Для вывода результатов используйте форматный вывод. Сохраните проект на диске.

4. Создать форму для вычисления среднего балла успеваемости студента. Число предметов обучения – четыре. На форме разместить пять окон для ввода/вывода данных (*TextBox*), 6 надписей (*Label*) и две кнопки (*CommandButton*)

4.1. Разработайте проект вычисления среднего балла успеваемости студента согласно примеру (рис. 12).

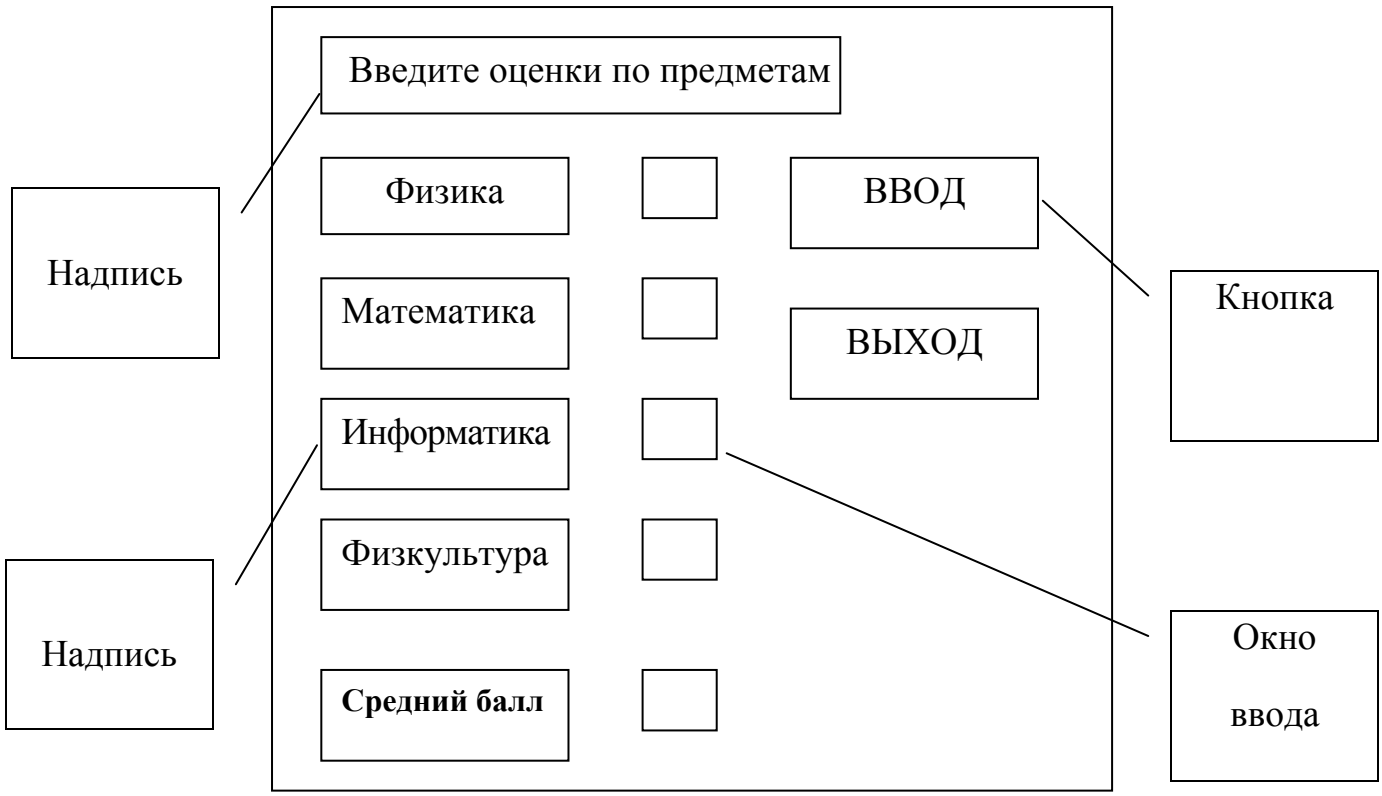


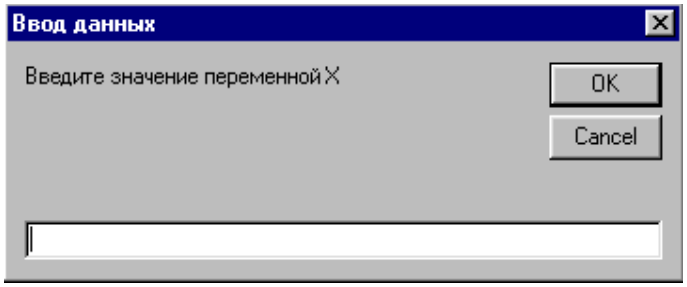
Рис. 12. Пример размещения элементов на форме

4.2. Разработайте форму для вычисления площади поверхности и объема фигуры по заданию преподавателя.

2.3. Порядок выполнения работы

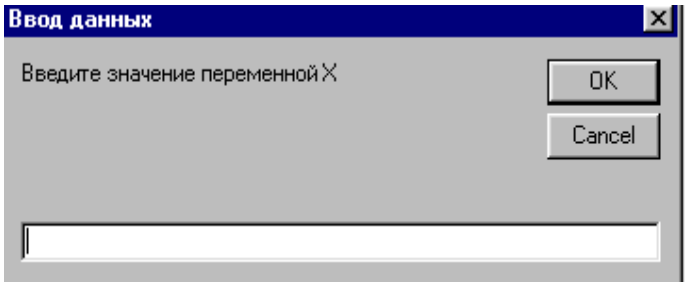
Практически любая создаваемая программа, выполняющая заложенные в ней действия должна получить от пользователя исходные данные для работы. Ввод данных, как правило, осуществляется сразу после запуска программы. Данные могут быть также запрошены для продолжения выполнения программы в процессе ее работы.

Задание 1. Создайте новый проект приложения и реализуйте ввод данных с помощью функции **InputBox** и элемента управления **TextBox.**



1. Реализовать ввод данных можно с помощью функции **InputBox** из модального диалогового окна по запросу приложения сразу после его запуска. При этом не имеет значения, какие объекты расположены на форме, она может быть даже пустой.Для этого необходимо в процедуру формы записать оператор (здесь и далее программный код, который должен быть написаны разработчиком приложения выделен жирным шрифтом):

```
Private Sub Form_Load()
x = InputBox("Введите значение
переменной X", "Ввод данных")
End Sub
```



2. Реализовать ввод данных можно из модального диалогового окна, после наступления некоторого события, активизирующего **InputBox,** например, при нажатии кнопки на форме Для этого необходимо в процедуру командной кнопки записать оператор:

```
Private Sub Command1_Click()
x = InputBox("Введите значение
переменной X", "Ввод данных")
End Sub
```

Функция **InputBox** возвращает строковое значение, поэтому если вы вводите числа, то лучше использовать функцию **Val,** которая преобразует строку в число:

```
x=Val(InputBox("Введите значение
переменной X", "Ввод данных"))
```

3. Реализовать ввод данных можно с помощью элемента управления **TextBox.** Для этого необходимо в процедуру элемента *TextBox* записать оператор:

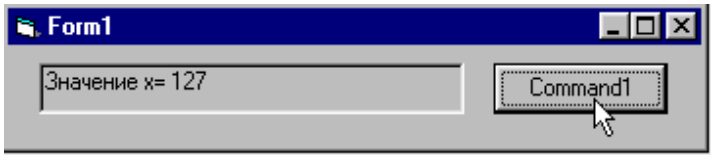
```
Private Sub Command1_Click()
a = Text1
End Sub
```

Задание 2.

1. Организуйте вывод значения переменной на форму с помощью процедуры **Print**.


```
Private Sub Command1_Click()
x = 127
Print "Значение x="; x
End Sub
```

2. Организуйте вывод значения переменной в окно **PictureBox** с использованием метода Print.



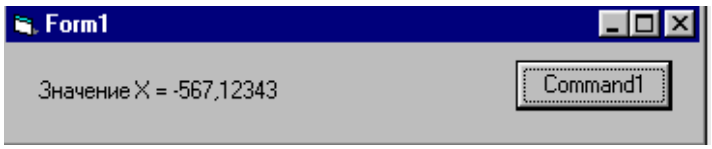

```
Private Sub Command1_Click()
x = 127
Picture1.Print "Значение x=";x
End Sub
```

3. Организуйте вывод значения переменной в окно **TextBox**


```
Private Sub Command1_Click()
x = -326.597
Text1 = x
End Sub
```

4. Организуйте вывод значения переменной на метку **Label**

Форма с размещенной на ней меткой *Label1*




```
Private Su мb Command1_Click()
St = "Значение X = "
X = -567.12343
Label1.Caption = St & X
End Sub
```

Задание 3. Написать программу вычисления площади сектора при известных: внешнем радиусе R, внутреннем радиусе r и центральном угле α. Площадь кольца вычислим по формуле: $S = \frac{\pi \cdot \alpha (R^2 - r^2)}{360}$. Пример формы рабочего окна программы (рис. 13).

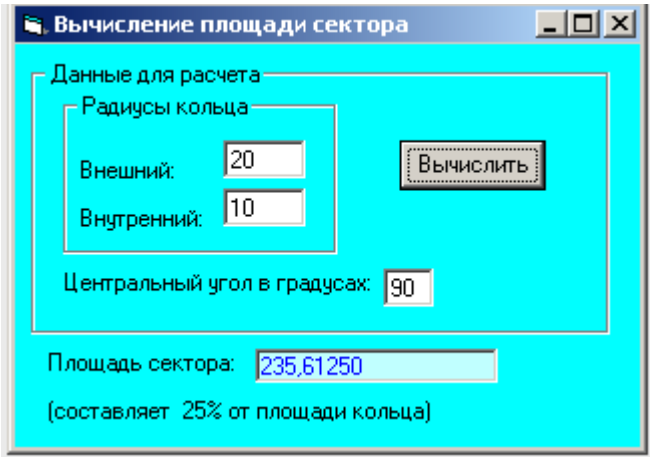


Рис. 13. Окно программы

На форме размещены:

- три текстовых поля (*TextBox*) для ввода исходных данных;
- графическое поле (*PictureBox*) для вывода результатов вычислений (значения площади сектора);
- метки (*Label*) для поясняющих надписей рядом с полями для ввода-вывода данных;
- метки (*Label*) для вывода текста и значения процента площади сектора от площади кольца (вывод данных на метку);
- фреймы (*Frame*) для для функционального объединения однотипных элементов в группы;
- кнопка (*CommandButton*) для запуска программы вычислений.

При создании формы изменены свойства объектов – их цвет и шрифт.

Может быть спроектирован и другой интерфейс программы с использованием различных вариантов оформления формы и организации ввода-вывода данных. Текст программы нахождения площади сектора (рис. 14).

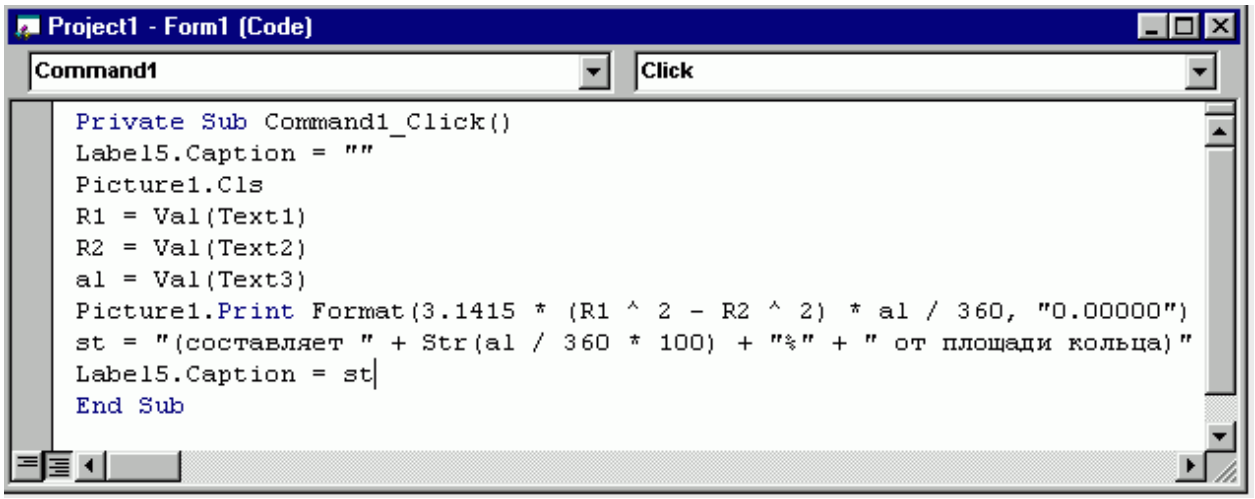


Рис. 14. Текст программы нахождения площади сектора

Задание 4.

1. Запустите программу *Visual Basic 6.*
2. Создайте новый проект командой *File, New Project*.
3. Присвойте форме имя и заголовок (*Name, Caption*) в окне *Свойств* (*Properties*).
4. Разместите на форме надписи (*Label*) и для каждой из них определите свойства:

- имя (*Name*);
- текст выводимый в окно метки (*Caption*);
- выравнивание текста – по центру (*Alignment, Center*).

5. Разместите на форме окна для ввода оценок, используя элемент управления *TextBox.*
6. Для каждого окна определите свойства:

- имя (*Name*);
- начальное значение поля – пусто (*Text*, удалите информацию);

7. Разместите командные кнопки, используя элемент управления *CommandButton.*
8. Для каждой кнопки определите свойства:

- имя (*Name*);
- наименование (*Caption*);

Остальные свойства объектов принимаются по умолчанию.

9. Откройте окно программы (*View Code)* и введите текст программы:
10. Запишите текст программы в обработчик события *Click* для кнопки “ВВОД” (имя кнопки *cmdVvod)*:

- откройте список объектов и выберите объект *cmdVvod*;
- откройте список свойств и методов и выбрите свойство *Click*;
- введите текст программы, например*: fiz = Val(txtFizika.Text)*.

Здесь *fiz* – имя переменной, *Val* – имя функции преобразования символьной переменной в числовую переменную, *txtFizika.Text* – значение свойства *Text* поля ввода *txtFizika* – оценки по физике. Аналогично вводятся оценки и по другим предметам обучения.

- введите формулу для вычисления среднего балла, например:

Sb = (Fiz + Mat + Inf + Fizk)/4

- значение среднего балла присвойте свойству *Text* окна “*Средний балл*” имя окна ввода *txtSrBall:* txtSrBall.Text= Str(Sb).

Функция *Str* преобразует числовую переменную в символьную.

Код программы:

```
Private Sub cmdVvod_Click()
    Cls
    fiz = Val(txtFizika.Text)
    mat = Val(txtMatematika.Text)
    inf = Val(txtInformatika.Text)
    fizk = Val(txtFizkultura.Text)
```

```
    Sb = (fiz + mat + inf + Fizk) / 4
        txtSrBall.Text = Str(Sb)
End Sub
```

11. Напишите текст программы для кнопки “ВЫХОД”, имя кнопки *cmdExit*:

```
Private Sub Д_Click()
    End
End Sub
```

12. Сохраните проект на диске в отдельной папке командой *Save As …,* укажите имя файла, например, *Успеваемость* и щелкните по кнопке *Сохранить*.

13. Запустите программу для отладки *Run/Start*.

14. После отладки сохраните проект.

15. При форматироровании проекта можно использовать такие свойства объектов, как *BackColor* – цвет фона, *ForeColor* – цвет символов в форме или цвет символов объектов *TextBox* и *Label, Font* – шрифт.

16. Сохраните оформленный проект на диске.

17. Выведите текст проекта на печать командой *File/Print,* установите флажки *Current Project* и *Code* и нажмите клавишу *OK***.**

2.4. Контрольные вопросы

1. Перечислите основные свойства *формы*.
2. Перечислите основные свойства *метки* (*Label*).
3. Перечислите основные свойства *окна ввода/вывода (TextBox*).
4. Перечислите основные свойства и события *кнопки (ComandButton*).
5. Приведите пример присвоения значения свойства объекта переменной.
6. Приведите пример присвоения значения переменной свойству объекта.
7. Приведите пример использования функции *InputBox*.
8. Приведите пример использования функции *MsgBox*.
9. Приведите пример использования *Окна ввода для ввода и вывода данных.*

2.5. Задания для самостоятельной работы

1	Вычислить площадь параллелепипеда.	7	Вычислить объем параллелепипеда.
2	Вычислить площадь призмы.	8	Вычислить объем призмы.
3	Вычислить площадь пирамиды.	9	Вычислить объем пирамиды.
4	Вычислить площадь кругового прямого цилиндра.	10	Вычислить объем кругового прямого цилиндра.
5	Вычислить площадь конуса.	11	Вычислить объем конуса.
6	Вычислить объем шара.	12	Вычислить площадь шара.

2.6. Справка

Параллелепипед: $S = 2(ab + bc + ca)$; $V = abc$.

S – площадь;

V – объем;

a, b, c – стороны 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: $S = \pi R(R + l)$; $V = \pi R^2 h/3$.

l – образующая конуса, $l = \sqrt{R^2 + h^2}$.

Шар: $S = 4\pi R^2$; $V = \pi R^3/3$.

Лабораторная работа № 3

Построение схем алгоритмов. Линейные программы

Цель: приобрести опыт самостоятельной разработки схем алгоритмов, закрепить начальные навыки работы в среде VB 6.0 при создании простейших приложений и линейных программ.

3.1. Методические указания

Алгоритмом называется последовательность действий, выполнение которой приводит к решению поставленной задачи.

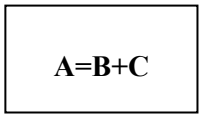
Любой алгоритм может быть задан словесно и графически. Словесное описание алгоритма не дает наглядного представления о порядке выполняемых действий.

При построении графической схемы алгоритма используются специальные блоки, обозначающие характерные виды действий.

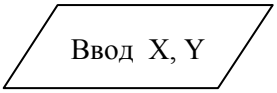
Могут использоваться следующие блоки.

1. **Начало/Конец.** Служит для обозначения точки начала или завершения работы алгоритма.

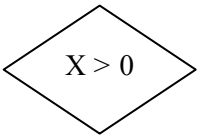
2. **Процесс.** Служит для обозначения одного или нескольких последовательно выполняемых действий.



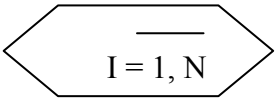
3. **Ввод/вывод данных.** Служит для обозначения действий по вводу или выводу информации без явного указания устройств ввода/вывода.



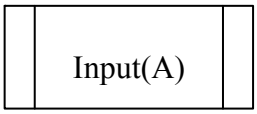
4. **Решение.** Служит для обозначения процессов проверки условий или разветвляющихся процессов.



5. **Модификатор.** Служит для обозначения циклических процессов с заданным количеством итераций.

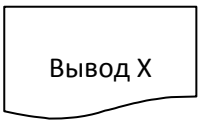


6. **Типовой (предопределенный) процесс.** Служит для обозначения вызовов процедур и функций.

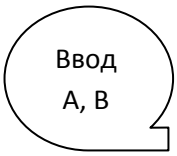


7. **Явное обозначение устройств ввода/вывода.**

Принтер

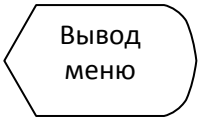


Память с послед. доступом (магнит. лента)



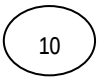
Магнитный диск

Дисплей



Перфокарта

8. **Узел.** Служит для обозначения переходов в пределах одной страницы или объединения нескольких потоков данных. Внутри указывается номер блока, к которому или от которого идет переход.



9. **Межстраничный соединитель.** Служит для обозначения переходов на другую страницу. Внутри указывается номер блока, к которому или от которого идет переход. Справа в качестве комментария указывается номер страницы, к которой или от которой идет переход.

10. **Комментарий.** Служит для вставки пояснительного текста или примечания в схему алгоритма.

При составлении схем алгоритмов блоки могут нумероваться. Номер блока ставится слева вверху над условным обозначением блока.

Направление последовательности выполняемых действий обозначается стрелками, если оно отличается от стандартного. Стандартным считается направление слева направо и сверху вниз.

Стандартом определены пропорции блоков при составлении схем алгоритмов: блоки, описанные в п.п. 2–7, должны вписываться в прямоугольник с соотношением ширины к высоте 3:2. Блок, описанный в п. 1, должен вписываться в прямоугольник с соотношением ширины к высоте 3:1. Размеры комментария произвольные. Остальные блоки должны вписываться в квадрат.

Пример фрагмента схемы алгоритма представлен на рис. 15.

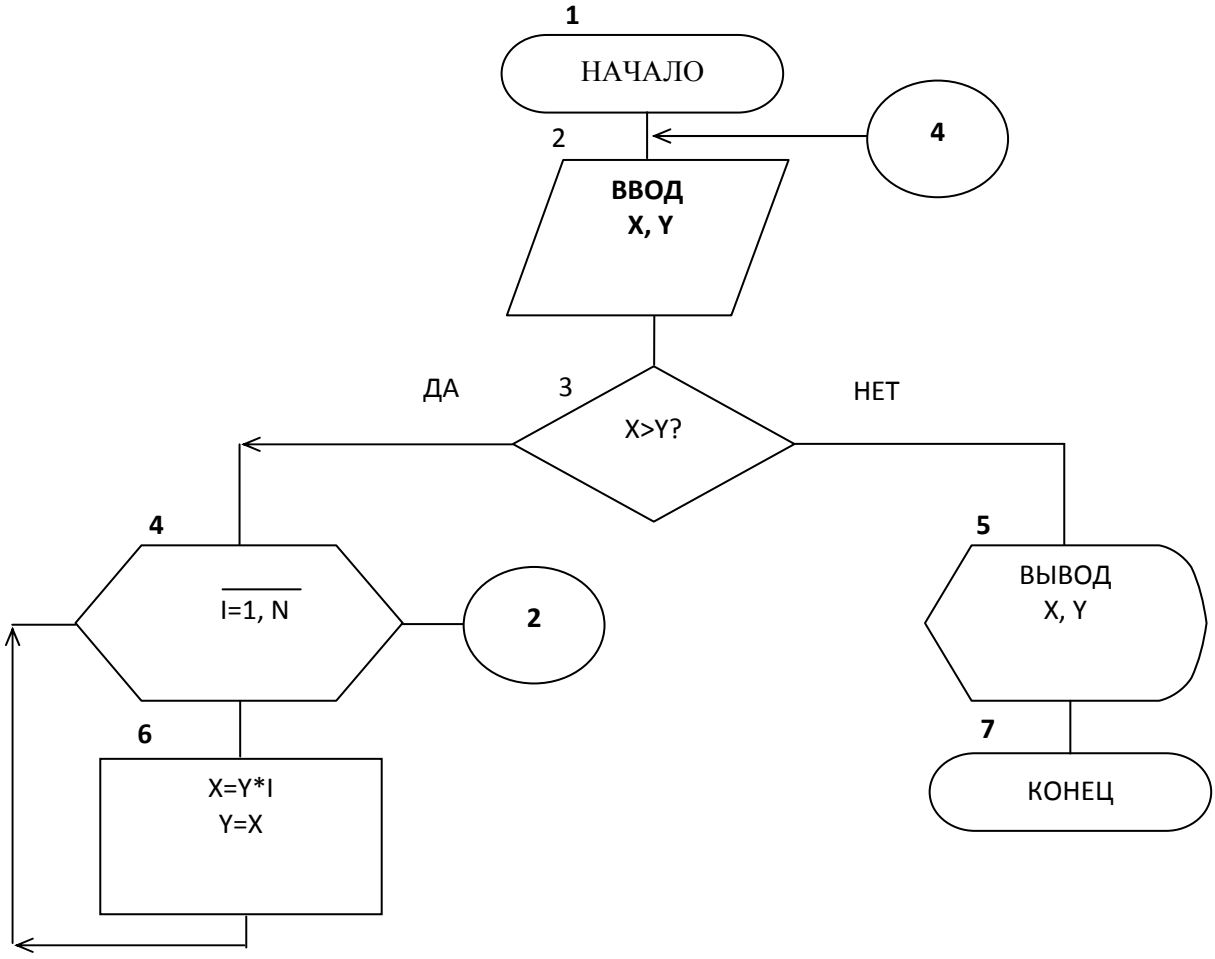


Рис. 15. Пример фрагмента схемы алгоритма

3.2. Задания

Построить блок-схемы следующих алгоритмов:

1. Алгоритма исследования квадратного уравнения $ax^2 + bx + c = 0$.
2. Алгоритм ввода стороны и высоты треугольника, расчет его площади и вывода результатов.
3. Алгоритм принадлежности точки *(x,y)* круговому кольцу с центром в начале координат и внутренним радиусом *r*, а внешним *R*.
4. Алгоритм ввода 10 чисел и определения их четности.
5. Алгоритм подготовки, выполнения и защиты студентом лабораторной работы.
6. Составить блок-схемы алгоритмов решения следующих задач. Записать их в виде исходного кода на языке Visual Basic:

6.1. Написать код ввода данных для вычисления значений выражений по следующим формулам:

f = (m * c * t * b + Abs(c * Sin(t))) ^ 1 / 3

z = m * Cos(b * t * Sin(t)) + c

Организовать вывод результатов (рис. 16) и протестировать для следующих данных: m = 2; c = –1; t = 1.2; b = 0.7.

