МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ
«Карачаево-Черкесский государственный университет имени У.Д. Алиева»

Программирование в Visual Basic 6.0

Салпагаров Х.М.
Бостанова М.М.

Программирование в Visual Basic 6.0

Учебно-методическое пособие

Карачаевск-2016

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФУДУРАЦИИ
«Карачаево-Черкесский государственный университет имени У.Д. Алиева»

Салпагаров Х.М.
Бостанова М.М.

Практикум решения задач на ЭВМ в среде Visual Basic 6.0

Карачаевск-2016

Печатается по решению редакционно-издательского совета Карачаево-Черкесского государственного университета им. У.Д. Алиева

УДК 004.42
ББК 32.81
Б85

Описание Для бакалавров по направлению подготовки
01.03.02 Прикладная математика и информатика
09.03.01 Информатика и вычислительная техника

Салпагаров, Х.М., Бостанова, М.М. «Практикум на ЭВМ. Программирование в Visual Basic». / Х.М.Салпагаров, М.М. Бостанова - Карачаевск: КЧГУ, 2016. – 168 с.

Рецензенты:
П.А. Кочкарова, к.ф.-м.н., доцент
А.М. Узденова , к.ф.-м.н., доцент
Х.Д. Шунгаров, к.ф.-м.н., доцент

Предисловие

Учебно-методическое пособие содержит теоретический и практический материал для изучения среды программирования Visual Basic. Пособие разработано для студентов направления подготовки «Прикладная математика и информатика».

Применение пособия по программированию в учебном процессе повышает уровень восприятия информации ее понимания, способствует развитию таких важных качеств, как интуиция, образное мышление. В практикуме использованы разнообразные интересные задания для увлекательного программирования. С целью повышения заинтересованности учащихся к программированию, эффективности изучения основ объектно-ориентированного программирования, в практикум включены экономические, математические, физические, графические и другие задачи «жизненной» направленности.

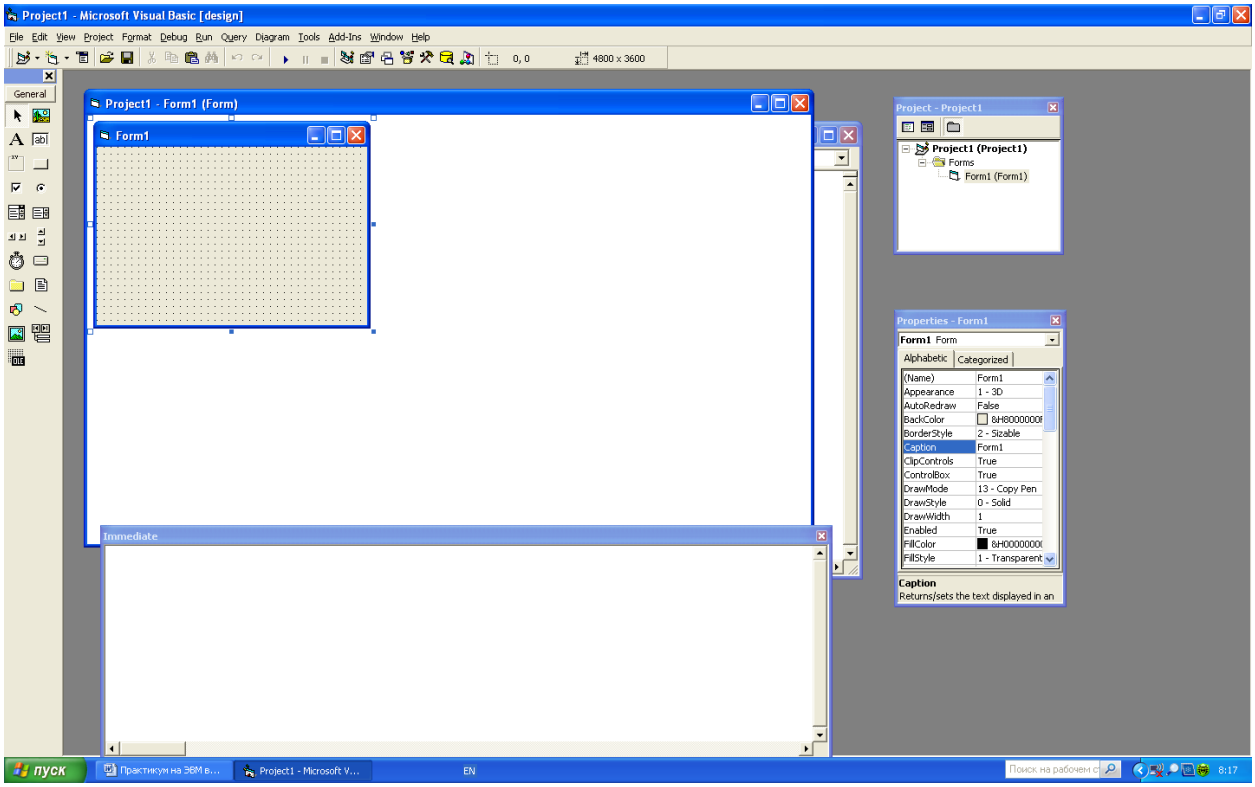
Здесь можно ознакомиться с основными понятиями Visual Basic, с типами данных, с арифметическими, логическими операциями, встроенными функциями, с назначением и работой разных операторов, где приведены примеры написания кодов программ, где можно найти необходимую информацию по теме практической работы, использовать данную вкладку как справочное руководство.

Следует отметить, что в практикум вошел не весь отобранный и переработанный материал, вошли не все задания, охвачены не все темы, не все программные моменты, а только самые важные темы по программированию в среде Visual Basic, его можно ещё доработать, учитывая ситуации, которые возникали во время апробации.

Успехов программирования в среде Visual Basic!

Глава 1. Среда программирования Visual Basic 6.0. Краткие сведения

Современные среды разработки программ являются интегрированными, то есть объединяют в себе несколько составляющих, таких как компилятор, редактор текстов программ, средства отладки, визуальный конструктор форм и т.п. На рисунке представлен общий вид интегрированной среды:



На рисунке представлены основные окна.

Меню – главное окно – верхнее окно, отображается при вызове имеющегося или нового проекта Visual Basic 6.0. При закрытии окна закрываются все окна. На нем присутствуют пункты меню, содержащие все команды приложения. Названия их видны при отображении проекта – **File, Edit, View, Project** и др.

File (файл) содержит команды, позволяющие сохранять проект Save Project As, сохранять изменения Save Project, открыть имеющийся проект Open Project и др.

Edit (редактировать) содержит команды и сочетания горячих клавиш, позволяющие редактировать текст программы.

View (Вид) содержит команды, позволяющие отображать основные и др. окна среды разработки.

Project (Проект) содержит команды, позволяющие добавлять и удалять *формы, модули и др*.

Окно ToolBox Панель управляющих элементов (*Отображается*: **View – ToolBox**). Содержит элементы управления конструирования формы. По умолчанию на панели располагаются только стандартные элементы управления, такие как **Label** (Надпись), **TextBox** (Текстовое поле), **CommandButton** (Командная кнопка) и т.п.

Окно Properties Window Окно свойств (*Отображается*: **View – Properties Window**). С формой и с каждым элементом управления формы связано окно свойств Properties Window, которое позволяет в режиме конструирования изменять свойства формы или элемента управления.

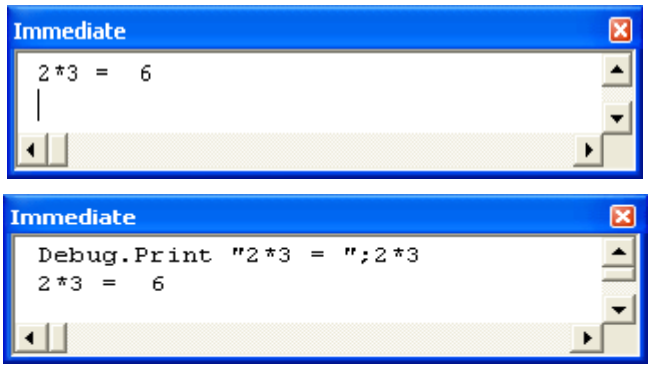
Окно Project Explorer Окно Проекта (*Отображается*: **View – Project Explorer**). Предназначено для отображения всех открытых проектов, а также их модулей, форм и ссылок на другие проекты.

Окно Object Окно формы (*Для отображения сначала в окне Проекта выделяется форма, а затем*: **View – Object**). На Form1 и

др. формы в режиме конструирования размещаются элементы управления из окна ToolBox.

Окно Code Окно модуля формы (*Для отображения сначала в окне Проекта выделяется форма, а затем*: **View – Code**). Предназначено для просмотра, написания и редактирования программы на языке Visual Basic.

Окно Immediate Window Окно проверки (*Отображается*: **View – Immediate Window**). В данном окне удобно выводить результаты выполненных действий. Команда **Debug.Print S;** выводит в окно Immediate выражение **S**, начиная с места, где был расположен курсор и переводит курсор в конец выражения **S** (обратите внимание на точку с запятой после команды), а в случае команды **Debug.Print S** курсор переводится на следующую строку. Таким образом, варьируя с точкой запятой, можно выводить информацию в строчку или в столбик, по желанию. Например, команда **Debug.Print "2*3 = "; 2 * 3** выводит строковое выражение **"2*3 = "** и значение числового выражения **2*3** в одну строку, а курсор переводит на следующую строку



Глава 2. Наиболее важные сведения для начала программирования в среде Visual Basic 6.0

События элементов управления. Подпрограммы. Процедуры обработки событий

Программы в языках высокого уровня, как правило, строятся из отдельных фрагментов программ, называемых подпрограммами.

Подпрограммы – это отдельные фрагменты программ, которые можно вызывать на выполнение из разных частей программы. Для этой цели их оформляют специальным образом. В этом пункте это обстоятельство не затрагивается.

Подпрограммы по способу запуска их на выполнение делятся на две категории:

1. Подпрограммы, которые выполняются только после их ***вызова*** из другой подпрограммы,
2. Подпрограммы, которые выполняются по происшествию определенного ***события***.

Последние называют *процедурами обработки событий*, кратко – обработчики событий. В этом пункте речь идет о процедурах обработки событий.

Форма и каждый элемент управления формы обладают сериями событий, которые происходят при определенных действиях пользователя. С каждым событием формы или элемента управления связана процедура обработки этого события, что позволяет запустить ее на выполнение, осуществляя определенное действие.

Одно из событий элемента управления по умолчанию является его главным событием. Оно удобно тем, что *заготовку* процедуры его

обработки можно создать двойным щелчком левой кнопкой мыши по элементу управления во время конструирования формы.

Первый пример. Например, событие **Load** (загрузка) является главным событием формы. Оно происходит во время запуска программы на выполнение, следовательно, во время запуска и выполняется обработчик указанного события Load формы.

Если осуществить двойной щелчок по форме Form1 в режиме конструирования ее (*осуществите двойной щелчок по* Form1), вызывается заготовка обработчика события Load в модуле формы:

```
Private Sub Form_Load()
End Sub
```

Процедура состоит из заголовка и из ее конца. Между заголовком и концом записывается код процедуры, но в данном случае – он ***пустой***.

Ключевое слово **Private** в заголовке указывает на область видимости процедуры – она видима в модуле формы, но невидима из других модулей. В случае ключевого слова **Public** она была бы видимой и из других модулей.

Ключевое слово **Sub** в заголовке указывает на то, что речь идет собственно о процедуре (Subroutine – подпрограмма, исторически процедуру называли подпрограммой).

Form_Load() – это имя процедуры, пустые скобки означают отсутствие параметров процедуры.

Load – это событие формы, происходит при загрузке программы на выполнение.

End Sub – это конец процедуры – завершение процедуры.

Событие Load формы происходит всего один раз при запуске программы на выполнение. Оно, как правило, используется для начальной инициализации, например, для установления надписей на кнопках и т.п.

Пример кода:

```
Option Explicit

Private Sub Form_Load() ' При запуске программы на выполнение
    Form1.AutoRedraw = True 'Установка режима перерисовки
    Form1.Font.Size = 14 ' Установка размера шрифта на форме
    Form1.Print "При загрузке программы Произошло событие Load формы!!!" 'Вывод сообщения на форме
    Form1.Print "Оно вынудило выполнение процедуры Form_Load()"
End Sub
```

Второй пример. Событие **Click** является главным событием кнопки CommandButton (и некоторых других элементов управления). Если на форму установить кнопку CommandButton1, и осуществить двойной щелчок по ней, то откроется заготовка процедуры обработки события Click кнопки Command1 (часть Button при этом отсекается)

```
Private Sub Command1_Click()
End Sub
```

Если теперь проект запустить на выполнение, и осуществить щелчок левой кнопкой мыши по кнопке Command1, то произойдет событие Click на кнопке Command1, что вынудит выполнение приве-

денной процедуры. Но при этом никакой реакции не произойдет, так как код процедуры – пустой, никаких действий не выполняет. Для последней цели необходимо между заголовком и концом процедуры описать код процедуры обработки события.

Событие Click (щелчок) кнопки происходит в следующих случаях:

- когда осуществляют щелчок левой кнопкой мыши по кнопке
- при нажатии на клавишу **Enter**, когда кнопка имеет *фокус ввода*
- при нажатии на клавишу **Пробел**, когда кнопка имеет *фокус ввода*

Фокус ввода является важным понятием языков программирования. Языки программирования высокого уровня – многооконные. При этом *активным* может быть только *одно* из окон, то окно, которое имеет фокус ввода. Например, окно приобретает фокус ввода, если по нему осуществить щелчок левой кнопкой мыши, имеются и другие способы ввода фокуса.

Аналогично обстоит дело и с элементами управления формы. Активным может быть либо сама форма, либо один из его элементов управления.

- При запуске программы на выполнение фокус ввода получает тот элемент управления, который первым установлен на форму.
- Фокус можно вводить программно. Например, при выполнении команды **Command1.SetFocus** фокус ввода приобретает кнопка Command1. Аналогично можно вводить фокус ввода и

в другие элементы управления с помощью указанного свойства.

- Фокус можно вводить и щелчком по элементу левой кнопкой мыши.

Упражнения 1

Упражнение 1.

Нанесите на форму элементы управления по своему усмотрению и осуществите по ним двойные щелчки. Вы увидите, для каких элементов, какие события являются главными, и создадите и откроете заготовки процедур обработки соответствующих событий (пустые).

Упражнение 2.

Нанесите на форму, например, элементы Label1, Text1, Command1 или др.

1. Откройте Модуль формы (View Code). Нажав на черный треугольник в левой части вверху окна модуля, откройте ниспадающий список элементов. Щелкнув на одном из них, активизируйте его.
2. Нажав на правый черный треугольник вверху окна модуля, откройте ниспадающий список событий выделенного элемента. Вы увидите события выделенного элемента.
3. Щелкнув на имени одного из событий, создадите заготовку процедуры обработки соответствующего события.
4. Вы научитесь создавать заготовки процедур обработки не только главных событий элементов управления, но и других

их событий. Однако для работы с такими процедурами необходимо знание того, когда происходят указанные события.

Переменные и типы данных

Понятие переменной и типа данных являются основными понятиями при программировании.

Переменная *– это область памяти компьютера, наделенная* ***идентификатором****. В переменную можно вводить данные, из переменной можно выводить данные, можно изменять значения переменной во время выполнения программы*.

Прежде первого использования переменных их необходимо описать. Для этой цели в языках программирования имеется богатый набор типов данных.

Типы данных в языке Visual Basic 6.0 для первого знакомства следующие:

String – строковый тип.

Byte, Integer, Long – целые типы, отличаются диапазоном значений.

Single, Double – числа с плавающей точкой (запятой) одинарной и двойной точности.

Boolean – логический тип.

Описание переменных: Сначала записывается ключевое слово Dim, затем идентификатор (имя) переменной, затем ключевое слово As, и затем тип переменной:

Dim *VariableName* [As *Type*]

Примечание. Язык Visual Basic является, так сказать, «Дружественным языком», т.е. позволяет некоторые отступления от общепринятых норм языков программирования. В частности позволяет не описывать переменные. Однако, описание переменных весьма желательно по ряду причин. Имеется опция обязательного объявления переменных **Option Explicit**. Рекомендуется всегда записывать эту опцию в начале модуля.

Пример

```
Option Explicit
Dim a As String
Dim b As Integer
```

Приведена опция обязательного объявления переменных, и описаны две глобальные переменные – переменная *a* строкового типа и переменная *b* целого типа.

При описании переменной в области памяти компьютера выделяется память под переменную и наделяется идентификатором переменной. Переменная строкового типа инициализируется при этом *пустой строкой* – **""**, а числового типа – *нулем* – **0**.

Отделение одного оператора от другого. В каждом языке программирования имеются свои правила отделения одного оператора от другого. В языке Basic при написании операторов в одну строчку они отделяются друг от друга двоеточием (**:**), а для написания операторов в столбик приходится нажимать на клавишу Enter, что попутно отделяет один оператор от другого.

В предыдущем примере 3 оператора были отделены друг от друга нажатием на клавишу Enter – для перевода строки требуется нажимать на клавишу Enter.

Логическая строка. Если в окне редактора кода Visual Basic 6.0 набирать текст (не более 1023 символов), нажимая на различные клавиши, но не на клавишу Enter, получим одну *логическую строку*. Если теперь нажать на клавишу Enter, курсор перейдет на новую логическую строку. Операторы, набранные в различных логических строках, считаются отделенными друга в языке Basic. Операторы, набранные в одной логической строке, отделяются друг от друга двоеточием (:).

Время жизни переменных.

Каждая переменная характеризуется своим временем жизни – периодом времени, в течение которого она существует в памяти.

- Одни переменные существуют недолго
- Другие переменные неоднократно создаются и уничтожаются
- Третьи существуют на протяжении всего времени выполнения программы

Переменная, описанная внутри процедуры, существует, пока выполняется процедура. По завершении выполнения процедуры уничтожается. При новом обращении к процедуре создается вновь, по завершении выполнения – вновь уничтожается. Такая переменная называется *локальной*.

Переменная, описанная вне всяких процедур, существует на протяжении всего времени выполнения программы. Такая пере-

менная называется глобальной. Глобальная переменная в языке Visual Basic 6.0 описывается перед всеми подпрограммами после опции обязательного объявления переменных.

Если переменная вводится и используется только в одной подпрограмме, то желательно ее описывать локально в этой подпрограмме.

Если переменная используется в нескольких подпрограммах, то ее необходимо описать глобально.

Ввод данных. Функции ввода и преобразования

Ввод данных подразумевает ввод данных в переменную, то есть ввод данных в область памяти компьютера, наделенной идентификатором. Ввод осуществляется разными средствами. Однако при первом знакомстве с Visual Basic будем иметь дело с функцией ввода InputBox (поле ввода).

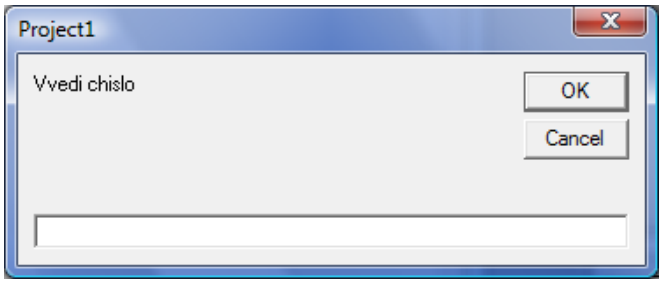
InputBox – функция ввода данных

- Функция отображает подсказку в диалоговом окне
- Ждет пользователя, чтобы ввести текст или щелкнуть кнопкой
- Возвращает строку типа String, содержащую содержание текстового поля

Функция имеет один обязательный и несколько необязательных параметров. Ознакомимся здесь только с обязательным параметром. Синтаксис:

Variable = **InputBox**(**Prompt**)

где **Variable** – переменная, **Prompt** – строка-подсказка, например: **Prompt** = “**Vvedi chislo**”. Если выполнить приведенную команду, то появится окно диалоговой функции



Если теперь в поле ввода ввести число и щелкнуть левой кнопкой мыши по кнопке OK, или просто нажать Enter, то это число введется в переменную Variable.

Задача. *Описать процедуру обработки события* Click *кнопки* Command1*, в которой в глобальную переменную a типа* Integer *вводится целое число*.

Решение:

```
Option Explicit
Dim a As Integer
Private Sub Command1_Click()
a = InputBox("Введи число")
End Sub
```

Примечание. Как выше сказано, функция **InputBox** возвращает значение ***строкового*** типа. Но в процедуре оно присваивается переменной ***числового*** типа. Это не совпадение типов.

Однако в данном случае компилятор автоматически преобразовывает представление числа из строкового типа в числовой тип.

Это можно было б сделать явно и с помощью функций преобразования **CInt**:

```
a = CInt(InputBox("Введи число"))
```

а лучше с помощью функции преобразования типов **Val**:

a = Val(InputBox("Введи число"))

Функции округления и преобразования в языке Visual Basic

Round(x)	Округляет число x до ближайшего целого.
Round(x, m)	Округляет число x до m знаков после десятичной запятой.
Int(x)	Округляет число x вниз до ближайшего целого.
Val(x)	Округляет x по недостатку до целого и преобразует его к целому типу Integer.
	Преобразовывает строковый тип в числовой тип
CInt(x)	Округляет x до ближайшего целого и преобразует его к целому типу Integer.
CSng(x)	Преобразует x к типу одинарной точности Single
CDbl(x)	Преобразует x к типу двойной точности Double
Str(x)	Преобразует x к строковому типу String
Val(s)	Преобразует строку ***s*** в числовой тип

Вывод данных

Под выводом здесь подразумевается вывод данных на какое-нибудь окно. Вывод осуществляется разными способами. Здесь остановимся на более простых методах вывода.

Первый способ – использование диалоговой функции **MsgBox.**

MsgBox – диалоговая функция вывода

- отображает сообщение в диалоговом окне
- ждет пользователя, чтобы щелкнуть кнопкой
- и возвращает Целое число, указывающее, какой кнопкой пользователь щелкал.

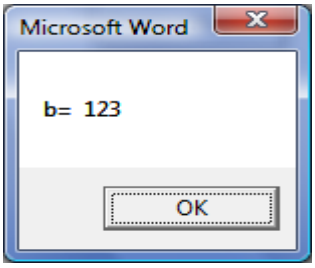
Имеет несколько вариантов синтаксиса. Один из них следующий:

MsgBox s

где *s* – строковое выражение.

Выражение может быть и чисто числовым. В этом случае компилятор автоматически преобразовывает числовое выражение в строковое. Однако, если выражение смешанное, числовую часть его необходимо преобразовать в строковую с помощью функции преобразования Str. Например,

- Если переменная ***a*** описана как строковая, и выполнена команда **a = "Строка",** то синтаксис **MsgBox a** допустим.
- Если переменная ***b*** описана как числовая, и выполнена команда **b = 123**, то синтаксис **MsgBox b** также допустим.
- Однако синтаксис **MsgBox " b = " + b** не допустим – строка складывается с числом.
- Допустим синтаксис **MsgBox " b = " + Str(b).** Выполнение команды приведет к вызову диалогового окна



Второй способ – вывод на **форму**.

Команда **Print S** выводит выражение **S** на форму, начиная с места позиции курсора, и переводит курсор на следующую строку. Так что при повторном выводе выведенное выражение окажется на следующей строке.

Для вывода выражений в строку после имени выражения ставят точку с запятой: **Print S;**

Команда **Print S1; S2** выводит написанные выражения в одну строку одно за другим. Можно выводит и большее число выражений.

Третий способ – вывод в окно отладки **Immediate Window**.

Окно вызывается: View (Вид) и Immediate Window.

Команда

Debug.Print S

выводит выражение S в окно отладки **Immediate Window**.

Четвертый способ. Данные можно выводить и на некоторые элементы управления. В этом пункте на них не останавливаемся.

Задача. *В одной процедуре обработки события ввести два целых числа. В другой процедуре сложить эти числа и результат сложения вывести на экран*.

Решение. Нанесите на форму кнопки Command1 и Command2. Перекопируйте или перепишите код нижеприведенной программы. Программа сложения готова.

Кнопку Command1 и его событие Click используем для ввода в переменные. Их необходимо описать глобально для сохранения их значений при использовании их в другой процедуре.

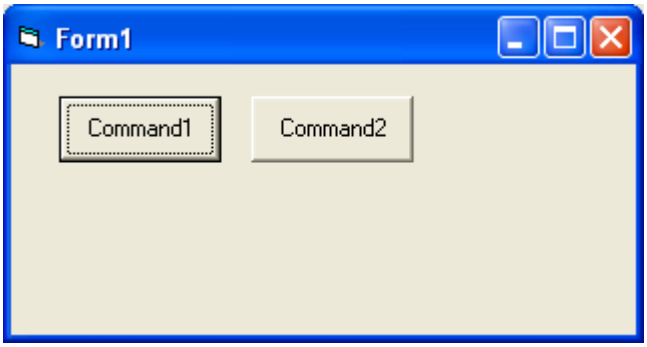
Кнопку Command2 и его событие Click используем для суммирования чисел и их вывода. Переменную для суммы можно описать локально, так как она в другой процедуре не используется.

Код программы:

```
Option Explicit
Dim a As Integer 'Opisanie globalnoy peremennoy celogo tipa Integer
Dim b As Integer
```

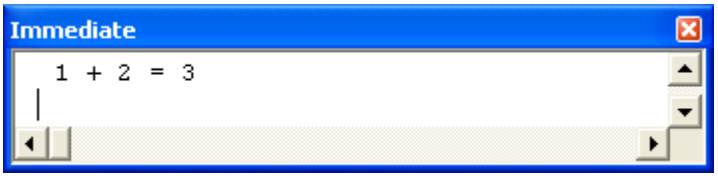
```
Private Sub Command1_Click() 'Pri tselchke po knopke
a = Val(InputBox("Введи число")) 'Vvod v peremennuyu
b = Val(InputBox("Введи число"))
End Sub

Private Sub Command2_Click() 'Pri tselchke po knopke
Dim S As Integer 'Opisanie lokalnoy peremennoy celogo tipa Integer
S = a + b 'Prisvoenie summy peremennyh tretey peremennoy
Debug.Print a; "+"; b; "="; S 'Vyvod na Immediate Window
End Sub
```

Запустив программу на выполнение, получим картину



Фокус ввода имеет кнопка Command1 – кнопка выделена прямоугольником. Поэтому, чтобы выполнить первую процедуру, достаточно нажать на Enter.