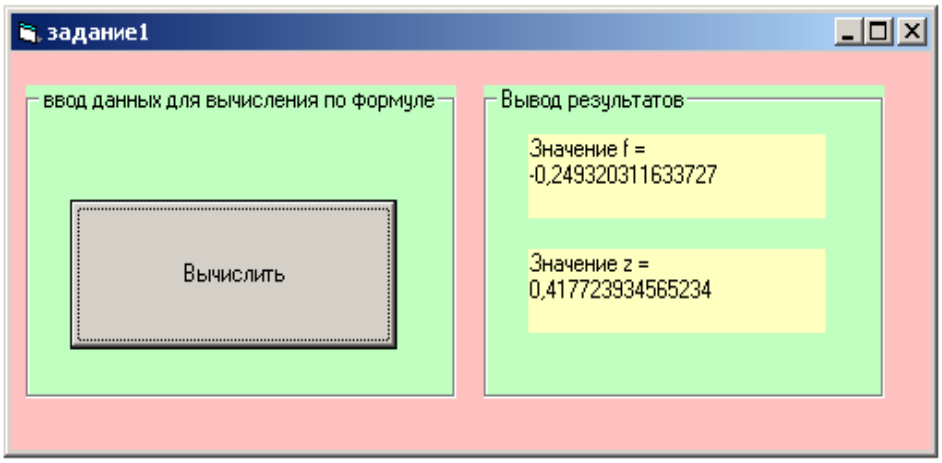


Рис. 16. Окно вывода результатов

6.2. Написать код вычисления длины (рис. 17). Длина выражена в сантиметрах. Выразить ее в дюймах. (1 дюйм = 2,5 см) и заполнить табл. 2.

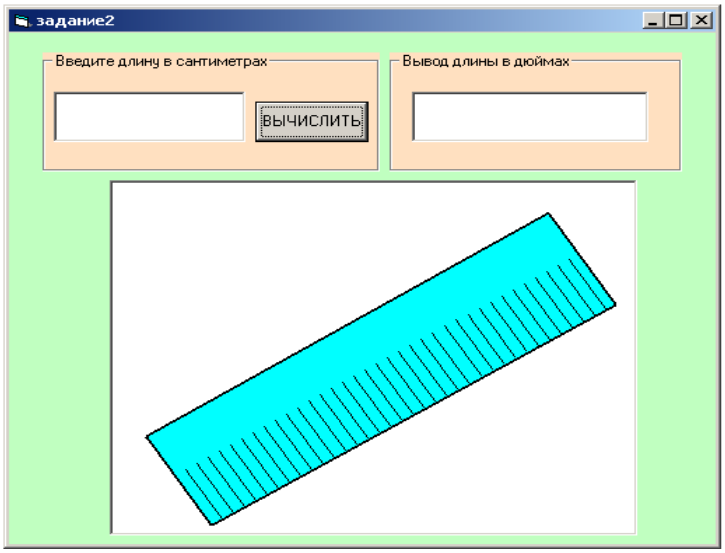


Рис. 17. Окно вывода результатов

Таблица 2

Объект	**Свойства**	**Значение**
Command 1	Caption	Вычислить
Command 2	..	….
Frame1		…
Frame2		…
Frame3		Введите длину в сантиметрах
Frame4		…
Label		…
Label		….
Label		введите обьем воды в 1ом сосуде
Label		….
Form1	Caption	задание1
Form1	BackColour	&H00C0C0FF&

6.3. Для двух целых чисел A и B определить сумму S, разность R и среднее арифметическое SR, создав форму следующего вида (рис. 18). Протестируйте для следующих данных (рис. 19).

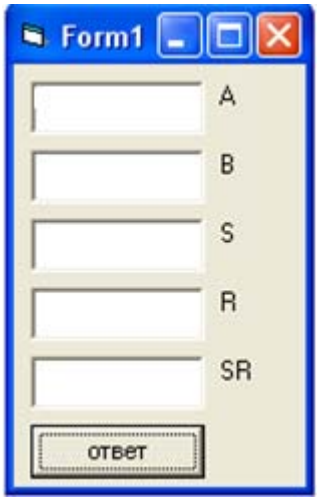


Рис. 18. Окно формы

Результат тестирования для следующих данных (рис. 19)

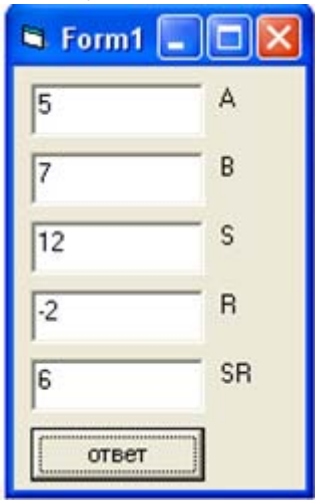


Рис. 19. Окно вывода результатов

3.3. Порядок выполнения работы

Задание 1.

Код программы:

```
Dim m As Integer, c As Single, t As Single, b As Single
m = Val(InputBox("Введите значение переменной m", "Ввод данных"))
c = Val(InputBox("Введите значение переменной c", "Ввод данных"))
t = Val(InputBox("Введите значение переменной t", "Ввод данных"))
b = Val(InputBox("Введите значение переменной b", "Ввод данных"))
f = (m * c * t * b + Abs(c * Sin(t))) ^ 1 / 3
z = m * Cos(b * t * Sin(t)) + c
```

```
St = "Значение f = "
Label1.Caption = St & f
St = "Значение z = "
Label2.Caption = St & z
End Sub
```

Задание 2.

Код программы:

```
Private Sub Command1_Click()
Dim x As Single, y As Integer
x = Val(Text1)
y = x / 2.5
Text2 = Str(y)
End Sub
```

Задание 3.

Код программы:

```
Private Sub Command1_Click()
n = InputBox(“введите A”)
m = InputBox(“введите B”)
a = Val(n)
m = Val(m)
s = a+b
r = a-b
sr = (a+b)/2
Text1. Text=a
Text2. Text=b
Text3. Text=s
Text4. Text=r
Text5. Text=sr
End Sub
```

3.4. Контрольные вопросы

1. Дайте понятие алгоритма, способы и формы его записи.
2. Перечислите основные типы данных.

Лабораторная работа № 4

Форматы вывода объектов. Изменение свойств объектов

Цель: приобрести опыт самостоятельной разработки простейших приложений, изучить правила организации форматного вывода значений.

4.1. Методические указания

Для форматирования данных (выражения) в Visual Basic используется функция *Format*.

Функция *Format* возвращает значение типа *Variant (String)*, содержащее выражение, отформатированное согласно инструкциям, заданным в описании формата

Общий вид функции (в скобках указаны необязательные параметры, которые можно опустить при обращении к функции):

Format (Expression[, Format[, FirstDayOfWeek[, FirstWeekOfYear]]])

Параметры функции

Параметр *Expression*

Обязательный аргумент – любое допустимое выражение, подлежащее форматированию.

Параметр *Format*

Необязательный аргумент – любое допустимое именованное или определяемое пользователем выражение формата. При форматировании чисел без указания аргумента *Format* функция *Format* выдает тот же результат, что и функция *Str*(*number*) – Возвращает строку, представляющую число, хотя эта функция учитывает национальную настройку. Отличие состоит в том, что при преобразовании положительного числа с помощью функции *Format* пробел в начале строки (на месте знака числа) теряется, а при преобразовании с помощью функции Str останется.

Таблица 3

Символы форматирования, применяемые для создания пользовательских форматов числовых величин

Символ	Назначение	Примечание
1	2	3
0	*Прототип цифры*. Выводит цифру или нуль.	Print **Format**(6.789, "00.0000") ' возвратит 06,7890
#	*Прототип цифры*. Выводит цифру или не выводит ничего.	Print **Format**(6.789, "##.0000") ' возвратит 6,7890

1	2	3
.	*Десятичный разделитель*. В некоторых национальных настройках десятичным разделителем служит запятая. Десятичный разделитель указывает, сколько цифр следует вывести в целой и дробной части форматируемого числа, т.е. слева и справа от десятичного разделителя.	Print **Format**(0.789, "##.0000") ' возвратит ,7890
%	*Процентный формат.* Выражение умножается на 100. Символ процентов (%) выводится в позиции, соответствующей позиции прототипа в строке формата	Print **Format**(0.17, "0%") ' возвратит 17%
,	*Разделитель групп разрядов*. В некоторых национальных настройках в качестве разделителя групп разрядов используется точка. Разделитель групп разрядов разделяет позиции тысяч и сотен в числе, целая часть которого состоит из четырех или большего числа цифр.	
:	*Разделитель компонентов времени*. В некоторых национальный настройках в качестве разделителя компонентов времени используется другой символ.	Print **Format**(Now, "h:m:s") ' возвратит к примеру 11:23:35
/	*Разделитель компонентов даты.* В некоторых национальный настройках в качестве разделителя компонентов даты используется другой символ.	
E- E+ e- e+	*Экспоненциальный формат*. Для вывода знака плюс перед положительными значениями показателя степени и знака минус перед отрицательными значениями показателя степени следует использовать символы "E+" или "e+"	
- + $ ()	Вывод указанного символа. Для вывода любого не перечисленных здесь символов следует поместить перед ним символ обратной косой черты (\) или заключить символ в прямые кавычки (" ")	
\	Вывод следующего символа из строки формата. Примерами символов, которые не могут быть включены в строку форматирования явным образом, являются символы форматирования даты и времени (a, c, d, h, m, n, p, q, s, t, w, y, / и :), символы форматирования чисел (#, 0, %, E, e, запятая и точка) и символы форматирования строк (@, &, <, > и !)	
"ABC"	Вывод строки, заключенной в прямые кавычки (" "). Для представления кавычек в аргументе формат в программе необходимо использовать функцию Chr(34) (34 является кодом символа прямых кавычек "))	

Параметр **FirstDayOfWeek**

Необязательный аргумент – константа, определяющая первый день недели.

Параметр **FirstWeekOfYear**

Необязательный аргумент – числовая константа, указывающая, какую неделю считать первой в году.

Ниже приведено окно формы программы, демонстрирующей примеры форматирования числовых данных с использованием функции *Format* (рис. 20).

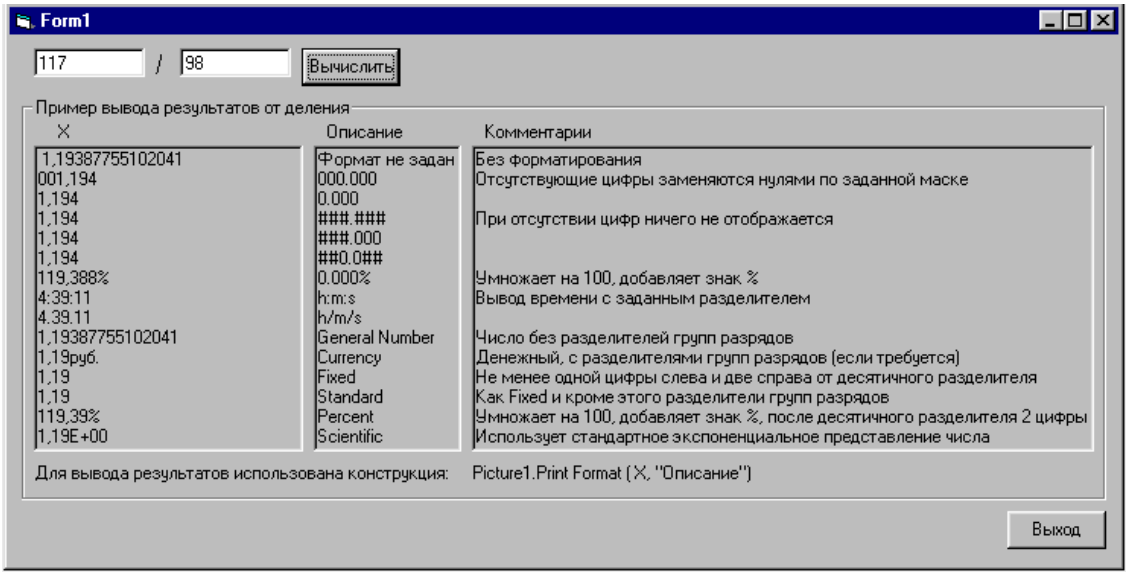


Рис. 20. Окно вывода результатов

Фрагмент текста программы, демонстрирующей примеры форматирования числовых данных с использованием функции *Format* (рис. 21).

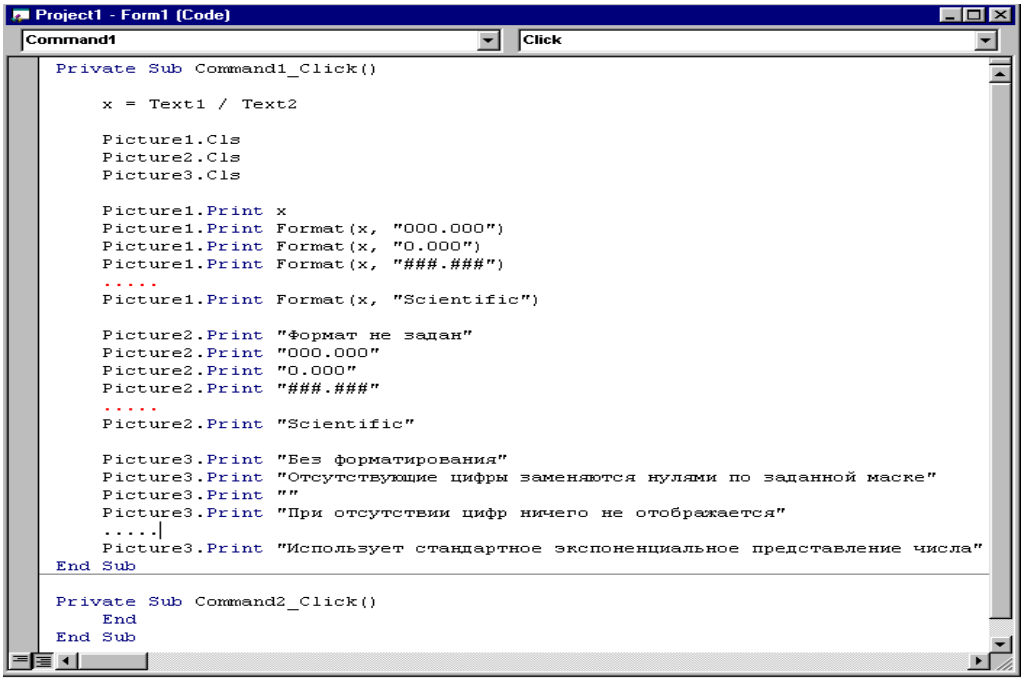


Рис. 21. Окно вывода результатов

Для настройки свойств формы, а также размещенных в ней объектов предназначено окно *Properties (Свойства*) (рис 22). В нем, например, содержатся такие свойства выбранного объекта, как название, высота, ширина, цвет и др.

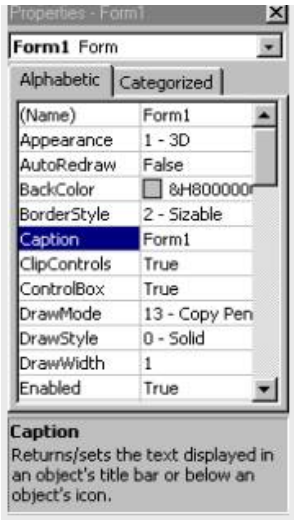


Рис. 22. Окно *Properties (Свойства*)

Диалоговое окно *Properties* вызывается командой *Properties* Window (*Окно свойств*) из меню *View (Вид*), кнопкой *Properties* Window на стандартной панели инструментов или командой *Properties* контекстного меню выбранного объекта. Поскольку форма и элементы управления каждый сами по себе являются объектами, набор свойств в этом окне меняется в зависимости от выбранного объекта. При помощи вкладок *Alphabetic* (*По алфавиту) и Categorized (По категориям*) свойства объекта можно просмотреть в алфавитном порядке или по группам (категориям) соответственно.

В нижней части окна вы всегда найдете подсказку, поясняющую назначение выбранного атрибута объекта. Более подробную информацию найдете в справочной системе Visual Basic 6, нажав клавишу <**Fl**>. При этом необходимо предварительно выделить интересующее вас свойство.

Используя диалоговое окно *Properties*, можно изменить установленные по умолчанию свойства объектов. Часть свойств объекта, например, размеры и расположение объектов, можно задать перемещением объекта и изменением его размеров с помощью мыши в конструкторе форм.

Ряд свойств, установленных на этапе разработки приложения в окне свойств *Properties*, допускается изменять при выполнении приложения, написав соответствующие коды в процедурах, создаваемых с помощью редактора кода.

В окне свойств *Properties* можно воспользоваться вкладкой *Categorized* (*По категориям*), которая выводит свойства, сгруппированные по типам (категориям). Вкладка *Alphabetic* (*По алфавиту*), расположит свойства в

алфавитном порядке их названий, что даст возможность быстрее найти нужное свойство.

Основные группы свойств, представленные на вкладке *Categorized* окна *Properties*.

Свойства, определяющие внешний вид объекта

В группе *Appearance* (*Оформление*) окна *Properties* содержатся свойства объекта, которые задают атрибуты его внешнего вида. Основные свойства данной группы:

Caption - задает текст в строке заголовка объекта;

BorderStyle - задает стиль рамки объекта;

Palette - устанавливает цветовую палитру;

Picture - назначает значок, картинку для объекта. Используя данное свойство формы, можно задать фоновое графическое изображение.

Свойства, определяющие поведение объекта

Свойства объектов, отвечающие за их поведение, собраны в группе *Behavior (Поведение*). *Основные свойства данной группы:*

Causes Validation - устанавливает признак проверки условия достоверности данных при выходе из объекта;

Enabled - разрешает или запрещает доступ к объекту;

MaxLength - устанавливает максимальную длину данных в объекте;

Visible - устанавливает видимость объекта.

Свойства, определяющие шрифт

Группа *Font (Шрифт*) содержит всего одно свойство *Font*, позволяющее с помощью диалогового окна Выбор шрифта задать шрифт текста объекта, размер, начертание. Свойства группы *Misc* задают общие атрибуты объекта, в том числе для его идентификации.

Name - задает имя объекта;

Text - устанавливает текст в поле по умолчанию;

Index - задает уникальный индекс объекта в коллекции;

По имени, указанному в свойстве *Name (Имя)*, объект идентифицируется в форме и в тексте программы. Поэтому необходимо иметь в виду, что в одной форме не может быть двух объектов с одинаковыми именами. По умолчанию это свойство устанавливается автоматически. Вместо имени, заданного по умолчанию, лучше использовать имя, отражающее его смысловое значение.

Свойства позиционирования

Left - задает положение объекта по горизонтальной оси от левого края формы или, в общем случае, от объекта-контейнера;

Top - задает положение объекта по вертикальной оси от его верхнего края до верхней стороны формы;

Width - задает горизонтальный размер (ширину) объекта;

Height - задает вертикальный размер (высоту) объекта.

Группа *Position (Расположение*) окна *Properties* служит для позиционирования объектов в системе координат формы и установки размеров объектов.

Свойства шкалы размеров объекта

Свойства группы *Scale (Масштаб*) устанавливают шкалу максимальных размеров объектов в системе координат формы.

ScaleLeft - задает максимальное положение объекта по горизонтальной оси;

ScaleTop - задает максимальное положение объекта по вертикальной оси;

ScaleWidth - задает максимальный горизонтальный размер (максимальная ширина);

ScaleHeight - задает максимальный вертикальный размер (максимальная высота).

Пример формы с размещенными на ней объектами и измененными свойствами (цвет, шрифт, строка заголовка формы, надписи на объектах и др.) приведен на рис. 23.

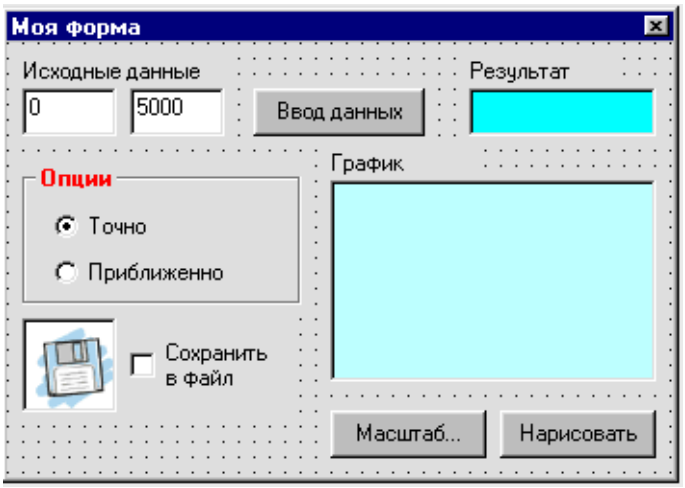


Рис. 23. Окно формы с размещенными на ней объектами

4.2. Задания

1. Пусть смешано V_1 литров воды температуры t_1 с V_2 литрами воды температуры t_2 и V_3 литрами воды температуры t_3 . Вычислить объем и температуру образовавшейся смеси (рис. 24).