Нажав на Enter, введем числа, например, 1 и 2. Чтобы их сложить, достаточно щелкнуть по кнопке Command2 (нажатие на Enter снова приведет к выполнению первой процедуры, так как фокус ввода остался на первой кнопке). Получим картину

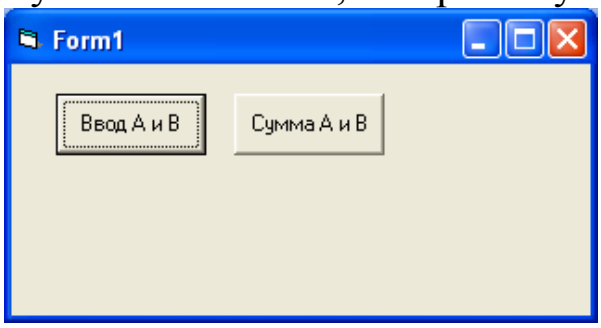


Написанную программу можно улучшать. Например, можно нанести надписи на кнопки, указывающие, какие действия они выполняют: «Ввод A и B» и «Сумма A и B». Надписи можно нанести в

режиме конструирования или программно. Для программного ввода надписей используется событие Load формы. Дополните код программы процедурой:

```
Private Sub Form_Load() 'Pri zapuske
Command1.Caption = "Ввод A и B"   'Vyvod nadpisey na knopki
Command2.Caption = "Сумма A и B"
End Sub
```

Запустив программу на выполнение, теперь получим картину



На кнопках появились соответствующие надписи.

Можно осуществлять и дальнейшие улучшения кода программы. Например, вводить фокус ввода на кнопки по желанию пользователя.

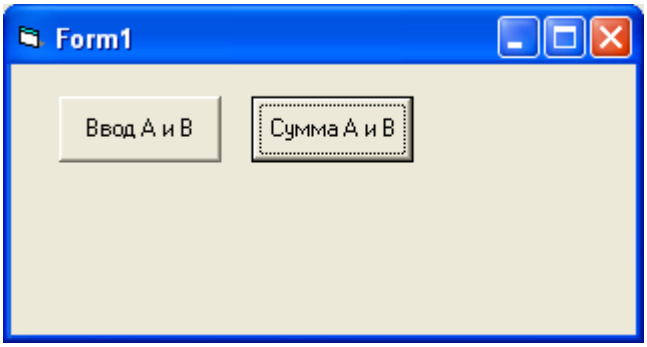
Дополните код кнопки Command1 командой

Command2.SetFocus

а код кнопки Command2 командой

Command1.SetFocus

Запустив программу на выполнение, введем, например, числа 2 и 3. Затем получим картину



Теперь фокуса ввода оказался на кнопке суммирования. И для нахождения суммы достаточно нажать на Enter – находится сумма, и фокус ввода переходит на кнопку ввода. И можно начать новую серию вычислений.

Упражнения 2

1. Опишите переменные в коде программы, используя разные типы целых чисел, например, Byte, Integer, Long. Вводя и выводя малые и большие числа, можно получить определенное представление о диапазонах значений указанных типов.
2. Опишите процедуры и других операций над целыми числами на свое усмотрение.
3. Опишите процедуры действий над числами с плавающей запятой. Помните, при вводе чисел применяется именно десятичная запятая, а не точка, а при записи вычислений - десятичная точка.

Глава 3. Подпрограммы

При решении задачи бывает удобным выделять отдельные ее части в отдельные фрагменты программ, называя их *подпрограммами*. Подпрограммы удобны тем, что их можно вызывать на выполнение неоднократно из разных частей программы. Для этой цели их описывают специальным образом. В каждом языке программирования свои способы их описания.

Выше было отмечено, что подпрограммы по способу их выполнения делятся на два типа: на подпрограммы, выполняемые по

происшествию определенного события (обработчики событий), и подпрограммы, которые непосредственно не запускаются на выполнение – выполняются только после их вызова из другой подпрограммы. Кстати, и обработчики событий также могут быть вызваны на выполнение из другой подпрограммы, если возникнет такая необходимость.

В данном пункте речь идет о подпрограммах второго типа, то есть о подпрограммах, которые запускаются на выполнение после их вызова из другой подпрограммы. Они по способу возвращения или не возвращения значения также делятся на два типа

1. подпрограммы, которые не возвращают значения
2. подпрограммы, которые возвращают значения.

Первые называют *процедурами*, вторые – *функциями* (не во всех языках).

Главные характеристики подпрограммы – это **имя** ее и **параметры**. По ним осуществляется вызов подпрограммы. При вызове подпрограммы в ней осуществляются какие-то определенные действия и обработка ее параметров. Они и используются в основной программе. Встречаются подпрограммы, у которых отсутствуют параметры. В этом случае используются только те действия, которые осуществляются в подпрограмме.

Мы описали подпрограммы кратко в общих чертах. Далее пойдет речь о подпрограммах касательно языка Visual Basic 6.0.

Подпрограммы, которые не возвращают значения, называют **Sub**-подпрограммы, или просто *процедуры*. Примерами Sub-

подпрограмм являются процедуры обработки событий, рассмотрение ранее.

Подпрограммы, которые возвращают значения, называют **Function**-подпрограммы, или просто *функции*. Процедуры обработки событий не возвращают значения, следовательно, и не являются функциями.

Описание процедур

Каждая подпрограмма имеет свой заголовок и свой конец.

End Sub – это конец процедуры, а

End Function – конец функции.

Между заголовком и концом идет описание кода подпрограммы – ***тело*** подпрограммы. В теле процедуры записываются те действия, которые необходимо осуществить в решаемой задаче.

Описание заголовка процедуры. Сначала записывается ключевое слово **Sub**. Затем записывается **имя** (идентификатор) процедуры, затем в круглых скобках идет описание **параметров.**

Понятия *параметр* и *переменная* здесь употребляются как синонимы. Таким образом, параметры процедуры – это области памяти, наделенные идентификаторами.

Примечание. Если в модуле набрать слово Sub, а затем идентификатор подпрограммы, и нажать на клавишу Enter, компилятор автоматически создаст заготовку процедуры с пустыми круглыми скобками. Затем можно будет отредактировать ее.

Параметры процедуры. Допустим, при решении определенной задачи задаются какие-то *данные*. По этим данным требуется

найти какие-то *величины*. Вот эти данные и величины и будут служить параметрами процедуры.

Формальные и фактические параметры процедуры. Вызов процедуры. Для вызова процедуры записывают имя процедуры и список параметров ее. Но параметры, записываемые при вызове процедуры, и параметры, перечисленные в заголовке процедуры при ее описании – это различные переменные в том смысле, что они занимают разные области памяти. Для их различения в речи параметры, перечисленные в заголовке процедуры, называют *формальными*, а параметры, перечисленные при вызове процедуры – *фактическими*. А как их обозначать – дело вкуса пользователя. Можно их обозначать даже одинаковыми идентификаторами – они все равно разные.

Обмен между параметрами при вызове процедуры. Передача по ссылке и по значению. При вызове процедуры происходит обмен между фактическими и формальными параметрами.

Обмен происходит не по именам, а номерам параметров.

1. Значение первого фактического параметра передается первому формальному параметру,
2. значение второго – второму,
3. и так далее

Затем происходит изменения формальных параметров в процедуре. Но эти изменения не всегда отражаются на фактических параметрах. Это зависит от способа передачи параметров.

1. Изменения формальных параметров, переданных *по значению, **не отражаются*** на фактических параметрах.

2. Изменения формальных параметров, переданных *по ссылке*, ***вызывают*** соответствующие изменения и фактических параметров.

Ключевое слово

1. **ByVal** используется при передаче по значению (Value значение),

2. **ByRef** – по ссылке (Reference – ссылка).

В Visual Basic 6.0 по умолчанию передача осуществляется по ссылке.

Сказанное разъясним на примерах.

Задача. *По двум сторонам прямоугольника найти его площадь*.

Решение. На примере решения этой задачи желаем продемонстрировать суть процедуры и ее параметров. Для этой цели алгоритм нахождения площади прямоугольника описываем в виде формальной процедуры.

Первый вопрос, на что требуется ответить – это имя процедуры. Какое дать имя – это дело вкуса пользователя. Однако рекомендуется давать осмысленное имя, по которому можно было узнать, что делает процедура. Наделяем именем RectangleArea (Rectangle – прямоугольник, Area – площадь).

Второй вопрос, на что требуется ответить – это параметры процедуры. Чтобы ответить на этот вопрос, задаемся вопросом, что дано, что требуется найти? Даны ***стороны*** прямоугольника, требуется найти его ***площадь*** – это и будут параметрами – две стороны и площадь.

Третий вопрос, как обозначить параметры? Это дело вкуса пользователя. Однако, чтобы показать отличие формальных параметров от фактических их обозначаем разными буквами: x, y, z – формальные параметры, a, b, s – фактические, первые два – это стороны, третьи – площадь.

Четвертый вопрос, в каком порядке располагать параметры? Это дело вкуса пользователя. Однако необходимо знать и помнить, какой параметр по счету, что означает. Мы выбираем: первый параметр – это площадь, вторые два – это стороны.

Пятый вопрос, как передавать параметры? По значению или ссылке? Стороны задаются в основной программе, они не изменяются, их можно передавать по значению. Площадь вычисляется в формальной подпрограмме, это значение используется в основной программе – ее необходимо передавать по ссылке.

Шестой вопрос, каков алгоритм нахождения площади прямоугольника? Ответ из математики: Произведение сторон прямоугольника равно его площади.

После ответов на эти вопросы формальная процедура нахождения площади прямоугольника по двум сторонам выглядит следующим образом:

```
Sub RectangleArea(ByRef z As Long, ByVal x As Long, ByVal y As Long) ' Выполняется после вызова из другой программы
   z = x * y 'Вычисляет площадь прямоугольника по сторона x и y. присваивает переменной z
End Sub
```

Указано имя процедуры, в заголовке перечислены параметры процедуры с указанием их порядка и способов передачи параметров.

Наконец, в теле процедуры описан алгоритм нахождения площади прямоугольника – Произведение вторых двух параметров присваивается первому параметру. По этому признаку и определяется, какому параметру присваивается площадь, а каким параметрам – стороны прямоугольника.

Если б записать в теле процедуры, например, `x = z * y`, то второй параметр означал бы площадь, а первый и третий – стороны.

Вызов процедуры. В данном примере процедура вызывается из процедуры обработки события (см. ниже). Для этой цели записано имя процедуры и перечислены фактические параметры без заключения их в скобки:

RectangleArea s, a, b – Это один из способов вызова процедуры. Другой способ вызова процедуры позволяет параметры заключать в скобки. Но при этом имя процедуры наделяется служебным словом **Call**:

Call RectangleArea (s, a, b) – Это другой способ вызова процедуры.

Процедура вызова подпрограммы:

```
Private Sub Command1_Click()
   Dim s As Long 'Описание переменных целого типа Long
   Dim a As Long
   Dim b As Long
   a = 3          ' Присвоение переменным значений
   b = 5
```

RectangleArea s, a, b ' **Вызов подпрограммы с указанием параметров**
Debug.Print "a ="; a; "b ="; b; "s ="; s ' **Вывод параметров на Immediate Window**
End Sub

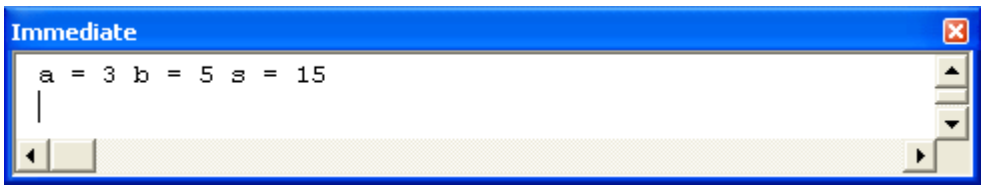
По щелчку по кнопке Command1 начинает выполняться процедура обработки. Сначала выполняются первые 5 операторов. При этом стороны получают значения a = 3, b = 5. Заметим, что значение s = 0 – это значение получено при описании переменной Dim s As Long.

Затем компилятор, встретив команду RectangleArea s, a, b, находит формальную подпрограмму вычисления площади прямоугольника и начинает ее выполнять. При этом значения s, a, b передает соответственно параметрам z, x, y, т.е. они становятся равными соответственно 0, 3, 5. Выполняется формальная процедура. При этом значения x и y не изменяются, изменяется z, становится равным 15. Так как параметр z описан по ссылке, то изменение параметра z вызывает соответствующее изменение фактического параметра s, т.е. и s становятся равным 15.

Завершив выполнение формальной процедуры, управление передается оператору, следующему за оператором вызова RectangleArea s, a, b. Это оператор

Debug.Print "a ="; a; "b ="; b; "s ="; s.

Он выводит значение фактических параметров в окно Immediate Window.



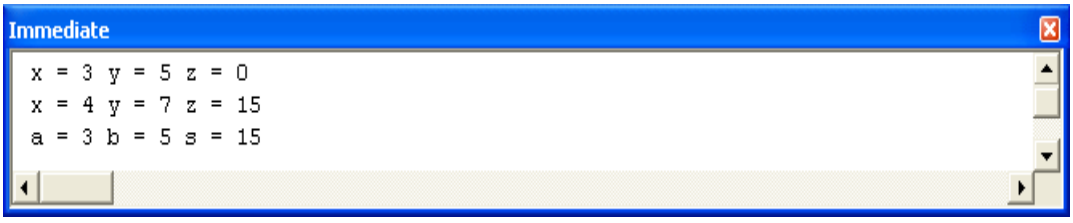
Заметьте, параметры при выводе записаны в ином порядке – сначала стороны, затем площадь. Они равны соответственно 3, 5, 15. То есть полученные значения фактических параметров можно использовать по своему усмотрению.

При желании можно использовать и значения формальных параметров. Для этой цели необходимые действия нужно записать в теле формальной процедуры.

Например,

```
Sub RectangleArea(ByRef z As Long, ByVal x As Long, ByVal y As Long)
Debug.Print "x ="; x; "y ="; y; "z ="; z; "Это значения фактических параметров, переданных формальным"
z = x * y                              'Изменения формальных параметров
x = 4: y = 7
Debug.Print "x ="; x; "y ="; y; "z ="; z; "Это значения измененных формальных параметров"
End Sub
```

Оператор Debug.Print "x ="; x; "y ="; y; "z ="; z сначала выведет значения формальных параметров, полученных от передачи им фактических параметров, а затем выведет изменения, осуществленные в самой формальной процедуре.



Обратите внимание, изменения формальных параметров *x*, *y*, переданных по значению не отразились на значениях фактических

параметров *a, b*, а значение параметра *z*, переданного по ссылке, отразилось на значении *s*.

Описание функции

Подпрограмма, возвращающее значение, называется *функцией*. Описание заголовка функции начинается со служебного слова **Function**, после которого идет *имя*, в круглых скобках *параметры*, заканчивается описанием типа возвращающего значения.

В теле функции должно быть присвоение имени функции значению выражения, возвращающему функцией. Иначе возвращаемое значение нельзя будет использовать в вызываемой подпрограмме. Более точно, не будет возвращаемого значения.

Описание функции завершается **End Function**.

Задачи, решаемые с помощью функций, можно решать и с помощью процедур, и, наоборот.

Однако одни задачи удобней решать с помощью процедур, другие – с помощью функций.

Задачу нахождения площади прямоугольника можно оформить и в виде ***функции***:

```
Option Explicit
Function RectangleArea(ByVal x As Long, ByVal y As Long) As
Long 'Выполняется после вызова
   RectangleArea = x * y            ' Значение площади присваивается
имени функции
End Function

Private Sub Command1_Click()
   Dim s As Long                    'Описание переменных целого
типа Long
   Dim a As Long
   Dim b As Long
   a = 3                  'Присвоение значений переменным
```

```
    b = 5
    s = RectangleArea(a, b)          'Вызов функции и Присвоение возвращаемого значения переменной
    Debug.Print "a ="; a; "b ="; b; "s ="; s      'Вывод значений параметров на Immediate Window
End Sub
```

Обратите внимание на отличительные особенности при использовании процедуры и функции.

Второй пример.

Задача. *По двум сторонам прямоугольника найти его площадь и периметр*. Решить задачу, используя подпрограмму-процедуру и подпрограмму-функцию.

Решение. *Замечание*. При решении задачи с использованием процедуры можно обойтись одной подпрограммой, у которой 4 параметра: 2 стороны, площадь, периметр.

При решении задачи с использованием функций нельзя обойтись одной функцией, так как в языке Visual Basic 6.0 не предусмотрены функции, возвращающие два значения (векторные функции) – предусмотрены функции, возвращающие только одно значения (скалярные функции).

Необходимо описать две функции: одна находит площадь, другая – периметр. Параметры у той и у другой функции одни и те же – стороны прямоугольника. Различаются возвращаемыми значениями. Необходимо также организовать два вызова: один вызов для площади, другой – для периметра.

Теперь сам читатель может определить, какой способ более удобен – использование процедур, или использование функций.

Упражнения 1

В следующих заданиях решения оформить в виде вызываемых и вызывающих подпрограмм.

Задание 1. Найти площадь треугольника по стороне и высоте к этой стороне.

Задание 2. Найти площадь треугольника по двум сторонам и углу между ними.

Указание. Угол может быть задан в радианах и градусах. В последнем случае возникает необходимость преобразования угловых градусов в радианы, для чего используется константа *pi* (пи) - отношение длины окружности к диаметру. Однако в Visual Basic 6.0 такая константа не предусмотрена. Ее можно описать как функцию без параметров

```
Function pi()
pi = 4 * Atn(1)
End Function
```

Задание 3. Вычислить площадь треугольника по трем сторонам.

Указание. Площадь треугольника вычисляется по формуле Герона. При этом используется полупериметр треугольника, обозначаемый через *p* и выражение *d* = p(p – a)(p – b)(p – c). Однако следует помнить, что переменные *p, d* не являются параметрами подпрограммы, они являются вспомогательными локальными переменными подпрограммы для удобства выкладок. При желании вовсе можно обойтись без этих вспомогательных переменных, однако выкладки усложнятся. Если бы, кроме площади треугольника,

требовался бы найти и полупериметр, в этом случае и полупериметр был бы параметром подпрограммы.

Задание 4. Найти площадь и периметр круга по его радиусу.

Указание. См. указание к заданию 2.

Задание 5. Решить линейное скалярное уравнение в зависимости от его коэффициентов.

Указание. Коэффициенты уравнения являются параметрами подпрограммы.

Задание 6. Решить приведенное квадратное уравнение.

Указание. Квадратное уравнение в зависимости от коэффициентов может иметь решение, а может и не иметь решение. Это зависит от знака дискриминанта. Знак дискриминанта определяется с помощью условного оператора If.

Упражнения 2

Рекомендуется решения оформлять в виде вызываемых и вызывающих подпрограмм

1. Даны два действительных положительных числа. Найти их среднее арифметическое и среднее геометрическое.
2. Найти сумму арифметической прогрессии по начальному члену, разности и числу членов.
3. Найти сумму геометрической прогрессии по начальному члену, разности и числу членов.
4. На плоскости даны две точки своими координатами. Найти расстояние между ними.

5. Треугольник на плоскости задан координатами своих вершин. Найти полупериметр и площадь треугольника.

Глава 4. Модули

В Visual Basic 6.0 вся программа строится из модулей. *Модуль* – это набор описаний и процедур на языке Visual Basic 6.0, собранных в одну программную единицу. Таким образом, все программы в Visual Bacic 6.0 содержатся в модулях. Существуют:

- Модули форм
- Стандартные модули
- Модули класса
- Модули форм

Модуль формы связан с определенной формой. В процессе создания формы мы создаем и модуль формы. Модули форм обычно содержат процедуры обработки событий, включающих ***вызовы*** процедур, добавленных в стандартные модули.

Для добавления объекта к проекту достаточно выполнить команды:

Project и **Add** с **именем** объекта (например, **Project** – **Add Form**).

Для удаления объекта – выделить удаляемый объект в окне Проектов (**Project Explorer**), и затем **Project** и **Remove** с **именем** удаляемого объекта.

С созданием формы автоматически создается и модуль формы, с удалением формы удаляется и модуль формы.

В небольших проектах, как правило, используется одна форма. Можно использовать и несколько форм. По умолчанию Form1 – стартовая форма. Можно сделать стартовой и другую форму:

Project – **Proect1 Properties**

и в открывшемся окне выделить стартовую форму.

Стандартные модули

Стандартным называется модуль, не связанный ни с одним объектом (например, формой). Обычно в стандартных модулях содержатся часто используемые функции. Они могут быть ***вызваны*** для выполнения из модулей форм.

Для добавления стандартного модуля выполнить команды **Project** – **Add Module**.

Модули класса, не связанный от форм, в этом пункте не затрагивается.

Упражнения 1

Упражнения. Добавьте в проект стандартный модуль. Решение заданий оформите в виде вызываемых и вызывающих подпрограмм. Вызываемые подпрограммы установите в стандартном модуле, вызывающие – модуле форм.

1. Даны катеты прямоугольного треугольника. Найти его гипотенузу и площадь.
2. Дана сторона правильного треугольника. Найти площадь треугольника.
3. Даны два действительных числа. Найти их максимум и минимум.

4. Даны три действительных числа. Найти их максимум и минимум.

Указание. Для решения двух последних заданий используется условный оператор If.

5. Опишите в стандартном модуле функцию *pi* без параметров – отношение длины окружности к диаметру. Найдите значения тригонометрических функции аргументов, заданных в градусной мере.

Указание. Аргументы стандартных тригонометрических функций задаются в радианах. Поэтому возникает необходимость преобразования градусной меры в радианную меру, если аргументы задаются в градусах. Напомним описание функции *pi*:

```
Function pi()
    pi = 4 * Atn(1)
End Function
```

6. Проверьте экспериментально справедливость равенства $\cos^2 x + \sin^2 x = 1$ и других тригонометрических тождеств.

Глава 5. Переменные и типы переменных

Переменной *называется имя, определяющее область памяти для хранения величины, которая может изменяться во время работы программы.*

В каждом языке программирования определены правила, определяющие, какие имена (идентификаторы) можно давать переменным, процедурам, константам (см. справочники).

Объявление переменных

Перед использованием переменной в программе ее необходимо сначала объявить, т.е. задать ее тип и область видимости (область использования переменной). Синтаксис объявления переменной имеет вид:

{Dim | Public | Private | Static} VariableName [As Type]

Т.е. сначала записывается одно из ключевых слов, что записаны внутри фигурных скобок, затем имя переменной, а затем тип переменной. Если тип переменной не указывается, то она принимает тип **Variant**.

Область видимости переменной

При объявлении переменной определяется ее область видимости (Scope), то есть область проекта, где она доступна. Область видимости переменной задается с помощью одного из четырех ключевых слов:

- Dim
- Public
- Private
- Static

Объявление переменной на уровне процедуры

Для объявления переменной на уровне процедуры используется ключевое слово Dim. Переменная, объявленная на уровне процедуры, называется *локальной*. Такая переменная доступна только в той процедуре, где она объявлена. Ниже представлен пример объявления локальных переменных:

```
Option Explicit

Private Sub Command1_Click()
Dim intVar1 As Integer 'Описаниелокальных переменных целого типа Integer
Dim intVar2 As Integer
Dim intVar3 As Integer
Dim strVar As String        ' Описание локальной переменной строкового типа String
intVar1 = 3                 'Присвоение значений переменным
intVar2 = 5
intVar3 = intVar1 + intVar2 'Присвоение суммы значений переменных третьей переменной
strVar = "Summa = " + Str(intVar3) 'Присвоение конкатенации переменных строковой переменной
MsgBox strVar          ' Вывод в окно MsgBox
End Sub
```

Внутри данной процедуры обработки события Click кнопки CommandButton1 (между заголовком Private Sub Command1_Click() и ее концом End Sub) объявлены три числовых переменных целого типа Integer и одна строковая переменная типа String. Первым двум переменным присвоены значения, равные 3 и 5, а третьей переменной значения сумм двух первых переменных. Четвертой переменной присвоено строковое выражение, которое выводится в окно диалоговой функции MsgBox.

Значения данных переменных сохраняются, пока выполняется процедура, по завершении ее выполнения уничтожаются. Поэтому

значения этих переменных нельзя использовать в другой подпрограмме.

Объявление переменной на уровне модуля

Для объявления переменной на уровне модуля используются ключевые слова Public и Private.

Переменная, объявленная на уровне модуля с ключевым словом Private, доступна ко всем процедурам этого модуля, но не доступна процедурам других модулей.

В случае использования ключевого слова Public переменная доступна во всех модулях проекта.

Пример.На уровне стандартного модуля Module1(Code) объявим константу

```
Option Explicit
Const nn = 123 ‘ Описание именованной константы на уровне стандартного модуля
```

На уровне модуля формы Form1(Code) поместим подпрограмму

```
Option Explicit

Sub Command1_Click()
Debug.Print nn ‘ Вывод константы на Immediate Window
End Sub
```

Запустим программу на выполнение. При щелчке на кнопку Command1 – появится сообщение:

```
---------------------------
Microsoft Visual Basic
Compile error: Ошибка компиляции
Variable not defined Переменная не определена и выделение переменной
ОК   Справка
```

Это означает, что константа из модуля Module1(Code) не видна в модуле формы Form1(Code).

Теперь команду **Const** nn = 123 исправим на **Public Const** nn = 123. Выполнение программы завершилось благополучно. Это значит, что теперь константа из модуля Module1(Code) стала видимой в модуле формы Form1(Code).

Упражнения 1

Задание. Добавить в проект стандартный модуль Module1. Вверху модулей описать опцию

Option Explicit ‘Опция обязательного объявления переменных

обязательного объявления переменных.

Задание 1. На стандартном модуле описать некоторые переменные – сначала со служебным словом Private, а затем со служебным словом Public. На модуле формы выполнить какие-нибудь действия с описанными переменными. Пронаблюдать, когда эти переменные видимы в модуле формы, а когда нет.

Задание 2. На стандартном модуле описать подпрограммы, выполняющие какие-нибудь действия – сначала со служебным словом Private, а затем со служебным словом Public, а затем вовсе без этих слов. Описанные подпрограммы вызвать на выполнение из модуля формы. Пронаблюдать, когда эти подпрограммы видимы в модуле формы, а когда нет.

Задание 3. В модуле формы опишите две процедуры обработки события:

Option Explicit

```
Private Sub Command1_Click()
Dim a As Integer
a = 123
Debug.Print a
End Sub

Private Sub Command2_Click()
Debug.Print a
End Sub
```

Выполните процедуры. Объясните результаты явлений.

Отключите опцию обязательного объявления переменных (Наделив ее апострофом ', превратите в комментарий). Снова выполните процедуры. Объясните результаты явлений.

Время жизни переменных. Статические переменные

Под *временем жизни переменной* понимается время, в течение которого переменная может иметь значение.

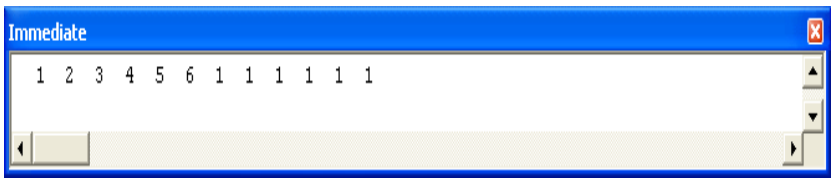
1. *Локальные* переменные имеют значение только во время выполнения процедуры, в которой они объявлены. Между вызовами процедуры такие переменные *не сохраняют* значения.

2. Использование перед именем переменной ключевого слова Static позволяет сохранять ее значение между вызовами процедуры, в которой эта переменная объявлена. Объявленные с помощью нее переменные называются статическими. Статические локальные переменные имеют значение в течение всего времени выполнения программы. Однако значения статических переменных нельзя использовать в других подпрограммах.

Ниже приведенные процедуры модуля формы демонстрируют статические и нестатические локальные переменные

```
Option Explicit
Sub Command1_Click()
Static n As Integer   ‘ Статическое описание локальной переменной
n = n + 1
Debug.Print n;        ‘ Вывод значения переменной в строчку
End Sub

Sub Command2_Click()
Dim n As Integer     ‘ Обычное описание локальной переменной
n = n + 1
Debug.Print n; ‘ Вывод значения переменной в строчку
End Sub
```

Если программу запустить на выполнение и кнопку Command1 щелкнуть, например, 6 раз, то в окне Immediate Window последовательно появятся числа 1 2 3 4 5 6. Щелчки же по кнопке Command2 дают числа 1 1 1 1 1 1.



Объяснение результатов предоставляется читателю.

Типы данных

Типы данных являются одним из важнейших элементов языков программирования. Visual Basic 6.0 поддерживает несколько типов данных, которые можно использовать для объявления типов данных или создания массивов.

При объявлении переменных тип данных указывается после ключевого слова **As**.

Каждый тип данных характеризуется объемом занимаемой ими *памяти* и *диапазоном* изменения значений (подобности см. справочники).

Основными типами данных являются:

Byte (байт) – 1 byte, от 0 до 255.

Boolean (логический) – 2 byte, True или False.

Integer (целый) – 2 byte, от -32768 до 32767.

Long (длинное целое) – 4 byte, прибл. от -2 млр до 2 млр. (- 2^31 … 2^31 – 1)

Single (с плавающей точкой одинарной точности) – 4 byte, до 7 знаков точности.

Double (с плавающей точкой двойной точности) – 8 byte, до 15 знаков точности.

String (строка переменной длины) – 10 byte + длина строки, от 0 до приблизительно 2 миллиардов.

И др. типы.

Задача. *Описать вычисление площади прямоугольника по двум сторонам, используя стандартный модуль*.

Решение. В проект добавим стандартный модуль (Project – Add Module).

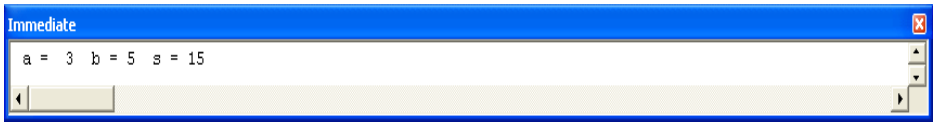
В окне стандартного модуля **Module1**(Code) поместим вызываемую подпрограмму, вычисляющую площадь прямоугольника:

```
Option Explicit
Public Sub RectangleArea(ByRef z As Long, ByVal x As Long, ByVal y As Long)
   z = x * y
End Sub
```

А в окне редактора кода формы **Form1**(Code) поместим процедуру обработки события, вызывающую подпрограмму вычисления площади:

```
Option Explicit
Private Sub Command1_Click()
    Dim s As Long
    Dim a As Long
    Dim b As Long
    a = 3
    b = 5
    RectangleArea s, a, b
    Debug.Print "a = "; a; " b ="; b; " s ="; s
End Sub
```

Эти две подпрограммы решают поставленную задачу (только для прямоугольника со сторонами 3 и 5).



Упражнения 2

Задание 1. Наверху используемых модулей опишите опции Option Explicit обязательного объявления переменных. Вызываемые подпрограммы расположите на стандартном модуле, вызывающие – на модуле формы. Измените, приведенные подпрограммы так, чтобы можно было вычислять площадь прямоугольника, стороны которого вводятся с клавиатуры. Опишите типы данных соответствующим образом и вычислите площадь прямоугольника со следующими данными.

1. Стороны прямоугольника – целые числа, не превосходящие 255.
2. Стороны прямоугольника – целые числа, не превосходящие 32 000.

3. Стороны прямоугольника – целые числа, не превосходящие 2 000 000 000.
4. Стороны прямоугольника – числа с плавающей запятой до 7 знаков точности.
5. Стороны прямоугольника – числа с плавающей запятой до 15 знаков точности.

Задание 2.

1. *Придумать задачи, решаемые с помощью вызываемых и вызывающих подпрограмм, например, задания из предыдущего пункта*.

Указание. При решении задания вызываемые подпрограммы поместить в стандартных модулях (в одном или более). Вызывающие подпрограммы поместить в модуль формы Form1(Code).

Задание 3.

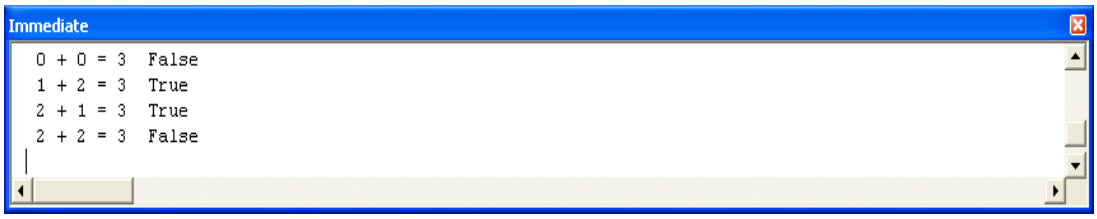
1. Найдите методом подбора несколько решений уравнения $x+y=3$.

Указание. Можно составить программу

```
Option Explicit

Function Tru1(x As Long, y As Long) As Boolean     ‘ Функция возвращает булево значение
   Tru1 = (x + y = 3) ‘ Булево значение равенства присваивается значению булевой функции
End Function

Private Sub Command1_Click()
   Dim a As Long
   Dim b As Long
   a = InputBox("a=", , 0)
   b = InputBox("b=", , 0)
```

```
    Dim Tru As Boolean              ‘ Описание локальной булевой переменной
    Tru = Tru1(a, b)
    Debug.Print a; "+"; b; "= 3  "; Tru    ‘ Выводится булево значение равенства
End Sub
```

Запустив программу на выполнение, вводя значения вместо переменных, можно получить, например, следующую картину



Найдены два решения.

Это задание преследует цель не только ознакомлению с решением уравнения, но ознакомлению и с типом Boolean.

Задание 4.

2. Найдите методом подбора несколько решений неравенства $x+y>3$.
3. Найдите методом подбора решение системы уравнений: $x+y=3$, $x-y=1$ в целых числах.
4. Найдите методом подбора несколько пар (x,y) целых чисел, удовлетворяющих хотя одному из уравнений: $x+y=3$, $x-y=1$
5. Найдите методом подбора несколько решений уравнения $y=x^2$ в целых числах.
6. Найдите методом подбора несколько решений неравенства $y>x^2$.

7. Найдите методом подбора несколько пар (x, y) целых чисел, не удовлетворяющих уравнению $y = x^2$:

8. Проверьте справедливость равенства $(x+y)^2 = x^2 + 2xy + y^2$ на нескольких примерах.

Глава 6. Операции над целыми числами

Целые числа в Visual Basic 6.0 представлены 3 типами

Byte (байт) – 1 byte, от 0 до 255 (0 ... 2^8 – 1)
Integer (целый) - 2 byte, от -32768 до 32767 (-2^15 … 2^15 -1)
Long (длинное целое – 4 byte, прибл. от -2 млр до 2 млр (-2^31 ... 2^31 – 1)

Над целыми числами осуществляются следующие операции:

^ Возведение в степень бинарное

* Умножение бинарное

\ Деление целочисленное бинарное

Mod Нахождение остатка от деления бинарное

+ Сложение унарное и бинарное

- Вычитание унарное и бинарное

Сложение чисел типа Byte

В проект добавьте стандартный модуль (Project – Add Module). В стандартном модуле наберите или перекопируйте код функции суммирования:

```
Option Explicit
Public Function Summa(ByVal x As Byte, ByVal y As Byte) As Byte
Summa = x + y
End Function
```

На форму установите кнопки Command1, Command2. В модуле формы наберите или перекопируйте фрагмент кода программы

```
Option Explicit
```

```
Private a As Byte      ‘ Глобальное объявление переменных целого типа Byte
Private b As Byte
Private Sub Command1_Click()
a = InputBox("a =", "Ввод чисел типа Byte", "0")
Debug.Print "a ="; a
b = InputBox("b =", "Ввод чисел типа Byte", "0")
Debug.Print "b ="; b
Command2.SetFocus ‘Передача фокуса ввода
End Sub
Private Sub Command2_Click()
Dim s As Byte
s = Summa(a, b)
Debug.Print "s ="; s
Command1.SetFocus ‘Передача фокуса ввода
End Sub
Private Sub Form_Load()
Command1.Caption = "Vvod A B"      ‘Надписи на кнопках в режиме выполнения
Command2.Caption = "Summa A B"
End Sub
```

Код программы суммирования двух целых чисел типа Byte готов.

Переменные *a, b* описаны глобально с ключевым словом Private. – Они видимы во всех процедурах модуля формы, но невидимы из стандартного модуля (они там и не используются).

В обработчике события Load формы устанавливаются надписи на кнопках.

В обработчике события Click кнопки Command1 осуществляется ввод чисел из диапазона 0 … 255. При попытке ввести числа вне этого диапазона вызовет ошибку времени выполнения – экспериментируйте. Для ввода используется стандартная функция ввода InputBox. Используются 3 параметра ее строкового типа String для

удобства ввода. Значение ее также имеет строковый тип, но компилятор автоматически преобразовывает строковый тип в целый тип Byte. По окончании ввода фокус ввода переводится на кнопку Command2 для удобства дальнейшей работы.

В обработчике события Click кнопки Command2 осуществляется вызов функции суммирования и присвоение ее значения переменной s. Переменная описана локально, так она используется только в этой процедуре. Если значение суммы будет вне диапазона 0 … 255, произойдет ошибка времени выполнения – экспериментируйте. Если же возвращаемого значения функции описать не как Byte, а скажем, как Integer или Long, то ошибка уже не будет происходить - экспериментируйте.

Наделение функции Summa в стандартном модуле ключевым словом Public делает видимой ее из других модулей, в частности из модуля формы. По умолчанию (т.е. если опустить ключевое слово) функция также видима из других модулей. Если же ключевое слово заменить словом Private, то станет невидимой из других модулей – экспериментируйте.

Упражнения 1

1. Наберите выше приведенный код программы суммирования чисел типа Byte. Осуществите эксперименты согласно выше приведенным комментариям.

2. Опишите программы действий и с другими операциями, используя подпрограммы-функции, или подпрограммы-процедуры на свое усмотрение. Вызываемые подпрограммы расположите на стандартных модулях (в одном или более), вызывающие подпро-

граммы – в модуле формы. Осуществите экспериментальные исследования.

3. В описанных программах замените тип Byte и другими типами и осуществите соответствующие исследования.

Сложение нескольких целых чисел, вводимых с клавиатуры

Задание. *Описать код программы, которая позволяет складывать несколько чисел, вводимых с клавиатуры*.

Решение. В переменную *Summa* (назовем ее накопитель суммы) вводятся частичные суммы, в переменную *a* – слагаемые, в переменную n – номера слагаемых.

Алгоритм сложения.

1. Осуществляется начальная инициализация переменных (накопителя суммы и номера слагаемых)
2. Вводится значение слагаемого в переменную.
3. Значение слагаемого добавляется к накопителю суммы. Продолжается снова с инструкции 2.

Примечание. Инструкция 1 выполняется 1 раз. Инструкции 2 и 3 выполняются столько раз, сколько это необходимо. Как правило, для этой цели используется оператор цикла. Но в ниже приведенной программе для этой цели вызывается обработчик события столько раз, сколько раз это требуется.

Чтобы начать складывать новую серию чисел, необходимо снова начинать с начальной инициализации.

Ниже идет более или менее подробный комментарий к алгоритму, и приводится код программы суммирования.

Сложение начинается с нейтрального элемента сложения, это число 0. Реализуется командами

n = 0
Summa = 0

Это начальная инициализация. Используем обработчик события Click кнопки Command1. Этот обработчик выполняется 1 раз при начале новой серии сложений.

Далее: вводится число, накопитель суммы увеличивается на это число. Кроме того, нумеруются слагаемые (только для удобства). Реализуется командами

a = InputBox("a =", "Ввод чисел типа Integer", "0")
n = n + 1
Summa = Summa + a

Это алгоритм суммирования. Используем обработчик события Click кнопки Command2. Этот обработчик выполняется столько раз, сколько необходимо сложить чисел.

(Можно было вводить число в одном обработчике, а складывать в другом обработчике. Но это дело вкуса пользователя).

Необходимо позаботиться и о выводе результатов. Раньше для этой цели было использовано окно Immediate Window. Теперь используем метку Label1:

Label1.Caption = "n = " + Str(n) + Chr(13) + Chr(10) + "Summa =" + Str(Summa)

В правой части оператора присваивания все слагаемые преобразованы в строковые типы.

Str(x) – преобразовывает число x в строковый тип

Chr(x) – преобразовывает десятичный код *x* символа в строковый тип.

Asc(x) – первый символ строки *x* преобразовывает в десятичный код.

Chr(13) + Chr(10) – переводит курсор на следующую строку, 13 – десятичный код клавиши Enter, **10** – десятичный код перевода строки.

+ – операция конкатенации

&– операция конкатенации. Этот знак предпочтительней знака +.

Например:

Label1.Caption = "n = " & Str(n) & Chr(13) & Chr(10) & "Summa =" & Str(Summa)

При использовании знака & компилятор автоматически преобразовывает числовой тип в строковый. Например:

Label1.Caption = "n = " & n & Chr(13) & Chr(10) & "Summa =" & Summa

Необходимо еще позаботится о надписях на кнопках. Это осуществим программно.

Упражнения 2

Нанесите на форму элементы Command1, Command2, Label1 и наберите или перекопируйте код – программа суммирования нескольких чисел готова:

```
Option Explicit
Dim n As Integer                ‘Глобальное описание переменных целого типа Integer
Dim Summa As Integer

Private Sub Command1_Click()
```

```
n = 0          ‘ Начальная инициализация счетчика слагаемых
Summa = 0      ‘ Начальная инициализация сумматора
Label1.Caption = "n = " + Str(n) + Chr(13) + Chr(10) + "Summa =" +
Str(Summa)     ‘Вывод значений на Label1
End Sub
Private Sub Command2_Click()
Dim a As Integer
a = InputBox("a =", "Ввод чисел типа Integer", "0")   ‘ Ввод слагае-
мого
n = n + 1          ‘ Увеличение счетчика слагаемых на 1
Summa = Summa + a‘ Увеличение сумматора на величину слагаемо-
го 1
Label1.Caption = "n = " + Str(n) + Chr(13) + Chr(10) + "Summa =" +
Str(Summa) ‘Вывод значений на Label1
End Sub
Private Sub Form_Load()
Command1.Caption = "Инициализация суммы"
Command2.Caption = "Суммирование"
End Sub
```

Приведенный пример «обучает» алгоритму суммирования и аналогичным операциям.

***Изучите алгоритм суммирования нескольких чисел, вводимых с клавиатуры*.**

Умножение нескольких целых чисел, вводимых с клавиатуры

Задание. *Описать код программы, которая позволяет умножать несколько чисел, вводимых с клавиатуры*.

Решение. В переменную *Product* вводятся частичные произведения, в переменную *p* – множители, в переменную n – номера множителей.

Алгоритм умножения

1. Осуществляется начальная инициализация переменных (произведения и номера произведения)
2. Вводится значение множителя в переменную.

3. Значение множителя умножается на произведение. Продолжается снова с инструкции 2.

Умножения начинается с нейтрального элемента умножения, это единица 1. Реализуется командами

n = 0
Product = 1

Это начальная инициализация.

Далее: вводится множитель p. Произведение Product умножается на множитель p. Кроме того, нумеруются множители (только для удобства). Реализуется командами

p = InputBox("p =", "Ввод чисел типа Integer", "0")
n = n + 1
Product = Product * p

Это алгоритм умножения.

Дальнейшие подробности предоставляются читателю.

Упражнения 3

Упражнение

1. Опишите программу умножения нескольких чисел, вводимых с клавиатуры.
2. Опишите программу вычитания из данного числа нескольких чисел, вводимых с клавиатуры.
3. Опишите программу целочисленного деления данного числа на нескольких чисел, вводимых с клавиатуры.

Примечание. Понятие «нейтральный элемент» для операций вычитания и деления не определено.

Упражнение

1. Для каких операций, известных вам, определены понятия *нейтральный элемент*, а для каких нет?
2. Для каких логических операций определены понятия *нейтральный элемент*? Чему равны нейтральные элементы операции конъюнкции ***And***? операции дизъюнкции ***Or***?
3. Для булевых элементов (для элементов типа Boolean) можно определить понятия *Логическая сумма нескольких элементов* и *Логическое произведение нескольких элементов*. Определите эти понятия на языке математики. Опишите программы нахождения логической суммы и логического произведения.

Условный оператор If. Делители числа

Цель. *На примере делители числа ознакомиться с условным оператором, операциями целочисленного деления и оператором цикла.*

Операции \, Mod находят соответственно целочисленное частное и остаток от деления.

Если остаток от деления одного числа на другое равен 0, то второе число называют делителем первого числа, а первое число – делимым второго числа.

Для проверки делимости одного числа на другое применяется условный оператор If. Он имеет несколько форм синтаксиса:

- строчная форма
- блочная форма
- расширенная блочная форма

Строчная форма синтаксиса имеет вид:

If *условие* **Then** *оператор*1 **Else** *оператор*2

Блочная форма имеет вид:

If *условие* **Then**

*Оператор*1
Else
*Оператор*2
End If

Если выражение *условие* принимает значение True, то

выполняется *оператор*1, следующий за ключевым словом **Then** (*Then-часть*),

в противном случае выполняется *оператор*2, следующий за словом **Else** (*Else-часть*).

Выполняемые операторы (*Оператор*1 и *Оператор*2) могут быть и составными. Составляющие их при блочной форме могут быть записаны в столбик, а при строчной форме отделены друг от друга двоеточием.

Обратите внимание на места, занимаемые выполняемыми операторами! Между чем и чем они расположены?

Else-часть можно опускать вместе со словом Else. Тогда приобретает более простой вид:

If *условие* **Then**
*Оператор*1
End If

Предложение

Если остаток от деления m на n равен 0, *то n – делитель m, m – делимое n*

может быть реализовано, например, командой:

```
If m Mod n = 0 Then
   Debug.Print n; " - делитель "; m
   Debug.Print m; " - делимое "; n
End If
```

Обратите внимание на запись составного оператора – она заключена между заголовком **If** m Mod n = 0 **Then** и концом **End If**.

Могут быть реализованы и более сложные предложения.

Расширенная блочная форма условного оператора в этом пункте не затрагивается. Она будет рассмотрена при необходимости ее применения.

Задание 1. *Даны два действительных числа* ***x, y****. Найти их максимум.*

Решение. Из двух чисел то число принимается за *максимум*, которое больше другого, а то число, которое меньше другого, за – *минимум*. в общем случае допускается равенство чисел. В этом случае каждое из этих чисел одновременно служат и максимумом и минимумом этих чисел. В общем случае можно дать такое определение максимума:

Если ***x*** *больше* ***y****, то* ***max = x****, в противном случае* ***max = y****.*

Здесь ***max*** – означает максимум указанных чисел. Приведенное предложение реализуется с помощью условного оператора If.

```
If x > y Then
   Max = x
Else
   Max = y
End If
```

Решение задания удобно организовать в виде подпрограммы:

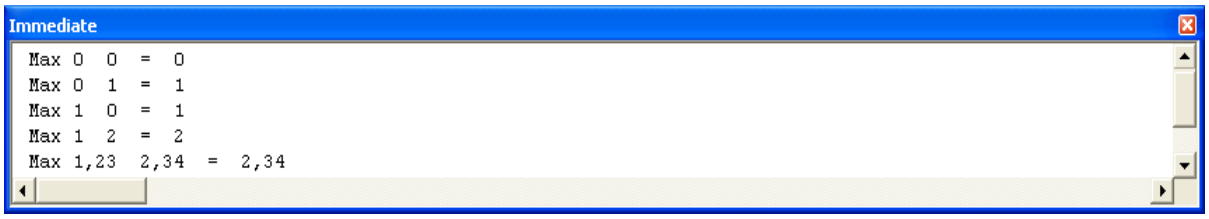
```
Sub Maximum(max As Double, x As Double, y As Double)
If x > y Then
   max = x
Else
   max = y
End If
End Sub
```

Здесь максимальное значение вторых двух параметров присваивается первому параметру.

Для фактического нахождения максимума заданных чисел необходимо подпрограмму вызвать из процедуры обработки события, введя фактические параметры:

```
Private Sub Command1_Click()
   Dim a As Double
   Dim b As Double
   Dim max As Double
   a = InputBox("a =", , 0)
   b = InputBox("b =", , 0)
   Maximum max, a, b
   Debug.Print "Max"; a; b; " = "; max
End Sub
```

Запустив программу на выполнение, можно получить картину:



Примечание. Можно предложить *второй* вариант решения. Оператор

```
If x > y Then
   Max = x
Else
   Max = y
End If
```

заменить оператором

```
Max = x
If Max < y Then
   Max = y
End If
```

Второй вариант решения в некоторых случаях – предпочтительней первого варианта.

Упражнения 4.

1. Даются два числа. a) Найти их максимум. b) Найти их минимум. c) Найти их максимум и минимум.

2. Даются три числа. a) Найти их максимум. b) Найти их минимум. c) Найти их максимум и минимум.

3. Даются два положительных действительных числа. A) Найти максимум между их средним арифметическим и средним геометрическим. B) Найти минимум между их средним арифметическим и средним геометрическим.

4. Даются три положительных действительных числа. A) Найти максимум между их средним арифметическим и средним геометрическим. B) Найти минимум между их средним арифметическим и средним геометрическим.

5. Даны два действительных числа a, b. Проверить, выполняется ли неравенство $a<b$.

6. Даны три действительных числа a, b, c. Проверить, выполняются ли неравенства $a<b<c$.

7. Функция $f(x)$ определена следующим образом: $f(x)=\begin{cases}-x \text{ïðè } x<0\\ x \text{ïðè } x\geq 0\end{cases}$

Опишите функцию.

Оператор цикла For … Next. Подсчет числа делителей данного числа

При выполнении нескольких операторов возникает необходимость в использовании операторов цикла. В Visual Basic 6.0 имеется несколько видов операторов цикла. В этом пункте затрагивается только оператор цикла For.

Оператор разъясним на примере перечисления делителей заданного числа.

Задача. *Посчитать число делителей заданного числа и вывести их на экран*.

Алгоритм словесный:

- Перечисляются возможные делители заданного числа – для этого необходим оператор *цикла*.
- Проверяется, являются ли они делителями данного числа или нет – для этого необходим *условный* оператор.
- Если являются делителями, то выполняются какие-то определенные *действия*.

Для реализации алгоритма и решения задачи необходимы переменные:

n – *Исследуемое число*.
i – *Счетчик цикла*
j – *Счетчик числа делителей*

Option Explicit

```
Private Sub Command1_Click()
Dim n As Long          ‘Исследуемое число
Dim i As Long          ‘Счетчик цикла
Dim j As Long          ‘Счетчик числа делителей
n = InputBox("n =", "Ввод чисел типа Long", 1)‘Ввод     Исследуемое число
Debug.Print "Делители  "; n
j = 0  ‘Начальная инициализация счетчика числа делителей
For i = 1 To n Step 1
If n Mod i = 0 Then
   j = j + 1         ‘Увеличение счетчика числа делителей
   Debug.Print i; ‘Вывод делителя
```

```
End If
Next i
Debug.Print
Debug.Print "Число делителей  "; j
End Sub
```

Голубым цветом выделены *заголовок* и *конец* цикла. Между ними расположено *тело* цикла. Step 1 – шаг изменения счетчика цикла, его можно опустить, по умолчанию шаг цикла равен 1.

Цикл For с положительным шагом выполняется следующим образом:

Сначала счетчик цикла принимает свое начальное значение (в данном случае 1) и выполняется тело цикла. Счетчик изменяется на величину шага (увеличивается на 1).

Если значение счетчика не стало больше конечного значения (в данном случае *n*), то тело цикла выполняется еще раз и снова изменяется счетчик.

Такой процесс продолжается и далее. Продолжается до тех пор, пока значение счетчика не станет равным конечному значению. При этом тело цикла выполняется еще раз, и выполнение цикла на этом завершается, (счетчик становится большим конечного значения цикла).

Примечание. Если начальное значение счетчика больше конечного значения, то цикл с положительным шагом ни разу не выполняется. Более точно, цикл выполняется до тех пор, пока счетчик не выйдет за пределы диапазона изменения его.

В теле цикла осуществляется проверка того, является ли *i* делителем данного числа. И если является делителем, то выполняются два действия

* счетчик числа делителей увеличивается на 1 единицу (считает число делителей),

* и делитель выводится в окно Immediate Window в строчку.

По окончании цикла курсор переводится на следующую строку, чтобы продолжить вывод с новой строки. А затем выводится и число делителей.

Упражнения 5.

1. Установите на форму кнопку Command1. Вызовите окно View – Immediate Window. Перекопируйте приведенный код программы в модуль формы. Программа готова.
2. Запустите программу на выполнение. Нажмите на кнопку Command1. В появившемся окне введите целое число. Программа выведет делители этого числа и число его делителей.

Расширенная блочная форма условного оператора If. Простые числа

Цель

- закрепить операции целочисленного деления
- организация счетчика объектов с определенным свойством
- получить определенное представление о расширенной блочной форме условного оператора
- продолжить знакомство с оператором цикла

Число называют *простым*, если оно имеет ровно 2 делителя, и *составным*, если имеет более двух делителей. Единицу 1 не относят ни к простым числам, ни составным.