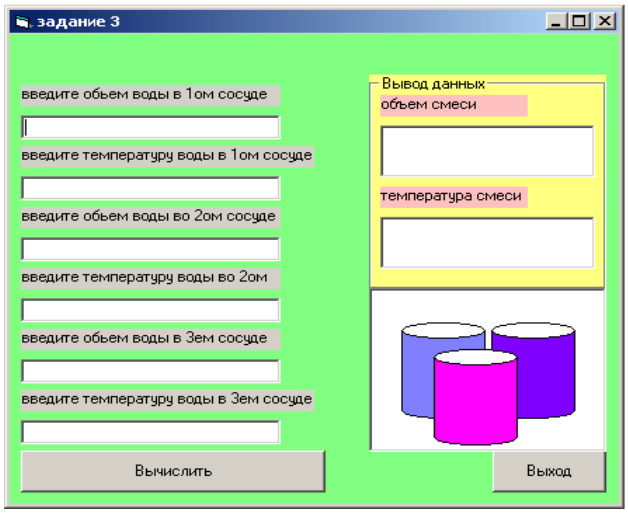


Рис. 24. Окно вывода результатов

Протестируйте для следующих данных (рис. 25)

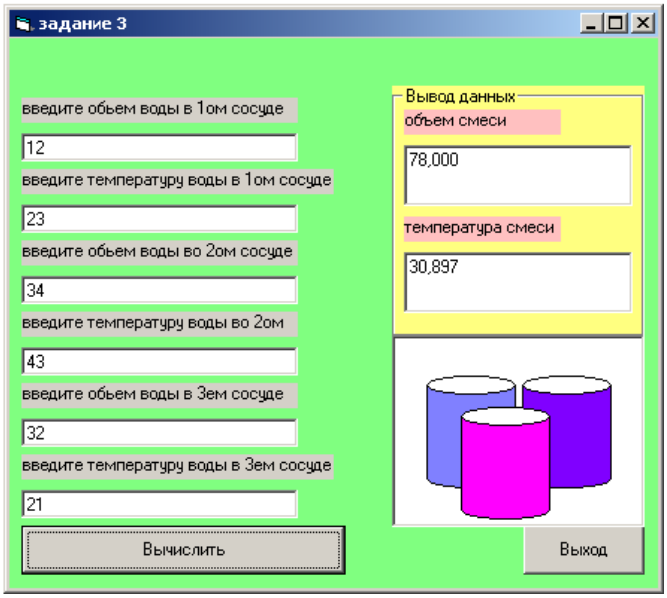


Рис. 25. Окно вывода результатов

2. Поменяйте между собой значения переменных *A* и *B*, воспользовавшись третьей переменной *C*, без использования третьей переменной. Пример формы рабочего окна программы (рис. 26).

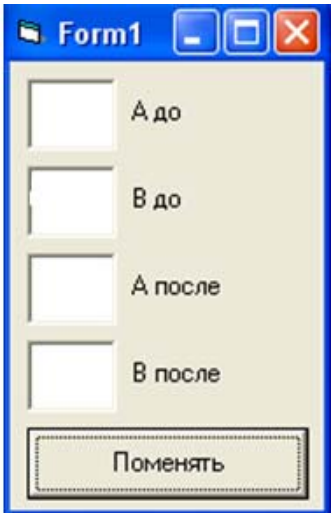


Рис. 26. Окно формы

3. Используя ранее созданный проект, написать код, который по щелчку меняет свойства формы: изменяет цвет, размер, местоположение, заголовок (рис. 27).

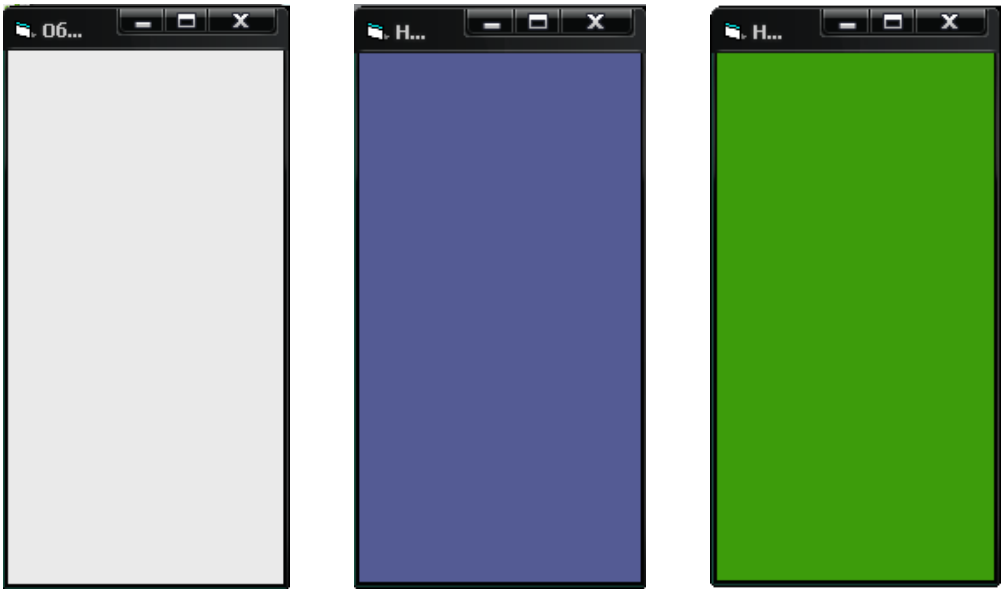


Рис. 27. Окно вывода результатов

4.3. Порядок выполнения работы

Задание 1.

Код программы:

```
Private Sub Command1_Click()
Dim V1 As Integer, t1 As Integer, V2 As Integer, t2 As Integer, V3 As Integer, t3 As Integer
V1 = Val(Text1)
t1 = Val(Text2)
V2 = Val(Text3)
```

```
t2 = Val(Text4)
V3 = Val(Text5)
t3 = Val(Text6)
S = V1 + V2 + V3
t = (V1 * t1 + V2 * t2 + V3 * t3) / (V1 + V2 + V3)
Text7 = Format(S, "#.000")
Text8 = Format(t, "#.000")
End Sub
Private Sub Command2_Click()
End
End Sub
```

Задание 2.

Код программы:

```
Private Sub Command1_Click()
 c=0
a = InputBox("введите число A")
b = InputBox("введите число B")
Text1.Text = a
Text2.Text = b
c=a
a=b
b=c
Text3.Text = a
Text4.Text = b
End Sub
```

Задание 3.

Код программы:

```
Private Sub Form_Click()
Dim intloop As Integer
Form1.DrawStyle = vbInsideSolid
Form1.DrawWidth = 2
Form1.ScaleHeight = 256
For intloop = 0 To 255
Form1.Line (0, intloop)-(Screen.Width, intloop - 1), RGB(0, 0, 255 - intloop), B
Next intloop
End Sub

Private Sub Form_Paint()
Caption = "Цветовая форма"
BackColor = vbWhite * Rnd
End Sub
```

4.4. Контрольные вопросы

1. Для чего функция *Format*?
2. Приведите общий вид функции *Format* и объясните назначение ее параметров?
3. Какие символы форматирования применяются для создания пользовательских форматов числовых величин?

Лабораторная работа № 5

Процедуры обработки событий

Цель: научиться генерировать различные процедуры обработки событий и приобрести начальные навыки работы в среде при создании простейших приложений при обработке событий.

5.1. Методические указания

Процедуры – структурная основная единица любой программы. Вся программа состоит из процедур. Процедуры, записанные в модуле формы, обрабатывают события как самой формы, так и всех входящих в нее элементов управления. В отличие от функции процедура не возвращает значений переменных. Начинается она всегда с ключевого слова *Sub* и заканчивается словом *End Sub*. Процедуры могут вызывать друг друга. Вызов процедурой самой себя (такие процедуры называются рекурсивными) не допускается. Процедуры имеют имена. Имя процедуры формы по умолчанию состоит из имени интерфейса, к которому она относится, и имени события, которое запускает данную процедуру, например, *Private Sub Command1_Click*. Это имя локальной процедуры, привязанной к командной кнопке *Command1* и запускаемой по щелчку мыши на этой кнопке (*Click* – щелчок). Процедура может содержать параметры, но если их нет, как в данном случае, то просто ставятся пустые скобки (они проставляются автоматически самим Visual Basic).

Процедуры модулей классов не привязаны к конкретным событиям, но могут быть вызваны из любой событийной или иной процедуры. В частности, писать их имеет смысл тогда, когда приходится выполнять много однотипных действий с различными исходными данными. Например, необходимо печатать на платежном поручении сумму платежа с прописью. Значения сумм различны, но операция преобразования их в пропись одна и та же. Поэтому удобно создать в модуле класса объект, обладающий способностью преобразовывать числа в пропись и использовать его везде, где требуется эта операция.

Событием называется подпрограмма с особым именем, состоящим из имени элемента, символа "_" и имени события (*Form_Load*). Событие вызывается при изменении в работе элемента управления. В событиях можно изменять свойства других элементов управления.

Например:

Private Sub Form_Click ()

... " Обработка события щелчка мыщью на форме.

End Sub

Методы обработки событий элементами управления: щелчок (*Click*), двойной щелчок (*DblClick*), кнопка в фокусе (*GotFocus*), кнопка вне фокуса (*LostFocus*).

События

Система Windows генерирует события (сообщения для приложения), которые предназначены для управления работой приложения. Приложение может реагировать или не реагировать на событие. Для того чтобы приложение реагировало на событие, необходимо поместить программный код в заготовке соответствующей событийной процедуры. Некоторые события поддерживают почти все управляющие элементы. Такими событиями, например, являются:

- *GotFocus (Получил фокус)* – генерируется в момент получения объектом фокуса клавиатуры. Пока объект обладает фокусом, все события клавиатуры происходят для него;
- *LostFocus (Потерял фокус)* – генерируется, когда фокус перемещается на другой объект.

Форма поддерживает события, некоторые из которых используются особенно часто. Приведем несколько примеров событий формы в табл. 4.

Таблица 4

Событие	Возникает
Activate (Активизировать)	Когда форма становится активной, т. е. отображается на экране
Deactivate (Активизировать)	Когда форма становится неактивной. Например, при активизации на экране другой формы
Initialize (Инициализация)	При создании объекта типа форма
Load (Загрузка)	В момент загрузки формы в память
Paint	При перерисовывании формой своего содержимого
QueryUnlload	Перед выгрузкой формы и используется для отмены выгрузки формы
Resize (Изменить размер)	При изменении размера формы
Terminate (Завершение)	В момент удаления формы
Unload (Выгрузка)	В момент выгрузки формы из памяти

Источником некоторых событий может быть мышь (табл. 5.).

Таблица 5

Событие	Возникает
1	2
Click	При нажатии и отпускании кнопки мыши (щелчок)
DragDrop	При окончании перетаскивания объекта
DragOver	При перетаскивании объекта
MouseDown	При нажатии кнопки мыши. Аргументы событийной процедуры позволяют определить, какая из кнопок мыши была нажата и не происходит ли это при нажатой клавише Shift или Ctrl или Alt
MouseMove	При перемещении мыши
MouseUp	При отпускании ранее нажатой кнопки мыши

Клавиатура также является источником событий. События клавиатуры (табл. 6.) происходят для того управляющего элемента, который в данный момент обладает фокусом клавиатуры. В случае, если ни один управляющий элемент не обладает фокусом, это событие получит форма. Но если свойству *KeyPreview* формы присвоить значение *True* (по умолчанию оно имеет значение *False*), то каждое событие клавиатуры будет происходить сначала для формы, затем для управляющего элемента, обладающего фокусом клавиатуры.

Таблица 6

События клавиатуры

Событие	Возникает
KeyDown	При нажатии клавиши. Применяется для отслеживания нажатия функциональных клавиш, клавиш навигации и других, используемых для управления программой
KeyPress	При нажатии и затем отпускании одной из символьных клавиш. При нажатии не символьных клавиш событие не происходит
KeyUp	При отпускании ранее нажатой клавиши. Применяется для отслеживания нажатия функциональных клавиш, клавиш навигации и других, используемых для управления программой

Процедуры обработки событий связаны с объектами, размещенными в формах Visual Basic, или с самой формой и выполняются при наступлении события, с которым они связаны. Для события, связанного с формой, процедура *Sub* имеет следующий синтаксис:

Private Sub Form имяСобытия (аргументы)
операторы
End Sub

Исходя из синтаксиса, наименование процедуры обработки события для формы содержит слово *Form*, затем размещается символ подчеркивания (_) и имя события. Например, имя процедуры, выполняемой при загрузке формы, будет *Form_Load*, а процедуры, выполняемой при щелчке мыши на форме — *Form_Click.* При формировании процедуры для формы *MDI* ее имя будет содержать перед словом *Form* приставку *MDI*, то есть записываться *MDiForm.*

Для события, связанного с элементом управления формы, процедура обработки событий *Sub* имеет следующий синтаксис:

Private Sub имяЭлементаУправления имяСобытия (аргументы)
операторы
End Sub

Наименование процедуры обработки события для элемента управления формы содержит имя элемента управления, заданное в свойстве Name, затем следует символ подчеркивания (_) и имя события. Например, имя процедуры,

выполняемой при щелчке мыши на кнопке управления, имеющей наименование *cmdPrint* будет *Form_Click.*

Visual Basic облегчает формирование имен создаваемых процедур. Разработчику необходимо выполнить для этого следующие действия:

1. В окне *Properties* с помощью свойства *Name (Имя)* задать имя объекта, для которого создается процедура. Если имя не будет задано, то при создании процедуры Visual Basic использует имя, присваиваемое объекту по умолчанию при его размещении в форме. При последующем изменении наименования объекта необходимо будет изменить и имя процедуры.

2. В окне редактора кода из списка *Object (Объект*) выбрать объект, для которого создается процедура.

3. Из списка *Procedure (Процедура)* выбрать событие, обработка которого будет выполняться.

После выполнения указанных действий в области размещения процедур редактора кода будут размещены операторы *Sub* и *End Sub* с указанием наименования процедуры (рис. 28), где необходимо разместить между этими операторами выполняемый при наступлении этого события программный код.

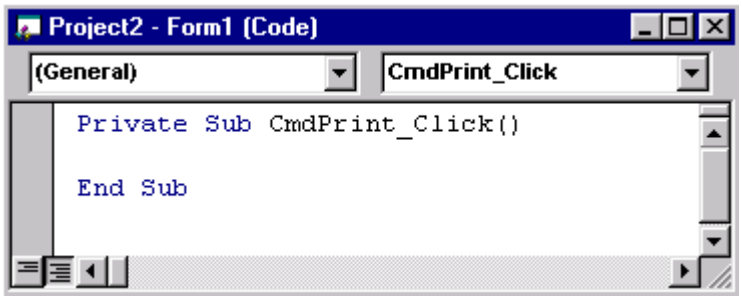


Рис. 28. Visual Basic формирует наименование процедуры

Создание исполняемого файла проекта

После завершения проектирования проекта, тестирования и отладки его в среде Visual Basic наступает завершающий этап — компиляция, то есть создание независимого от среды исполняемого файла (с расширением *exe*), библиотеки динамической компоновки (с расширением *dll*) или компонента *ActiveX (с расширением ocx).*

Для запуска процесса компиляции и создания исполняемого файла проекта приложения необходимо выполнить следующие действия:

1. Настроить параметры компиляции на вкладках *Make (Создать*) и *Compile (Компиляция)* диалогового окна *Project Properties* свойств проекта.

2. Выполнить команду *Make <имя проекта>.exe меню File (Файл*). При этом появляется диалоговое окно *Make Project* .

3. В поле Имя файла диалогового окна *Make Project (Создать проект*) ввести имя исполняемого файла или оставить имя, предлагаемое Visual Basic по умолчанию исходя из имени проекта.

4. Нажать кнопку *Options (Параметры*) и в открывшемся диалоговом окне *Project Properties* свойств проекта ввести номер версии исполняемого файла.

5. Нажатием кнопки ОК запустить процесс компиляции.

5.2. Задания

1. Известна длина окружности. Найти площадь круга, ограниченного этой окружностью (рис. 29).

Рис. 29. Окно вывода результатов

2. Создайте форму с несколькими объектами и измените их свойства и разместите на Форме Командную кнопку. Написать код, который при получении фокуса Кнопкой выполняет действия: передвигает Кнопку по Форме; изменяет её цвет, использовав переменные с обязательным объявлений (Option Explicit). Пример формы рабочего окна программы (рис. 30).

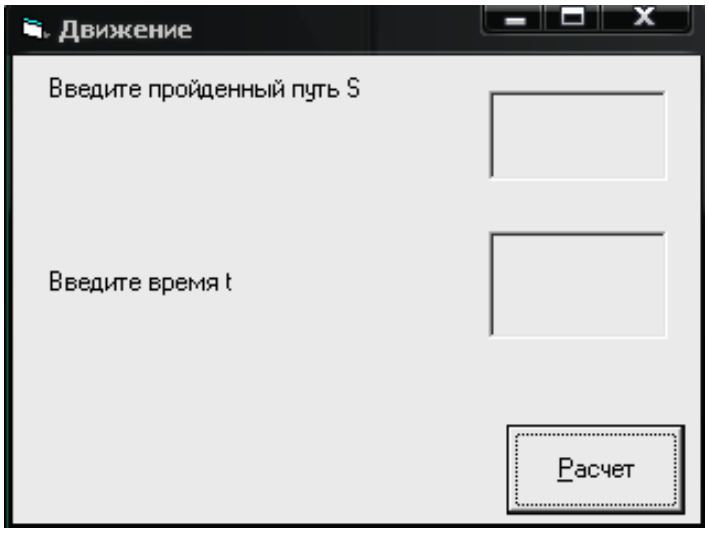


Рис. 30. Окно вывода результатов

3. Вычислить периметр, диагональ и площадь прямоугольного треугольника по заданным длинам двух катетов *a* и *b* (рис. 31).

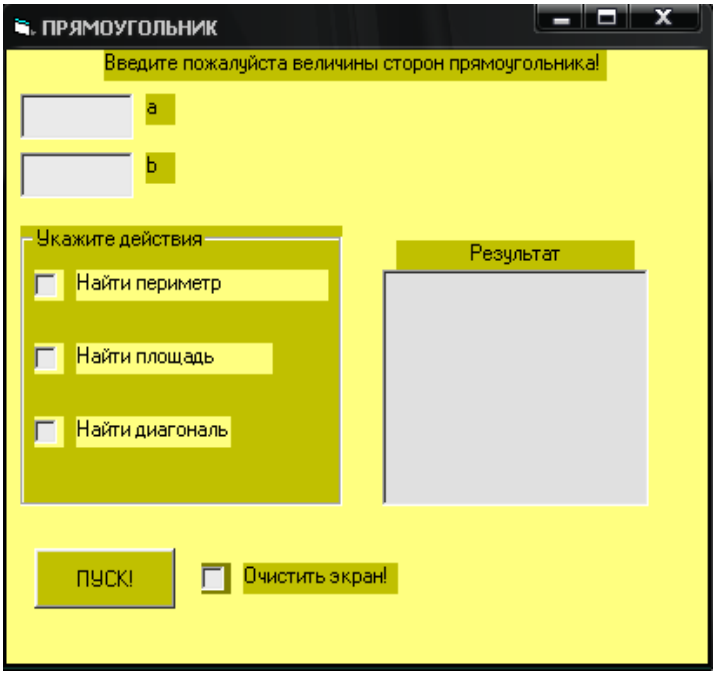


Рис. 31. Окно вывода результатов

5.3. Порядок выполнения работы

Задание 1.

Результат выполнения задания приведен на рис. 32.

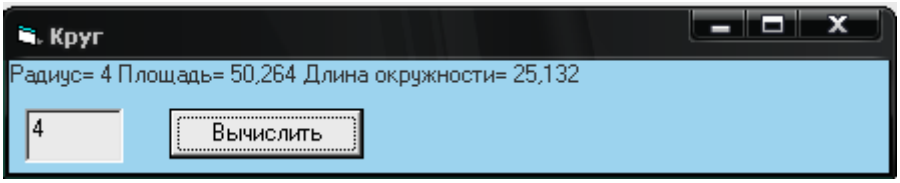


Рис. 32. Окно вывода результатов

Код программы:

```
Private Sub Command1_Click()
r = Val(Text1)
 Print "Радиус="; r; "Площадь="; 3.1415 * r ^ 2; "Длина окружности="; 2 * 3.1415 * r
End Sub
```

Задание 2.

Результат выполнения задания приведен на рис. 33.

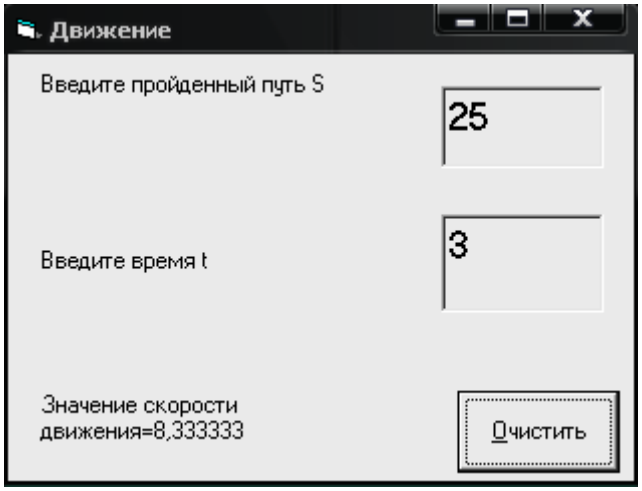


Рис. 33. Окно вывода результатов

Код программы:

```
Option Explicit
Private Sub Command1_Click()
Dim put_S As Single, vrema_t!
If Command1.Caption = "&Расчет" Then
Command1.Caption = "&Очистить"
put_S = Val(Text1.Text)
vrema_t = Val(Text2.Text)
Label3.Caption = "Значение скорости движения=" & put_S / vrema_t
Else
Text1.Text = ""
Text2.Text = ""
Label3.Caption = ""
Command1.Caption = "&Расчет"
Text1.SetFocus
End If
End Sub
```

Задание 3.

Код программы:

```
Private Sub Command1_Click()
a = Val(Text1)
b = Val(Text2)
If Check1.Value Then p = (a + b) * 2
Picture1.Print p
If Check2.Value Then s = a * b
Picture1.Print s
```

```
If Check3.Value Then d = Sqr(a ^ 2 + b ^ 2)
Picture1.Print d
If Check4.Value Then Picture1.Cls
End Sub
```

5.4. Контрольные вопросы

1. Каково назначение процедур событий?
2. Укажите синтаксис процедур формы.
3. Перечислите и укажите назначение процедур событий формы.
4. Как сформировать имя создаваемой процедуры?