Простые числа представляют собой удобный пример для демонстрации и изучения условного оператора и операторов цикла.

Задача. *Перед собой поставим цель – перечислить простые числа*.

Мы не ставим перед собой цель, перечислить простые числа самым эффективным способом. Читатель же может поставить себе такую цель.

Решение. Сначала построим функцию, которая находит число делителей целого числа. Такую функцию нетрудно построить с учетом программы предыдущего пункта. Назовем ее **CisloDelit**:

```
Option Explicit

Public Function CisloDelit(ByVal n As Long) As Long
Dim i As Long  ‘Счетчик цикла
Dim k As Long  ‘Счетчик делителей
            k = 0 ‘Начальная инициализация счетчика делителей
For i = 1 To n
      If n Mod i = 0 Then
            k = k + 1   ‘Подсчет делителей
      End If
Next i
CisloDelit = k            ‘Число делителей
End Function
```

Эту функцию желательно установить на стандартном модуле.

Обратите внимание на счетчик делителей числа:

```
If n Mod i = 0 Then
k = k + 1
```

End If

С помощью построенной функции можно проверить, является число простым или составным. Можно, например, предложение

Если число делителей равно 1*, то число и не простое и не составное,*

Иначе если число делителей равно 2*, то число простое,*

Иначе число составное.

реализовать в форме:

```
d = CisloDelit(n)
If d = 1 Then
   Debug.Print n; "- число не простое и не составное"
ElseIf d = 2 Then
   Debug.Print n; "- число простое"
Else
   Debug.Print n; "- число составное"
End If
```

Здесь приведен пример расширенной блочной формы условного оператора – состоит из трех блок частей: **If**-часть, **ElseIf**-часть, **Else**-часть. Выделены голубим цветом операторы указанных частей. Операторы могут быть пустыми, простыми или составными.

- If-часть – обязательная часть – всегда 1 часть.
- ElseIf-часть – необязательная часть – могут быть несколько частей.
- Else-часть – необязательная часть – может быть 1 часть, а может и не быть.
- End If – завершение обязательно.

В данном примере

- Проверяется условие d = 1. Если оно верно, выполняется If-оператор.
- Иначе проверяется условие d = 2. Если оно верно, выполняется ElseIf-оператор.
- Иначе выполняется Else-оператор.

В следующем примере условный оператор состоит из 4 блок частей:

```
d = CisloDelit(n)
If d = 1 Then
   Debug.Print n; "- имеет 1 делитель "
ElseIf d = 2 Then
   Debug.Print n; "- имеет 2 делителя "
ElseIf d = 3 Then
   Debug.Print n; "- имеет 3 делителя "
Else
   Debug.Print n; "- имеет больше 3 делителей "
End If
```

Опишите словесно, как выполняется приведенный условный оператор.

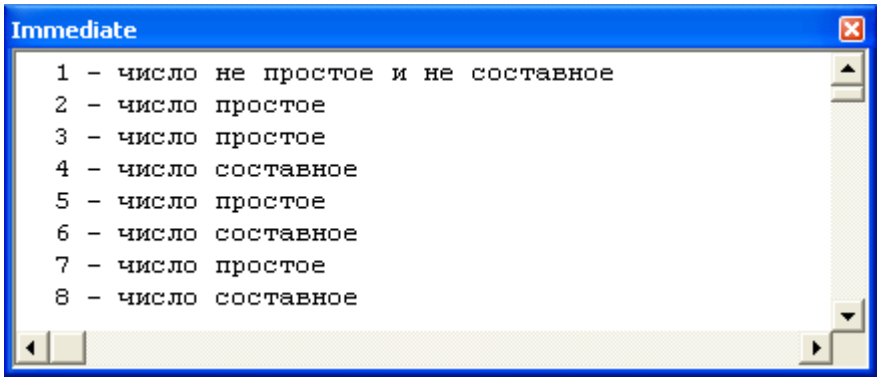
Задача. *Перечислите целые числа от* 1 *до* 100 *с указанием того, что число простые или составные*.

Решение. Наберите в стандартном модуле код функции **CisloDelit**, приведенной выше. На форму установите кнопку Command1. На модуле формы наберите следующий код кнопки, (улучшите код программы). Код программы решения задачи готов.

```
Option Explicit

Private Sub Command1_Click()
Dim n As Long  'Счетчик цикла – Исследуемые числа
Dim d As Long  'Число делителей исследуемые чисел
```

```
For n = 1 To 100
        d = CisloDelit(n)
        If d = 1 Then
           Debug.Print n; "- число не простое и не составное"
        ElseIf d = 2 Then
           Debug.Print n; "- число простое"
        Else
           Debug.Print n; "- число составное"
        End If
Next n
End Sub
```

Запустив программу на выполнение и щелкнув по кнопке Command1, получим фрагмент из ответов:



Упражнения 6

Упражнение 1.

1. Перечислите простые числа в диапазоне 1 … 100.
2. Найдите число простых чисел в диапазоне 1 … 100.
3. Найдите число простых чисел и в других диапазонах (нескольких) на свое усмотрение.

Задача 4. *Вывести на экран дисплея все простые числа первой сотни*.

Указание. В цикле, параметр *n* которой изменяется от 1 до 100, вызвать функцию числа делителей командой d = CisloDelit(n). Вывести на экран дисплея те числа *n*, для которых *d* = 2.

Задача 5. *Вывести все простые числа второй сотни, третьей сотни, четвертой сотни*.

Задача 6. *Вывести все простые числа из данного диапазона чисел*.

Задача 7. *Вывести все числа первой сотни, которые имеют ровно* 3 *делителя: ровно* 4 *делителя*.

Задача 8. *Вывести все числа из данного диапазона, которые имеют ровно* 3 *делителя; ровно* 4 *делителя*.

Задача 9. *Вывести все числа из данного диапазона, которые имеют либо* 3 *делителя, либо* 4 *делителя*.

Задача 10. *Вывести все составные числа из данного диапазона*.

Задача 11. *Дается целое n. Вычленить и вывести все десятеричные цифры числа*.

Указание. Команда n mod 10 находит последнюю (первую с конца) цифру числа, а команда n / 10 находит часть числа, что остается от него после отбрасывания последней цифры. Например, если n = 123456, то n mod 10 равно 6, а n / 10 равно 12345. Следовательно, для решения задачи достаточно команды ch:= n mod 10; n:= n / 10; выполнять в соответствующем цикле, пока *n* не станет равным 0. Попутно, при желании, можно осуществлять какие-нибудь действия со значениями ch.

Задача 12. *Дается целое n. Преобразовать его последовательно в двоичное, троичное, восьмеричное, шестнадцатеричное числа* (найти цифры).

Указание. Команда n mod 2 находит первую с конца цифру двоичного разложения числа, а команда n / 2 находит часть числа, что остается от него после отбрасывания последней цифры.

При записи шестнадцатеричных чисел латинская буква *A* означает цифру для 10 единиц, *B* – для 11, *C* – для 12, *D* – для 13, *E* – для 14, *F* – для 15.

Оператор цикла Do … Loop. Нахождение числа делителей данного числа

Выше была описана функция **CisloDelit**(n), которая находила число делителей данного числа *n*. Однако она не эффективна в том смысле, что если число *n* большое, допускает много проверок. Например, если n = 1000000, то требует миллион проверок. Имеются другие более эффективные методы нахождения делителей.

Один из них следующий: Можно проверять не все числа, а только те, квадраты которых не превосходят исследуемого числа. Второй делитель можно найти делением исследуемого числа на найденный делитель. Например, если n = 1000000, то теперь потребуется всего 1000 проверок.

Хотя и в этом случае для проверок можно было б использовать цикл **For** … **Next**, но в данном случае удобней использовать другой цикл **Do .. Loop**.

Этот цикл используется либо с условием завершения цикла **While** ***условие***, либо с условием завершения цикла **Until** ***условие***.

Со служебным словом **While** (*пока*) цикл выполняется пока ***условие*** истинно.

Со служебным словом Until (*пока не станет*) цикл выполняется до тех пор, пока ***условие*** не станет истинным.

Каждый из указанных служебных слов также имеют по две формы синтаксиса: либо в начале цикла – *предусловие*, либо в конце цикла – *постусловие*. Таким образом, имеются 4 формы синтаксиса

Do While ***Условие***
Тело цикла
Loop

Do
Тело цикла
Loop While ***Условие***

Do Until ***Условие***
Тело цикла
Loop

Do
Тело цикла
Loop Until ***Условие***

В цикле с *предусловием* сначала проверяется условие, затем выполняется тело цикла.

В цикле с *постусловием* сначала выполняется тело цикла, затем проверяется условие.

В последнем случае цикл, по крайней мере, один раз всегда выполняется независимо от условия. В первом случае цикл может

ни разу и не выполняться в зависимости от условия. Этим, по сути, и отличаются цикл с предусловием от цикла с постусловием.

Здесь используется только одна из форм синтаксиса цикла:

```
Public Function CisloDelit2(ByVal n As Long) As Long
Dim i As Long  ‘счетчик цикла
Dim k As Long  ‘счетчик числа делителей
k = 0
i = 0
Do  ‘Считает число делителей, квадраты которых не превосходят исследуемое число
    i = i + 1     ‘Пробегает проверяемые числа
    If n Mod i = 0 Then    ‘Условие увеличения счетчик числа делителей
       k = k + 1
    End If
Loop Until i * i >= n          ‘Условие завершения цикла
If n = 2 Then    ‘Расширенная блочная форма условного оператора, определяет число делителей
  CisloDelit2 = 2
ElseIf i * i = n Then
  CisloDelit2 = k * 2 - 1
ElseIf i * i > n Then
  CisloDelit2 = k * 2
End If
End Function
```

Часть кода

```
k = 0
i = 0
Do
i = i + 1
If n Mod i = 0 Then
  k = k + 1
End If
Loop Until i * i >= n
```

вычисляет число делителей, квадраты которых не превосходят *n*. Переменная *i* – счетчик цикла, переменная *k* – счетчик числа дели-

телей. Если счетчик цикла *i* является делителем, то счетчик делителей *k* увеличивается на 1 единицу – считает делители. Обратите внимание, счетчики сначала инициализируются.

Цикл выполняется следующим образом:

Сначала выполняется тело цикла – При этом счетчик *i* становится равным 1.

Проверяется условие i * i >= n, если оно ложно, тело цикла снова выполняется, счетчик становится равным 2.

Этот процесс продолжается и далее. Продолжается пока условие i * i >= n не станет истинным. Как только это условие становится истинным, цикл перестает выполняться.

Упражнения 7

1. Опишите функцию нахождения числа делителей данного числа, используя и 3 другие формы синтаксиса оператора цикла Do … Loop.

TextBox как элемент управления формы для ввода и вывода данных. Перечисление простых чисел

Задача. *Перечислить простые числа первой сотни*.

Алгоритм перечисления словесный. Перебираются числа от 1 до 100, и те из них, которые имеют 2 делителя, выводятся на экран.

Элементы можно выводит на разные объекты. Но в виде ознакомления с текстовым полем элементы будем выводить на элемент Text1 (часть Box при установке на форму автоматически усекается). Для этой цели ознакомимся некоторыми свойствами Текстового поля.

Текстовое поле TextBox по умолчанию представляет собой однострочный редактор.

При выводе элементов желательно его сделать многострочным и установить вертикальную полосу прокрутки. В режиме конструирования установить:

Свойство **MultiLine** на **True**
свойство **ScrollBars** на **Vertical**.
Используемые команды вывода:

Команда

Text1.Text = S

заменяет текст поля Text1 строкой *S*. Команда может понадобиться при выводе данных, удаляя при этом предыдущие данные.

Команда

Text1.SelText = S

заменяет выделенную часть текста поля Text1 строкой *S*. При отсутствии выделенной части заменяется место курсора, по иному, выводится, начиная с места курсора. Команда может понадобиться при выводе данных, оставляя предыдущие данные.

Выражение *S* может быть и чисто числовым. В этом случае компилятор числовое выражение преобразовывает в строку. Но если выражение смешанное, числовую его часть необходимо преобразовать в строку функцией Str. Например,

Text1.SelText = Str(n) + " "

Здесь число *n* преобразуется в строку, после чего *конкатенируется* (складывается) со строкой *пробел* (внутри кавычек). Запись

Text1.SelText = n + " " приводит к ошибке времени выполнения – несовпадение типов (экспериментируйте).

Приведенная команда может использоваться при выводе чисел в строчку, отделяя их друг от друга пробелами.

Команда

Text1.SelText = Str(n) + Chr(13) + Chr(10)

может использоваться при выводе чисел в столбик. Выражение **Chr(13) + Chr(10)** переводит курсор на следующую строку. Функция Chr преобразует числовой код в символ, 13 – код клавиши Enter, 10 – код перевода строки.

Команда

Text1.SelText = Chr(13) + Chr(10)

может быть использована для перевода курсора на следующую строку. Например, чтобы начать выводит с новой строки, если вывод осуществлялся в строку.

Команда ввода в переменную

a = Text1. Text

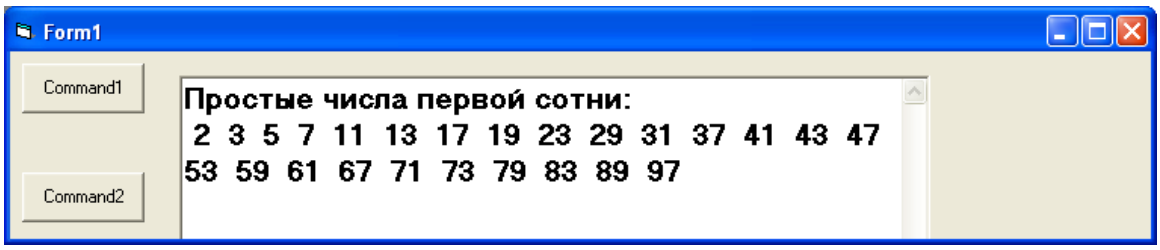
вводит в переменную *a* текст поля Text1. Текст поля Text1 представляет собой строку, и, если *a* – переменная числового типа, то компилятор преобразовывает строку в соответствующий числовой тип. Эта команда может использоваться для ввода.

Следующий обработчик выводит простые числа первой сотни на текстовое поле Text1. Предполагается, что уже описана вышеприведенная функция **CisloDelit2**(n) нахождения числа делителей *n*.:

```
Private Sub Command2_Click()
Dim n As Long  'Исследуемые числа
```

```
Dim d As Long  ‘Число делителей
Text1.SelText = "Простые числа первой сотни:" + Chr(13) + Chr(10)
For n = 1 To 100        ‘Цикл
        d = CisloDelit2(n)
        If d = 2 Then
                Text1.SelText = Str(n) + " "        ‘Выводит в строчку
        End If
Next n
Text1.SelText = Chr(13) + Chr(10)    ‘Перевод курсора на следующую строку
End Sub
```

Предполагается, что уже описана функция **CisloDelit2**(n) нахождения числа делителей *n*.



Упражнения 8.

1. Отредактируйте выше приведенный обработчик Command2_Click() так, чтобы он выводил список простых чисел из любого данного диапазона целых чисел. Например, первая сотня целых чисел – это есть диапазон целых чисел.

2. Выведите целые числа из данного диапазона, которые имеют ровно 3 делителя.

3. Выведите целые числа из данного диапазона, которые имеют ровно 4 делителя.

4. Выведите целые числа из данного диапазона, которые имеют 3 или 4 делителя.

5. Выведите целые числа из данного диапазона, для которых число делителей не равно ни 3, ни 4.

6. Выведите целые числа из данного диапазона, для которых число делителей больше 2, но меньше 5.

7. Выведите составные числа из данного диапазона.

8. Выведите список простых чисел из диапазона, который начинается числом, имеющим 1 делитель, и заканчивается первым числом, имеющим 12 делителей.

9. Найдите первое число, которое имеет ровно 32 делителя.

10. Найдите второе число, которое имеет ровно 32 делителя.

11. Найдите в первой тысяче число, которое имеет наибольшее число делителей.

Указание. *Используйте булевы операции*.

Булевы операции

Булево выражение описывается как тип Boolean. Над ними осуществляют операции:

Not – Отрицание (*negation*), одноместная операция.

And – Конъюнкция (*conjunction*), двуместная.

Or – Дизъюнкция (*disjunction*), двуместная.

False, True – диапазон принимаемых значений булевыми выражениями. Допускается ввод их с клавиатуры.

False – нейтральный элемент дизъюнкции: ***a* Or False = *a*.**

True– нейтральный элемент конъюнкции: ***a* And True = *a*.**

Операция отрицания не имеет нейтрального элемента.

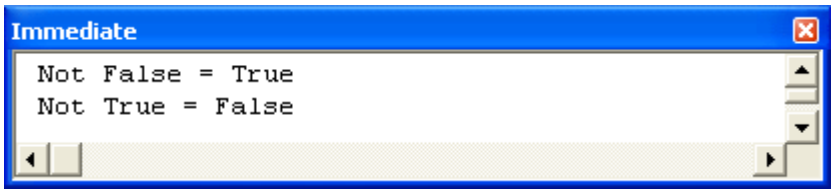
Упражнения 9.

1. Описать процедуры операций над булевыми выражениями.
2. Описать отрицание дизъюнкции двух выражений.
3. Описать конъюнкцию отрицаний двух выражений.
4. Описать отрицание конъюнкции двух выражений.
5. Описать дизъюнкцию отрицаний двух выражений.

6. Описать отрицание отрицания выражения.
7. Покажите: Отрицание дизъюнкции двух выражений есть конъюнкция отрицаний этих выражений.
8. Покажите: Отрицание конъюнкции двух выражений есть дизъюнкция отрицаний этих выражений.
9. Покажите: Отрицание отрицания выражения есть само выражение.
10. Сформулируйте предыдущие предложения для нескольких булевых выражений. Как показать математически их справедливость.

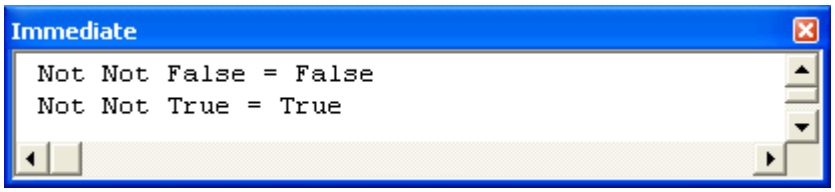
Указание. *Пример*. Операция отрицания:

```
Private Sub Command1_Click()
Dim a As Boolean
a = InputBox("a =", "Ввод False или True", False)
Debug.Print "Not "; a; " = "; Not a
End Sub
```



Указание. *Пример*. Отрицание отрицания:

```
Private Sub Command1_Click()
Dim a As Boolean
a = InputBox("a =", "Ввод False или True", False)
Debug.Print "Not Not "; a; " = "; Not Not a
End Sub
```



Дизъюнкцию *нескольких выражений называют* ***логической суммой*** *указанных выражений.*

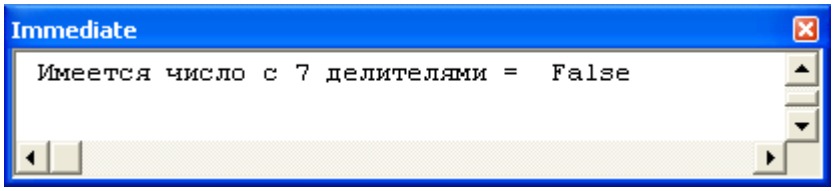
Конъюнкцию *нескольких выражений называют* ***логическим произведением*** *указанных выражений.*

Пример построения логической суммы

Задание. Построить логическое значение предложения: *В первой сотне чисел имеется число, имеющее 7 делителей.*

Построение. Следующая процедура строит указанное значение:

```
Private Sub Command6_Click()
Dim n As Long
Dim d As Long
Dim Tru As Boolean
Tru = False       ‘Нейтральный элемент логического сложения
For n = 1 To 100
d = CisloDelit2(n)
Tru = Tru Or (d = 7)              ‘Логическое сложение
Next n
Debug.Print "Имеется число с 7 делителями =  "; Tru
End Sub
```



Ответ: Среди первой сотни чисел нет числа с 7 делителями.

Значение искомого предложения *Tru* – логической суммы строит фрагмент:

```
Tru = False
For n = 1 To 100
d = CisloDelit2(n)
```

Tru = **Tru Or (d = 7)**
Next n

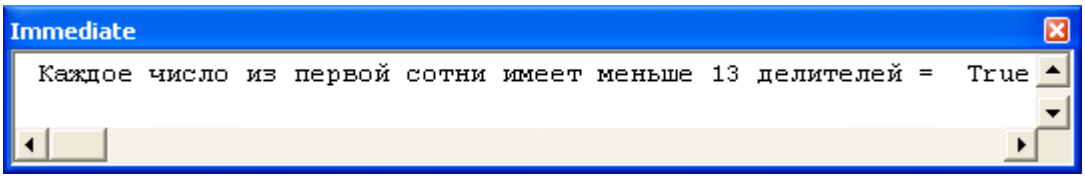
Пример построения логического произведения.

Задание. Построить логическое значение предложения: *Число делителей каждого числа первой сотни чисел меньше* 13.

Построение. Следующая процедура строит указанное значение:

Private Sub Command6_Click()
Dim n As Long
Dim d As Long
Dim Tru As Boolean
Tru = True
For n = 1 To 100
d = CisloDelit2(n)
Tru = **Tru And (d < 13)**
Next n
Debug.Print "Каждое число из первой сотни имеет меньше 13 делителей = "; Tru

End Sub



Ответ: Да, действительно, Каждое число из первой сотни имеет меньше 13 делителей.

Логическая сумма *нескольких выражений соответствует* ***кванторy существования*** *по этим выражениям.*

Логическое произведение *нескольких выражений соответствует* ***квантору общности*** *по этим выражениям.*

Упражнения 10

Постройте логические значения предложений:

1. В первой сотне целых чисел есть число, имеющее 1 делитель.
2. Во второй сотне нет ни одного числа, которое имело бы 1 делитель.

3. Дан диапазон чисел. Любое число из данного диапазона чисел имеет либо 2 делителя, либо имеет 3 делителя.
4. Дан диапазон чисел. Ни одно число из данного диапазона не имеет ни 2 делителей, ни 3 делителей.

Избранные задания, связанные с целыми числами

Задание 1. **Степень с натуральным показателем** определяется по рекурсии

1) Степень с показателем 0 равен 1: $a^0 = 1$
2) Степень с другим показателем равен произведению степени с предыдущим показателем на основание степени: $a^n = a^{n-1} \cdot a$, n = 1, 2, 3, …

a) Опишите как подпрограмму Function степень с натуральным показателем по основанию 2.

b) Опишите как подпрограмму Function степень с натуральным показателем по основанию 3.

Задание 2. **Числа Фибоначчи** определяются по рекурсии

1) Первые два числа Фибоначчи задаются.
2) Число Фибоначчи (отличное от первых двух) равно сумме двух предыдущих.

a) Выведите числа Фибоначчи в строчку (в несколько строк).

b) Выведите числа Фибоначчи в столбик.

Задание 3. **Факториальная функция** определяется по рекурсии

1) Факториал 0 равен 1: 0! = 1.
2) Факториал числа, отличного от 0, равен произведению факториала предыдущего числа на само число: n! = (n – 1)! n, n= 1, 2, 3, …

a) Опишите факториальную функцию как подпрограмму Function.

Задание 4. **Биноминальные коэффициенты – Сочетания из m элементов по n элементов** обозначаются русской транскрипции как C_m^n, в иностранной транскрипции как $\binom{m}{n}$. Определяется

1) $C_m^n = 0$, если $m<0$ или $n<0$.
2) $C_m^n = 1$, если $m\geq 0$ и $n=0$.
3) $C_m^n = C_m^{n-1} \cdot \frac{m-n+1}{n}$, если $m\geq 0$ и $n>0$

a) Опишите биноминальные коэффициенты как подпрограмму Function, обозначьте как $C(m,n)$.

b) Выведите в одну строку биноминальные коэффициенты с индексами m = 0 (например, начиная с n = 0), в следующую – с индексами m = 1, и так далее.

c) Найдите суммы биноминальных коэффициентов последовательно с индексами: m = 0, m = 1, m = 2, …. Сделайте определенные выводы.

Глава 7. Массивы. Динамические массивы.

Опция Option Base

Массив (array) – *это группа однотипных переменных, которые объединены одним общим именем*.

Их можно считать одной из разновидностей переменных. Различие в том, что в массивах можно хранить не одно, а несколько значений. Доступ к заданному элементу массива осуществляется с помощью индекса. Синтаксис объявления массива отличается от синтаксиса объявления переменных тем, что требуется указать размерность массива и границы изменения индексов. Массивы

бывают одномерные, двумерные, трехмерные, и т.д. до 60 размерности.

Пример. Команда

```
Dim A (1 To 10) As Integer
```

объявляет одномерный массив целых чисел с именем **A**. Граница изменения индексов задается в диапазоне от 1 до 10.

Допускается в описании опускать нижний индекс. В этом случае нижний индекс по умолчанию становится равным 0.

Пример. Команда

```
Dim B (10) As Integer
```

объявляет одномерный массив целых чисел с именем **B**. Граница изменения индексов задается в диапазоне от 0 до 10.

Впрочем, значение нижнего индекса по умолчанию можно изменять с помощью опции **Option Base** 0, либо опции **Option Base** 1. В первом случае значение нижнего индекса будет равно 0, во втором – 1. Опции записываются в начале модуля сразу после опции **Option Explicit**, например:

```
Option Explicit
Option Base 1
```

Пример. Команда

```
Dim C (1 To 10, 1 To 5) As Double
```

объявляет двумерный массив чисел с плавающей точкой двойной точности с именем **C**. Граница изменения первых индексов с опцией задается в диапазоне от 1 до 10, вторых индексов – в диапазоне от 1 до 5.

Пример. Команда

```
Dim D (10, 5) As Double
```

объявляет двумерный массив чисел с плавающей точкой двойной точности с именем **D.**

Граница изменения первых индексов с опцией **Option Base** 1 задается в диапазоне от 1 до 10, вторых индексов – в диапазоне от 1 до 5.

Граница изменения первых индексов с опцией **Option Base** 0 задается в диапазоне от 0 до 10, вторых индексов – в диапазоне от 0 до 5.

Для объявления границы изменения индексов массива удобно использовать именованную константу. Например, фрагмент

```
Const nn = 10
Dim D (nn) As Integer
Dim E (nn) As Integer
```

объявляет два массива с границами изменения индексов в диапазоне от 0 до 10 (если не указана опция **Option Base** 1).

Для изменения границ индексов обоих массивов достаточно изменить значение именованной константы в одном месте.

Одномерные массивы. Примеры ввода и вывода массивов

Одномерные массивы на практике часто отождествляют с векторами. Над массивами можно осуществлять разнообразные операции. Одной из важных операций является ввод значений элементов массива.

Ввод значений элементов одномерного массива.

Для ввода значений элементов массива удобно использовать цикл. Следующий фрагмент программы вводит в массив **A** квадраты целых чисел в диапазоне от 1 до 10.

```
Dim A (1 To 10) As Integer
Private Sub Command1_Click()
Dim i As Integer      ‘Счетчик цикла
For i = 1 To 10       ‘Заголовок цикла
      A(i) = i * I ‘Массив задан своей закономерностью
Next I                ‘Конец цикла
End Sub
```

Массив **A** описан глобально для возможного использования его в других подпрограммах.

Следующий фрагмент программы вводит в массив **A** 10 случайных чисел, заключенных между 0 и 7.

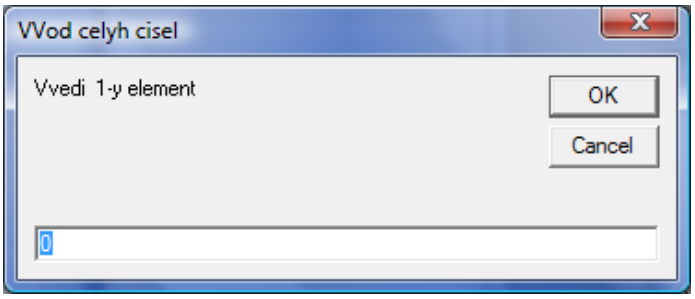
```
Dim A (1 To 10) As Integer
Private Sub Command1_Click()
Randomize             ‘Позволяет каждую серию начинать со случайного числа
Dim i As Integer
For i = 1 To 10
      A(i) = 7 * Rnd  ‘Массив из 10 целых случайных, меньших 7
Next i
End Sub
```

Функция 7 * Rnd генерирует случайные числа с плавающей запятой между 0 и 7. Однако, так как массив A описан как целочисленный, то при присвоении A(i) = 7 * Rnd компилятор автоматически округляет числа до целых. Таковое допускается не во всех языках программирования. Более правильной было присвоение осуществлять командой A(i) = Round(7 * Rnd), что соответствовало бы подходам и других языков программирования.

Для ввода значений элементов массива из клавиатуры можно использовать диалоговую функцию InputBox:

```
Dim A(1 To 10) As Integer

Private Sub Command1_Click()
Dim Prompt As String     ‘Строка подсказки
Dim Title As String      ‘Строка для заголовка
Title = "VVod celyh cisel"      ‘Заголовок окна ввода
Dim i As Integer
For i = 1 To 3
      Prompt = "Vvedi " + Str(i) + "-y element" ‘Подсказка
      A(i) = InputBox(Prompt, Title, 0)     ‘Ввод с клавиатуры 3 элементов массива
Next i
End Sub
```

При работе с этим фрагментом появляется окно диалога. Обратите внимание на строки-подсказки. После введения числа в поле диалога и нажатия OK это число вводится в соответствующую переменную A(i) массива A.



Для наглядности обозрения вводимых данных удобно ввод данных сопровождать с одновременным выводом их на экран. В следующем фрагменте ввод данных сопровождается с одновременным выводом их в окно отладки Immediate.

```
Dim A(1 To 10) As Integer

Private Sub Command1_Click()
```

```
Dim Prompt As String
Dim Title As String
Title = "VVod celyh cisel"
Dim i As Integer
For i = 1 To 3
      Prompt = "Vvedi " + Str(i) + "-y element"
      A(i) = InputBox(Prompt, Title, 0)     ‘Ввод с клавиатуры
      Debug.Print A(i);                     ‘Вывод в окно Immediate
Window
Next i
Debug.Print                                 ‘Перевод курсора на следу-
ющую строку
End Sub
```

Команда **Debug.Print A(i)**; выводит значение **A(i)** в окно отладки, начиная с позиции, где был расположен курсор, и переводит курсор вправо на следующую позицию. Команда **Debug.Print** переводит курсор на следующую строку. Обратите внимание на присутствие точки с запятой в первом операторе и отсутствие таковой во втором.

Вывод значений элементов одномерного массива

Здесь под выводом понимается вывод значений на экран дисплея. Удобно выводить данные в окно **формы**, в окно отладки **Immediate**, в текстовое поле **TextBox** и др.

Следующий фрагмент программы выводит массив в строчку, как в окно **формы**, так и окно отладки **Immediate**. Чтобы вывести в столбик, необходимо удалить точку с запятой (;) в соответствующих командах.

```
Private Sub Command2_Click()
Dim i As Integer
For i = 1 To 10
```

```
        Form1.Print A(i);     ‘Вывод в строчку в окна формы Form1
и
        Debug.Print A(i);     ‘Immediate Window. В столбик – снять
точку с запятой
Next i
Form1.Print
Debug.Print
End Sub
```

Следующий фрагмент программы выводит массив в строчку в поле элемента **TextBox1**.

```
Private Sub Command2_Click()
Dim i As Integer
For i = 1 To 10
        Text1.SelText = Str(A(i)) + " "‘Вывод в строчку в поле
Text1
Next i
        Text1.SelText = Chr(13) + Chr(10)        ‘Перевод курсора
на следующую строку
End Sub
```

Следующий фрагмент программы выводит массив в строчку на метке **Label1**.

```
Private Sub Command2_Click()
Dim S As String
        S = ""                          ‘Инициализация конкатенации
Dim i As Integer
For i = 1 To 10
        S = S + Str(A(i)) + " "      ‘ Конкатенация элементов масси-
ва в строчку
Next i
Label1.Caption = S                  ‘Вывод результата конкатенации
End Sub
```

Следующий фрагмент программы выводит массив в столбик на метке **Label1**.

```
Private Sub Command2_Click()
Dim S As String
        S = “”                      ‘Инициализация конкатенации
```

```
Dim i As Integer
For i = 1 To 10
      S = S + Str(A(i)) + Chr(13) + Chr(10)    ‘ Конкатенация
элементов массива в столбик
Next i
Label1.Caption = S                ‘Вывод результата конкатенации
End Sub
```

Упражнения 1.

1. Опишите два массива одинаковой размерности. Введите в первый массив элементы по своему усмотрению. Элементы первого массива присвойте элементам второго массива. Выведите оба массива в строчку или столбик по своему усмотрению.

2. Опишите два массива одинаковой размерности. Введите в первый массив элементы по своему усмотрению. Элементы первого массива присвойте элементам второго массива в обратном порядке. Выведите оба массива.

3. Опишите два числовых массива одинаковой размерности. Введите в первый массив элементы по своему усмотрению. Квадраты элементов первого массива присвойте второму массиву. Выведите оба массива.

4. Найдите суммы квадратов элементов введенного массива.

5. Найдите индекс элемента массива с наибольшим значением и индекс элемента с наименьшим значением. Поменяйте местами значения этих элементов.

6. Осуществите циклическую перестановку элементов массива (в одном и в другом направлениях).

7. Возведите первый элемент массива в первую степень, второй элемент – во вторую степень, и так далее. Присвойте полученные значения другому массиву.

Динамические массивы. Построение массива простых чисел

Динамические массивы используются в том случае, когда размерность массива нельзя определить заранее. В момент своего объявления такой массив не содержит ни одного элемента:

Dim A ( ) as Long

Чтобы использовать массив и добавить в него элементы, необходимо использовать инструкцию **ReDim**:

ReDim A (100)

Теперь к элементам массива можно обращаться точно так же, как и в случае массива с заранее определенным числом элементов.

При необходимости количество элементов массива можно снова изменить путем повторного использования инструкции **ReDim**:

ReDim A (1000)

Внимание. При изменении размерности массива все значения, хранящиеся в массиве, теряются.

Чтобы сохранить значения при изменении размерности массива, следует использовать ключевое слово **Preserve**, например:

ReDim Preserve A (1000)

Задача. *Построить массив простых чисел из первой сотни.*

Решение – идея. Перебираются числа из диапазона 1 … 100. Первое встретившееся простое число наделяем номером и вводим его в

массив с указанным номером. Номер следующего встретившегося простого числа увеличиваем на 1 единицу и с этим номером вводим в массив. Эту процедуру продолжаем и далее. Продолжаем далее до тех пор, пока не исчерпаются все испытуемые числа.

Нумерацию простых чисел в принципе можно начинать с любого целого числа. Как правило, нумерацию начинают либо с 0 либо 1. Здесь предпочтение отдается 1. Таким образом, первое встретившееся простое число получает номер 1, следующее 2, и т.д.

При этом предполагается, что описана функция **CisloDelit2** нахождения числа делителей данного числа.

Так как число простых чисел заранее неизвестно, то необходимо использовать динамический массив и цикл **Do … Loop**. Обозначим массив через A. Массив опишем глобально на случай возможного его использования в других подпрограммах.

При первой встрече простого числа динамический массив расширяется до номера этого числа, используя ключевые слова **ReDim Preserve**. При второй встрече массив снова расширяется до номера встретившегося числа. И так далее. Продолжается до тех пор, пока не исчерпаются искомые простые числа.

Dim A() As Long
n – Переменная, пробегающая испытуемые номера.

i – Номера простых чисел.

Для удобства заново приводится функция **CisloDelit2** и обработчик, строящий массив простых чисел из диапазона 1 … 100.

```
Option Explicit
Dim A ( ) As Long
Public Function CisloDelit2(ByVal n As Long) As Long ‘ Число делителей n
Dim i As Long
Dim k As Long
k = 0
i = 0
Do
i = i + 1
If n Mod i = 0 Then
   k = k + 1
End If
Loop Until i * i >= n

If n = 2 Then
   CisloDelit2 = 2
ElseIf i * 1 = n Then
   CisloDelit2 = k * 2 - 1
ElseIf i * i > n Then
   CisloDelit2 = k * 2
End If
End Function
Private Sub Command3_Click()
ReDim A(0)          ‘Изменяет размер массива без сохранения изменений
Dim n As Long  ‘Исследуемое число
Dim d As Long  ‘Число делителей
Dim i As Long  ‘Номер простого числа
i = 0          ‘ Начальные инициализации
n = 0
Do             ‘ Заголовок цикла
n = n + 1   ‘ Счетчик цикла, увеличивается на 1
      d = CisloDelit2(n)      ‘Число делителей
      If d = 2 Then
            i = i + 1 ‘Счетчик простых чисел
            ReDim Preserve A(i) ‘Изменяет размер массива с сохранением предыдущих изменений
```

```
            A(i) = n     ‘Ввод простого числа в массив
        End If
Loop Until n >= 100 ‘Конец цикла
n = i ‘Число простых чисел
For i = 1 To n
        Debug.Print A(i);       ‘Вывод простых чисел в строчку в окно Immediate Window
Next i
Debug.Print       ‘Перевод курсора в следующую строку

Debug.Print "Число простых чисел = "; n  ‘Вывод числа простых чисел
End Sub
```

Переменная i – номер простого числа, начинается с начальной его инициализации, n – испытуемое число, d – число делителей испытуемого числа.

Если испытуемое число простое (см. фрагмент кода), то

- Номер простого числа увеличивается на 1 единицу: i = i + 1
- Размер динамического массива расширяется до индекса i с сохранением предыдущих значений: ReDim Preserve A(i)
- Простое число вводится в массив с построенным индексом: A(i) = n
- Элемент массива выводится на экран (для проверки правильности выполнения программы). Элементы массива можно использовать и в другом обработчике, например, их выводить.
- По завершении цикла значение индекса i становится равным числу построенных простых чисел.

Приведенный код программы *показывает, как можно использовать динамический массив*.

Упражнения 2.

1. Постройте массивы а) из простых чисел первой сотни, б) из простых чисел второй сотни, в) из простых чисел третьей сотни. Определите число элементов массивов.

2. Постройте массивы а) из составных чисел первой сотни, б) из составных чисел второй сотни, в) из составных чисел третьей сотни. Определите число элементов массивов.

3. Разбейте числа первой сотни на два массива – на массив простых чисел и массив составных чисел.

4. Разбейте числа первой сотни на четыре массива. Первый массив – числа, имеющиеся по 1 делителю, второй – по 2 делителя, третий – 3 делителя, четвертый – все остальные.

5. Постройте два массива по своему усмотрению. Объедините их в один общий массив.

Разложение целого числа на произведение простых множителей

Цель – Демонстрация примера использования массива простых чисел, демонстрация еще раз динамического массива и условного оператора.

Вспомним алгоритм разложения числа на простые множители.

1. Записываем число, которое необходимо разложить на множители, назовем его *исследуемым* числом, и рядом проводим вертикальную черту.

2. Мысленно фиксируем первое простое число.

3. Последовательно перебираем простые числа, начиная с фиксированного числа, пока не найдем число, на которое делится

исследуемое число. Найденное число записываем рядом, и исследуемое число делим на него, записываем частное.

4. То же самое поступаем с полученным частным. То есть пункт 3 повторяем заново, перебирая теперь простые числа, начиная с записанного простого числа.

5. Приведенную процедуру продолжаем до тех пор, пока в частном не получится 1.

Примечание. В этом алгоритме мы встречаемся с особым статусом числа 1 – не является ни простым числом, ни составным. Этот факт используется для завершения цикла (алгоритма).

Реализация алгоритма. Переменные в процедуре.

- i – Номера (индексы) простых чисел. Нумерация начинается с 1 – вспомогательная переменная процедуры.
- pr – Простое число с индексом i – вспомогательная переменная процедуры.
- n – Разлагаемое число – параметр процедуры.
- B – Массив множителей разлагаемого числа – параметр процедуры.
- k – Номера множителей разлагаемого числа – параметр процедуры. Нумерация начинается с 1. По завершению выполнения алгоритма значение k становится равным числу множителей разлагаемого числа.

Приведенный алгоритм реализует процедура **ProductProst**:

```
Sub ProductProst(n As Long, B() As Long, k As Long)
Dim pr As Long     ‘Простое число
Dim i As Long      ‘Номер простого числа
i = 1      ‘Номер первого простого числа
k = 0
```

```
Do
   pr = A(i)    ‘Простое число с номером i
   If n Mod pr = 0 Then
      n = n / pr        ‘Делим на простой множитель
      k = k + 1         ‘Номер простого множителя увеличивается
      ReDim Preserve B(k)‘Расширяется массив простых множителей
      B(k) = pr         ‘Ввод в массив простого множителя
      Debug.Print B(k); ‘Вывод простого множителя
   Else
      i = i + 1 ‘Увеличивается номер простого числа
   End If
Loop Until n = 1   ‘Завершение цикла при получении в частном 1 – числа не простого и не составного

   Debug.Print   ‘Перевод курсора на следующую строку
End Sub
```

Используется цикл Do … Loop с условием завершения Until n = 1. Цикл начинается с проверки первого простого числа – начальная инициализация i = 1 до цикла и присвоение pr = A(i) в теле цикла. A – Массив простых чисел.

Если исследуемое число делится на проверяемое простое число: n Mod pr = 0, то

- исследуемое число делят на это простое число: n = n / pr,
- индекс массива множителей увеличивается на 1: k = k + 1,
- размер массива множителей расширяется: ReDim Preserve B(k),
- множитель вводится в массив B(k) = pr,
- цикл повторяется заново без изменения найденного простого числа.

Если же исследуемое число не делится на проверяемое простое число,

- то проверяется следующее простое число: i = i + 1.

Цикл завершается, когда частное от деления становится равным 1.

Глава 8. Операции над вещественными числами.

Вещественные числа в Visual Basic 6.0 представлены 2 типами

Single (с плавающей точкой одинарной точности) – 4 byte, до 7 знаков точности.

Double (с плавающей точкой двойной точности) – 8 byte, до 15 знаков точности.

Над вещественными числами осуществляются следующие операции:

+	Сложение унарное и бинарное
-	Вычитание унарное и бинарное
*	Умножение бинарное
/	Деление бинарное
^	Возведение в степень бинарное

Вещественные числа на ЭВМ представляются *приближенно* в виде конечных десятичных дробей в двух формах: стандартной и показательной. При представлении вещественных чисел на ЭВМ они автоматически округляются, число знаков округления зависит от того типа, в котором представляется число.

Округление при вычислениях занимает важное место. Для этой цели используются стандартные функции языков программирования.

Round(x) – Округляет число x до ближайшего целого.

Round(x, m) – Округляет число x до m знаков после десятичной запятой.

Int(x) – Округляет число x вниз до ближайшего целого.

Val(x) – Округляет x по недостатку до целого и преобразует его к целому типу Integer.

Функцию Int(x) можно использовать для округления чисел, превосходящих по величине целые типы. Например, Int(100*x + 0.5)/100 округляет положительное число *x* до 2 знаков после десятичной точки.

Для округления до *m* знаков после десятичной точки можно сначала найти коэффициент округления *mm* с помощью фрагмента

```
mm = 1
For i = 1 to m
mm = mm * 10
next
```

Тогда **Int** (mm * x + 0.5) / mm округляет положительное число *x* до m знаков после десятичной точки.

Упражнения 1

Задача 1. *Даются два вещественных числа. Описать процедуры операций над этими числами. Используя эти процедуры описать программу действий над двумя вещественными числами.*

Указание. См. аналогичную задачу действий над целыми числами предыдущего раздела.

Вычисление значений операций над вещественными числами с заданной точностью их представления

Как сказано выше, вещественные числа представляются на ЭВМ приближенно с определенной точностью, с точностью, зависящей от типа представления. При осуществлении операций над вещественными числами происходит накопление ошибок округлений, подчиняющихся правилам действий над погрешностями. Поэтому возникает проблема вычисления значений операций с заданной точностью их представления.

Задача. *Дается операция, зависящая от своих аргументов. Требуется вычислить ее значение с требуемой точностью его представления.*

Решение задачи. Всякая операция характеризуется значениями, как самой операции, так и значениями ее аргументов. Чем точнее представляются значения аргументов, тем точнее вычисляется значение самой операции.

По этой причине:

- Сначала значения аргументов операции округляются (представляются) с некоторым числом знаков после десятичной точки, вычисляется значение операции по правилам действий над десятичными дробями. Полученное значение операции округляется с требуемой в задаче точностью его представления.
- Затем значения аргументов снова округляются, но уже с большим числом знаков после десятичной точки, снова вы-

числяется значение операции. Полученное значение операции округляется снова с той же требуемой в задаче точностью его представлении

- Такая процедура продолжается и далее. Продолжается до тех пор, пока округленные значения операции не перестанут изменяться, то есть не стабилизируются к некоторой постоянной величине.

Найденная постоянная величина и будет значением операции с требуемой в задаче точностью его представления.

Примечание.

- Если округленные значения операции действительно перестают изменяться, начиная с некоторой точности представления значений ее аргументов, то операция называется ***сходящейся***.
- Если же округленные значения операции не перестают изменяться, с какой бы точностью ни представляли значения аргументов, то операция называется ***расходящейся***. В последнем случае операцию не наделяют значением.

В ы в о д.

1. *Решить задачу представления значения операции – это построить такую точность представления значений ее аргументов, что, начиная с которой, значения операции больше не будут изменяться при дальнейших более точных представлениях аргументов*.

2. *И та постоянная, к которой стабилизируются значения операции, и будет значением операции с точностью требуемой в задаче.*

Пример 1. *Вычислить сумму* $\sqrt{2} + \sqrt{5}$ *с точностью до целых. С какой точностью необходимо представлять слагаемые, чтобы вычислялась сумма с точностью до целых*?

Решение. Имеем

$$\sqrt{2} = 1{,}41421356237309504880168872420 97,$$

$$\sqrt{5} = 2{,}23606797749978969640917366873 13$$

с точностью представления до 32 знаков. (Значения вычислены на 32 разрядном калькуляторе).

Слагаемые представим с точностью до целых, сложим, и результат сложения округлим до целых, как того требует задача:

$$\sqrt{2} + \sqrt{5} = 1 + 2 = 3.$$

Слагаемые представим с точностью до десятых, сложим, и результат округлим снова до целых, как того требует задача:

$$\sqrt{2} + \sqrt{5} = 1.4 + 2.2 = 3.6 = 4.$$

Слагаемые представим с точностью до сотых, сложим, и результат округлим до целых:

$$\sqrt{2} + \sqrt{5} = 1.41 + 2.24 = 3.65 = 4.$$

Эту процедуру можно продолжить и далее. У нас получится последовательность

3, 4, 4, 4, …

округленных значений сумм. Округленные значения сумм перестали изменяться, начиная с точностью представления слагаемых до десятых. По иному, округленные значения сумм стали равняться

одному и тому значению 4, начиная с представления слагаемых до десятых. – То есть выполняется условия сходимости операции.

О т в е т : $\sqrt{2} + \sqrt{5} = 4$. *Слагаемые необходимо представлять с точностью до десятых, чтобы сумму вычислить с точностью до целых*.

Упражнения 2.

8. Вычислите сумму $\sqrt{2} + \sqrt{5}$ последовательно с точностями до сотых, до тысячных, до десятитысячных, и т.д.

Сложение и умножение нескольких чисел. Использование циклов

Во многих задачах приходится складывать несколько чисел. Эти числа могут быть построены разными способами: они могут быть введены с клавиатуры, они могут быть элементами массива, они могут быть заданы определенной закономерностью, и т.д. Число этих чисел заранее может быть известно, а может быть и неизвестно.

Если через *p* обозначить *текущие* значения слагаемых, через Sum - *текущие* значения сумм, то сложение нескольких чисел осуществляет следующий алгоритм:

1. Сначала осуществляется начальная инициализация текущей суммы: Sum = 0.
2. Находится слагаемое p, добавляется к текущей сумме: Sum = Sum + p
3. Затем снова находится слагаемое p и снова добавляется к текущей сумме: Sum = Sum + p
4. и так далее.

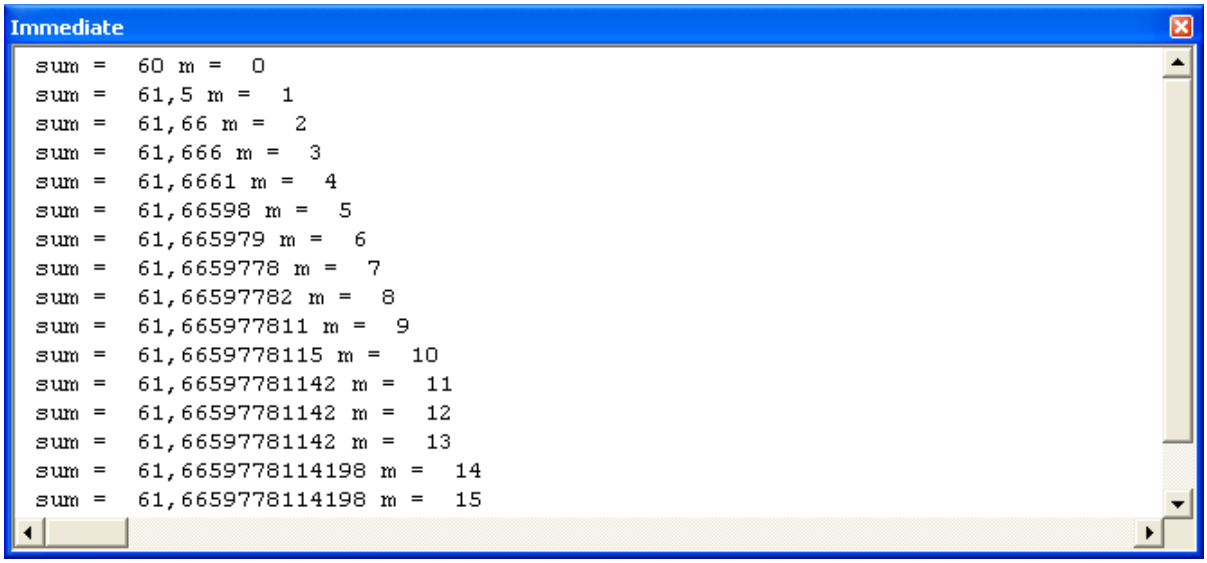
Для каждой серии складываемых чисел команда начальной инициализации выполняется заново, команды сложения выполняются столько раз, сколько раз необходимо складывать числа.

Задача. Вычислить сумму $\sum_{i=1}^{20}\sqrt{i}$ с требуемой точностью представления (до 15 знаков):

Решение Программа суммирования:

```
Option Explicit
Private Sub Command1_Click()
Dim m As Byte ‘Число знаков округления
m = InputBox("m =", "Ввод целых чисел < 16", "0")
Dim p As Double       ‘Текущие слагаемые
Dim q As Double       ‘Округленные значения слагаемых
Dim sum As Double  ‘Текущие суммы
Dim i As Long  ‘Счетчик цикла
      sum = 0    ‘Инициализация текущей суммы
For i = 1 To 20
      p = Sqr(i)  ‘Слагаемые задаются своей закономерностью
      q = Round(p, m)        ‘Округление слагаемых
      sum = sum + q  ‘Вычисление текущих сумм
Next i
Debug.Print "sum = "; sum; "m = "; m‘Вывод результата
End Sub
```

Запустим программу на выполнение



Число знаков округления слагаемых придадим последовательно равным 0, 1, 2, и т.д., получим картину:

Начиная с округления m = 0, перестали изменяться первые цифры сумм – 6. Следовательно, она точная цифра.

Начиная с округления m = 1, перестали изменяться первые 2 цифры – 61. Следовательно, 61 – точная часть искомой суммы.

Начиная с округления m = 2, перестали изменяться первые 4 цифры – 61,66. Следовательно, 61,66 – точная часть суммы.

Дальнейшие суждения предоставляется читателю.

Задача. Вычислить сумму $\sum_{i=1}^{20}\sqrt{i}$ с требуемой точностью до 4 знаков после десятичной точки:

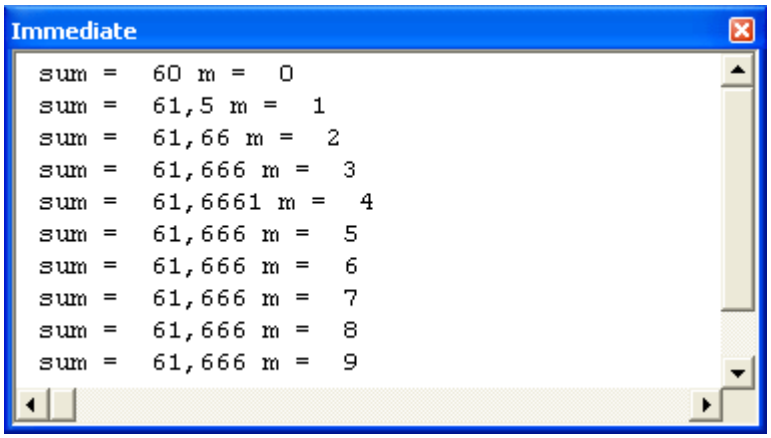
Решение. В предыдущей **Программе суммирования** команду

Debug.Print "sum = "; sum; "m = "; m‘Вывод результата

заменим командой

Debug.Print "sum = "; **Round**(sum, **4**); "m = "; m ‘Вывод результата

Запустим программу на выполнение. Число знаков округления слагаемых придадим последовательно равным 0, 1, 2, и т.д., получим картину:



Ответ: Округленные значения текущих сумм до 4 знаков после десятичной точки *стабилизировались* к постоянной величине 61,666 начиная с округления слагаемых до m = 5.