Лабораторная работа № 6

Пользовательские подпрограммы. Отладка проекта

Цель: приобрести опыт разработки собственных библиотек подпрограмм *Sub* и *Function*

6.1. Методические указания

При программировании широко используются процедуры, позволяющие разбивать программные коды на небольшие логические блоки, которые, во-первых, легче отлаживать, а во-вторых, можно в свою очередь использовать при создании других процедур. В Visual Basic существуют следующие виды процедур:

Sub

Function

Функции и процедуры в VB

Процедуры и функции представляют собой отдельные блоки, из которых складывается код программы, каждая процедура выполняет какую-то задачу или ее часть. Процедуры обработки событий после вызова постоянно находятся в ожидании событий. Процедуры и функции не связанные с событиями выполняют отдельные действия и могут быть использованы неоднократно. Назовем их общими.

Процедуры общего назначения вызываются на выполнение в коде программы. Использование процедур экономит время и позволяет избежать лишних ошибок. Функции отличаются от процедур тем, что возвращают какое-то значение.

Под *процедурой или функцией* понимается последовательность операций, которую нужно многократно выполнять в различных местах приложения. При этом требуемый блок команд записывается в коде только один раз, после чего к нему можно обращаться из любой части программы.

Функция – это подпрограмма, вызываемая для выполнения каких-то расчетов или проверок, которая при завершении работы возвращает управление вызывающей программе и передает ей результат расчета.

Процедура – это тоже подпрограмма, вызываемая для выполнения каких-либо действий без возврата основной программе каких-то значений.

Синтаксис объявления процедуры и функции:

[Public/Private][Static] Sub <Имя процедуры>(<Параметры>)
<Операторы>
End Sub

Function <Имя функции> [As тип]
<Операторы>

End Function

Процедуры, объявленные с ключевым словом *Public*, можно вызвать в любом модуле приложения (каждая форма – это отдельный модуль).

Процедуры объявленные как *Private*, можно вызывать только в текущем модуле.

Слово *Static* означает, что все переменные, объявленные в процедуре, будут статическими, т.е. их значения сохраняются между вызовами.

Параметры обеспечивают связь процедуры с приложением. Это данные, передаваемые в процедуру при вызове.

Процедуры обработки событий вызываются в том, случае если произошло какое-либо событие. При этом существенным является как имя элемента, та и вид события, который с ним произошел.

Пользовательские процедуры – группы операторов, создаваемые разработчиком для выполнения определенных задач и не зависящие от текущего состояния приложения или произошедших в тот или иной момент событий.

Встроенные функции – определенные наборы команд, имеющиеся в языке Visual Basic и предназначенные для вычисления тех или иных значений на основании исходных данных. Встроенными являются, в частности, как математические, так и строковые функции (*Abs, Cos, Sin, Mid, Len* и т.д.)

Пользовательские функции – группы операторов, аналогичные пользовательским процедурам. Однако между ними есть ряд отличий.

Основные отличия функции от процедуры состоят в следующем.

1. Функция имеет тип (аналогично переменной) и может возвращать в программу значение, которое присваивается функции при помощи оператора:

<Имя функции> = значение

2. Вызов функции, как правило, осуществляется посредством указания в правой части какого-либо оператора ее имени и параметров. С другой стороны, процедура вызывается при помощи отдельного оператора:

Call <Имя процедуры> (Параметры)

Или

<Имя процедуры> (Параметры)

Если при вызове процедуры используется ключевое слово *Call*, то список параметров должен быть указан в скобках. Если же процедура вызывается без использования *Call*, то ее параметры перечисляются без скобок.

Необходимо отметить, что вызываемая процедура может не иметь параметров. В этом случае (если использовалось служебное слово Call) после имени процедуры следует ставить пустые скобки.

Пользовательские процедуры обычно используются при необходимости выполнения одной и тоже последовательности операций.

Например, в программе требуется неоднократно вводить в цикле значения массива *arrA,* состоящего из пяти элементов. В этом случае заполнение массива лучше всего оформить в виде процедуры.

Команда *Add Procedure* меню *Tools* позволяет добавить процедуру или функцию.

Пример 1.

Пусть процедура *Cir* вычерчивает эллипс с координатами *x, y,* которые передаются в процедуру как параметры. Создавая процедуру *Cir* командой *Add Procedure,* нужно указать имя процедуры и выбрать область видимости *Public* или *Private*. Вписать параметры в скобки и написать текст процедуры. В списке параметров рекомендуется указывать тип переменных.

Код:

```
Private Sub Cir(x As Integer, y As Integer)
Circle (x,y),500,,,2
End Sub
```

Пользовательский тип данных

Для определения пользовательского типа данных используется оператор *Type*, размещаемый в секции *(General) (Declarations*) модуля:

```
{Public | Private} Type имя_типа
имя_элемента [([нижняя_гр To верхняя_гр],..)] As тип_данных
...
End Type
```

Доступ к элементам переменной пользовательского типа осуществляется в следующей форме: *имя_переменной.имя_элемента[([индекс],..)]*.

Переменным пользовательского типа можно присваивать значения других переменных того же типа.

Обработка ошибок

Классы, определяемые пользователем, должны иметь собственные обработчики ошибок. Если внутри класса невозможно устранить ошибку, то эту задачу должна взять на себя вызывающая процедура. Для этого используется объект *Err*:

```
Err.Raise number [,source ][ ,description ][,helpfile,helpcontext ]
```

Метод *Raise* использует некоторые именованные аргументы.

Аргумент *number* содержит уникальный код ошибки. Для кодов ошибок, определяемых пользователем, к непосредственному номеру следует прибавить константу *vbObjectError*. При этом нельзя выходить за пределы области значений переменных типа *Long*.

Аргумент source содержит ссылку на источник ошибки, т.е.на класс. При задании аргумента source следует использовать структуру *ProjectName.ClassName.*

Аргумент description содержит текст, описывающий возникшую ошибку. В процедуре обработки ошибки этот текст можно получить, воспользовавшись свойством *Description* объекта *Err*.

Аргументы *helpfile* и *helpcontext* указывают справочный файл и в нем - тему справки.

С помощью метода *Raise* ошибка передается вызывающе процедуре, которая должна иметь соответствующий обработчик ошибок.

Реакция среды на ошибки

Реакцию среды разработки на ошибку можно настроить, воспользовавшись вкладкой *General* диалогового окна *Options*, которое открывается одноименно командой меню Tools. На вкладке General разработчик может выбрать один из трех вариантов реакции среды разработки на ошибку:

Break on All Errors – при возникновении ошибки среда разработки переходит в режим прерывания (отладки) независимо от того, имеется или нет активная процедура обработки ошибок, и независимо от того, находится ли ошибочный код в модуле класса.

Break in Class Module – возникновение любо необрабатываемой ошибки в модуле класса приводит к переключению среды разработки в режим отладки и выделению строки кода, вызвавшей ошибку. Если при отладке компонентов ActiveX вы запускаете тестовую клиентскую *ActiveX* программу в другом проекте,то при установке этой опции обработка ошибки будет выполняться в модуле класса компонента *ActiveX*,а не в тестовой программе.

Break on Unhandled Errors – если процедура обработки ошибок активна, то возникшая ошибка обрабатывается без переключения среды в режим отладки. Если же такая процедура отсутствует, то происходит переключение среды в режим отладки. Если ошибка произошла в модуле класса, то выполнение программы останавливается в вызывающей процедуре, а не в модуле класса.

Отладка проекта

Отладка проекта – это отдельная большая тема, которой мы подробнее займемся позже. Здесь же по этой теме даны самые начальные сведения.

Отладка – это поиск и исправление ошибок в программе. Даже внимательный визуальный анализ программы не всегда позволяет обнаружить допущенную ошибку. В помощь программисту в VB предусмотрено несколько средств, облегчающих поиск ошибок. Различают три вида ошибок в программах:

синтаксическая ошибка. Обнаруживается компилятором. Возникает при нарушении синтаксиса инструкций. До устранения синтаксических ошибок VB не позволит запустить программу;

ошибки выполнения. Они возникают после запуска программы при возникновении ситуации, когда выполнять программу далее невозможно (не обнаруженная при компиляции синтаксическая ошибка, деление на ноль; обращение к гибкому диску, когда устройство не готово);

логические ошибки. Эти ошибки появляются следствием невнимательности программиста или его заблуждений при разработке алгоритма или при кодировании алгоритма. В итоге программа работает не правильно. Результаты ее работы не соответствуют тестам.

О синтаксических ошибках и ошибках выполнения сообщает система, останавливая дальнейшее выполнение программы. Их невозможно не заметить. Логические ошибки коварны. Программа работает, выдает результаты. Но программист обязан заметить, что программа дает неверные результаты. Для этого программист испытывает программу на тестах. Однако, обнаружив дефект в работе программы, следует найти причину этого дефекта. Для этого удобно воспользоваться предоставляемой средой VB возможностью остановить выполнение программы перед выполнением заранее выбранной инструкции и затем продолжить ее дальнейшее выполнение по шагам (по одному оператору) с остановкой после каждого шага и контролем полученных значений переменных.

Для отладки приложения в пошаговом режиме нужно открыть панель инструментов отладки *Debug* (Отладка), которую можно отобразить, выбрав команду меню *View* (Вид), затем *Toolbars* (Панели инструментов) и, щелкнув на строке *Debug*. На рис. 34 показана панель инструментов *Debug* (Отладка).

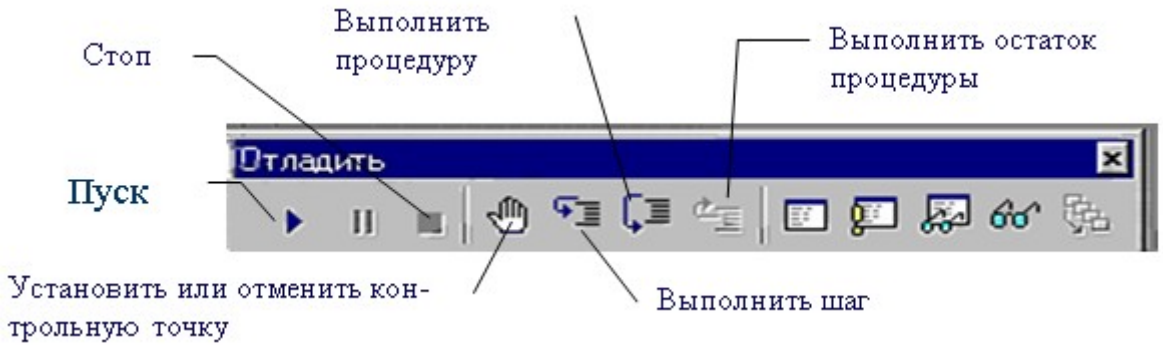


Рис. 34. Панель инструментов отладки

Можно рекомендовать следующую последовательность действий при поиске логических ошибок:

1. Откройте окно Code с программным кодом.
2. Откройте панель инструментов отладки.
3. В подозрительном месте программы, где Вы предполагаете существование логической ошибки, выберите строку программы, перед выполнением которой требуется остановить программу.
4. Установите перед этой строкой контрольную точку. Это можно сделать, например, следующим способом. Щелкните в окне Code слева от выбранной строки на серой вертикальной полосе. Строка будет выделена коричневым цветом. Для отмены контрольной точки достаточно повторить щелчок.
5. Запустите проект. Выполнение программы будет остановлено перед выполнением строки программы с контрольной точкой.
6. После остановки программы можно продолжать ее выполнение по шагам (по одной строке программы). Для выполнения одного шага нужно

щелкнуть на кнопке панели инструментов "Выполнить шаг". Если же в следующей строке находится обращение к процедуре или к функции, то они могут быть выполнены за один шаг, если воспользоваться кнопками "Выполнить процедуру" или "Выполнить остаток процедуры".

7. После выполнения каждого шага можно наблюдать значение любой переменной. Для этого достаточно в окне Code переместить курсор мыши на имя переменной. Появится окно, в котором будет отображено значение переменной, имя которой находится в фокусе мыши.

8. Продолжая так выполнение программы по шагам, Вы, в конце концов, обнаружите то место, где программа делает не то, что Вы ожидали. После этого следует внести необходимые исправления в программу и продолжить отладку до устранения всех ошибок

6.2. Задания

1. Написать функцию для вычисления дискриминанта квадратного уравнения и процедуру, которая будет выводить полученные значения в метки.

2. Написать процедуру, которая будет выводить на экран сообщение "*Hello World*!":

3. Написать функцию для вычисления квадрата числа.

4. Вычислить факториал заданного числа с использованием процедур и функций (рис. 35).

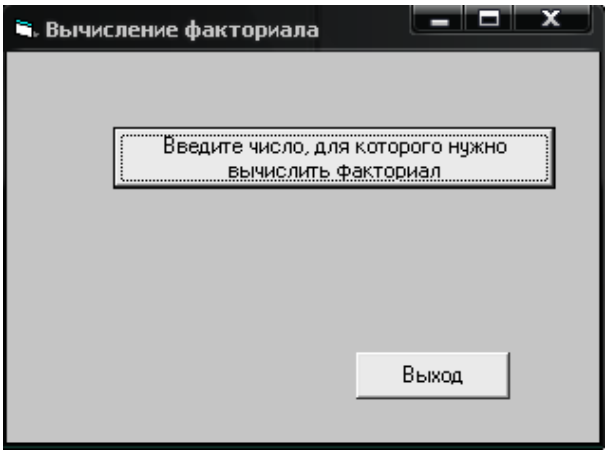


Рис. 35. Окно вывода результатов

6.3. Порядок выполнения работы

Задание 1.

Чтобы вычислить дискриминант необходимо знать 3 параметра – коэффициенты **a, b и c**. Функция будет возвращать значение – дискриминант. Для автоматического создания заготовки для этой функции выберите в главном меню VB: *Tools->Add Procedure...* В появившемся окне указать вид добавляемой процедуры и отметить радиокнопку *Function*. В поле "*Name*:", указать имя добавляемой функции, например, как *CalcDiscriminant*, нажатье OK и Visual Basic создаст для вас следующую заготовку:

Public Function CalcDiscriminant ()

End Function

Необходимо модифицировать её так, чтобы функция принимала 3 аргумента (параметра) типа *Double*, и возвращала значение, которое тоже будет иметь тип *Double* и добавить код в нашу заготовку:

Public Function CalcDiscriminant (a As Double, _

b As Double, c As Double) As Double

* CalcDiscriminant = (b * b) - (4 * a * c)*

End Function

Задание 2.

Private Sub ShowMessage()

* MsgBox "Hello World!"*

End Sub

Процедура имеет тип *Private*, т.е. доступна будет только из кода именно той формы (модуля), где она объявлена (вспомните типы объявления переменных). Данная процедура не содержит параметров,т.к.присутствуют пустые круглые скобки. Назначение – вывести сообщение *Hello World* на экран. *MsgBox* – это встроенная функция Visual Basic, которая выводит на экран окно с сообщением, заданным в качестве параметра.

Задание 3.

У функции будет один параметр типа *Integer* – число для возведения в квадрат. Функция будет возвращать значение квадрата параметра. Тип возвращаемого значения - *Long*:

Public Function Square(number As Long) As Long

* Square = number * number*

End Function

Вызвать функцию можно следующими способами:

1. *b = Square (5)*
2. *ShowMessage Square (5)*
3. *Square 5*

Задание 4.

Код программы:

```
Option Explicit
Private Sub Command1_Click()
Dim bytn As Byte
Dim strinput
Dim intR As Integer
strinput = InputBox("Ввод факториала:", "Ввод")
bytn = CByte(strinput)
intR = MsgBox(Fact(bytn), , "Вывод")
End Sub
Function Fact(byta As Byte) As Integer
Dim byti As Byte
Dim intfact As Integer
```

```
intfact = 1
For byti = 1 To byta
intfact = intfact * byti
Next
Fact = intfact
End Function
Private Sub Command2_Click()
End
End Sub
```

6.4. Контрольные вопросы

1. Приведите синтаксис процедур и функций. Какие виды процедур и функций определены в VB?
2. Как вызвать процедуру, функцию?
3. Поясните синтаксис объявления процедуры, синтаксис объявления функции.
4. Что означает передача аргумента процедуры по значению? Чем может быть такой аргумент при обращении к процедуре?
5. Что означает передача аргумента процедуры по ссылке? Чем может быть такой аргумент при обращении к процедуре?
6. Где должна быть объявлена процедура или функция, чтобы к ней можно было обращаться из любой процедуры и функции формы? Как осуществляется вызов функций?

Лабораторная работа № 7

Разработка приложений с разветвляющимися алгоритмами

Цель: изучение возможностей разработки приложений с разветвляющимися алгоритмическими структурами

7.1. Методические указания

Управляющие конструкции Visual Basic

В Visual Basic, как и во всех языках программирования, существуют управляющие конструкции, предназначенные для управления порядком выполнения команд. Различают два основных типа управляющих операторов

If

Select Case

Конструкция *If* используется в том случае, когда необходимо, чтобы группа операторов выполнялась при соблюдении определенных условий. Конструкция *Select case* позволяет на основании анализа значения заданного выражения выполнять те или иные действия.

В свою очередь, управляющие операторы *If* бывают двух видов:

If...Then

If...Then...Else

Конструкция *If...Then* применяется, когда необходимо выполнить определенные действия в зависимости от некоторого условия. Управляющая конструкция *If...Then...Else* используется в том случае, когда необходимо выполнить разные действия в зависимости от условия.

Условные выражения

Основанием для принятия решений в управляющих конструкциях являются условные выражения, поэтому предварительно необходимо сказать несколько слов об этих выражениях и работе с ними.

Условные выражения — это такие выражения, которые возвращают одно из двух значений *True (Истина)* или *False (Ложь)*. В условных выражениях используются операторы сравнения, приведенные в табл. 7.

Таблица 7

Оператор	Назначение
=	Равно
>	Больше
<	Меньше
<>	Не равно
>=	Больше или равно
<=	Меньше или равно

Над условными выражениями можно выполнять действия логической математики (логические операции), а именно:

AND (И) — возвращает значение *True (Истина),* если все участвующие в операции выражения имеют значение True. В остальных случаях возвращается значение False (Ложь);

OR (ИЛИ) — возвращает значение *True*, если хотя бы одно из участвующих в операции выражений имеет значение *True*. В случае, когда все выражения имеют значение *False*, возвращается значение *False*;

XOR (*Исключающее ИЛИ)* — возвращает значение *True (Истина*), если только одно из участвующих в операции выражений имеет значение *True*. В остальных случаях возвращается значение *False*;

NOT (НЕ) — операция отрицания. Возвращает обратное для значения выражения значение, то есть если выражение равно *True*, то возвращается *False* и наоборот, если значение выражения равно *False*, то возвращается значение *True*.

Синтаксис использования логических операций такой же, как и у арифметических операций. Например:

(выражение1 *And* выражение2 *And* выражение3) *Or* (выражение4 *Хог* выражение5)

Скобки в условных выражениях действуют так же, как и в арифметических, то есть первыми всегда выполняются операции в скобках.

Сложные выражения можно предварительно вычислить и хранить в логических переменных типа *Boolean*. Например, предыдущий код с использованием переменных можно представить следующим образом:

Dim bVar1 As Boolean
Dim bVar2 As Boolean
bVarl = выражение! And выражение2 And выражение3
bVar2 = (выражение4 Xor выражение5)
Итоговым будет следующее выражение:
bVarl Or bVar2

Конструкция If... Then

Конструкция *If.. .Then* применяется в том случае, когда необходимо выполнить один или группу операторов при соблюдении определенного условия, то есть когда значение заданного условия равно *True*. Существует две разновидности данного оператора: однострочный и многострочный. Однострочный оператор имеет следующий синтаксис:

If условие *Then* конструкция

В этом операторе условие и выполняемые при соблюдении условий действия располагаются в одной строке. Если при выполнении условия требуется выполнение блока операторов, используется многострочный оператор, имеющий следующий синтаксис:

If условие *Then*
конструкции

End If

Исходя из синтаксиса, приведенные ниже программные коды выполняют одни и те же действия:

' Однострочный оператор

If y>20 Then y=2

' Многострочный оператор

If y>20 Then

У=2

End If

После имени конструкции *If* должно следовать логическое выражение, содержащее условие. В качестве условия могут выступать следующие логические выражения:

1. сравнение переменной с другой переменной, константой или функцией;

2. любая переменная, выражение, поле базы данных или функция, принимающие значения *True* или *False*.

Ключевое слово *End If* обозначает конец многострочной конструкции и его наличие в команде в этом случае обязательно. Если указанное условие выполняется, то есть результат проверки равен *True*, то Visual Basic выполнит конструкции, следующие за ключевым словом *Then*. Если условие не выполняется, то Visual Basic переходит к выполнению операторов, следующих за указанным оператором.

Конструкция If... Then... Else

Конструкция *If.. .Then.. .Else* аналогична конструкции *If..*T*hen*, но позволяет задать действия, исполняемые как при выполнении условий, так и в случае их невыполнения.

Конструкция имеет следующий синтаксис*:* *IF* условие *Then* .

Конструкции для обработки истинного условия *Else*

Конструкции для обработки ложного условия *End If*

Ключевые слова *IF и End if* имеют тот же смысл, что и в конструкции *If...Then*. Если заданное в конструкции условие не выполняется (результат проверки равен *False*), и конструкция содержит ключевое слово *Else*, Visual Basic выполнит последовательность конструкций, расположенных следом за Else. После чего управление перейдет к конструкции, следующей после End If.

Пример 1.

If x >= 0 *Then*

Label1.Caption = "Значение больше или равно 0"

Else

Labell.Caption = "Значение меньше 0"

End If

Команда *If* может проверить только одно условие. Если вам потребуется осуществить переход управления в зависимости от результатов проверки нескольких условий, то такая возможность существует. Дополнительное

условие можно задать с помощью оператора *ElseIf*. Оно будет анализироваться только в том случае, если предыдущее условие ложно (пример 2).

Пример 2.

If x > 0 *Then*
Label1.Caption = "Значение положительное"
Elself x = 0 *Then*
Label1.Caption = "Значение равно 0"
Else
Labell.Caption = "Значение отрицательное"
End If

Конструкция Select Case

Конструкция *Select Case* позволяет обрабатывать в программе несколько условий и аналогична блоку конструкций *If.. .Then.. .Else*. Эта конструкция состоит из анализируемого выражения и набора операторов case на каждое возможное значение выражения. Работает эта конструкция следующим образом. Сначала Visual Basic вычисляет значение заданного в конструкции выражения. Затем полученное значение сравнивается со значениями, задаваемыми в операторах case конструкции. Если найдено искомое значение, выполняются команды, приписанные данному оператору case. После завершения выполнения конструкций управление будет передано конструкции, следующей за ключевым словом *End Select*.

Синтаксис конструкции *Select Case* следующий:

Select Case сравниваемоеЗначение
CASE значение1
конструкция1
CASE значение2
конструкция2
. . .
End Select

В начале конструкции расположены ключевые слова *Select Case*, указывающие, что расположенный рядом с ними параметр сравниваемоеЗначение будет проверяться на несколько значений. Далее следуют группы команд, начинающиеся с ключевого слова *Case*. Если параметр сравниваемоеЗначение равен значению, указанному в текущем операторе case, то будут выполняться команды, расположенные между этим и следующим ключевым словом case.