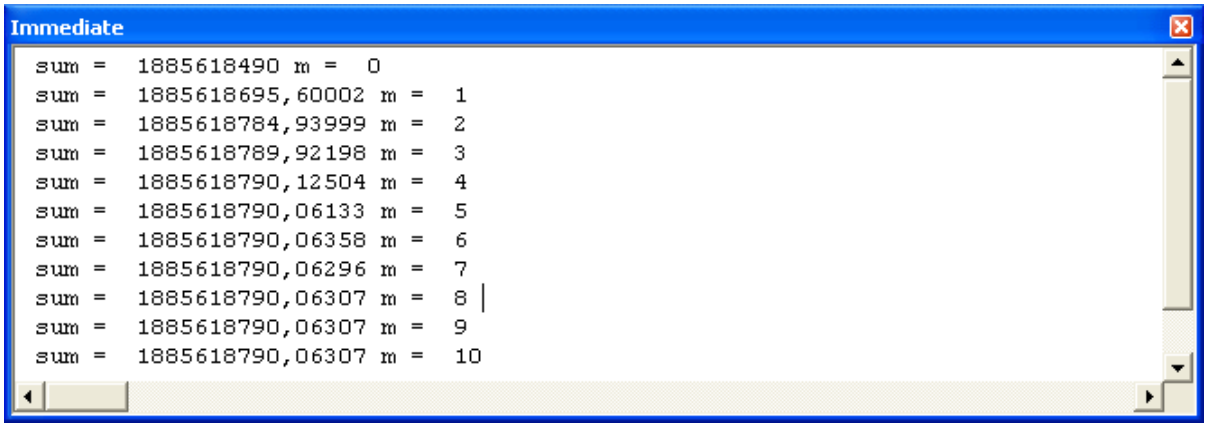
Сумма равна 61,666 с точностью до 4 знаков после десятичной точки.

Упражнения 3.

Задача 1. *Вычислить значения сумм* $\sum_{i=1}^{n}\sqrt{i}$ *последовательно при значениях n* = 10, 100, 1000, 1000000. *С какими точностями необходимо представлять слагаемые, чтобы значения сумм находились с точностью до целых, с точностью до десятых, с точностью до сотых, и т.д.*?

Указание. В предыдущей программе заголовок цикла For i = 1 To 20 изменить на For i = 1 To n. Значение n вводить с клавиатуры.

При суммировании $\sum_{i=1}^{2000000}\sqrt{i}$ получаем картину



Из этой картины выясняется, чтобы найти целую часть суммы 1885618790 (0 знаков после десятичной запятой), необходимо слагаемые округлять до 4 знаков после десятичной запятой, а чтобы

найти сумму с 15 точными знаками – слагаемые округлять до 8 знаков после запятой.

Глава 9. Суммирование числовых рядов. Последовательность чисел. Предел последовательности

Понятие последовательности и ряда тесно связаны между собой. Подходы к понятиям предела последовательности и суммы числового ряда здесь отличаются от традиционного подхода.

Сходимость к пределу с требуемой точностью его округления

Определение. *Последовательность чисел называется сходящейся к своему пределу с требуемой точностью его округления, если последовательность округленных значений ее членов перестает изменяться, начиная с некоторого номера, то есть становится равным к некоторой постоянной величине*.

Эта величина принимается за предел последовательности с требуемой точностью округления.

Пример. *Исследовать последовательность* $\frac{1}{n}$, $n = 1, 2, 3, \ldots$ *на сходимость*.

Решение. Исследуемая последовательность хорошо известна из анализа. Ее предел находится с абсолютной точностью и равен 0.

Однако наша задача иная – на примере этой последовательности продемонстрировать проблемы, возникающие из приведенного определения.

Округлим члены последовательности соответственно с точностями до 0 знаков, 1 знака, 2 знаков и т.д. после десятичной запя-

той. Для каждой серии выведем определенное число членов. При этом m означает число знаков округления.

При m = 0 получается картина:

1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0

Обращает на себя внимание тот факт, что последовательность округленных значений стабилизировалась к постоянной величине 0, начиная с номера n = 2. Следовательно, можно применить приведенное определение сходимости:

Последовательность сходится к 0 (0 целых), начиная с номера n = 2.

При m = 1 получается картина:

1 0,5 0,3 0,2 0,2 0,2 0,1 0,1 0,1 0,1 0,1 0,1 0,1 0,1 0,1 0,1 0,1
0,1 0,1 0 0 0 0 0 0 0 0 0 0

Последовательность стабилизировалась к постоянной величине 0 (0 целых 0 десятых), начиная с номера n = 20.

При m = 2 получается картина:

1 0,5 0,33 0,25 0,2 0,17 0,14 0,12 0,11 0,1 0,09 0,08 0,08 0,07
0,07 0,06 0,06 0,06 0,05 0,05 0,05 0,05 0,04 0,04 0,04 0,04 0,04
0,04 0,03 0,03 0,03 0,03 0,03 0,03 0,03 0,03 0,03 0,03 0,03 0,02
0,02 0,02 0,02 0,02 0,02 0,02 0,02 0,02 0,02 0,02 0,02 0,02 0,02
0,02 0,02 0,02 0,02 0,02 0,02 0,02 0,02 0,02 0,02 0,02 0,02 0,02
0,01 0,01 0,01 0,01 0,01 0,01 0,01 0,01 0,01 0,01 0,01 0,01 0,01
0,01 0,01 0,01 0,01 0,01 0,01 0,01 0,01 0,01 0,01 0,01 0,01 0,01
0,01 0,01 0,01 0,01 0,01 0,01 0,01 0,01 0,01 0,01 0,01 0,01 0,01
0,01 0,01 0,01 0,01 0,01 0,01 0,01 0,01 0,01 0,01 0,01 0,01 0,01
0,01 0,01 0,01 0,01 0,01 0,01 0,01 0,01 0,01 0,01 0,01 0,01 0,01
0,01 0,01 0,01 0,01 0,01 0,01 0,01 0,01 0,01 0,01 0,01 0,01 0,01
0,01 0,01 0,01 0,01 0,01 0,01 0,01 0,01 0,01 0,01 0,01 0,01 0,01
0,01 0,01 0,01 0,01 0,01 0,01 0,01 0,01 0,01 0,01 0,01 0,01 0,01
0,01 0,01 0,01 0,01 0,01 0,01 0,01 0,01 0,01 0,01 0,01 0,01 0,01

0,01 0,01 0,01 0,01 0,01 0,01 0,01 0,01 0,01 0,01 0,01 0,01 0,01
0,01 0,01 0,01 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Последовательность стабилизировалась к постоянной величине 0 (0 целых 0 сотых), начиная с номера n = 200.

Этот процесс можно продолжить и далее. Такое исследование и дает картину сходимости исследуемой последовательности.

Полученная при этом последовательность номеров 2, 20, 200, …, начиная с которых последовательности стабилизировались, показывает *скорость* сходимости исследуемой последовательности.

Следующий код программы выводит округленные значения выше рассмотренной последовательности.

Упражнения 1

Задание. Установите на форму кнопку **Command1** и текстовое поле **Text1**большого размера. Свойство **MultiLine** установите на **True**, а полосу прокрутки на **ScrollBars** на **Vertical**.

Запустите проект на выполнение. Нажмите на Command1. Введите число знаков округления (небольшое) и число членов. Методом подбора числа членов добейтесь стабилизации выводимых членов – найдете предел последовательности и определите характер скорости сходимости.

Можно составить более эффективную программу – попытайтесь!

```
Option Explicit
Private Sub Command1_Click()
```

```
Dim m As Byte ‘Число знаков округления членов последовательности
m = InputBox("m = ", "Число знаков округления", 0)
Dim n1 As Long ‘Число выводимых членов последовательности
n1 = InputBox("n1 = ", "число выводимых членов", 0)
Dim n As Long ‘Номера членов последовательности
Dim p As Double ‘Значения членов последовательности
Dim q As Double ‘Округленные значения членов последовательности
For n = 1 To n1
p = 1 / n ‘Значения членов последовательности – формула общего члена
q = Round(p, m) ‘Округление членов
Text1.SelText = Str(q) & " " ‘Вывод через пробел в строчку на текстовое поле
‘Для вывода в столбик – знак пробела " " заменить на перевод строки Chr(13) & Chr(10)
Next n
Text1.SelText = Chr(13) & Chr(10)
Text1.SelText = “n = “ & Str (n1)
Text1.SelText = Chr(13) & Chr(10)
End Sub
```

Изменив соответствующим образом команду **p = 1 / n** в программе, можно выводить округленные значения членов и другой последовательности.

Задача 1. *Исследовать на сходимость последовательности от натурального аргумента*:

$\frac{1}{n^2}$, $\frac{1}{n^3}$, 2^{-n}, 3^{-n}, e^{-n}, $\frac{n-1}{n}$, $n \sin\frac{1}{n}$, $(1+\frac{1}{n})^n$.

Числовые ряды

Числовым рядом называют выражение вида

$$\sum_{k=1}^{\infty} a_k = a_1 + a_2 + a_3 + \ldots$$

где a_k – данные числа. Встречаются как сходящиеся, так и расходящиеся числовые ряды (ряды, которым можно приписать сумму, и

ряды, которым нельзя приписать сумму). Если ряд сходится, то общий член его сходится к 0, но этого недостаточно для сходимости ряда. Имеются числовые ряды, общий член которых сходится к 0, в то время как ряд расходится. Примером такого ряда может служить расходящийся гармонический ряд

$$\sum_{n=1}^{\infty}\frac{1}{n} = 1+\frac{1}{2}+\frac{1}{3}+...+\frac{1}{n}+...$$

Ряды на ЭВМ суммируются приближенно, приближенно с наперед заданной точностью.

Задача суммирования ряда. Дается ряд. Требуется найти его сумму с требуемой точностью ее представления.

Правило суммирования ряда. (*Рекомендуется выучить*)

1. Слагаемые представляются с некоторой точностью, суммируются по правилам сложения десятичных дробей – суммируются до тех пор, пока слагаемые не обратятся в 0. Результат суммирования округляется с требуемой в задаче точностью представления суммы.

2. Затем слагаемые представляются с большей точностью и суммируются, а результат суммирования округляется снова с требуемой точностью представления суммы.

3. Этот процесс продолжают и далее. Продолжают до тех пор, пока округленные значения сумм не перестанут изменяться – то есть не начнут становиться равными одной и той постоянной величине.

4. Если округленные значения сумм действительно перестанут изменяться, начиная с некоторого представления слагаемых, то ряд называется *сходящимся*, и указанная постоянная величина принимается за сумму ряда с требуемой точностью ее представления.

5. Если же округленные значения сумм не перестают изменяться, как бы точно ни представлялись слагаемые, то ряд называется расходящимся. Расходящийся ряд не наделяют суммой.

Примечание. На Visual Basic 6.0 числа с точностью более, чем в 15 знаков не представляются.

Задача 1. *Исследовать на сходимость гармонический ряд*

$$\sum_{n=1}^{\infty}\frac{1}{n} = 1+\frac{1}{2}+\frac{1}{3}+...+\frac{1}{n}+...$$

Решение. Члены ряда будем округлять последовательно до 0 знаков, до 1 знака, до 2 знаков, и т.д. А результаты суммирования, для определенности, будем округлять до целых. При этом

m – Число знаков округления.

n – Число складываемых (ненулевых) слагаемых.

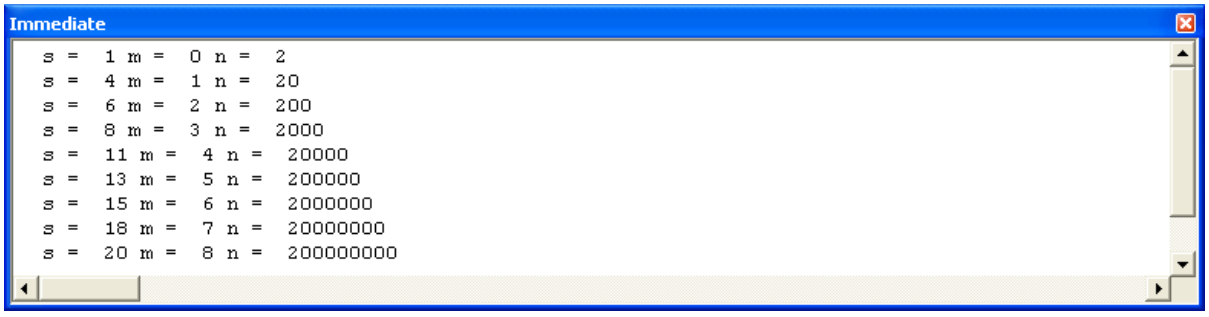
s – Частичные суммы.

Код программы суммирования гармонического ряда:

```
Option Explicit

Private Sub Command1_Click()
Dim m As Byte  'Число знаков округления слагаемых
m = InputBox("Vvod chisla znakov okrug. m = ", "Vvod Ch. type Byte", 0)
Command1.Caption = "m = " + Str(m)
Dim s As Double        'Текущая сумма
```

```
Dim p As Double       ‘Значение слагаемых
Dim q As Double       ‘Округление слагаемых
Dim n As Long  ‘Номера слагаемых

s = 0 ‘Начальная инициализация
n = 0
Do    ‘Цикл
      n = n + 1    ‘Счетчик номеров слагаемых
      p = 1 / n:   ‘Значение слагаемых
      q = Round(p, m)        ‘округление слагаемых
      s = s + q    ‘Текущие суммы
Loop Until q = 0        ‘Завершение цикла

s = Round(s)     ‘Округление суммы до целых
Debug.Print " s = "; s; "m = "; m; "n = "; n   ‘Вывод на Immediate Win-
dow
End Sub
```

Запустив программу на выполнение, придавая числу знаков округления слагаемых m последовательно значения 0, 1, 2, и т.д., получим картину:



Из этой картины видно, что частичные суммы s не перестают изменяться, как бы точнее ни представляли слагаемые – ряд расходится.

Последовательности s и n указывают на характер расходимости. Например, с увеличением числа знаков округления на 1 единицу

частичная сумма s увеличивается примерно на $\ln 10$, а число ненулевых слагаемых n – в 10 раз.

Задача 2. *Просуммировать гармонический ряд с показателем* 2

$$\sum_{n=1}^{\infty}\frac{1}{n^2} = 1+\frac{1}{2^2}+\frac{1}{3^2}+...+\frac{1}{n^2}+...$$

соответственно с точностями до 0 *знаков, до* 1 *знака, до* 2 *знаков, и т.д. после десятичной точки.*

С какой точностью необходимо при этом округлять слагаемые, чтобы получить указанную точность суммы,

И сколько необходимо сложить слагаемых.

Решение.

Сумма ряда с точностью до целых.

Код суммирования гармонического ряда с показателем 2 получается из кода суммирования гармонического ряда заменой команды **p = 1 / n** на команду **p = 1 / n / n**.

Будем округлять слагаемые последовательно до 0 знаков, 1 знака, 2 знаков, и т.д. и суммировать их.

Результаты суммирования округляемых ***до целых***. Получим картину

Частичные суммы перестали изменяться, начиная с округления *m=2*, а число слагаемых с *n = 15*.

Ответ: Ряд сходится. Сумма ряда равна 2 (с точностью до целых). Чтобы найти сумму ряда с точностью до целых, достаточно округлить слагаемые до 2 знаков и сложить 15 членов.

Сумма ряда с точностью до десятых

Результаты суммирования округляемых ***до десятых*** – в коде программы команду **s = Round(s)**

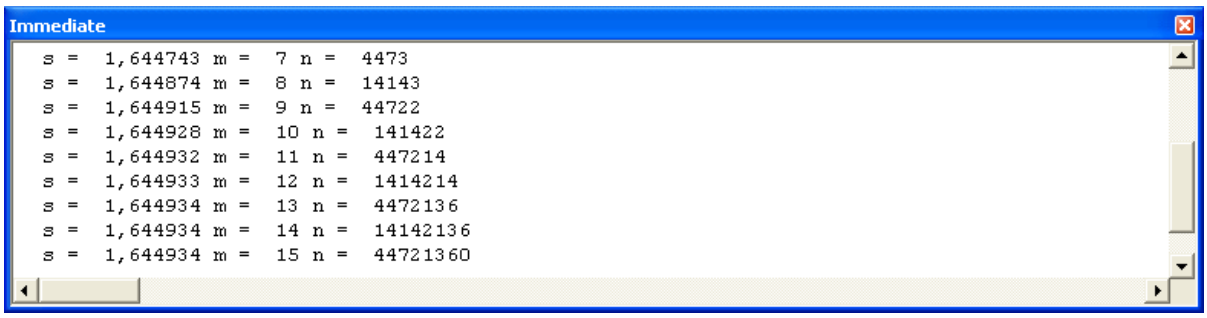
Заменим командой **s = Round(s, 1)**. Получим картину

Ответ: Сумма ряда равна 1.6 (с точностью до десятых). Чтобы найти сумму ряда с точностью до десятых, достаточно округлить слагаемые до 2 знаков и сложить 15 членов.

Сумма ряда с точностью до 6 *знаков после десятичной точки*

Результаты суммирования округляемых ***до десятых*** – в коде программы команду **s = Round(s)**

Заменим командой **s = Round(s, 6)**. Получим картину



Ответ: Сумма ряда равна 1.644934 (с точностью до 6 знаков после десятичной точки). Чтобы найти сумму ряда с точностью до 6 знаков после десятичной точки, достаточно округлить слагаемые до 13 знаков и сложить 4472136 членов.

Изучим полученные таблицы. Чем с большей точностью представляются слагаемые, тем с большей точностью вычисляются суммы, и при этом приходится складывать тем большее число слагаемых.

Характеристика скорости сходимости ряда

Пусть n_0, n_1, n_2, … число слагаемых, отличных 0 при представлении их соответственно с точностями 0 знаков, 1 знак, 2 знака, … после десятичной точки. В нашем примере $n_0 = 1$, $n_1 = 4$, $n_2 = 14$, … Такая последовательность играет важную роль при исследовании скорости сходимости рядов. Назовем ее ***Характеристической*** последовательностью. Она ведет себя как геометрическая прогрессия. Составляем для нее геометрическую характеристику – знаменатель последовательности:

$q = \lim \frac{n_{k+1}}{n_k}$ – геометрическая характеристика скорости сходимости.

Признак сходимости ряда: *Если* $\lg q < 1$, *то ряд сходится, если* $\lg q \geq 1$, *то ряд расходится*.

Характеристика числа точных знаков суммы. При увеличении точности слагаемых ряда на 1 знак число точных знаков суммы увеличивается примерно на $1-\lg q$ знаков.

Так, для гармонического ряда $q=10$, $\lg q=1$, $1-\lg q=0$.

Гармонический ряд расходится по только что приведенному признаку.

При увеличении точности слагаемых точность суммы не увеличивается.

Для гармонического ряда с показателем 2 $q = \sqrt{10}$ и $1-\lg q = \frac{1}{2}$. Т.е. при увеличении точности слагаемых на 2 знака число точных знаков увеличивается на 1 единицу.

По иному, чтобы найти сумму гармонического ряда с показателем 2 с некоторой точностью, точность слагаемых необходимо представлять в 2 раза большей точностью.

Правило суммирования. *Сначала установить начальное значение текущей суммы. Затем в цикле*

a) *найти текущее значение слагаемого по какому-нибудь правилу*
b) *округлить слагаемое с соответствующей точностью*
c) *увеличить текущее значение суммы на соответствующее значение слагаемого*
d) *цикл выполнять до тех пор, пока не исчерпаются все складываемые слагаемые*

Упражнения 2

Задание 1. *Найти сумму* $\sum_{n=1}^{\infty}\frac{1}{n^2}$

a) *С точностью до целых,*

b) *С точностью до* 5 *знаков после десятичной точки*,

c) *С большей точностью*.

С какой точностью необходимо представлять слагаемые и сколько при этом необходимо взять слагаемых, чтобы сумма находилась с соответствующей точностью?

Оцените характеристики скорости сходимости и числа точных знаков суммы.

$q = \lim \frac{n_{k+1}}{n_k}$, $1-\lg q$

Указание.

Задание 2. *То же самое для суммы* $\sum_{n=1}^{\infty}\frac{1}{n^3}$

Задание 3. *То же самое для суммы* $\sum_{n=1}^{\infty}\frac{1}{n^4}$

Задание 4. *То же самое для суммы* $\sum_{n=0}^{\infty}\frac{1}{n!} = 1 + \frac{1}{1} + \frac{1}{1\cdot 2} + \frac{1}{1\cdot 2\cdot 3} + \frac{1}{1\cdot 2\cdot 3\cdot 4} + \ldots$

Указание. Последний ряд отличается от предыдущих рядов способом нахождения общего члена. В предыдущих рядах общий член задавался определенной формулой. В данном примере он связан с факториальной функцией n!. Она определяется *рекурсивно*:

a) факториал 0 принимается равным 1:

0! = 1

b) факториал числа, отличного от 0, принимается равным произведению факториала предыдущего числа на само число:

$n! = (n - 1)!\, n.$

Изучим закономерность образования слагаемых.

Первое слагаемое равно 1. Это обстоятельство можно задать командами

n = 0: p = 1

Второе слагаемое получается из предыдущего (первого) делением его на 1. Это обстоятельство можно задать командами

n = n + 1: p = p / n

При выполнении этих команд в первый раз (предполагая выполненными предыдущие команды) значение n становится равным 1, и, следовательно, значение p получается из предыдущего его значения делением на эту полученную 1.

Третье слагаемое получается из предыдущего (второго) делением его на 2. Это обстоятельство также можно задать командами

n = n + 1: p = p / n

При выполнении этих команд во второй раз значение n становится равным 2, и значение p получается из предыдущего его значения делением на эту полученную 2.

Четвертое слагаемое получается из предыдущего (третьего) делением его на 3. Это обстоятельство также можно задать командами

n = n + 1: p = p / n

И т.д.

Таким образом, чтобы найти соответствующее слагаемое, необходимо команды **n = 0: p = 1** выполнить 1 раз, а команды **n = n + 1: p = p / n** – необходимое число раз.

Следующая программа суммирует приведенный ряд

```
Option Explicit
Private Sub Command1_Click()
Dim m As Byte            ‘Число знаков округления слагаемых
m = InputBox("Vvod chisla znakov okrug. m = ", "Vvod Ch. type Byte", 0)
Command1.Caption = "m = " + Str(m)
   Dim s As Double       ‘Текущая сумма
   Dim p As Double       ‘Текущее слагаемое
   Dim q As Double       ‘Округление слагаемых
   Dim n As Long         ‘Номера слагаемых
s = 0                        ‘Начальная инициализация
   n = 0: p = 1
   q = Round(p, m): s = s + q      ‘Сумма первого слагаемого
   Do                    ‘Цикл
   n = n + 1: p = p / n  ‘Общий член ряда
   q = Round(p, m): s = s + q      ‘Текущая сумма
  Loop Until q = 0       ‘Завершение цикла
  's = Round(s)
   Debug.Print " s = "; s; "m = "; m; "n = "; n
   End Sub
```

Задание 5. *Просуммировать ряды*

a) $\exp x = \sum_{n=0}^{\infty} \frac{x^n}{n!}$

b) $\cos x = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n}}{(2n)!}$

c) $\sin x = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{(2n+1)!}$

для значений x, водимых с клавиатуры, с точностями 0 *знаков,* 5 *знаков,* 15 *знаков. Сравнить полученные значения со стандартными функциями*.

Указание. Общие члены в соответствующем цикле можно задать командами

для $\exp x$:

n = n + 1: p = p * x / n

для $\cos x$ и $\sin x$:

n = n + 1: p = p * x / n: n = n + 1: p = p * x / n: p = -p:

Остается установить еще и начальные данные.

Задание 6. *Просуммировать ряды двумя способами:* ***аналитически*** *и на* ***ЭВМ***. *Сравнить результаты*. *Определить характеристики скорости сходимости. С какой точностью можно вычислить суммы рядов на ЭВМ? Аналитически? Сколько при этом необходимо взять слагаемых?*

a) $\sum_{n=1}^{\infty} \frac{1}{n(n+1)}$

b) $\sum_{n=1}^{\infty} \frac{1}{n(n+1)(n+2)}$

c) $\sum_{n=1}^{\infty} \frac{1}{n(n+1)(n+2)(n+3)}$

d) $\sum_{n=1}^{\infty} \frac{1}{(2n-1)(2n+1)}$

e) $\sum_{n=1}^{\infty} \frac{1}{(3n-2)(3n+1)}$

Задание 7. *Вычислите сумму* $\sum_{n=1}^{\infty} \frac{1}{n^2}$ *с большой точностью*.

Указание. Разложите $\frac{1}{n^2}$ на сумму $\frac{1}{n(n+1)} + r_3$. Остаток r_3 в свою очередь разложите на сумму $\frac{1}{n(n+1)(n+2)} + r_4$, и так далее. Воспользуйтесь результатами задания 6.

Задание 8. *Вычислите сумму* $\sum_{n=1}^{\infty}\frac{1}{n\sqrt{n}}$ *с возможно большей точностью. Сколько точных знаков удалось построить? Чему равна характеристика скорости сходимости*?

Глава 10. Векторы

Скалярное умножение векторов. Длина вектора. Нахождение косинуса угла и угла между векторами

Сложение векторов и умножение вектора на число

Под *вектором* в данном разделе понимается *упорядоченный* набор чисел, называемых *элементами* или *компонентами* вектора.

Элементы вектора *нумеруются*, удобно начинать нумерацию с 1. Таким образом, вектор характеризуется своим *первым* элементом, *вторым* элементом, и т.д., *последним* элементом. Число элементов вектора называют *размером* вектора. Размер вектора здесь будем обозначать через *nn*.

Числовые векторы на ЭВМ можно реализовать как *одномерные* массивы чисел соответствующих типов (см. раздел: *Массивы. Динамические массивы*).

Напомним: Под одномерным *массивом* понимается набор однотипных элементов – переменных, хранимых под одним и тем же именем – *идентификатором*. Массив характеризуется именем, индексами – номерами своих элементов, нижним и верхним индексами, типом элементов. Нижний индекс здесь принимается равный 1, а верхний индекс удобно описать как *именованную* константу, например:

Const nn = 5

К элементам массива обращаются по его индексам: записывается имя массива и в круглых скобках указывается искомый индекс, например,

a(1) – элемент массива *a* с индексом 1,

a(2) – элемент массива *a* с индексом 2,

Массивы-параметры подпрограмм описывают как *динамические* массивы.

Над векторами осуществляют различные операции.

Сложение векторов осуществляется по компонентам: первые их элементы складываются, результат сложения принимается за первый элемент суммы. Точно так же поступают и с остальными элементами.

Вычитание векторов также осуществляется по компонентам.

Умножение на число вектора: первый элемент вектора умножается на число и принимается за первый элемент произведения. Точно также поступают и с остальными элементами.

Умножение вектора на вектор не осуществляется в обычном его понимании.

Скалярное произведение двух векторов: соответствующие компоненты перемножаются и складываются, т.е. произведение первых компонент плюс произведение вторых компонент, плюс и т.д.

Длина вектора берется как корень квадратный из суммы квадратов элементов вектора – корень квадратный из скалярного квадрата вектора.

Косинус угла между двумя векторами – это отношение скалярного произведения векторов на произведение их длин.

Ввод и **вывод** векторов также можно отнести к операциям над векторами.

Над векторами осуществляются и **много других** операций. Например, нахождение **максимального** и **минимального** элементов, **перестановка** двух элементов *одного* вектора, **перестановка** элементов *двух* векторов, **сортировка** вектора по возрастанию или убыванию его элементов, **поиск** элемента среди элементов вектора, и т.д.

Ввод вектора

Для ввода вектора с клавиатуры удобно использовать диалоговую функцию **InputBox** и удобно описать в виде пользовательской процедуры. Она характеризуется своим заголовком

Sub VvodVectora(X() As Double, n As Long)

и двумя параметрами. Для фактического ввода описаны два массива **A** и **B** глобально на случай их использования в других подпрограммах. В обработчике **Command1_Click()** вводится массив **A**,

для наглядности и одновременно выводится на поле **Text1**. Аналогично можно ввести и другие фактические, вызывая процедуру ввода **VvodVectora**.

На форму установите кнопку **Command1** и текстовое поле **Text1**, свойство **MultiLine** установите на **True**, и установите вертикальную полосу прокрутки. Перекопируйте следующий фрагмент программы в модуль формы (из стандартного модуля поле Text1 невидимо) – программа ввода вектора **A** готова (*улучшите* программу).

Ввод вектора

```
Option Explicit
Const nn = 5     ‘Размер вектора. Для изменения размеров всех векторов достаточно изменить здесь
Dim A(nn) As Double     ‘Фактический параметр-вектор подпрограммы
Dim B(nn) As Double     ‘Параметр-вектор. Здесь не используется – используйте по своему усмотрению

Sub VvodVectora(X() As Double, n As Long)    ‘Подпрограмма ввода
Dim i As Long  ‘Счетчик элементов вектора - цикла
For i = 1 To n          ‘Цикл вводит элементы и выводит на Text1 в столбик
   X(i) = InputBox("Vvedy" + Str(i) + "-y el", "Vvod type Double", 0)
   Text1.SelText = Str(X(i)) + Chr(13) + Chr(10)
Next i
End Sub

Private Sub Command1_Click()
Text1.SelText = "Vector A" + Chr(13) + Chr(10) ‘Заголовок вводимого вектора
VvodVectora A, nn  ‘Вызов подпрограммы ввода вектора A.  Для ввода другого вектора необходимо указать его имя
End Sub
```

Подпрограмму **VvodVectora**(X() As Double, n As Long) можно использовать для ввода нескольких векторов, вызывая ее с соответствующим именем вводимого вектора, как например, осуществлено выше. Значение **n** означает верхний индекс вводимого вектора, в данном случае – константа **nn**.

Вывод вектора

Вектора можно выводить на *форму*, на Immediate Window, на текстовые поля Text и др. Вектор, по желанию, можно выводить в столбик или в строку. Вывод удобно организовать в виде формальной подпрограммы. Например,

Вывод вектора на поле Text1 *в столбик*:

```
Sub VyvodVectoraText1(X() As Double, n As Long)
Dim i As Long
For i = 1 To n
   Text1.SelText = Str(X(i)) + Chr(13) + Chr(10)
Next i
End Sub
```

Это формальная процедура вывода. Для фактического вывода необходимо вызвать формальную процедуру вывода заменой формальных параметров на фактические выводимые векторы, например,

VyvodVectoraText1 A, nn

Вывод вектора на метку Label1 *в строчку*:

```
Sub VyvodVectoraLabel1(X() As Double, n As Long)
Dim i As Long
‘Сначала вектор числовой преобразуется в строковой вектор в строчку:
Dim S As String
S = ""
```

```
For i = 1 To n
S = S + Str(X(i)) + " "
Next i
‘Строковой Вектор выводится на метку:
Label1.Caption = S
‘При попытке выводить каждый элемент по отдельности – предыдущий сотрется
End Sub
```

Вывод вектора на Picture1 *в столбик*, начиная с центра управляющего элемента:

```
Sub VyvodVectoraPicture1(X() As Double, n As Long)
Picture1.Scale (-10, 10)-(10, -10)     ‘Преобразование в масштабный прямоугольник
Picture1.PSet (0, 0)  ‘Установка курсора в центр прямоугольника
Dim i As Long
For i = 1 To n
   Picture1.Print X(i) ‘Вывод элемента и перевод курсора на следующую строку
Next i
End Sub
```

Команда **Picture1.Scale** (-10, 10)-(10, -10) преобразовывает элемент **Picture1** в масштабный прямоугольник, у которого левый верхний угол имеет координаты (-10, 10), а правый нижний угол – (10,-10).

Команда **Picture1.PSe**t (0, 0) переводит курсор в центр прямоугольника, а команда Picture1.Print X(i) выводит значение X(i), начиная с места курсора, а сам курсор переводит на следующую строку.

Для фактического вывода вектора необходимо вызвать формальную подпрограмму с указанием имени фактически выводимого вектора.

Операции над векторами

Упражнения 1

1. **Скалярное произведение двух векторов** – сумма произведений соответствующих компонент. Опишите подпрограмму Function нахождения скалярного произведения двух векторов. Назовите функцию: ScalarProduct.

2. **Длина вектора** – корень квадратный из суммы квадратов элементов вектора - корень квадратный из скалярного квадрата вектора (скалярное произведение вектора самого на себя). Назовите DlinaVectora.

3. **Косинус угла между двумя векторами** – отношение скалярного произведения векторов на произведение их длин. Опишите функцию.

4. **Введите с клавиатуры** два вектора. Найдите их скалярное произведение, их длины, косинус угла между ними.

5. **Найдите угол** между двумя векторами.

Указание. Для нахождения угла между векторами используется функция Арккосинус. Среди стандартных функций она отсутствием. Она вне интервала [-1, 1] не определена. Однако ниже описанной подпрограмме она продолжена и за пределы указанного интервала.

```
Function pi()
    pi = 4 * Atn(1)
End Function
```

```
Function ArcCos(x As Double) As Double
Dim w As Double
  If x = 0 Then
        w = pi / 2
  ElseIf x > 0 And x <= 1 Then
         w = Atn(Sqr(1 - x * x) / x)
  ElseIf x < 0 And x >= -1 Then
        w = pi + Atn(Sqr(1 - x * x) / x)
  ElseIf x > 1 Then
        w = 0
  ElseIf x < -1 Then
        w = pi
  End If
  ArcCos = w
End Function
```

6. **Максимальный элемент** вектора – опишите подпрограмму нахождения максимального элемента вектора, необходимо указать и индекс этого элемента.

7. **Минимальный элемент** вектора – опишите подпрограмму.

8. **Поменяйте местами** значения двух избранных элементов вектора, необходимо указать индексы выбранных элементов.

9. **Поменяйте местами** значения максимального и последнего элементов вектора.

10. **Минимальный и максимальный** элементы вектора – найдите. Поменяйте значения минимального элемента с первым, максимального с последним.

Суммирование векторов

Сумма двух векторов

Подпрограмма сумму вторых двух векторов присваивает первому вектору.

```
Sub SummaVectorov(z() As Double, x() As Double, y As Double, n As Long)
Dim i As Long
For i = 1 To n
   z(i) = x(i) + y(i)
Next
End Sub
```

Упражнения 2

1. С клавиатуры введите два вектора. Найдите их сумму. Результаты действий выведите на экран.

Указание. Фактические параметры-векторы обозначьте A, B, C. Векторы A, B введите с клавиатуры. Вызовите подпрограмму сложения векторов командой

SummaVectorov C, A, B, nn

из какого-нибудь обработчика события

2. Опишите подпрограмму умножения вектора на число. С клавиатуры введите вектор и число. Найдите произведение введенного вектора на введенное число. Выведите результат.

Сложение нескольких векторов

Алгоритм сложения векторов:

1. Переменная S сначала инициализируется:

Описывается подпрограмма

```
Sub NullVector( x() As Double, n As Long)
Dim i As Long
```

```
For i = 1 To n
   x(i) = 0
Next
End Sub
```

Команда

NullVector S, nn

инициализирует вектор S.

2. Вводится вектор p.
3. Команда

SummaVectorov S, S, p, nn

увеличивает вектор S на величину вектора p.

Глава 11. Сортировка и поиск

Встречаются многие задачи, где требуется среди данных элементов найти искомый элемент. Например, в словарях ищется слово, которое требуется перевести. Для этой цели данные элементы **сортируют**, сортируют либо по возрастанию, либо по убывания. Затем, используя бинарный **поиск,** находят нужный элемент.

Имеются много видов сортировок, которые носят имена их открывателей, например, метод Шелтона, метод Хоора и др. Среди них наиболее простой для первоначального знакомства является **Метод пузырька**. Здесь рассматриваются указанный метод, для ознакомления с другими методами рекомендуются соответствующие справочники.

Сортировка методом пузырька

рассмотрим на примере.

Пример 1. *Требуется отсортировать массив* 1, 3, 2, 9, 2, 4, 6, 5, 0 *по возрастанию*.

Решение.

Первый проход. *Первый шаг*. Сравним первый и второй элементы массива **1, 3**, 2, 9, 2, 4, 6, 5, 0. Если первый элемент больше второго – поменяем их местами, в противном случае оставляем без изменения. Массив примет вид 1, **3, 2**, 9, 2, 4, 6, 5, 0.

Второй шаг. Сравним второй и третий элементы массива и если второй элемент больше третьего, снова их поменяем местами. Массив примет вид 1, 2, **3, 9**, 2, 4, 6, 5, 0.

Последующие шаги. Такой процесс продолжается и дальше, сравниваем третий и четвертый элементы, четвертый и пятый элементы и т.д., пока не будут сравнены предпоследний и последний элементы. Последовательность массивов примет вид

1, 3, 2, 9, 2, 4, 6, 5, 0
1, **3, 2**, 9, 2, 4, 6, 5, 0
1, 2, **3, 9**, 2, 4, 6, 5, 0
1, 2, 3, **9, 2**, 4, 6, 5, 0
1, 2, 3, 2, **9, 4**, 6, 5, 0
1, 2, 3, 2, 4, **9, 6**, 5, 0
1, 2, 3, 2, 4, 6, **9, 5**, 0
1, 2, 3, 2, 4, 6, 5, **9, 0**
1, 2, 3, 2, 4, 6, 5, 0, **9**

После завершения первого прохода максимальный элемент 9 массива поднимается наверх, как и пузырек. Отсюда название метода.

Второй проход. Проделываем то же самое, что и в первом проходе. Последовательность массивов примет вид

1, 2, 3, 2, 4, 6, 5, 0, 9
1, **2, 3**, 2, 4, 6, 5, 0, 9
1, 2, **3, 2**, 4, 6, 5, 0, 9
1, 2, 2, **3, 4**, 6, 5, 0, 9

1, 2, 2, 3, **4, 6**, 5, 0, 9
1, 2, 2, 3, 4, **6, 5**, 0, 9
1, 2, 2, 3, 4, 5, **6, 0**, 9
1, 2, 2, 3, 4, 5, 0, **6, 9**
1, 2, 2, 3, 4, 5, 0, **6**, 9

Во втором проходе следующий наибольший элемент 6 всплывает наверх и занимает свое место в последовательности. И два наибольших элемента сортируются и занимают свои места в массиве.

Такие проходы осуществляются и далее. При третьем проходе сортируются три наибольших элемента

1, 2, 2, 3, 4, 0, **5, 6, 9** – третий проход.

Последовательность проходов можно представить в виде

1, 2, 2, 3, 0, **4, 5, 6, 9** – четвертый проход.
1, 2, 2, 0, **3**, **4, 5, 6, 9** – пятый проход.
1, 2, 0, **2**, **3**, **4, 5, 6, 9** –шестой проход.
1, 0, **2**, **2**, **3**, **4, 5, 6, 9** –седьмой проход.
0, **1, 2, 2, 3, 4, 5, 6, 9** –восьмой проход.
0, **1, 2, 2, 3, 4, 5, 6, 9** –девятый проход.

Для обмена значений двух элементов массива можно использовать процедуру

```
Sub Swap(x As Double, y As Double)
Dim z As Double
   z = x
   x = y
   y = z
End Sub
```

Для сравнения элементов массива используется условный оператор If:

```
If x(i) > x(i + 1) Then
   Swap x(i), x(i + 1)
End If
```

Для прохода элементов массива можно использовать цикл For … Next:

```
For i = 1 To n - 1
If x(i) > x(i + 1) Then
   Swap x(i), x(i + 1)
End If
Next i
```

Так осуществляется один проход.

Для осуществления всех проходов, пока массив не будет отсортирован, можно использовать цикл Do … Loop. Подпрограмма сортировки приобретает вид:

```
Sub Sortirovka(x() As Double, n As Long)
Dim Tru As Boolean
Dim i As Long
Do
      Tru = True
      For i = 1 To n - 1
      If x(i) > x(i + 1) Then
         Swap x(i), x(i + 1)
         Tru = False
      End If
      Next i
Loop Until Tru = True
End Sub
```

Булево выражение Tru принимает значение False, если происходит хотя бы один обмен между элементами массива. Цикл Do … Loop выполняется, пока выражение Tru не примет значение True, пока не будет осуществлен проход без единого обмена.

Упражнения 1.

1. Откройте новый проект, сохраните. Добавьте стандартный модуль. В стандартный модуль перекопируйте подпрограммы Swap и Sortirovka, описанные выше.

2. Введите с клавиатуры числовой массив и отсортируйте его.

3. Введите массив с помощью случайной функции Rnd, как целочисленный, так и с плавающей точкой, и отсортируйте его.

4. Опишите подпрограмму сортировки массива строк. Отсортируйте список фамилий студентов группы.

5. Опишите элементы массива в подпрограмме сортировки как тип Variant. Тогда эта подпрограмма сортирует как числовые массивы, так и строковые. Экспериментируйте. (Если при описании переменной опустить тип переменной, то по умолчанию переменная приобретает тип Variant, Экспериментируйте).

6. Опишите подпрограмму сортировки по убыванию

Сортировка методом вставки.

Дополнительное задание

Суть метода разъясним на примере.

Пример 1. *Требуется отсортировать массив* 1, 3, 2, 9, 2, 4, 6, 5, 0 *по возрастанию*.

Первый шаг. Сортируется один первый элемент массива – *первый* элемент остается на месте:

1, 3, 2, 9, 2, 4, 6, 5, 0

Выделяются отсортированные элементы.

Второй шаг. Вставляется *второй* элемент в отсортированную часть. Он сравнивается с *первым* отсортированным элементом – он больше его – элементы остаются на местах.

1, 3, 2, 9, 2, 4, 6, 5, 0

Третий шаг. Вставляется *третий* элемент.

Сравнивается со *вторым* элементом. Он меньше его – элементы меняются местами.

Сравнивается с *первым* элементом. Он больше его – элементы остаются на местах.

1, 2, **3**, 9, 2, 4, 6, 5, 0

Четвертый шаг. Вставляется *четвертый* элемент.

Сравнивается с *третьим* элементом. Он больше его – элементы остаются на местах.

1, 2, **3**, **9**, 2, 4, 6, 5, 0

Пятый шаг. Вставляется *пятый* элемент.

Сравнивается с *четвертым* элементом. Он меньше его – элементы меняются местами.

Сравнивается с *третьим* элементом. Он меньше его – элементы меняются местами.

Сравнивается со *вторым* элементом. Он больше (либо равно) его – элементы остаются на местах.

1, 2, **2**, **3**, **9**, 4, 6, 5, 0

Шестой шаг. Вставляется *шестой* элемент.

Шестой элемент меняется с *пятым* элементом.

1, 2, **2**, **3**, **4**, **9**, 6, 5, 0

И так далее.

На *последнем* шаге *девятый* элемент последовательно меняется с *восьмым, седьмым, и т.д. с первым* элементами.

Упражнения 2.

7. Опишите подпрограмму сортировки методом **вставки**.

Сортировка методом Шелла. *Дополнительное задание*

Суть метода разъясним на примере.

Пример 1. *Требуется отсортировать массив* 1, 3, 2, 9, 2, 4, 6, 5, 0 *по возрастанию*.

Выбираются числа 1, 2, 4, 8, 16, 32, …

Выбирается первое число среди них, которое превосходит число элементов рассматриваемого массива. Это 16. Делим число на 2. Получится 8.

Первый шаг Шелла. Сортируются каким-нибудь методом элементы массива, индексы которых отстоят на 8 единиц. Это номера 1, 9.

1, 3, 2, 9, 2, 4, 6, 5, **0** Выделены номера 1, 9. Других номеров, отстоящих на 8 единиц, нет.

0, 3, 2, 9, 2, 4, 6, 5, 1

Второй шаг Шелла. Сортируются элементы массива, индексы которых отстоят на 4 единицы. Это номера (1, 5, 9), (2, 6), (3, 7),.(4, 8):

0, 3, 2, 9, **2**, 4, 6, 5, **1** Выделены первая и вторая группы номеров.

0, 3, 2, 5, **1**, 4, 6, 9, **2**

Третий шаг Шелла. Сортируются элементы массива, индексы которых отстоят на 2 единицы. Это номера (1, 3, 5, 7, 9), (2, 4, 6, 8):

0, 3, **2**, 5, **1**, 4, **6**, 9, **2** Выделены – первая группа номеров, не выделены – вторая группа.

0, 3, **1**, 4, **2**, 5, **2**, 9, **6**

Последний шаг Шелла. Сортируются элементы массива, индексы которых отстоят на 1 единицу. Это все номера 1, …, 9. То есть идет полная сортировка.

0, 3, **1**, 4, **2**, 5, **2**, 9, **6** Частично отсортированный массив на предыдущих шагах.

0**,** 1, 2, 2, 3, 4, 5, 6, 9 Полностью отсортированный массив.

Упражнения 2.

8. Опишите подпрограмму сортировки методом **Шелла**.

Поиск бинарный

Бинарный поиск рассмотрим на примере.

Задача. В массиве **0**, **1, 2, 2, 3, 4, 5, 6, 9** найти число **7.**

Решение разбивается на два этапа.

Первый этап. Массив индексирован номерами 1, 2, 3, 4, 5, 6, 7, 8, 9. См. таблицу: нижний ряд – это индексы, верхний ряд – это их значения.

Обозначим: *a* – нижний индекс, *b* – верхний индекс, c = (a + b) \ 2 (целочисленное частное).

a = 1, b = 9, c = (1 + 9) \ 2 = 5.

0	1	2	2	3	4	5	6	9
1 = a	2	3	4	**5 = c**	6	7	8	**9 = b**

Индекс **c** разбивает массив элементов на два массива: *левый* и *правый*.

Сравниваем искомое число со значением элемента с индексом *c*. Если искомое число больше элемента с индексом c, то выбираем *правый* массив – полагаем **a = c** (b остается без изменения), в противном случае выбираем *левый* массив, полагаем **b = c** (a остается без изменения).

Такое сравнение продолжаем несколько раз – несколько раз, пока разность **b – a** не станет равной **1**.

Первое сравнение (Искомое число 7 > значения 3 элемента с индексом *c*) приобретает вид - выбирается правый массив:

0	1	2	2	3	4	5	6	9
1	2	3	4	**5 = a**	6	**7 = c**	8	**9 = b**

Второе сравнение (Искомое число 7 > значения 5 элемента с индексом *c*) приобретает вид – снова выбирается правый массив:

0	1	2	2	3	4	5	6	9
1	2	3	4	5	6	**7 = a**	**8 = c**	**9 = b**

Третье сравнение (7 > элемента 6 с индексом *c*) приобретает вид:

0	1	2	2	3	4	5	6	9
1	2	3	4	5	6	7	**8 = a**	**9 = b**

Разность **b – a** стала равной **1**. Дальнейшие сравнения не изменяют картину, так как (8 + 9) \ 2 = 8 (целочисленное частное). Поэтому сравнения на этом завершаются.

Второй этап. Далее проводятся *новые* сравнения:

Искомое число сравнивается с элементом массива с индексом *a*. Если совпадает – искомое число найдено в массиве элементов с индексом *a*.

В противном случае искомое число сравнивается с элементом с индексом b. Если совпадает – искомое число найдено.

В противном случае – искомое число отсутствует среди элементов массива.

```
Sub Poyisk(x() As Double, n As Long, L As Double, i As Long)
Dim a As Double
Dim b As Double
Dim c As Double
a = 1
b = n
Do
        c = (a + b) \ 2
        If L > x(c) Then
                a = c
        Else
                b = c
        End If
Loop Until b - a = 1

If L = x(a) Then
        i = a
        Debug.Print L; "-элемент массива с индексом"; i
ElseIf L = x(b) Then
        i = b
        Debug.Print L; "-элемент массива с индексом"; i
Else
        i = 0
        Debug.Print L; "-элемент отсутствует в массиве"
End If
End Sub
```

Глава 12. Матрицы

Под матрицей понимается прямоугольная *таблица*. Каждая матрица состоит из *строк* и *столбцов*. Они нумеруются двумерными индексами: первые номера – строки, а вторые – столбцы. Нумерацию строк и столбцов здесь будем начинать с 1. Если число строк и столбцов совпадает, то матрица называется *квадратной*. В этой работе рассматриваются квадратные числовые матрицы.

i – Переменная, в которую вводятся номера строк.

j – Переменная, в которую вводятся номера столбцов.

nn – Именованная константа – размер квадратной матрицы.

Матрица в языках программирования реализуется как **двумерный массив**. Так, что матрица и двумерный массив у нас будут служить как *синонимы*.

Пример описания квадратной числовой матрицы 5-го порядка:

```
Const nn = 5
Dim A(1 To nn, 1 To nn) as Double
```

Над матрицами осуществляют различные операции. Ввод и вывод матриц также относится к операциям над матрицами.

Ввод матрицы

Ниже приведенная программа вводит матрицу A. Для этой цели описана формальная подпрограмма ввода: **VvodMatrix(x() As Double, n As Long)**. x() – формальный параметр – матрица ввода, описан как динамический массив, параметр n - размерность матрицы. Для фактического ввода формальный параметр x() необходимо

заменить фактическим параметром, предварительно описав его размерность, см. вызов **VvodMatrix A, nn** в процедуре Command1_Click(). Для ввода другой матрицы необходимо вызвать подпрограмму ввода с указанием идентификатора другой матрицы.

Установите на форму элементы, указанные в процедуре Form_Load(), Установите свойство MultiLine поля Text1 на True, перекопируйте код программы в модуль формы (из стандартных модулей элемент Text1 формы не виден). Программа ввода матрицы готова.

Проект PrMatrix

```
Option Explicit
Option Base 1
Const nn = 3
Dim A(nn, nn) As Double

Sub VvodMatrix(x() As Double, n As Long)
Dim Title As String   ‘Заголовок окна ввода
Title = Str(nn) + "*" + Str(nn) + " Matrix"
Dim i As Long  ‘Нумерация строк матрицы
Dim j As Long  ‘Нумерация столбцов матрицы

For i = 1 To n
      For j = 1 To n
         x(i, j) = Val (InputBox("Vvedy" + Str(i) + Str(j) + "-y el", Title,
0))
         Text1.SelText = Str(x(i, j)) + " "      ‘Вывод строк матрицы
      Next j
      Text1.SelText = Chr(13) + Chr(10)     ‘Перевод строки
Next i
End Sub

Private Sub Command1_Click()
Text1.Text = "" ‘Очистка поля
```

```
Text1.SelText = "A Matrix" + Chr(13) + Chr(10) ‘Заголовок выводимой матрицы
VvodMatrix A, nn  ‘Вызов подпрограммы с указанием фактических параметров
End Sub

Private Sub Form_Load()  ‘Для инициализации
Command1.Caption = "Vvod A Matrix"       ‘Надпись на кнопке
Text1.Font.Size = 14  ‘Размер выводимого шрифта
End Sub
```

Вывод матрицы

В предыдущем пункте с вводом матрицы одновременно осуществлялось и вывод матрицы на текстовое поле Text1. Вывод можно осуществлять и на другие объекты. См. примеры вывода векторов, рассмотренные раньше.