В качестве примера 3 воспользуемся конструкцией *Select Case* для решения предыдущей задачи:

Пример 3.

Select Case x
CASE 1 To 9
Labell.Caption = "Значение больше 0"
CASE 0

Labell.Caption = "Значение равно 0"
CASE -9 To -1
Label1.Caption = "Значение меньше 0"
End Select

Конструкция *Select Case* может выполнить не более одной из содержащихся в ней последовательностей конструкций. После того как одно из условий оказалось равно *True*, и была выполнена соответствующая последовательность конструкций, *Select Case* завершит свою работу. Остальные условия проверяться не будут.

Ключ (флажок)

Ключ (*CheckBox*) позволяет сделать активной какую либо опцию приложения. На рисунке показан фрагмент формы с двумя расположенными на ней ключами (рис. 36).

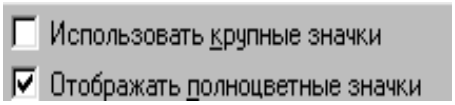


Рис. 36. Фрагмент формы с ключами

Пример 6. Проверка работы переключателя, то есть при нажатии на кнопку Проверка выводит на экран сообщение о состоянии переключателя.

Установить на форму объекты: кнопку со свойством *Caption* – проверить, переключатель со свойством *Caption* – *Переключить*, и кнопку с *Caption* – *Выход*.

Private Sub Command1_Click()
If Check1.Value = 1 Then
MsgBox "Переключатель включен"
Else
MsgBox "Переключатель выключен"
End If
End Sub

Private Sub Command2_Click()
End
End Sub

If Check1.Value = 1 *Then* – программа проверяет – если свойство *Value* элемента Check1 равно 1, то есть флажок установлен то...

MsgBox "Переключатель включен" – вывести сообщение, говорящее о том что переключатель включен

Else – иначе (то есть если свойство *Value* элемента *Check1* не равно 1) то...

MsgBox "Переключатель выключен"-вывести сообщение, говорящее о том что переключатель выключен.

End If – закончить проверку.

Переключатель (поля выбора или «радиокнопки»)

Переключатель (*OptionButton*) позволяет выбрать одну из возможных опций, представленных в форме в виде списка. На рисунке показан фрагмент формы с двумя расположенными на ней переключателями (рис. 37).

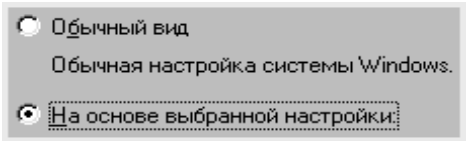


Рис. 37. Фрагмент формы с переключателями

7.2. Задания

1. Даны два числа. Вычислить их сумму, разность, произведение и частное (рис. 38).

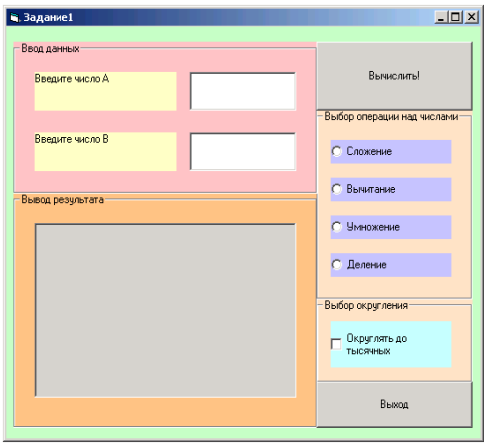


Рис. 38. Окно вывода результатов

2. Заданы три положительных числа *a*, *b* и *c*. Определить, являются ли они последовательно стоящими элементами арифметической прогрессии. Если являются, то определить разность прогрессии (рис. 39).

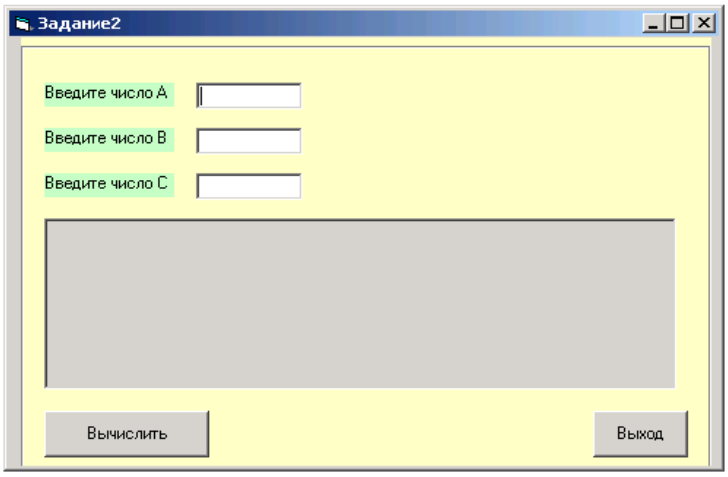


Рис. 39. Окно вывода результатов

3. Создайте проект и напишите программу вычисления корней квадратного уравнения a*x^2+b*x+c=0 (рис. 40). Предусмотрите условный переход и вывод результата решения уравнения для равных корней (x1=x2 при D=0). Для вывода результатов используйте форматный вывод. Создайте вариант проекта с использованием конструкции *Select Case.*

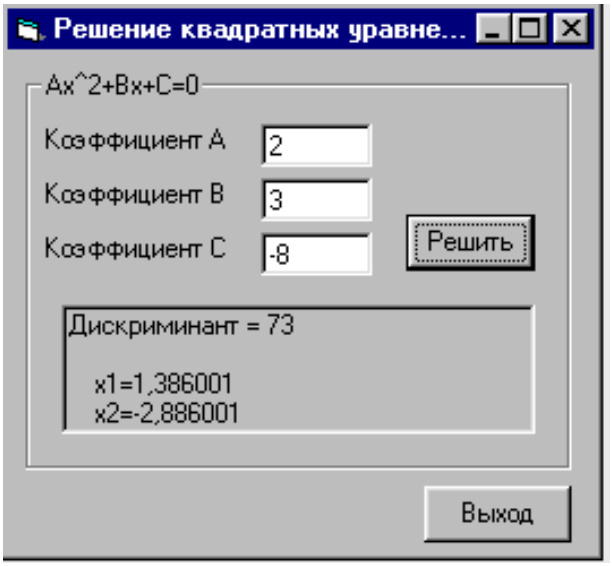


Рис. 40. Окно вывода результатов

7.3. Порядок выполнения работы

Задание 1.

Результат выполнения задания приведен на рис. 41.

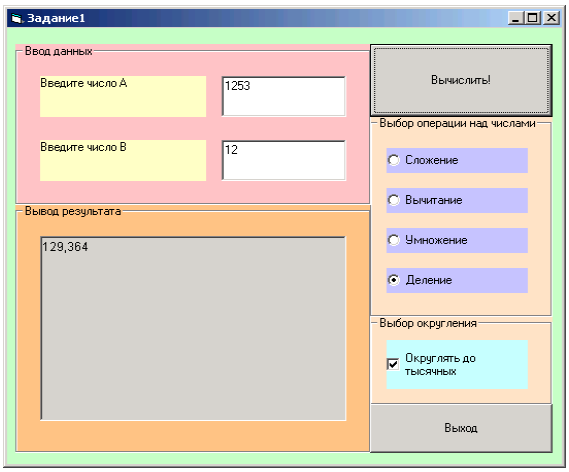


Рис. 41. Окно вывода результатов

Код программы:

Private Sub Command1_Click()

If Option1.Value *Then* X = Val(Text1) + Val(Text2)

If Option2.Value *Then* X = Val(Text1) - Val(Text2)
If Option3.Value *Then* X = Val(Text1) * Val(Text2)
If Option4.Value *Then* X = Val(Text1) / Val(Text2)
If Check1.Value *Then* Picture1.Print Format(X, "#.000") *Else* Picture1.Print X
End Sub
Private Sub Command2_Click()
End
End Sub

Таблица 8

Объект	Свойство	Значение
Form1	Caption BackColour	Задание1 &H00C0FFC0&
Command1 Command2	Caption Caption BackColour	Вычислить Выход
Frame1 Frame2 Frame3 Frame4	Caption Caption Caption Caption BackColour	Ввод данных Выбор операции над числами Вывод результата Выбор округления
Label1 Label2	Caption Caption BackColour	Введите число A Введите число B
Option1 Option2 Option3 Option4	Caption Caption Caption Caption BackColour	Сложение Вычитание Умножение Деление
Check1	Caption BackColour	Округлять до тысячных

Задание 2.
Результат выполнения задания приведен на рис. 42.

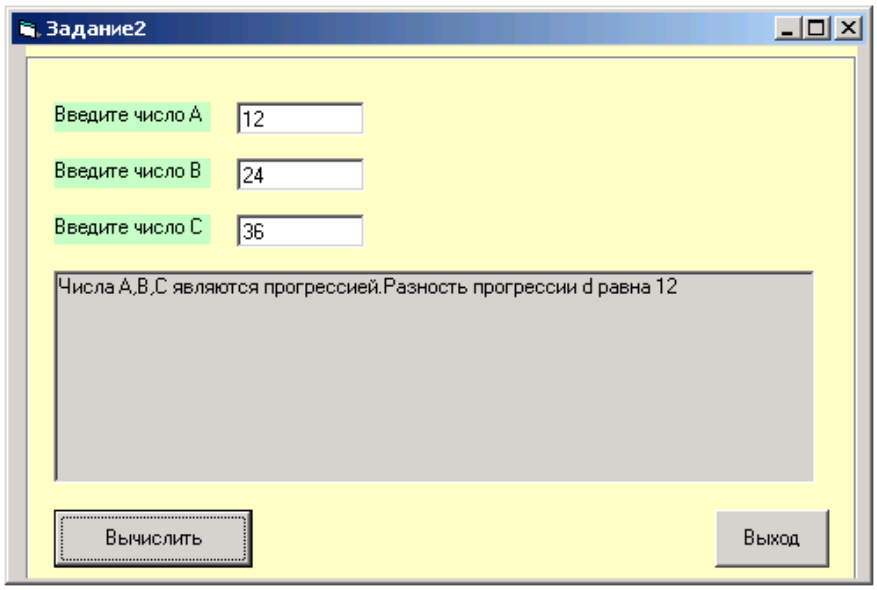


Рис. 42. Окно вывода результатов

Код программы:

```
Private Sub Command1_Click()
A = Text1
B = Text2
C = Text4
d = B - A
x = C - B
If d = x Then Picture2.Print "Числа A, B, C являются прогрессией. Разность прогрессии d равна"; d Else Picture2.Print "Числа A,B,C не являются последовательно стоящими элементами арифметической прогрессии"
End Sub
Private Sub Command2_Click()
End
End Sub
```

Таблица 9

Объект	Свойство	Значение
Form1	Caption	Задание2
Label1	Caption	Введите число A
Label2	Caption	Введите число B
Label3	Caption	Введите число C
	BackColour	
Command1	Caption	Вычислить
Command2	Caption	Выход

7.4. Контрольные вопросы

1. Какие операторы используются для организации условного перехода?
2. Какие логические отношения могут использоваться в операторах условного перехода?
3. Что такое логическое выражение?
4. Какие логические операции могут использоваться в логических выражениях?
5. В каких случаях используется Конструкция *Select Case*?
6. Для чего используется объект *Ключ*?
7. Для чего используется объект *Переключатель*?

7.5. Задания для самостоятельной работы

Вариант задания	Функция	Границы отрезка и шаг
1.	$y = 3\sin\sqrt{x} + 0{,}35x - 3{,}8$	[2,3] 0,1
2.	$y = 0{,}25x^3 + x - 1{,}2502$	[0,2] 0,2
3.	$y = \sin(\ln x) - \cos(\ln x) + 2\ln x$	[0,4;1] 0,05
4.	$y = \cos\frac{2}{x} - 2\sin\frac{1}{x} + \frac{1}{x}$	[1;3] 0,2
5.	$y = 3x - 4\ln x - 5$	[1;2] 0,1
6.	$y = \sqrt{1-x} - \cos\sqrt{1-x}$	[2;4] 0,2
7.	$y = tgx - \frac{1}{3}tg^3x + \frac{1}{5}tg^5x - \frac{1}{3}$	[0;1] 0,1

Лабораторная работа № 8

Разработка приложений с циклическими алгоритмами

Цель: изучение возможностей разработки приложений с циклическими алгоритмическими структурами

8.1. Методические указания

Циклы

В программах Visual Basic для выполнения повторяющихся действий используются циклы. Они бывают следующих типов:

For...Next

For Each...Next

Do. . . Loop

Цикл с использованием конструкции For...Next

Конструкция *For...Next* выполняет последовательность команд определенное число раз. Такую конструкцию называют циклом, а выполняемые ею программные коды — телом цикла.

Синтаксис конструкции *For.. .Next* следующий:

For счетчик = начЗначение *To* конЗначение [*Step* шаг]

конструкции

Next[счетчик]

Первый аргумент конструкции — счетчик — определяет имя переменной, которая будет "считать" количество выполнении цикла. Параметр начЗначение указывает числовое значение, которое присваивается переменной-счетчику перед первым проходом цикла. Цикл выполняется до тех пор, пока значение счетчика не превысит конечного значения, указанного после ключевого слова то. После каждого прохода цикла значение счетчика изменяется на величину шаг, указанную за ключевым словом *step*. Ключевое слово *Next* обозначает конец тела цикла и является обязательным.

Перед каждым проходом цикла Visual Basic сравнивает значения счетчика и аргумента конЗначение. Если значение счетчика не превышает установленного значения конЗначение, выполняются конструкции тела цикла. В противном случае управление переходит к следующей за *Next* конструкции.

Цикл с использованием конструкции For Each... Next

Цикл с использованием конструкции *For Each...Next* похож на цикл *For.. .Next*, но используется для обработки всех элементов некоторого набора объектов или массива. Его особенно удобно использовать в том случае, когда количество обрабатываемых элементов не известно.

Синтаксис конструкции *For Each.. .Next* следующий:

For Each элемент *In* группа

конструкции

Next элемент

Пример 4.

Dim objControl *As* Control
For Each objControl *In* Controls
objControl.Caption = "Test " & objControl.Caption
Next objControl

При использовании конструкции *For Each.. .Next* необходимо иметь в виду, что для набора объектов параметр элемент может быть только переменной типа *Variant*, общей переменной типа *Object* или объектом, перечисленным в *Object Browser*. Для массивов параметр элемент может быть только переменной типа *Variant*.

Цикл с использованием конструкции Do...Loop

Цикл, задаваемый конструкцией *Do... Loop*, выполняется до тех пор, пока истинно задаваемое в цикле условие.

Синтаксис конструкции *Do. . .Loop* следующий:

Do While условие
конструкции
Loop

Аргумент конструкции условие является логическим выражением, значение которого проверяется перед каждым проходом цикла. Если это значение равно *True*, то выполняется последовательность команд, которые расположены между *Do while* и ключевым словом *Loop*. Эти конструкции образуют тело цикла. Если при очередном проходе цикла условие равно *False*, то происходит выход из цикла и управление передается конструкции, следующей за *Loop*. Возможна ситуация, при которой операторы цикла не выполняются ни разу. Она возникает в том случае, если при первой проверке условия оно оказывается ложным.

В Visual Basic существует еще один вид цикла конструкции *Do... Loop*. Он отличается от рассмотренного ранее местом расположения условия. Если в предыдущей конструкции условие, по которому выполняется цикл, расположено в заголовке, то в этой конструкции условие располагается в конце цикла:

Do
конструкции
Loop While условие

При использовании этой формы оператора тело цикла выполняется хотя бы один раз, после чего осуществляется проверка заданного условия.

Есть еще две разновидности конструкции цикла *Do. . .Loop*. Они аналогичны рассмотренным ранее, но отличаются тем, что цикл выполняется до тех пор, пока условие ложно, а не истинно. Данные операторы имеют следующий синтаксис:

Do Until условие
конструкции
Loop

и

Do
конструкции
Loop Until условие

Пример 5.

```
nCounter = 2
Do While nCounter < 10
   nDecades(nCounter) = nCounter * 2
   nCounter = nCounter * 2
Loop
```

Пример 1.

```
For nCountVar = 1 To 10 Step 2
nNextWeek(nCountVar) = nCountVar * 2
Next
```

Здесь цикл выполняется пять раз при значениях счетчика *nCountVar* 1, 3, 5, 7 и 9. Переменная-счетчик используется в теле цикла в качестве обычной переменной. Шаг изменения счетчика может быть отрицательным.

Пример 2.

```
For nCounter = 100 To 1 Step -10
nDecades(nCounter) = nCounter *2
Next
```

В этом случае цикл будет выполняться до тех пор, пока *nCountVar* больше 1. Если значение шага цикла отрицательно, то начальное значение счетчика должно быть больше конечного.

Ключевое слово *Step* можно опустить. В этом случае значение шага по умолчанию принимается равным 1.

Возможны ситуации, при которых выполнение цикла невозможно или, наоборот, его выполнение становится бесконечным.

Пример 3.

1. Невыполняемый цикл: начальное значение счетчика больше конечного при положительном шаге цикла

```
For nCounter=100 To 1
nDecades (nCounter) = nCounter
Next
```

2. Бесконечный цикл: значение счетчика изменяется в теле цикла и никогда не превысит 10

```
For nCounter = 1 To 10
nCounter = 1
Next
```

8.2. Задания

1. Табулировать функцию $y = x^2 - x \ln x$ с границами [1; 2] и шагом 0.1 (рис. 43).

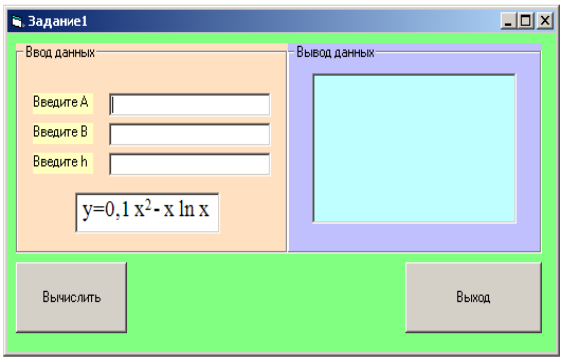


Рис. 43. Окно вывода результатов

2. По заданной формуле члена последовательности с номером *k* составить две программы:

– программу вычисления суммы первых *n* членов последовательности

$$S = \sum_{k=1}^{n} \frac{1}{(3i^2 + 7) \cdot (i^2 + 1)} \quad (k=1,2,3..,n);$$

– программу вычисления суммы всех членов последовательности, не меньших заданного числа *e*.

3. Табулировать функцию при заданных исходных данных, диапазоне и шаге изменения аргумента.(t = 20.3, a = 0.5, b = 2, dx = 0.1)

8.3. Порядок выполнения работы

Задание 1.

Результат выполнения задания приведен на рис. 44.

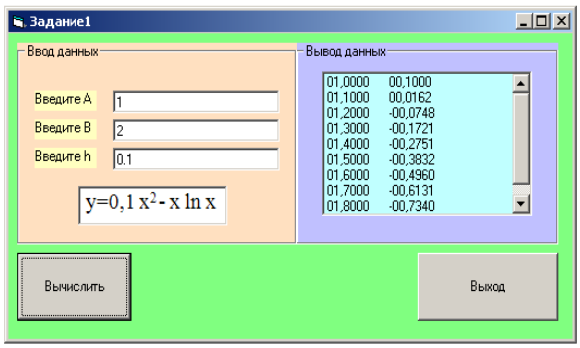


Рис. 44. Окно вывода результатов

Код программы:

```
Dim a As Single, b As Single, h As Single, x As Single, y As Single
Private Sub Command1_Click()
List1.Clear
a = Val(Text1)
```

```
b = Val(Text2)
h = Val(Text3)
For x = a To b Step h
y = 0.1 * x ^ 2 - x * Log(x)
List1.AddItem Format(x, "00.0000") & "      " & Format(y, "00.0000")
Next x
End Sub

Private Sub Command2_Click()
End
End Sub
```

Таблица 10

Объект	Свойство	Значение
Form1	Caption BackColour	Задание1 &H00C0FFC0&
Command1	Caption	Вычислить
Command2	Caption BackColour	Выход
Frame1	Caption	Ввод данных
Frame2	Caption BackColour	Вывод данных
Label1	Caption	Введите A
Label2	Caption	Введите B
Label3	Caption BackColour	Введите h
Picture1	Picture	Bitmap

Задание 2.

Результат выполнения задания приведен на рис. 45.

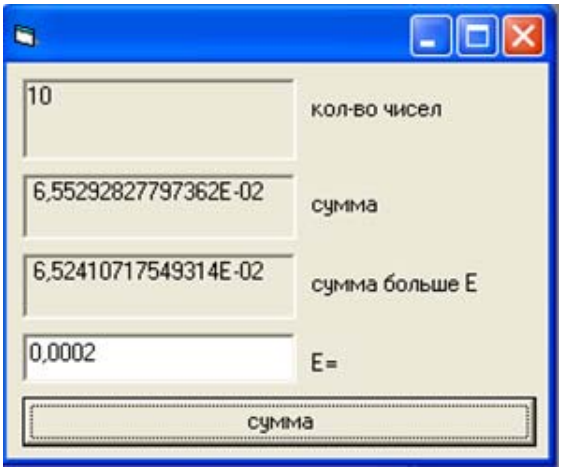


Рис. 45. Окно вывода результатов

Код программы:

```
Private Sub Command1_Click()
Picture1.Cls
```

```
Picture2.Cls
Picture3.Cls
n = InputBox("введите количество чисел")
s=0
For  i=1  To  n
ba(i)= 1/(3*i^2+7)*(i^2+1)
s =s+a(i)
Next
Picture1.Print  n
Picture1.Print  s
End Sub
```

8.4. Контрольные вопросы

1. Какой оператор используется для организации цикла при известном числе повторений?

2. Какие операторы используются для организации циклов при известном заранее числе повторений?

3. Каким образом организовать выход из цикла до исчерпания значений его параметра?

8.5. Задания для самостоятельной работы

Вариант задания	Функция	Границы отрезка и шаг
1.	$y = 3\sin\sqrt{x} + 0{,}35x - 3{,}8$	[2,3] 0,1
2.	$y = 0{,}25x^3 + x - 1{,}2502$	[0,2] 0,2
3.	$y = \sin(\ln x) - \cos(\ln x) + 2\ln x$	[0,4;1] 0,05
4.	$y = \cos\frac{2}{x} - 2\sin\frac{1}{x} + \frac{1}{x}$	[1;3] 0,2
5.	$y = 3x - 4\ln x - 5$	[1;2] 0,1
6.	$y = \sqrt{1-x} - \cos\sqrt{1-x}$	[2;4] 0,2
7.	$y = tgx - \frac{1}{3}tg^3x + \frac{1}{5}tg^5x - \frac{1}{3}$	[0;1] 0,1

Лабораторная работа № 9

Операции над массивами

Цель: изучение возможностей разработки приложений и получение практических навыков решения типовых задач с использованием одномерных и двумерных массивов

9.1. Методические указания

Массивы (списки)

Массив (вектор) – это набор однотипных переменных, объединенных одним именем и доступных через это имя и порядковый номер переменной в наборе. Количество элементов массива теоретически может быть бесконечным, ограничения накладываются конкретными языком программирования и операционной системой. Элементы массива обладают непрерывной нумерацией определённого диапазона.

Примером обработки табличных данных может служить следующая задача. В процессе полета летательного аппарата через определенный интервал времени фиксируются значения скорости и высоты полета. После завершения полета данные обрабатываются, и находится максимальная скорость и максимальная высота полета.

В программировании табличные данные представляются в виде массивов.

Массивы могут быть одномерными и многомерными.

Размерностью массива называется количество его измерений.

Размером массива называется количество хранящихся в нем элементов. Размер массива ограничен объемом оперативной памяти и типом данных элементов массива

В Visual Basic массивы определяются следующим образом:

Dim myArray (10) As Long

Определение массива отличается от определения обычной переменной только индексом, указанным в скобках. Этот индекс указывает размерность массива. В данном случае массив *myArray* будет содержать 11 элементов, потому что нижняя граница массива начинается с нуля. *[0,1,2.....9,10]*. Чтобы задать определённую размерность можно использовать зарезервированное слово *To*:

Dim myArray (5 To 10) As Long

Здесь определяется массив, размерность которого 6 элементов (5, 6, 7, 8, 9, 10).

Общий синтаксис определения массива следующий:

Dim ИмяМассива{НомПерв1 To НомПосл1, НомПерв2

To НомПосл2, ...) [As [New] ИмяТипа]

Многомерные массивы

Массивы можно делать многомерными. Например, объявим массив – таблицу поля шахматной доски:

Dim chessTable (1 To 8, 1 To 8) As String

Этот массив представляет собой таблицу с восьми ячейками по вертикали и горизонтали.

К элементам массива нужно обращаться по индексу, к примеру, чтобы изменить нулевой элемент массива *myArray* нужно написать:

myArray(0) = 1234

Или, например:

chessTable (2,3) = "Пешка"

Массивы переменной размерности (динамические)

Динамические массивы – это такие массивы, размерность которых может меняться в ходе работы программы. Пожалуй, динамические массивы используются даже чаще статических. Рассмотрим характерный пример использования такого массива. Пусть у нас есть процедура, которая загружает содержимое двоичного файла в массив, который можно определить так:

Dim fileContent (119) As Byte

Но это если файл имеет длину 120 байт. Если мы длина загружаемого файла неизвестна, то используют динамический массив, опеределяемый следующим образом:

Dim myArray () As Byte

В отличие от массивов статичных размеров, когда обращаться к элементам можно сразу после его объявления, к элементам динамического массива сразу обращаться нельзя, т.к. они ещё не инициализированы. Для начала нужно указать его новую размерность. Для это в VB есть оператор *ReDim*. Работает он следующим образом:

ReDim myArray (4)

Теперь массив *myArray* имеет одну размерность с индексами от 0 до 4 (т.е. всего 5 элементов) и обращаться к такому массиву можно точно так же, как и к статичному. Если в дальнейшем возникнет необходимость снова изменить размерность массива, можно ещё раз использовать *ReDim*.

Рассмотрим пример:

```
Dim myLong As Long
Dim myArray() As Long ' объявляем массив
ReDim myArray (2) ' одна размерность [0,1,2]
myArray (1) = 234 ' присваиваем второму элементу число 234
myLong = myArray (1) ' сохраняем его в переменной myLong
ReDim myArray (3) ' снова меняем размерность-теперь [0,1,2,3]
myLong = myArray (1) ' снова пытаемся сохранить второй элемент
```

На последней строке, переменной *myLong* присвоится 0 вместо 234! Это происходит потому, что оператор *ReDim* заново инициализирует (сбрасывает) все элементы массива к значению по умолчанию (как помните, для чисел – это

0, для строк ""). Для изменения размеров массива и сохранения всех старых элементов нужно после оператора *ReDim* поставить слово *Preserve*. Например:

ReDim Preserve myArray (3) ' сохраняем старые элементы

myLong = myArray (1) ' всё в порядке

Массивы могут храниться в переменных типа *Variant*. Если массив имеет тип *Variant*, то отдельные элементы могут содержать данные разных типов. Например, одни элементы могут быть числами, другие — строками или объектами. Чтобы сохранить какой-либо массив в переменной типа *Variant* необходимо просто присвоить этой переменной нужный массив:

Dim myVariantArray ' переменная *Variant* по умолчанию

myVarianrArray = chessTable

Теперь можно использовать копию как обычный массив:

myVarianrArray (0) = "Это копия"

Встроенные функции Visual Basic – *Lbound* и *UBound* используют, если потребуется в коде программы узнать текущие размеры массива. Первая функция возвращает нижнюю границу массива, вторая верхнюю

9.2. Задания

1. В одномерном массиве, содержащим *N* элементов найти сумму положительных элементов (рис. 46).

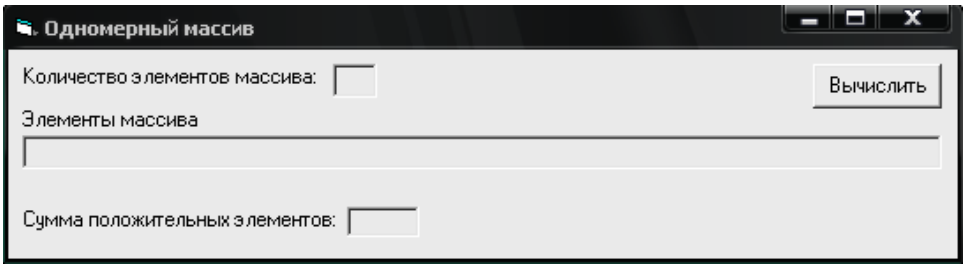


Рис. 46. Реализации формы приложения

2. В двумерном массиве размерности N на M найти сумму элементов, кратных числу 3 (рис. 47).

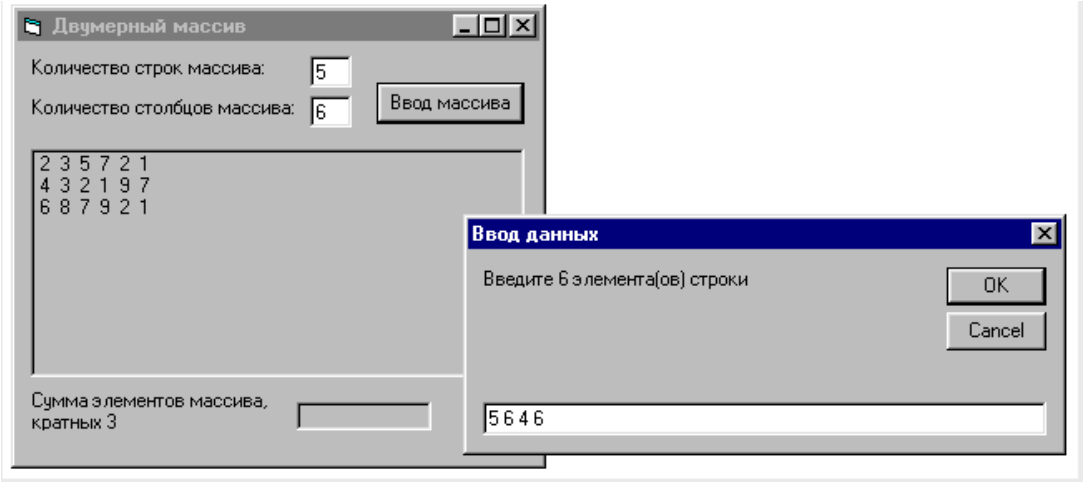


Рис. 47. Реализации формы приложения

3. Дан двумерный массив размером n*m, заполненный случайными числами. Определить, есть ли в данном массиве строка, содержащая больше положительных элементов, чем отрицательных (рис. 48).

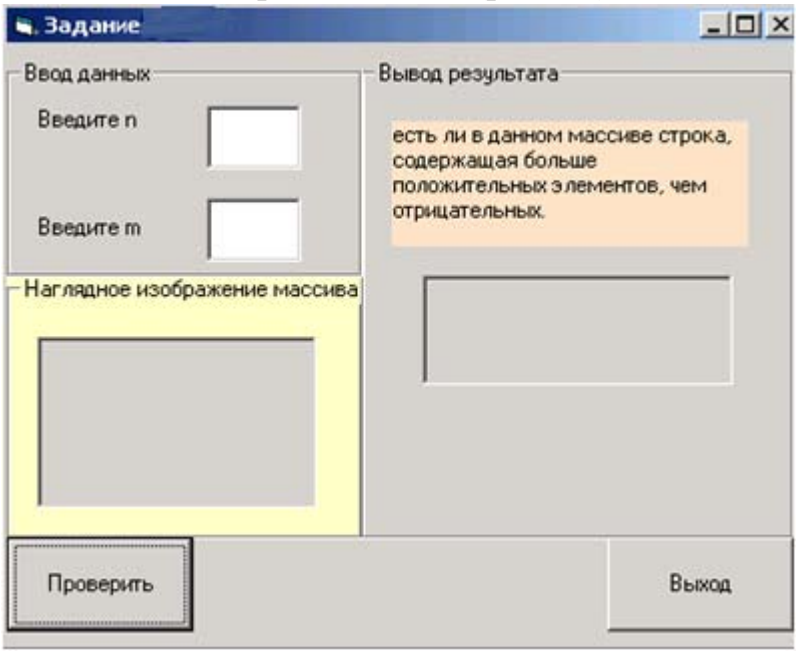


Рис. 48. Реализации формы приложения

9.3. Порядок выполнения работы

Задание 1.

Разработка алгоритма решения задачи начинается с составления блок-схемы алгоритма (рис. 49).

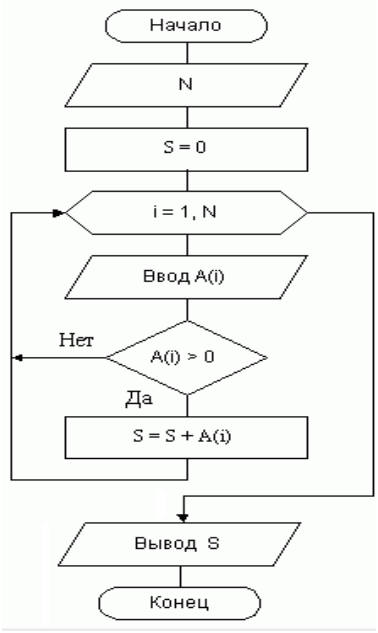


Рис. 49. Блок схема алгоритма решения задачи

Код программы:

```
Dim Am(50) As Double
Private Sub Command1_Click()
Picture1.Cls
s=0
av=Split (Text2, " " ) ' разбивает строку по разделителю
For i = 0 To Text1 - 1
        Am(i) = Val(av(i))
        If Am(i) >= 0 Then s=s+Am(i)
Next i
Picture1.Print s
End Sub
```

Задание 2.

Блок схема алгоритма решения задачи приведена на рис. 50.

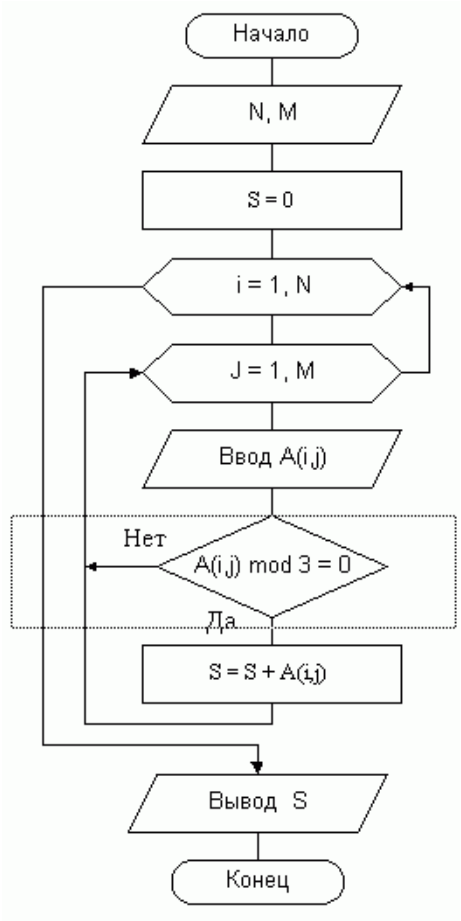


Рис. 50. Блок схема алгоритма решения задачи

Ввод элементов массива – построчный из модального диалогового окна по запросу приложения с помощью функции **InputBox.** Вывод элементов двумерного массива в графическое поле (**PictureBox**).

Код программы:

```
Dim A(50,50) As Double
Private Sub Command1_Click()
   Picture1.Cls
   s=0
   txt="Введите"+ Text2+"элемент(ов) строки "
   For i = 0 To Text1 – 1
   x=unputBox(txt, "Ввод данных" )
    av=Split (x, " " )
         If (Am(i, j) Mod 3)>= 0 Then s=s+ Am(i, j)
         Picture1.Print Am(i, j)
   Next i
   av= " "
   Picture2.Print s
   End Sub
```

Задание 3.

Результат выполнения задания приведен на рис. 51.

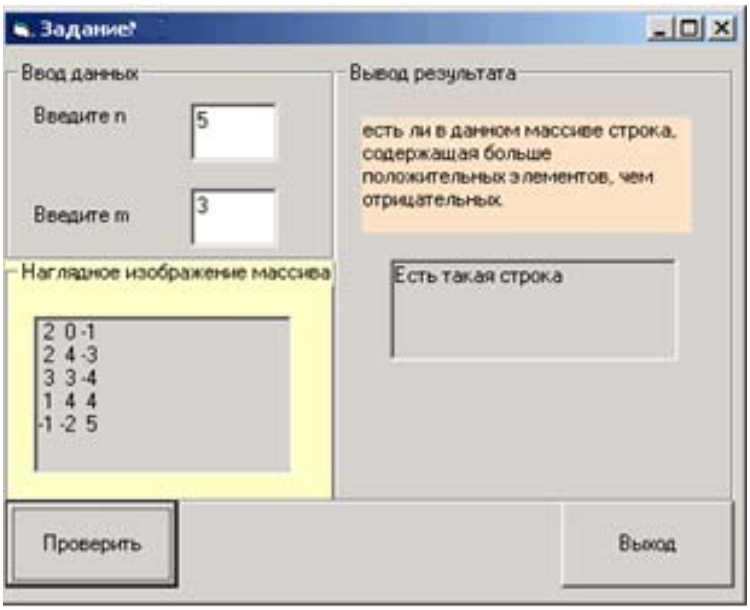


Рис. 51. Реализации формы приложения

Код программы:

```
Dim A(100) As Integer
Private Sub Command1_Click()
n = Val(Text1)
Picture1.Cls
Picture2.Cls
If n > 0 Then
 Randomize
 k = 1
```

```
For i = 0 To n - 1
    A(i) = Int((20 - -10 + 1) * Rnd - 10)
    Picture1.Print A(i)
     If A(i) >= Abs(5) Then
          Else
                k = k * A(i)
                l = k
    End If
 Next i
 Picture2.Print l
Else
  Picture1.Print "N-ДолжноБыть>0"
End If
End Sub
```

9.4. Контрольные вопросы

1. Что такое массив, его размер?
2. Могут ли элементы одного и того же массива иметь разный тип?
3. Как выполняется обращение к элементам массива?
4. Что понимается под диапазоном значений индекса массива?
5. Как задается размерность массива при его описании?
6. Какое значение принимает нижняя граница диапазона значений индекса по умолчанию?
7. Чем ограничена верхняя граница диапазона значений индекса?
8. Как можно организовать ввод (вывод) элементов одномерного (двумерного) массива?
9. Для чего при работе с массивами используется функция Split?

9.5. Задания для самостоятельной работы

(одномерные массивы)

Решить задачи с использованием ввода элементов из текстового поля. Для вывода результатов расчета используйте графические окна и форматный вывод.

Номер варианта	**Задание**
1	***2***
1.	Дан массив целых чисел из ***n*** элементов, заполненный случайным образом из промежутка [–10; 10]. Найти сумму элементов, имеющих нечетное значение. Вывести индексы тех элементов, значения которых больше заданного числа ***A.*** Определить, есть ли в данном массиве положительные элементы, кратные заданному числу ***K.***

1	*2*
2.	Дан массив целых чисел из ***n*** элементов, заполненный случайным образом из промежутка [–15; 15]. Найти произведение элементов, имеющих четное значение. Вывести индексы тех элементов, значения которых меньше заданного числа ***A.*** Определить, есть ли в данном массиве положительные элементы, делящиеся на заданное число *k* с остатком 2.
3.	Найти сумму элементов, имеющих нечетные индексы в массиве целых чисел из ***n*** элементов, заполненном случайным образом из промежутка [–10; 10].

Лабораторная работа № 10

Комбинированный тип данных (записи)

Цель: приобрести практический опыт по созданию и использованию переменных структурного типа – записей.

10.1. Методические указания

Пользовательский тип данных

Кроме встроенных типов данных, таких как *Integer, Long* и т.п., VB поддерживает также типы данных, определяемые пользователем. Они могут быть созданы как на основе стандартных типов данных, так и на основе ранее определенных пользователем типов данных.

Для определения пользовательского типа данных используется оператор *Type*, размещаемый в секции *(General) (Declarations*) модуля:

{Public | Private} Type имя_типа
имя_элемента [([нижняя_гр To верхняя_гр],..)] As тип_данных
...
End Type

Доступ к элементам переменной пользовательского типа осуществляется в следующей форме:

имя_переменной.имя_элемента[([индекс],..)]

Объявление массива элементов типа запись:

Dim ИмяПеременной As ИмяТипа

Переменным пользовательского типа можно присваивать значения других переменных того же типа.

Запись – это структура данных, состоящая из фиксированного числа компонентов разного типа. Составляющие запись компоненты называется полями записи. Записной тип еще называют комбинированным типом.

Записной тип данных предоставляет возможность объединить в одну связанную структуру различные по типу и смыслу элементы. Элементами записи могут быть и структурированные типы данных, например массивы и другие подчиненные записи. Для обработки доступна как вся запись целиком, так и отдельные ее поля.

Запись – это новый, определяемый тип данных, который состоит из одной и более переменных внутри. Например, необходимо в программе хранить массив студентов. Причём каждый студент имеет свои характеристики: ФИО, Возраст, наличие Грамот. Для хранения таких данных можно использовать, например, массив, имеющий две размерности, но в этом случае лучше использовать запись, из которой затем можно будет сделать массив. Чтобы определить запись в программе нужно использовать зарезервированное слово *Type*. Заканчивается запись словами *End Type*:

Private Type Student ' вместо Private могло быть и Public
* FIO As String*
* Age As Byte*
* HasGramot As Boolean*
End Type

Перед именем переменной *Dim* указывать не нужно. После определения записи в программе можно объявлять переменные, имеющий тип – *Student* (т.е. наша новая запись). Например: *Dim newStud As Student*

Слово *Student* синим выделяться не будет, т.к. синюю подсветку имеют только зарезервированные слова, встроенные в Visual Basic

Теперь, к полям записи можно обращаться при помощи точки:

newStud.FIO = "Василий Васильевич Васильченко"
newStud.Age = 19
newStud.HasGramot = False

Visual Basic предоставляет возможность не указывать каждый раз имя переменной типа запись, при обращении к её элементам. Это особенно полезно, когда запись имеет много внутренних членов. Для этого есть инструкция *With*:

With newStud
.FIO = "Егор Алексеевич Бабаев"
.Age = 20
.HasGramot = True
End With

Объявление массива элементов типа запись (точнее типа *Student*):

Dim myStudArray (20) As Student

Здесь объявлен массив из 21 студента. Теперь можно обращаться к элементам массива точно так же, как мы это делали раньше:

myStudArray(0).FIO = " Лев Алексеевич Вайнилович"

Инструкция With

Инструкция *With* позволяет указывать объект только один раз для последовательности инструкций. Инструкция *With* ускоряет выполнение процедур и помогает избежать повторного задания имени объекта.

Command1.Caption = "Вычислить"
Command1.Font.Bold = True
Command1.Visible = False

С применением инструкции *With* этот программный код следовало бы записать так:

With Command1
.Caption = "Вычислить"
.Font.Bold = True
.Visible = False
End With

Для увеличения эффективности программы возможно создание вложенных инструкций *With*. Это показывает следующий пример 1.

Пример 1.

Command1.Caption = "Вычислить"
Command1.Font.Name = "Arial"
Command1.Font.Bold = True
Command1.Font.Size = 15
Command1.Visible = False

Эквивалентный по результату действия программный код:

With Command1
.Caption = "Вычислить"
With .Font
.Name = "Arial"
.Bold = True
.Size = 15
End With
.Visible = False
End With

Доступ к элементам переменной пользовательского типа осуществляется, по аналогии с доступом к свойствам, путем указания точки после имени переменной. При этом переменные одинакового типа можно присваивать не поэлементно, а напрямую как в примере 2.

Пример 2.

Type Субъект
Фамилия As String
ТабельныйНомер As Integer
End Type
Читатель As Субъект, Пользователь As Субъект
Private Sub Commandl_Click()
Пользователь.Фамилия = "Иванов И.И."
Пользователь.ТабельныйНомер = 218739
Читатель = Пользователь
End Sub

Переменные *Читатель* и *Пользователь* относятся к одному типу *Субъект*. Поэтому они присваиваться напрямую, а не поэлементно.

Пользовательские типы данных могут быть составными. В этом случае важна последовательность определения типов. Сначала нужно определить базисный тип, который будет использоваться далее в составных типах. Если не соблюдать это правило, то после запуска программы появится сообщение об ошибке. В примере 3 показано как можно использовать составные пользовательские типы данных:

Пример 3.

Type Персона
Имя As String
Фамилия As String

End Type
Type Клиент
Идентификатор As Персона
ДеньРождения As Date
End Type
Dim Покупатель As Клиент
Private Sub Command1_Click()
Покупатель.Идентификатор.Имя = "Иван"
Покупатель.Идентификатор.Фамилия = "Петров"
End Sub

Данные пользовательского типа рекомендуется использовать при обработке данных неизменной структуры.

Таблицы указателей

При сортировке элементов данных, программа организует из них некоторое подобие структуры данных. Этот процесс может быть быстрым или медленным в зависимости от типа элементов. Перемещение целого числа на новое положение в массиве может быть намного быстрее, чем перемещение определенной пользователем структуры данных. Если эта структура представляет собой список данных о сотруднике, содержащий тысячи байт информации, копирование одного элемента может занять достаточно много времени.

Для повышения производительности при сортировке больших объектов можно помещать ключевые поля данных, используемые для сортировки, в таблицу индексов. В этой таблице находятся ключи к записям и индексы элементов другого массива, в котором и находятся записи данных. Например, предположим, что вы собираетесь отсортировать список записей о сотрудниках, определяемый следующей структурой:

Type Emloyee
ID As Integer
LastName As String
FirstName As String
<и т.д.>
End Type
‘ Выделить память под записи.
Dim EmloyeeData(1 To 10000)

Чтобы отсортировать сотрудников по идентификационному номеру, нужно создать таблицу индексов, которая содержит индексы и значения *ID values* из записей. Индекс элемента показывает, какая запись в массиве *EmployeeData* **с**одержит соответствующие данные.

Type IdIndex
ID As Integer
Index As Integer
End Type

‘ Таблица индексов.

Dim IdIndexData(1 To 10000)

Проинициализируем таблицу индексов так, чтобы первый индекс указывал на первую запись данных, второй — на вторую, и т.д.

For i = 1 To 10000

IdIndexData(i).ID = EmployeeData(i).ID

IdIndexData(i).Index = i

Next i

Отсортируем таблицу индексов по идентификационному номеру *ID.* После этого, поле *Index* в каждом элементе *IdIndexData* указывает на соответствующую запись данных. Например, первая запись в отсортированном списке — *это EmployeeData(IdIndexData(1).Index).*

Для того чтобы сортировать данные в разном порядке, можно создать несколько различных таблиц индексов и управлять ими по отдельности. В приведенном примере можно было бы создать еще одну таблицу индексов, упорядочивающую сотрудников по фамилии. Подобно этому списки со ссылками могут сортировать список различными способами. При добавлении или удалении записи необходимо обновлять каждую таблицу индексов независимо.

Сортировка — одна из наиболее активно изучаемых тем в компьютерных алгоритмах по ряду причин.

Во-первых, сортировка — это задача, которая часто встречается во многих приложениях. Почти любой список данных будет нести больше смысла, если его отсортировать каким-либо образом. Часто требуется сортировать данные несколькими различными способами.

Во-вторых, многие алгоритмы сортировки являются интересными примерами программирования. Они демонстрируют важные методы, такие как частичное упорядочение, рекурсия, слияние списков и хранение двоичных деревьев в массиве.

10.2. Задания

1. Определить тип данных *Товар,* объявив переменную *Инструмент* типа *Товар*, и установить конкретные значения составляющих этой переменной.
2. Определить запись типа *TComputer*, к полям которой будет производиться доступ, используя инструкцию *With*
3. Разработать структуру записи для хранения следующей информации: ФИО студента, домашний адрес, дата рождения. Организовать и ввести с клавиатуры массив записей из 5 элементов. Отсортировать записи по полю "ФИО студента" в алфавитном порядке.
4. Разработать структуру записи для хранения следующей информации: название товара, цена, количество. Организовать и ввести с клавиатуры массив записей из 5 элементов. Рассчитать стоимость каждого товара и суммарную стоимость всех товаров.

5. Разработать структуру записи для хранения следующей информации: ФИО абонента, домашний адрес, номер телефона. Организовать и ввести с клавиатуры массив записей из 5 элементов. По указанному значению поля "ФИО абонента" определить домашний адрес и номер телефона.

10.3. Порядок выполнения работы

Задание 1.

```
Type Товар
Название As String
Цена As Currency
Код As String
End Type
Dim Инструмент As Товар
Инструмент.Название = "Отвертка"
Инструмент.Цена =120
```

Задание 2.

```
Public Type TComputer
   OS As String
   CPU As String
   RAM As Byte
End Type
Dim Computer As TComputer
With Computer
   .OS = "Windows NT 4.0"
   .CPU = "AMD Athlon"
   .RAM = 128
End With
```

или

```
Computer.OS = "Windows NT 4.0"
Computer.CPU = "AMD Athlon"
Computer.RAM = 128
```

10.4. Контрольные вопросы

1. Как определить пользовательский тип данных?
2. Что такое запись?
3. Поясните как объявить массив элементов типа запись?
4. Как осуществляется доступ к элементам переменной пользовательского типа данных?
5. Что означает инструкция *With*?

Лабораторная работа № 11

Работа с файлами и строками

Цель: приобрести практические навыки в работе с файлами последовательного доступа.

11.1. Методические указания

Текстовые файлы последовательного доступа предназначены для записи и чтения неструктурированных данных. Достоинством файлов последовательного доступа является простота их создания и использования. При необходимости, файл последовательного доступа может быть создан или отредактирован любым текстовым редактором. Разделителем текста в файлах последовательного доступа является символ возврата каретки, который формируется автоматически при нажатии клавиши Enter.

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.

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*

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.

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.

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. 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.

Работа с файлами последовательного доступа состоит из двух самостоятельных операций: создания файла и использования файла.

Создание файла последовательного доступа:

1. Открытие файла ‘ (команда *Open* или *Append* с опцией *Output*).
2. Запись данных в файл (операторы *Write #* или *Print #*).
3. Закрытие файла ‘ (команда *Close*).

Чтение данных из файла последовательного доступа:

1. Открытие файла ‘ (команда *Open* с опцией *Input*).
2. Чтение данных из файла (оператор *Input #* или *Line Input #*).
3. Закрытие файла ‘ (команда *Close*).

Запись данных в файл последовательного доступа

Для записи данных в файл последовательного доступа используются операторы *Print # и Write #.*

При использовании оператора *Print* числовые данные, записываемые в файл, необходимо преобразовывать в строку символов, особенно это касается вещественных чисел, так как десятичную точку программа воспринимает как разделитель данных. Поэтому при работе с числами предпочтительнее использовать оператор *Write #.*

Чтение данных из файла последовательного доступа осуществляется операторами *Input #, Line Input # .*

Оператор ***Line Input #*** считывает из файла строку данных. Разделителем данных в файле в этом случае должен быть символ возврата каретки. Строка данных не должна превышать 255 символов.

Оператор ***Input #*** имеет следующий синтаксис:

Input # <номер канала>[, “текстовое сообщение”],<список переменных>

Переменные в списке разделяются запятыми.

Пример 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 2. Использование файла последовательного доступа.

```
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 #.

11.2. Задания

1. Добавить в файл с именем *Data.dat* имя и фамилию пользователя, которые он напишет в текстовых полях.Разместите на форме следующие элементы :

Наклейка (*Label1*) – с параметром Caption - Имя

Наклейка (*Label2*) – с параметром Caption - Фамилия

Текстовое окно (*Text1*) – в окне свойства Text удалите всё

Текстовое окно (*Text2*) – в окне свойства Text удалите всё

Кнопка (*Button1*) – с Caption'ом – Записать

Кнопка (*Button2*) – с Caption'ом – Выйти

2. Считать все данные из файла *autoexec.bat* который расположен в корневом каталоге C:\ и вывести их в текстовое окно. Создайте форму, расположив на ней такие объекты:

Текстовое окно (*TextBox*): *Name-Text1*, *Caption*-ничего *Multiline-True*

Кнопка (*Button): Name-Button1, Caption*-"Загрузить"

Кнопка *(Button): Name-Button2, Caption*-"Выход"

3. Считать все данные из файла *autoexec.bat*, который расположен в корневом каталоге C:\ и выводить их в текстовое окно. Создайте форму, расположив на ней следующие объекты:

Текстовое окно *(TextBo): Name-Text1, Caption*-ничего *Multiline-True*

Кнопка *(Button): Name-Button1, Caption*-"Загрузить"

Кнопка *(Button): Name-Button2, Caption*-"Выход"

11.3. Порядок выполнения работы

Задание 1. Запись в файл

Результат выполнения задания приведен на рис. 52.

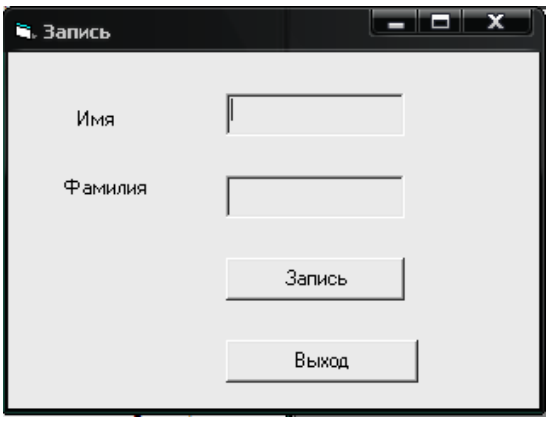


Рис. 52. Реализация формы приложения

```
Option Explicit
Dim fn As String
Dim ln As String
Private Sub Command1_Click()
fn = Text1.Text
ln = Text2.Text
```

Open "data.dat" For Append As #1
Print #1, fn;" ";ln
Close #1
End Sub
Private Sub Command2_Click()
End
End Sub

Осуществите проверку программы, при необходимости откомпилируйте ее. Запустите файл с расширением *.exe*, введя свое имя и фамилию, нажмите Записать. Теперь введите ещё чьи-нибудь имя и фамилию и опять запишите их. Выйдите из программы и зайдите в тот каталог, откуда вы запускали ее. Найдите там файл *Data.dat* и просмотрите его любым текстовым редактором, там должны быть те данные, которые вы вписали.

Задание 2. Считывание данных из файла

Результат выполнения задания приведен на рис. 53.

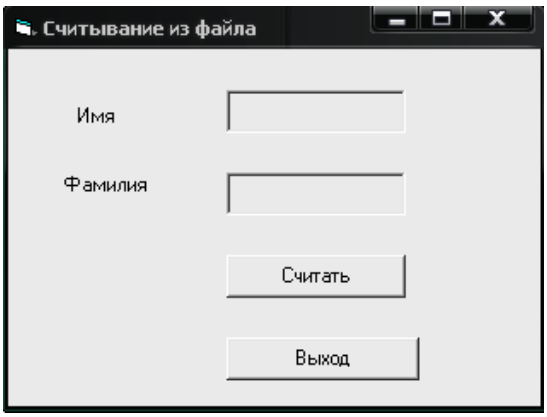


Рис. 53. Реализации формы приложения

Код программы:

Option Explicit
Dim st as String
Dim AllText as String
Open "C:\autoexec.bat" for Input as #1
Do While Not EOF (1)
Loop
Close #1
Text1.Text = AllText

Проверьте программу, открыв файл *data.dat из примера 1*.

11.4. Контрольные вопросы

1. Какие типы файлов данных Вам известны, и чем они отличаются?
2. Приведите синтаксис команды *Open* и *Close.*
3. Какие команды используются для чтения, записи данных из файла последовательного доступа?
4. Какая последовательность команд необходима для создания и чтения файла последовательного доступа?

Лабораторная работа № 12

Работа с графикой

Цель: изучение возможностей разработки графических приложений. приобретение навыков построения графиков функций.

12.1. Методические указания

Есть два объекта контейнера, способные содержать в себе точечный рисунок из графического файла и позволяющие рисовать на своей поверхности с помощью графических методов – это форма и элемент *PictureBox (графическое поле)*. Оба они могут содержать в себе другие управляющие элементы и обладают графическими методами. Как форма, так и графическое поле обладают системой координат. По умолчанию начало отсчета находится в левом верхнем углу объекта. Ось X направлена вправо, ось Y вниз. Единицу измерения координат можно выбрать. За это отвечает свойство *ScaleMode*. По умолчанию в качестве единицы измерения выбран твип. Эта единица соответствует 1/1440 дюйма или 0,0176 миллиметра. Можно также выбрать в качестве единицы измерения пиксели (размер точки на рисунке), пункты или символы (применяется для текста), дюймы, сантиметры или миллиметры.

Графические управляющие элементы

Это три элемента управления: *Image (рисунок), Line (линия) и Shape (фигура)*. Элемент *Shape* за счет выбора соответствующих значений его свойства *Shape* способен превращаться в одну из шести геометрических фигур (окружность, овал, прямоугольник, квадрат, а также прямоугольник и квадрат со скругленными углами). С применением этих трех элементов управления можно создавать графику проще, чем посредством графических методов. Однако эта цель достигается за счет ограничения других возможностей – они не могут служить в качестве контейнеров для других элементов управления и не могут получать фокус в период выполнения.

Работа с изображениями

Изображения могут быть помещены в форме, в элементе управления графическое поле *(PictureBox*) и в элементе управления рисунок *(Image)*. VB позволяет загружать в приложение файлы *.jpg и .gif*, а также *.bmp, .dib, .ico, .cur, wmf и .emf.*

Во время разработки изображение может быть добавлено в форму или элемент управления двумя способами:

1. из графического файла с помощью свойства *Picture*;
2. через буфер обмена.

В последнем случае следует скопировать графику из другого приложения в буфер обмена, вернуться в VB, выбрать форму, рисунок или графическое поле и в меню *Edit (правка*) выбрать *Paste (вставить*).

Для удаления рисунка из объекта следует выделить этот объект, затем в окне свойств выбрать свойство *Picture*, двойным щелчком на значении этого свойства выделить его и, наконец, нажать клавишу *Delete*.

В период выполнения можно выполнить загрузку изображения несколькими способами.

Например, можно использовать функцию *LoadPicture* для присвоения полного имени файла свойству *Picture*. Следующая инструкция загружает файл *bmp1.bmp* в элемент управления pic1:

pic1.Picture = LoadPicture("c:\Pict\bmp1.bmp")

Можно также скопировать изображение из одного объекта в другой. Следующая инструкция копирует графику из элемента управления – рисунок img1 в элемент управления – графическое поле pic1:

img1.Picture = pic1.Picture

Для удаления изображения в период выполнения без замены его другим изображением можно воспользоваться функцией *LoadPicture*.

Следующая инструкция удаляет изображение из графического поля img1:

img1.Picture = LoadPicture("")

Использование графических методов

Для создания графики VB, в дополнение к графическим элементам управления, имеет несколько графических методов для применения в форме и графическом поле *(PictureBox*), приведенных далее в табл. 11.

Таблица 11

Имя метода	Назначение метода
Line	Рисует линию, прямоугольник или заполненное окно
Circle	Рисует круг, эллипс или дугу
Cls	Очищает всю графику и вывод метода *Print*
PaintPicture	Закрашивает графику в произвольно выбранных местах
Point	Возвращает значение цвета выбранной точки
Pset	Устанавливает цвет отдельной точки

Очистка области рисования выполняется применением метода *Cls*:

[Объект.] Cls

Объект в этом и в последующих синтаксических определениях графических методов можно не указывать. Если объект опущен, то действие метода относится к текущей форме.

Для нанесения точки применяется метод *Pset*:

[Объект.]Pset(x,y)[,Цвет], где x и y – координаты точки.

Например, нанесет точку синего цвета инструкция: *Pset(300,400), RGB(0,0,255)*

Здесь и в следующих определениях аргумент Цвет можно задавать функцией *RGB(R, G, B)*. Эта функция определяет цвет как смесь трех цветов: красного (R), зеленого (G) и синего (B), интенсивность каждого из которых задается числом из диапазона от 0 до 255.

Стереть точку можно инструкцией, которая задает для нее цвет фона:

Pset(300,400), BackColor

Для рисования линии, соединяющей точку *(x1, y1)* с точкой *(x2, y2),* следует применить метод *Line*:

[Объект.]Line [(x1, y1)] – (x2, y2)[, Цвет]

Линия включает первую точку, но не включает вторую, конечную точку. Это полезно при рисовании замкнутого контура. Последнюю точку линии можно обозначить инструкцией

Pset Step (0, 0)[, Цвет].

Слово *Step* перед координатами означает, что они отсчитываются относительно последней точки рисования.

Например, инструкция *Line (100, 200)–(150, 250)* эквивалентна инструкции *Line (100, 200)–Step(50, 50)*.

Если первая пара координат опущена, то линия будет нарисована от позиции текущей точки, которой является последняя точка рисования, до точки с координатами (x2, y2). Текущая точка может быть также задана с помощью координат текущей точки рисования *CyrrentX* и *CyrrentY*.

Например, следующие инструкции нарисуют треугольник:

CurrentX = 500

CurrentY = 1500

Line –(2000, 3000)

Line –(1000, 3000)

Line –(500, 1500)

Если аргумент Цвет опущен, то цвет линии определяется значением свойства *ForeColo*r объекта.

Для рисования прямоугольников вместе с методом *Line* используется аргумент B, например инструкция *Line (200, 200)–Step(1000, 1000), , B*
нарисует квадрат со стороной 1000 твипов. Две запятые перед аргументом B указывают, что аргумент *Цвет* опущен.

Для рисования окружностей, эллипсов, дуг и секторов может быть применен метод Circle. Ниже показан синтаксис этого метода:

[Объект.]Circle [Step](x, y), Радиус[, [Цвет][, [Нач], [Кон][, Вид]]]

Здесь приняты некоторые новые обозначения:

x, y – координаты центра;

Радиус – радиус окружности, для эллипса – размер более длинной полуоси;

Нач и Кон – выраженный в радианах угол начала и угол конца дуги, которые присутствуют или отсутствуют в обращении к методу только вместе;

Вид – отношение вертикальной полуоси эллипса к горизонтальной полуоси (по умолчанию равен 1).

Например, процедура *Form_Click* в режиме выполнения после щелчка на форме нарисует на форме дугу окружности с центром в точке с координатами 2000, 1500, радиуса 1000 твипов, от начальной точки, расположенной под

углом pi / 2, против часовой стрелки, до конечной точки, находящейся под углом pi / 3:

Private Sub Form_Click()
Const pi = 3.14159265
Circle (2000, 1500), 1000, , pi / 2, pi / 3
End Sub

Если перед началом или перед концом дуги поставить знак минус, то при рисовании дуги эта точка дуги будет соединена прямой с центром окружности.

Обращение *Circle (600, 1000), 800, , , , 2* нарисует эллипс, вертикальная ось которого равна 1600, а горизонтальная ось равна 800.

При рисовании прямоугольника, круга или эллипса эти фигуры могут быть заполнены сплошным цветом или цветными линиями.

Свойство *FillStyle* отвечает за вид заполнения рисуемой фигуры, предлагая на выбор один из восьми видов заполнения. По умолчанию имеет значение 1 – *Transparent* (прозрачное, фигура выглядит незаполненной). Значение этого свойства 0 – *Solid* обеспечивает заполнение фигуры сплошным цветом, установленным свойством *FillColor*. Остальные значения 2 – 7 предлагают заполнение вертикальными, горизонтальными, наклонными линиями или в клетку.

Толщина линии при выводе задается значением свойства *DrawWidth* объекта.

Будет линия сплошной или различного вида прерывистой – определяет значение свойства *DrawStyle* объекта.

Свойство *AutoRedraw* формы или графического поля разрешает (при его значении, равном *True*) или запрещает (при его значении, равном *False*) перерисовывание результатов работы графических методов, например при изменении размеров окна.

Пример 1. Создать проект **"Цапфа"** чертежа некоторой детали из стали, по геометрическим размерам, которой, выраженным в миллиметрах, вычисляется ее вес детали по удельному весу материала, выраженному в граммах на кубический сантиметр, используя следующие способы построения чертежа:

- с применением графических управляющих элементов для построения чертежа детали;
- с использованием детали графических методов вместо графических объектов для получения чертежа

Разработаем проект с применением графических управляющих элементов для построения чертежа детали следующим образом:

1. Запустите систему VB, в диалоговом окне *New Project* выберите значок *Standard EXE* и нажмите *OK*.
2. Для сохранения формы и проекта в рабочей папке щелкните на кнопке *Save Project* панели инструментов, перейдите в корневой каталог вашего

диска, с помощью кнопки *Создание новой папки* создайте свою рабочую папку, откройте ее, щелкнув на кнопке *Сохранить* окна *Save Project As* для сохранения формы и еще раз щелкните на этой же кнопке для сохранения проекта.

3. С помощью управляющих элементов *Line* и *Shape* получите на форме чертеж цапфы (рис. 54). В соответствии с этим же рисунком поместите на форме кнопку, четыре надписи и пять текстовых полей.

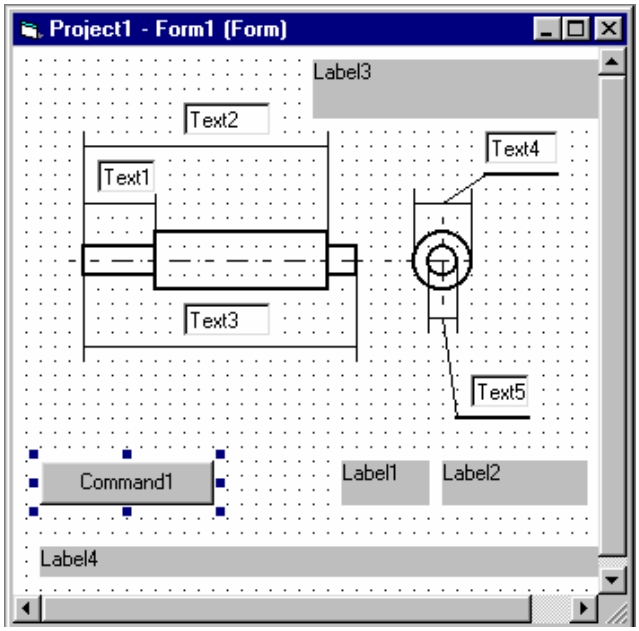


Рис. 54. Расположение объектов

4. Сохраните проект и форму в своей рабочей папке.
5. Установите значения свойств объектов проекта, чтобы интерфейс соответствовал рис. 55.

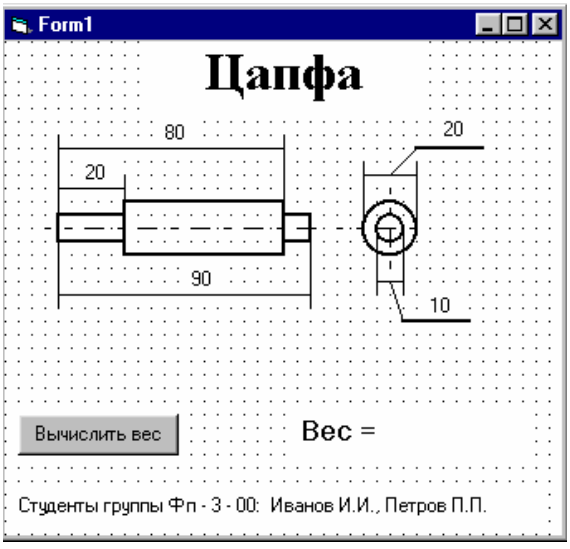


Рис. 55. Интерфейс проекта

6. Сохраните проект.

7. Введите программный код:

```
Option Explicit
Private Sub Command1_Click()
Const pi As Single = 3.14159
Dim УдельныйВес As Single
УдельныйВес = InputBox _
("Введите удельный вес материала" & _
"в граммах на кубический сантиметр")
Label2.Caption = Format((Text1.Text _
* pi * Text5.Text ^ 2 / 4 + (Text2.Text _
- Text1.Text) * pi * Text4.Text ^ 2 / 4 + _
(Text3.Text - Text2.Text) * pi * _
Text5.Text ^ 2 / 4) / 1000 * УдельныйВес, _
"###0.0 г.")
End Sub
```

Эта программа при щелчке на кнопке *Command1* вычисляет вес детали по удельному весу материала, выраженному в граммах на кубический сантиметр, и по геометрическим размерам детали, выраженным в миллиметрах.

8. Запустите приложение, щелкнув на кнопке *Command1*, наберите в окне функции значение удельного веса стали (7,8 г/см^3) и щелкните на кнопке ОК. Результат работы Вашего приложения должен соответствовать (рис. 56).

В режиме выполнения приложения можно изменять размеры детали, изменяя содержание соответствующих текстовых полей, и пересчитывать вес детали после нового щелчка на кнопке *Command1*.

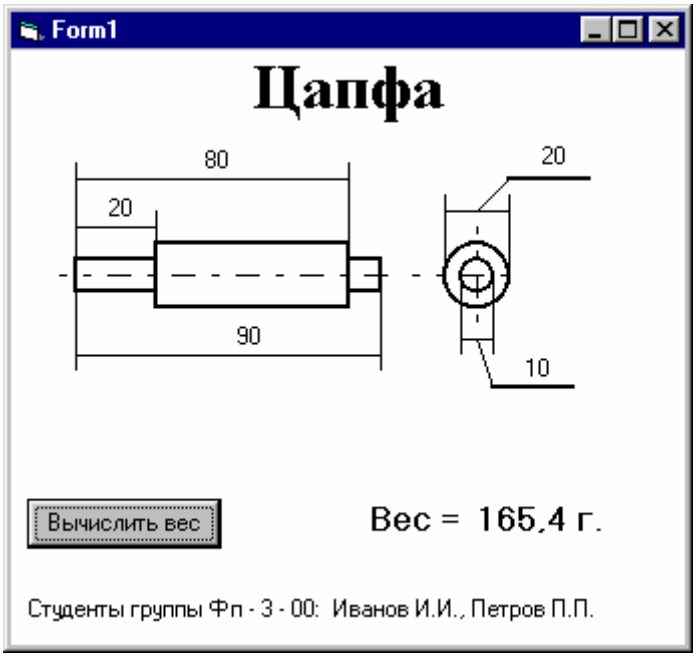


Рис. 56. Работа проекта, в котором применены графические объекты

Можно модернизировать проект с целью применения для получения чертежа детали графических методов вместо графических объектов. Для этого :

1. Удалите на форме все графические объекты *Line* и *Shape*.

2. Добавьте в программный код объекта следующие три процедуры:

```
Private Sub СтрелкаГЛ(x As Single, y As Single)
Line (x, y)-(x + 150, y - 20)
Line (x, y)-(x + 150, y + 20)
End Sub
Private Sub СтрелкаГП(x As Single, y As Single)
Line (x, y)-(x - 150, y - 20)
Line (x, y)-(x - 150, y + 20)
End Sub
Private Sub Form_Resize()
AutoRedraw = True
DrawWidth = 1
DrawStyle = 3
Line (200, 2000)-(2500, 2000)
Line (3000, 2000)-(4500, 2000)
Line (3750, 2500)-(3750, 1500)
DrawStyle = 0
DrawWidth = 2
Line (250, 2200)-(800, 1800), , B
Line (800, 2400)-(2000, 1600), , B
Line (2000, 2200)-(2300, 1800), , B
Circle (3750, 2000), 400
Circle (3750, 2000), 200
DrawWidth = 1
Line (250, 3000)-(250, 1000)
Line (2300, 3000)-(2300, 2200)
Line (250, 2900)-(2300, 2900)
СтрелкаГЛ 250, 2900
СтрелкаГП 2300, 2900
Line (2000, 1600)-(2000, 1000)
Line (250, 1100)-(2000, 1100)
СтрелкаГЛ 250, 1100
СтрелкаГП 2000, 1100
Line (250, 1450)-(800, 1450)
СтрелкаГЛ 250, 1450
СтрелкаГП 800, 1450
Line (250, 1450)-(800, 1450)
Line (800, 1600)-(800, 1280)
Line (3330, 2000)-(3330, 1000)
Line (4150, 2000)-(4150, 1000)
```

Line (3330, 1120)-(4150, 1120)
СтрелкаГЛ 3330, 1120
СтрелкаГП 4150, 1120
Line (3550, 2000)-(3550, 3000)
Line (3950, 2000)-(3950, 3000)
Line (3550, 2880)-(3950, 2880)
СтрелкаГЛ 3550, 2880
СтрелкаГП 3950, 2880
End Sub

Здесь процедура *Form_Resize* работает при каждом изменении размеров или загрузке формы. Она обеспечивает с помощью обращений к методам Line и *Circle* рисование чертежа детали. Процедуры *СтрелкаГЛ* и *СтрелкаГП* обеспечивают рисование на горизонтальной размерной линии соответственно левой или правой стрелки. Сохраните проект.

3. Запустите приложение, щелкните на кнопке *Command1*, наберите в окне функции значение удельного веса стали (7,8 г/см^3) и щелкните на кнопке *ОК*. Результат работы Вашего приложения должен соответствовать рис. 57.

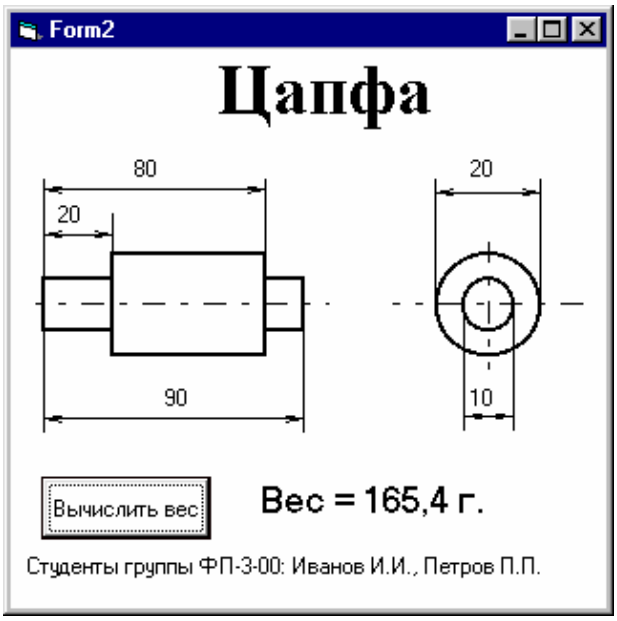


Рис. 57. Работа проекта, в котором применены графические методы

Остановите работу приложения

4. Добавьте в программный код процедуры *Form_Resize* после инструкции *AutoRedraw = True* следующие строки:

Dim X As Integer
For X = 0 To Height
Line (0, X)-(Width, X), _
RGB(X / (Height / 255), _
175, (Height - X) / (Height / 255))
Next

Этот фрагмент программы обеспечивает закраску формы изменяющимся цветом. Закраска обеспечивается рисованием в цикле цветных горизонтальных линий от левого края формы до правого края. Цвет линии в цикле изменяется.

Пример 2. Построить график функции y = -0.25x3 + 0.14x2 + 0.25 x – 25 на отрезке значений аргумента[-10,+10]. Анализ показывает, что область значений функции на этом отрезке: -270 < y < +270.

```
Option Explicit
Function Primer(x As Single) As Single
Primer = -0.25 * x ^ 3 + 0.14 * x ^ 2 + _
0.25 * x - 25
End Function
Private Sub Form_Click()
Dim x As Single
Scale (-10, -270)-(10, 270) 'установка масштаба
Cls 'очистка экрана
DrawWidth = 1 'установка толщины линии
Line (-10, 0)-(10, 0) 'рисование оси X
Line (0, -270)-(0, 270) 'рисование оси Y
CurrentX = -10 'текущее значение X
CurrentY = Primer(-10) 'текущее значение Y
For x = Step 0.5
Line -(x, Primer(x)) 'построение графика
Next
End Sub
```

Построенный график показан на рис. 58.

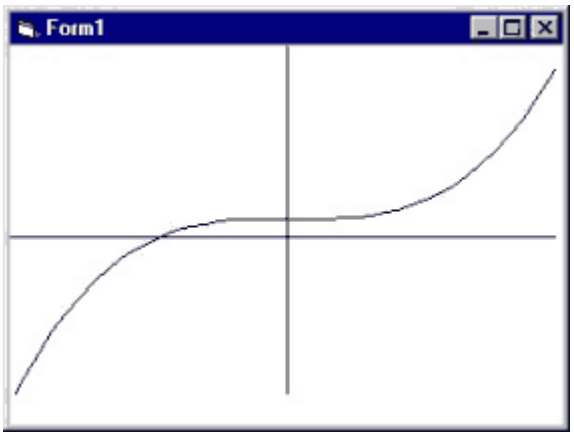


Рис. 58. Результат работы программы

12.2. Задания

1. Построить график функции $y = 3x^2$.

2. Создайте проект и напишите программу "Светофор", позволяющую по нажатию кнопки "включать" заданный свет светофора.

3. Построить график функции Y = Cos(x) + 1 на отрезке [0; 2*Pi]

12.3. Порядок выполнения работы

Задание 1.

Построенный график приведен на рис. 59.

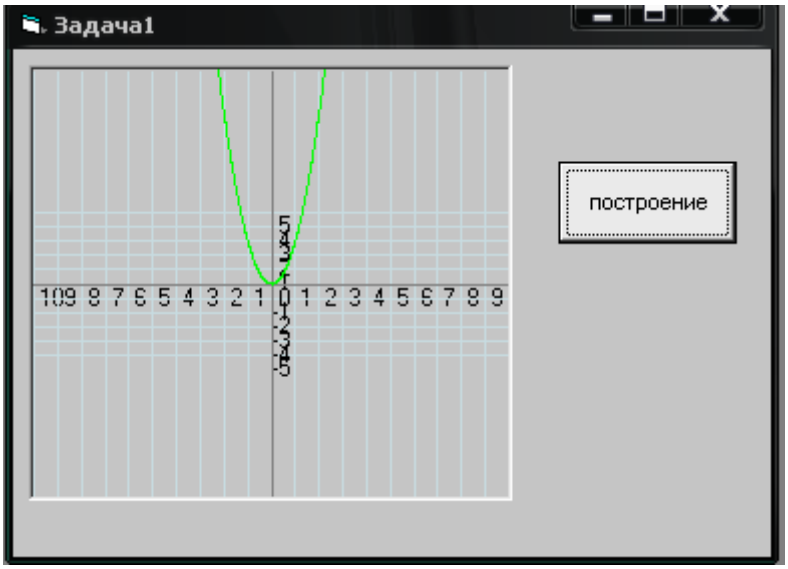


Рис. 59. Результат работы программы

Код программы:

```
Private Sub Command1_Click()
Picture1.Scale (-10, 15)-(10, -15)
For i = -10 To 10
Picture1.Line (i, -15)-(i, 15), RGB(200, 220, 225)
Picture1.PSet (i, 0)
Picture1.Print i
Next
For i = -5 To 5
Picture1.Line (-10, i)-(10, i), RGB(200, 220, 225)
Picture1.PSet (0, i)
Picture1.Print i
Next
Picture1.Line (-10, 0)-(10, 0), QBColor(8)
Picture1.Line (0, -15)-(0, 15), QBColor(8)
For x = -5 To 5 Step 0.01
y = 3 * x ^ 2
Picture1.PSet (x, y), vbGreen
Next
End Sub
```

Задание 2.

Созданный проект показан на рис. 60.

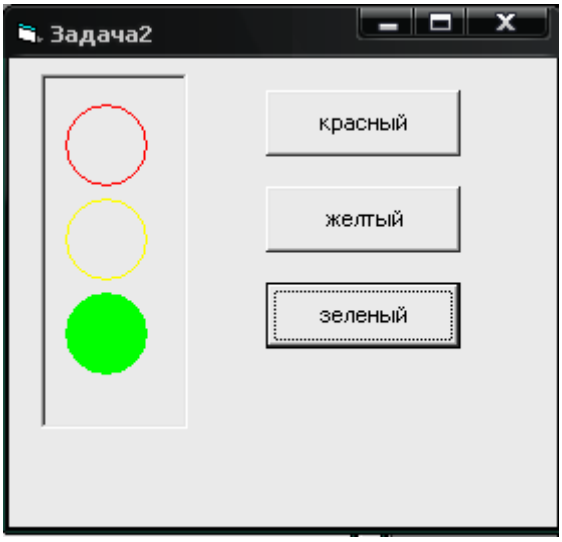


Рис. 60. Результат работы программы

```
Private Sub Form_Activate()
Picture1.Circle (450, 500), 300, RGB(255, 0, 0)
Picture1.Circle (450, 1200), 300, RGB(255, 255, 0)
Picture1.Circle (450, 1900), 300, RGB(0, 255, 0)
End Sub
Private Sub Command1_Click()
Picture1.FillColor = &H8000000F
Picture1.FillStyle = 0
Picture1.Circle (450, 1200), 300, RGB(255, 255, 0)
Picture1.Circle (450, 1900), 300, RGB(0, 255, 0)
Picture1.FillColor = QBColor(12)
Picture1.FillStyle = 0
Picture1.Circle (450, 500), 300, RGB(255, 0, 0)
End Sub
Private Sub Command2_Click()
Picture1.FillColor = &H8000000F
Picture1.FillStyle = 0
Picture1.Circle (450, 500), 300, RGB(255, 0, 0)
Picture1.Circle (450, 1900), 300, RGB(0, 255, 0)
Picture1.FillColor = QBColor(14)
Picture1.FillStyle = 0
Picture1.Circle (450, 1200), 300, RGB(255, 255, 0)
End Sub
Private Sub Command3_Click()
Picture1.FillColor = &H8000000F
```

```
Picture1.FillStyle = 0
Picture1.Circle (450, 500), 300, RGB(255, 0, 0)
Picture1.Circle (450, 1200), 300, RGB(255, 255, 0)
Picture1.FillColor = QBColor(10)
Picture1.FillStyle = 0
Picture1.Circle (450, 1900), 300, RGB(0, 255, 0)
End Sub
```

Задание 3.

Построенный график показан на рис. 61.

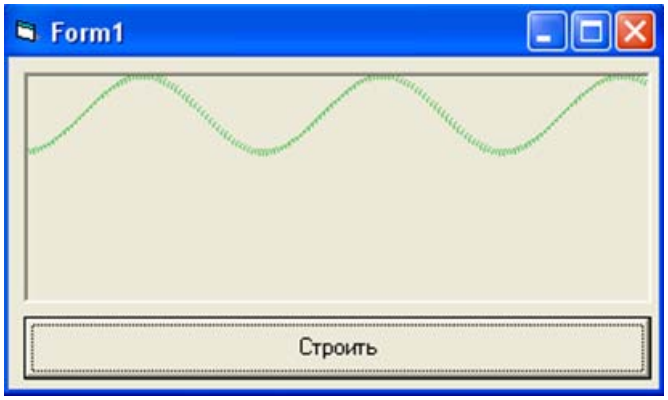


Рис. 61. Результат работы программы

```
Private Sub Command1_Click()
For i = 0 To 2 * 10 * 3.14 Step 0.1
y = Cos(x) + 1
Picture1.Circle (300 * x, 300 * y), 40, RGB(100, 200, 100), 2.4, 3.5
Next
End Sub
```

12.4. Контрольные вопросы

1. Какие управляющие элементы VB относятся к графическим элементам?
2. Какие свойства графического объекта определяют вид линии, ее толщину, ее цвет, а также вид заполнения фигуры?
3. Каково назначение и синтаксис метода Line?
4. Каково назначение и синтаксис метода Circle?
5. На что влияет значение свойства FillStyle?
6. На что влияет значение свойства FillColor?
7. На что влияет значение свойства DrawWidth?

ПРИЛОЖЕНИЯ

Приложение 1

Порядок выполнения всех лабораторных работ

1. Запустить программу Visual Basic.
2. В порядке, описанном в методических указаниях, изучить основные компоненты интегрированной среды разработки приложений Visual Basic.
3. Выполнить задания, приведенные в методических указаниях.
4. Сохранить результаты работы в личной папке.
5. Показать результаты работы преподавателю.
6. Хранить файлы результатов работы в личной папке до завершения семестра.

Требования к оформлению отчета

Отчет должен содержать тему, цель занятия, описание порядка выполнения пунктов задания, ответы на контрольные вопросы, т.е. по результатам работы должен быть составлен отчет, содержащий:

1) название, цель и задачи лабораторной работы;
2) краткое описание основных компонентов интегрированной среды разработки приложений Visual Basic;
3) описание приемов создания формы приложения;
4) файл с проектом, представляющим собой форму с размещенными на ней объектами (на дискете);
5) файл с проектом простейшего приложения (на дискете);
6) заключения и выводы.

Приложение 2

Таблица П2.1

Встроенные функции VisualBasic. Математические функции

Функции	Возвращаемое значение
Abs (число)	Модуль (абсолютная величина) числа
Atn (число)	Арктангенс
Cos (число)	Косинус
Exp (число)	Экспонента, т. е. результат возведения основания натурального логарифма в указанную степень
Log (число)	Натуральный логарифм
Rnd (число)	Случайное число из интервала [0; 1). Если число меньше нуля, то Rnd возвращает каждый раз одно и то же число. Если число больше нуля или опущено, то Rnd возвращает следующее случайное число в последовательности. Если число равняется нулю, то Rnd возвращает случайное число, возвращенное при предыдущем вызове этой функции. Перед вызовом функции Rnd используют инструкцию Randomize.
Sgn (число)	Знак числа
Sin (число)	Синус
Sqr (число)	Квадратный корень из числа
Tan (число)	Тангенс
Fix (число) Int (число)	Обе функции Int и Fix отбрасывают дробную часть числа и возвращают целое значение. Различия между функциями Int и Fix состоит в том, что для отрицательного значения параметра число функция Int возвращает ближайшее отрицательное целое число, меньшее либо равное указанному, а Fix – ближайшее отрицательное целое число, большее либо равное указанному

Таблица П2.2

Функции проверки типов

Функция	Проверка
IsArray (переменная)	Является ли переменная массивом.
IsDate (переменная)	Является ли переменная датой.
IsEmpty (переменная)	Была ли переменная описана инструкцией Dim.
IsError (переменная)	Является ли переменная кодом ошибки.
IsNull (переменная)	Является ли переменная пустым значением (Null).
IsNumeric(переменная)	Является ли переменная числовым значением.
IsObject (переменная)	Является ли переменная объектом.

Таблица П2.3

Функции преобразования форматов

Функция	Описание
Val (строка)	Возвращает число, содержащееся в строке, как числовое значение.
Str (число)	Возвращает значение типа Variant (String), являющееся строковым представлением числа.
Format	Возвращает значение типа Variant (String), содержащее выражение, отформатированное согласно инструкциям, заданным в описании формата. Синтаксис, например: *Format (Выражение [, Формат [, ПервыйДеньНедели [, ПерваяНеделяГода]]])* *Выражение* – любое допустимое значение; *Формат* – любое допустимое именованное или определяемое пользователем выражение формата. *ПервыйДеньНедели и ПерваяНеделяГода* – используются при задании формата даты. *Задают Первый День Недели и Первая Неделя Года*, соответственно.

Таблица П2.4

Именованные числовые форматы

Имя формата	Описание
General Number	Число без разделителя тысяч.
Currency	Использует установки страны в Панели управления. Отображает две цифры справа от десятичной точки.
Fixed	Отображает, по крайней мере, одну цифру справа и две справа от десятичной точки.
Standard	Отображает, по крайней мере, одну цифру справа и две справа от десятичной точки и выводит разделитель тысяч.
Percent	Отображает число в виде процентов и выводит две цифры справа от десятичной точки.
Scientific	Использует формат с плавающей десятичной точкой.
Yes/No	Отображает No, если число равно 0, и Yes – в противном случае.
True/False	Отображает False, если число равно 0, и True – в противном случае.
On/Off	Отображает Off, если число равно 0, и On – в противном случае.

Таблица П2.5

Именованные форматы даты и времени

Имя формата	Описание
General Date	Выводит дату или время. Если нет дробной части, то выводит только дату.
Long Date	Выводит дату в соответствии с полным форматом Windows для даты.
Medium Date	Выводит дату в соответствии с обычным форматом Windows для даты.
Short Date	Выводит дату в соответствии с сокращенным форматом Windows для даты.
Long Time	Выводит часы, минуты и секунды.
Medium Time	Выводит часы и минуты в 12-часовом формате.
Short Time	Выводит часы и минуты в 24-часовом формате.

Таблица П2.6

Функции преобразования типа

Функция	Тип, в который преобразуется выражение
CBool (Выражение)	Boolean
Cbyte (Выражение)	Byte
CCur (Выражение)	Currency
CDate (Выражение)	Date
CDbl (Выражение)	Double
Cdec (Выражение)	Decimel
CInt (Выражение)	Integer
CLng (Выражение)	Long
CSng (Выражение)	Single
CVar (Выражение)	Variant
CStr (Выражение)	String

Таблица П2.7

Функции обработки строк

Функция	Возвращаемое выражение
Asc	Возвращает ASCII код начальной буквы строки. *Синтаксис*: Asc(string)
Chr	Преобразует ASCII код в строку. *Синтаксис*: Chr(charcode) *Например*: Chr(13) – переход на новую строку. Chr(97) – возвращает букву "a". Возвращает ASCII код начальной буквы строки
LCase	Преобразует строку к верхнему регистру. *Синтаксис*: LCase(string)
UCase	Преобразует строку к нижнему регистру. *Синтаксис*: UCase(string)
Left	Возвращает подстроку, состоящую из заданного числа первых символов исходной строки. *Синтаксис*: Left(string, length)
Length	число возвращаемых символов подстроки.
String	строковое выражение, из которого извлекается подстрока.
Right	Возвращает строку, состоящую из заданного числа последних символов исходной строки. *Синтаксис*: Right(string, length)
Mid	Возвращает подстроку строки, содержащую указанное число символов. *Синтаксис*: Mid(string, start [, length])
Start	позиция символа в строке string, с которого начинается нужная подстрока.
Len	Возвращает число символов строки. *Синтаксис*: Len(string)
LTrim	Возвращает копию строки без пробелов в ее начале. *Синтаксис*: LTrim(string)
Space	Возвращает строку, состоящую из указанного числа пробелов. *Синтаксис*: Space(number)
RTrim	Возвращает копию строки без пробелов в ее конце. *Синтаксис*: RTrim(string)
Split	Преобразует строку в одномерный массив, нумеруемый с нуля. *Синтаксис*: Split(Expression, [Delimiter], [Limit], [Compare])

ЛИТЕРАТУРА

1. Волченков, Н.Г. Программирование на языке VB6: учеб. пособие. Часть 1 / Н.Г. Волченков. – М.: Инфра М, 2000. – 99 с.

2. Глаголев, В.Б. Visual Basic 6.0: Сборник заданий для лабораторно-практических занятий. Часть 2 / В.Б. Глаголев – М.: МЭИ, 2001. – 233 с.

3. Козлов, С.М. Программирование на языке VB / С.М. Козлов – Минск: ВУЗЮНИТИ, 2000. – 33 с.

4. Райтингер, М., Муч Г. Visual Basic 6.0 / М. Райтингер, Г. Муч; пер. с нем. – К.: Издательская группа BHV, 2000. – 99 с.

5. Сайлер, Б. Использование Visual Basic 6 / Брайн Сайлер, Джефф Споттс. – М.: Бином, 1999. – 178 с.

6. Симонович, С.В. Информатика базовый курс учебник для ВУЗов / С.В. Симонович, изд. дом “Питер”, 2004. – 65 с.

7. Угринович, Н.Д. Информатика и информационные технологии / Н.Д. Угринович. – 2 изд. – М.: БИНОМ. Лаборатория знаний, 2005.– 205 с.

8. Хальворсон, М. Microsoft Visual Basic 6.0 для профессионалов. Шаг за шагом / Микаэл Хальворсон. – М: Эком Год издания: 2005. – 364 с.