Формальная подпрограмма вывода матрицы

```
Sub VyvodMatrix(x() As Double, n As Long, S As String)
S = ""
Dim i As Long
Dim j As Long
For i = 1 To n
        For j = 1 To n
        S = S + Str(x(i, j)) + " "
        Next j
         S = S + Chr(13) + Chr(10)
Next i
End Sub
```

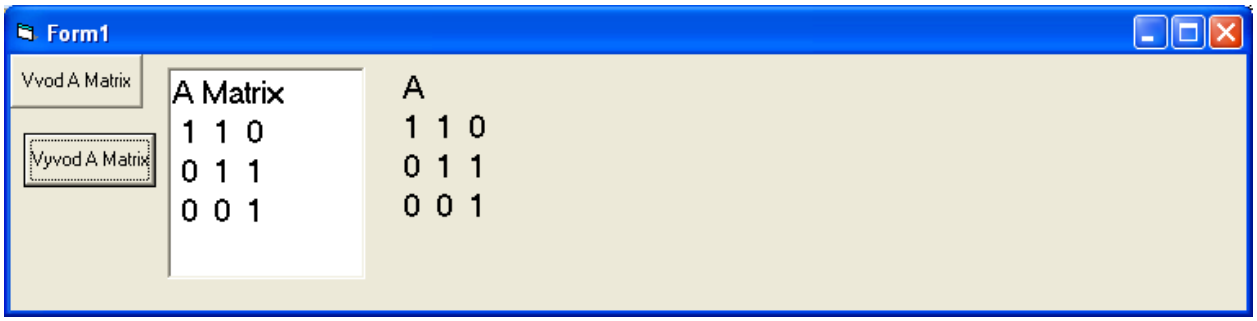
При вызове подпрограммы командой

VyvodMatrix A, nn, S

переменную S вводится матрица, преобразованная в строковой тип. Теперь можно вывести S на желаемом объекте. Например,

```
VyvodMatrix A, nn, S
S = "A" + Chr(13) + Chr(10) + S
```

Label1.Caption = S

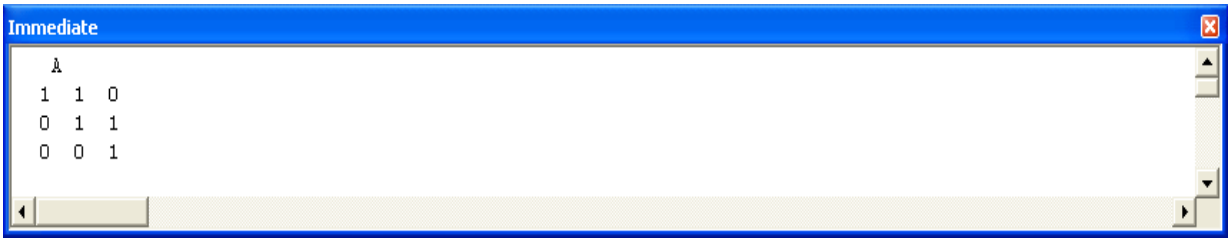


Или

VyvodMatrix A, nn, S
S = "A" + Chr(13) + Chr(10) + S
MsgBox S

Или

VyvodMatrix A, nn, S
S = "A" + Chr(13) + Chr(10) + S
Debug.Print S



Упражнения 1.

1. Экспериментируйте, вводя и выводя матрицы на различные объекты. Употребите собственную фантазию.

Заполнение матрицы случайными числами

Функция **Rnd** генерирует случайные числа, равномерно распределенные на открытом интервале (0, 1). Например, выражение 2 * Rnd генерирует – из интервала (0, 2), а выражение 1 + 2 * Rnd – из интервала (1, 3).

Если команду

```
n = Int(2 * Rnd)
```

выполнять многократно, то в переменную n будут вводиться числа 0 и 1 случайным образом - имитирует бросание игральной монеты. Появление их равновероятны.

Если же команду

```
n = Round(2 * Rnd)
```

выполнять многократно, то в переменную n будут вводиться числа 0, 1, 2 случайным образом. Но появления их не будут равновероятны. Число 1 будет появляться чаще, чем 0 или 1. *Проверьте и объясните*.

Команда

```
n =  Int(1+ 6 * Rnd)
```

имитирует бросание игральной кости шестигранной.

Чтобы случайные числа при каждом обращении начинались с новой серии, необходимо использовать служебное слово **Randomize** перед командами.

Упражнения 2.

1. Команда n = Round(10 * Rnd) выполняется 10000 раз. Посчитайте, сколько раз при этом появляются каждое из чисел из диапазона 0 … 10?

Задача. Заполнить матрицу случайными целыми числами из диапазона 0 … 9.

Решение. Задачу решает формальная подпрограмма.

```
Sub VvodMatrixRnd(x() As Double, n As Long)
Dim i As Long
Dim j As Long
Dim k As Long
Randomize
For i = 1 To n
        For j = 1 To n
                k = Int(10 * Rnd)
        x(i, j) = k
        Next j
Next i
End Sub
```

2. Наберите формальную подпрограмму в стандартном модуле. Вызовите ее из модуля формы, заменив формальные параметры фактическими. Выведите введенные матрицы несколько раз. Понаблюдайте.

Сложение двух матриц

Задача. Описать формальную подпрограмму, в которой сумма вторых двух матриц присваивается первой матрице. Введите две фактические матрицы, сложите их на основе описанной формальной подпрограммы, выведите результат.

Решение. Сложение матриц осуществляется по компонентам. Формально сложение матриц решает подпрограмма:

```
SummaMatrix(z() As Double, x() As Double, y() As Double, n As Long)
‘Сумма вторых двух матриц присваивается первой матрице‘
Dim i As Long  ‘Номера строк
Dim j As Long  ‘Номера столбцов
For i = 1 To n
For j = 1 To n
   z(i, j) = x(i, j) + y(i, j)     ‘Присвоение суммы элементов 2 матриц элементу третьей матрицы
Next j
Next i
End Sub
```

Упражнения 3.

1. Наберите формальную подпрограмму сложения матриц на стандартном модуле.
2. На модуле формы опишите 3 матрицы A, B, C с указанием их размерности.
3. Введите элементы матриц A, B (способом на свое усмотрение).
4. Выведите матрицу A на метке Label1, матрицу B – на метке Label2.
5. Вызовите подпрограмму сложения матриц из модуля формы командой

SummaMatrix C, A, B, nn

6. Выведите матрицу C на метке Label3.
7. Расположите метки наглядным образом

Сложение нескольких матриц

Алгоритм сложения

1. Сложение матриц начинается с начальной инициализации. Через S обозначим частичные суммы матриц. Ее необходимо обнулить. Для этой цели необходимо описать формальную подпрограмму обнуления матриц.
2. Опишите в стандартном модуле формальную подпрограмму обнуления матрицы с заголовком **Sub NullMatrix**(x() As Double, n As Long)
3. Опишите в стандартном модуле формальную подпрограмму ввода матрицы.
4. Одну из кнопок CommandButton используйте для обнуления матрицы – соответствующая надпись на кнопке.
5. Одну из кнопок CommandButton используйте для ввода матрицы A – соответствующая надпись на кнопке.
6. Сложения матриц осуществляется нижеприведенной командой – Значение матрицы S увеличивается на значение матрицы A, и снова присваивается матрице S. (Предполагается, подпрограмма **SumaMatrix** уже описана)

SummaMatrix S, S, A, nn

7. Одну из кнопок CommandButton используйте для сложения матриц – соответствующая надпись на кнопке. Записать команду из пункта 6.
8. В этой же кнопке организуйте вывод матрицы S.

Алгоритм вычисления сложений

1. Чтобы начать сложение, обнулить S – нажать на кнопку обнуления.
2. Ввести матрицу – нажать на кнопку ввода матрицы.

3. Сложить введенную матрицу – нажать на кнопку сложения.
4. Пункты 2 и 3 выполнять столько раз, сколько раз необходимо складывать. Чтобы начать новую серию сложений, необходимо снова начать с пункта 1 обнуления матрицы.

Упражнения 4.

1. Опишите проект сложения нескольких матриц. Сохраните проект с соответствующим именем.
2. Поэкспериментируйте, складывая несколько матриц.

Умножение двух матриц

Задача. Описать формальную подпрограмму, в которой произведение вторых двух матриц присваивается первой матрице.

Решение. Необходимо вспомнить алгоритм умножения матриц. Формально умножение матриц решает подпрограмма:

```
ProductMatrix(z() As Double, x() As Double, y() As Double, n As Long)
'Произведение вторых двух матриц присваивается первой матрице'
Dim i As Long   'Номера строк
Dim j As Long   'Номера столбцов
Dim k As Long 'Пробегает номера столбцов второй матрицы и строк третьей

For i = 1 To n
For j = 1 To n
    z(i, j) =0   'Начальная инициализация элемента первой матрицы
    For k = 1 to n   'Сумма произведений строки второй матрицы и столбца третьей
        z(i, j) = z(i, j) + x(i, k) * y(k, j)   'присваивается
    Next k      ' элементу первой матрицы
Next j
Next i
```

End Sub

Упражнения 5.

1. Наберите формальную подпрограмму умножения матриц на стандартном модуле.
2. На модуле формы опишите 3 матрицы A, B, C с указанием их размерности.
3. Введите элементы матриц A, B (способом на свое усмотрение).
4. Выведите матрицу A на метке Label1, матрицу B – на метке Label2.
5. Вызовите подпрограмму умножения матриц из модуля формы командой

ProductMatrix C, A, B, nn

6. Выведите матрицу C на метке Label3.
7. Можно использовать другие объекты для вывода матриц.

Как заметил читатель, умножение двух матриц отличается от сложения двух матриц формальной подпрограммой умножения.

Умножение нескольких матриц

Алгоритм умножения нескольких матриц отличается от алгоритма сложения матриц

1. Если при сложении матрица S частичных сумм инициализируется *обнулением*, то при умножении матрица P частичных произведений преобразовывается в единичную матрицу – матрицу, у которой по главной диагонали 1, а остальные 0.
2. Если сложение матриц осуществлялось командой **SummaMatrix** S, S, A, nn, то умножение матриц осуществляется коман-

дой **ProductMatrix** P, P, A, nn – матрица P умножается на матрицу A, результат присваивается матрице P.

3. Умножение матриц в общем случае не коммутативно. Команда **ProductMatrix** P, A, P, nn матрицу A умножает на матрицу P, результат снова присваивает матрице P.

Упражнения 6.

1. Опишите процедуру, которая матрицу преобразует в единичную матрицу. Наделите процедуру заголовком Sub OneMatrix(x() as Double, n as Long).
2. Опишите программу умножения нескольких матриц.

Избранные задания, связанные с матрицами

Упражнения 7.

1. Опишите процедуру умножения матрицы на вектор.
2. Опишите процедуру умножения матрицы на число.
3. Опишите процедуру деления матрицы на число.
4. Опишите процедуру возведения матрицы в степень с натуральным показателем.
5. Найдите матричную сумму

$$I + A + A^2 + \ldots + A^n$$

где I - единичная матрица, A – заданная матрица, n – заданное натуральное число, большее либо равное 0.

6. Найдите матричную сумму

$$I + \frac{A}{1!} + \frac{A^2}{2!} + \ldots + \frac{A^n}{n!}$$

7. Найдите бесконечную матричную сумму

$$\sum_{n=0}^{\infty} \frac{A^n}{n!} = I + \frac{A}{1!} + \frac{A^2}{2!} + \ldots + \frac{A^n}{n!} + \ldots$$

8. Найдите бесконечную матричную сумму

$$\sum_{n=0}^{\infty}(-1)^n \frac{A^{2n}}{(2n)!} = I - \frac{A^2}{2!} + \frac{A^4}{4!} - \frac{A^6}{6!} + \ldots$$

9. Найдите бесконечную матричную сумму

$$\sum_{n=0}^{\infty}(-1)^n \frac{A^{2n+1}}{(2n+1)!} = A - \frac{A^3}{3!} + \frac{A^5}{5!} + \ldots$$

10. Найдите сумму элементов матрицы
11. Найдите сумму абсолютных величин элементов матрицы
12. Найдите максимальный элемент матрицы – необходимо указать индексы максимального элемента.
13. Найдите минимальный элемент матрицы – необходимо указать индексы минимального элемента.
14. Поменяйте местами *i*-ую строку матрицы с *k*-ой строкой. Переменные *i, k* являются параметрами процедуры. *Такая операция является* ***элементарным*** *преобразованием матрицы*.
15. Умножьте *i*-ую строку матрицы на *число* и сложите к *k*-ой строке. Умножаемое число наряду с номерами строк являются параметрами процедуры. *Такая операция является* ***элементарным*** *преобразованием матрицы*.
16. **Получение нулей под элементом индексами (1, 1).** Если элемент матрицы с индексами (1, 1) – ненулевой, то умножьте первую строку на такое число и сложите ко второй строке, так чтобы элемент с индексами (2, 1) стал нулевым. Аналогично получите нули у элементов с индексами (3, 1) и так далее.

Глава 13. Графика в среде Visual Basic 6.0

Графику можно осуществлять на **форме** или на графическом элементе **PictureBox**. Для этого свойство **AutoReDraw** этих элементов необходимо установить на **True**. Графический экран представляет собой прямоугольник и состоит из точек, называемых Pixel-ями. Число точек по горизонтали составляет около 15000, по вертикали – около 10000. Изменяя форму можно изменять и число точек. Каждая точка имеет две декартовые координаты. Первая координата берется по оси абсцисс (направлена слева направо), вторая – по оси ординат (направлена сверху вниз). Точка на левом верхнем углу имеет координаты (0,0), на правом верхнем углу – около (15000,0), на левом нижнем углу – около (0,10000). Координаты других точек определяются с помощью вычислений. Таким образом, ось абсцисс направлена слева направо, а ось ординат – сверху вниз.

Есть понятие графический ***курсор***. По умолчанию он расположен на левом верхнем углу, т.е. имеет координаты (0,0).
Местоположение курсора можно изменять разными свойствами графических элементов:

Pset(x, y)	Установить курсор в точку с координатами (x,y)
Line (x1, y1) – (x2, y2)	Провести отрезок из точки (x1, y1) в точку (x2, y2), и установить курсор в точку (x2, y2).
Line – (x2, y2)	Провести отрезок из точки, где был расположен курсор, в точку (x2, y2), и

	установить курсор в точку (x2, y2).
Print S	Вывести надпись S, начиная из точки, где расположен графический курсор.
Cls	Очистить графический экран.
DrawWidth = m	Устанавливает ширину линии в m Пикселей
Scale (x1, y1) – (x2, y2)	Преобразовывает установленную форму или элемент PictureBox в прямоугольник, у которого (x1, y1) – это координаты левой верхней вершины, а (x2, y2) – координаты правой нижней вершины, координаты остальных точек прямоугольника находятся соответствующими вычислениями.

Здесь приведены только основные свойства графических элементов, с помощью которых удается изображать графики различных функций. Имеются и другие команды, с помощью которых можно изображать, например, окружности, прямоугольники, провести линию с определенным цветом, и т.д. (см. справочники).

Примеры записи команд:

Form1.Pset(100, 200)	Установить курсор в точку (100, 200) на форме Form1. Надпись Form1. допускается опускать
Picture1.Pset(100, 200)	Установить курсор в точку (100, 200) на элементе PictureBox1. Надпись Picture1. не допускается опускать
Form1.Line (100,200) – (200,400), vbRed	Провести отрезок красного цвета из точки (100, 200) в точку (200,400) и установить в конечную точку

Если будет выполнена инструкция

Form1.Scale (-10, 10) – (10, -10)

то левая верхняя вершина формы будет иметь координаты (-10, 10), правая верхняя вершина – (10, 10), левая нижняя вершина – (-10, -10), правая нижняя вершина – (10, -10).

Отрезок

Form1.Line (-10, 0) – (10, 0)

можно принять за горизонтальную ось абсцисс. Она направлена слева направо.

Отрезок

Form1.Line (0, -10) – (0, 10)

можно принять за вертикальную ось ординат. Она направлена снизу вверх.

Пересечение осей координат будет иметь координаты **(0, 0).**

Точки, координаты которых превосходят параметры шкалы **Scale**, не будут видимы на графике.

При изображении графиков кривых бывает удобным преобразовать форму в графический экран с помощью шкалы **Scale**, удачно выбирая при этом ее параметры. Этого можно добиться с помощью исследования кривой, либо экспериментально, выбирая различные параметры шкалы.

Задание. *Изобразить кривую* $f(x)=x^2-2$ *на отрезке* $[-10,10]$.

Решение

Функция задана явным уравнением. Ее удобно описать как подпрограмму-**Function**:

```
Function f(x As Double) As Double
f = x * x – 2       ‘Описание функции формулой
End Function
```

Для изображения графика кривой необходимо осуществить хотя бы минимальное исследование ее. Установим шкалу

Form1.Scale (-10, 10) - (10, -10)

Эта команда преобразует форму в прямоугольник. Левый верхний угол будет иметь координаты (-10, 10), правый нижний угол – (10, 10). Весь график функции не не поместится в этот прямоугольник, но на первый раз воспользуемся им.

При этом аргумент ***x*** функции будет изменяться от -10 до 10 и – используется цикл **For** … **Next.**

Для изображения графика можно использовать инструкцию **Line** – (x, y). При этом необходимо установить начало изображения кривой

```
x = -10: y = f (x): PSet (x, y)       'Ustanovka nacala krivoy
Цикл
For x = -10 To 10 Step 0.1 'Izobrajenie krivoy s shagom 0.1
   y = f (x)
    Form1.Line -(x, y), vbBlue
Next x
```

изображает кривую на отрезке от -10 до 10 голубым цветом. Step 0.1 – шаг изображения, его можно изменять.

Для изображения другой кривой инструкцию **f = x * x - 2** необходимо заменить уравнением изображаемой кривой. Уравнения некоторых кривых могут быть не определены в некоторых точках отрезка [-10, 10], и некоторые точки, желаемые для их изображения, могут не умещаться в масштабный прямоугольник. В этом случае необходимо внести соответствующие изменения в код программы.

Инструкции

Form1.**Line** (-10, 0) - (10, 0) 'gorizontalnaya os
Form1.**Line** (0, -10) - (0, 10) 'vertikalnaya os
изображают горизонтальную и вертикальную оси координат.

Инструкции

Form1.DrawWidth = 3 'Tolshina krivoy
For x = -10 **To** 10 Step 2 'Koordinatnaya setka
Form1.**PSet** (x, 0): Print x
Next x
Form1.DrawWidth = 1
устанавливают масштабную сетку по горизонтальной оси с шагом Step 2. Шаг можно изменять по желанию. Аналогично можно установить и вертикальную сетку.

Команда **Form1**.**DrawWidth** = 3 устанавливает толщину изображения в 3 Pixel. Команда **Form1**.**DrawWidth** = 1 отменяет предыдущую инструкцию. Если не отменять предыдущую инструкцию, то в дальнейшем все кривые изображались бы толщиной в 3 Pixel. Кривые по умолчанию изображаются в 1 Pixel.

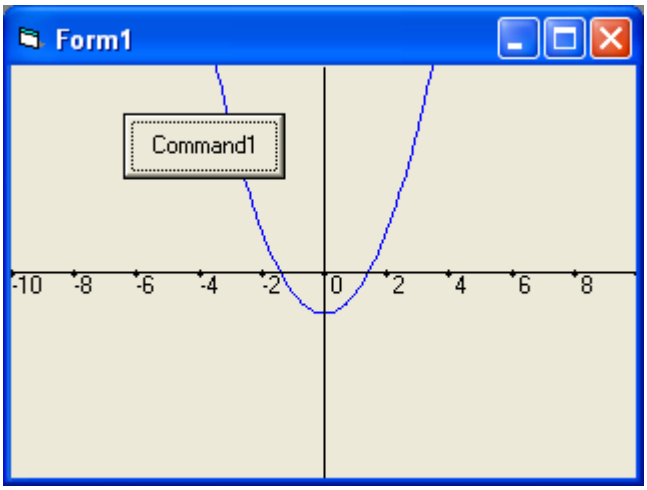
Проект Graphic Krivoy

Option Explicit
Function f(x As Double) As Double
f = x * x - 2
End Function

Private Sub Command1_Click()
Form1.Scale (-10, 10)-(10, -10) 'Scale Form1

Dim x As Double ‘Probegaet os abscis
Dim y As Double ‘Probegaet os ordinat
x = -10: y = f (x) 'Ustanovka nacala krivoy
Form1.PSet (x, y)
For x = -10 **To** 10 **Step** 0.1 'Izobrajenie krivoy
y = f (x)
Form1.Line -(x, y), **vbBlue**

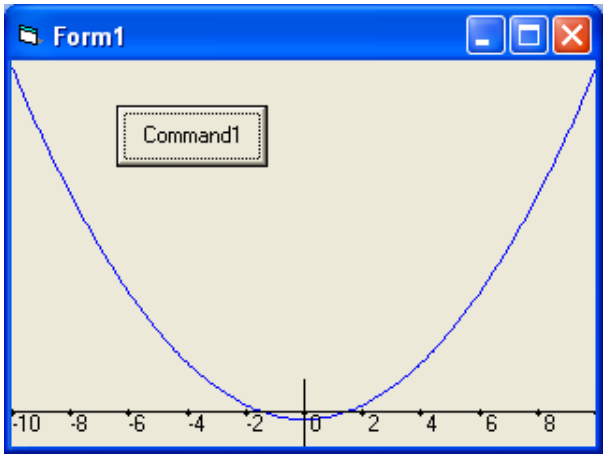
```
Next x
Form1.Line (-10, 0)-(10, 0) 'gorizontalnaya os
Form1.Line (0, -10)-(0, 10) 'vertikalnaya os
Form1.DrawWidth = 3          'Tolshina krivoy
For x = -10 To 10 Step 2     'Koordinatnaya setka
   Form1.PSet (x, 0)
   Form1.Print x
Next x
Form1.DrawWidth = 1          ‘Otmena predydushey tolshiny
End Sub
```



Некоторые точки не уместились в прямоугольник. Изменим масштаб

```
Form1.Scale (-10, 100)-(10, -10)      'Scale Form1
```

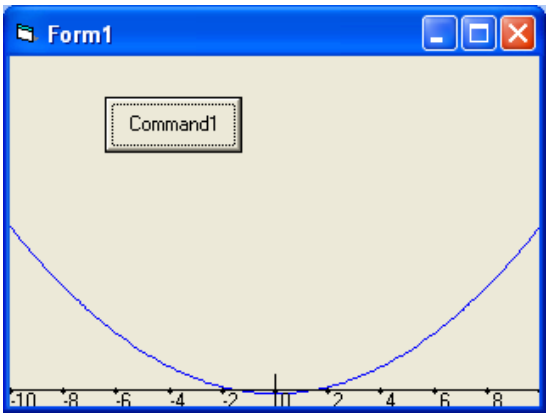
График приобретет вид



Изменим еще раз масштаб

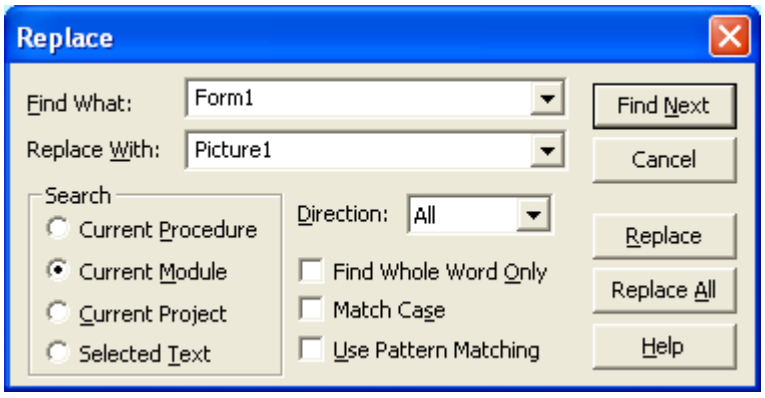
```
Form1.Scale (-10, 200)-(10, -10)      'Scale Form1
```

График приобретет вид

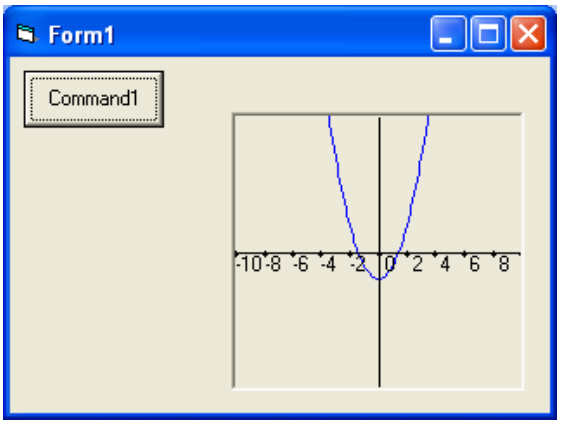


Упражнения 1.

1. Установите на форму элемент Picture1.
2. Вызовите окно Replace (замена) из меню Edit. В предыдущем проекте замените всюду Form1 на Picture1.



3. Программа готова, запустите ее на выполнение. Вы получите изображение кривой на Picture1.



4. Изобразите кривую на Picture2.
5. Построить графики линейной функции $y=k\,x$ для $k=0.1,$ 0.2, …, 1.

Select Case условный оператор. Изображение нескольких кривых

Упражнения 2.

Задание.

1. Откройте новый проект.
2. Добавьте стандартный модуль Module1. На него перекопируйте следующий код:

Проект 2 Graphic Krivoy

```
Option Explicit
Public n As Byte
Function g(x As Double, n As Byte) As Double
Select Case n
   Case 0
      g = x * x * x - 3 * x
   Case 1
      g = Cos(x)
   Case 2
      g = Sin(x)
   Case 3
      g = Tan(x)
   Case 4
      g = Atn(x)
   Case 5
      g = exp(x)
   Case Else
      g = x
End Select
End Function
Function f(x As Double) As Double
   f = g(x, n)
End Function
```

Выше представлен пример применения оператора **Select Case**. В приведенном примере выполняет одну из нескольких инструкций в зависимости от значения *n*.

При n = 0 выполняется оператор, следующий за Case 0.

При n = 1 выполняется оператор, следующий за Case 1.

И так далее.

При n = 5 выполняется оператор, следующий за Case 5.

Если *n* не принимает ни одно из приведенных значений, выполняется оператор, следующий за Case Else.

Описанные функции используются для описания функции $f(x)$. Если с клавиатуры ввести значение n = 0, то она становится равной, функции, следующей за Case 0, и так далее.

Задание продолжение.

3. Установите на форму элементы, указанные в процедуре Form_Load().
4. Перекопируйте в модуль формы ниже приведенный код. Программа изображения кривых, описанных выше, готова.

Проект 2 Graphic Krivoy - *продолжение*

```
Option Explicit
Dim aa As Double
Dim bb As Double
Private Sub Command1_Click()
   Dim x As Double
   Dim y As Double
   x = -aa: y = f(x)    'Ustanovka nacala krivoy
   Form1.PSet (x, y)
For x = -aa To aa Step 0.21  'Izobrajenie krivoy
   y = f(x)
    Form1.Line -(x, y), vbBlue
Next x
Form1.Line (-aa, 0)-(aa, 0) 'gorizontalnaya os
Form1.Line (0, -bb)-(0, bb) 'vertikalnaya os
Form1.DrawWidth = 3         'Tolshina krivoy
For x = -aa To aa Step 2        'Koordinatnaya setka
```

```
   Form1.PSet (x, 0): Form1.Print x
Next x
Form1.DrawWidth = 1
End Sub
Private Sub Command2_Click()
   Cls                        'Ochistka ekrana
End Sub
Private Sub Command3_Click()
n = Text1.Text
Command3.Caption = "Numer Functin" + Str(n)
End Sub
Private Sub Command4_Click()
aa = InputBox("aa =", "Vvod Scale", 10)
bb = InputBox("bb =", "Vvod Scale", 10)
Form1.Scale (-aa, bb)-(aa, -bb)     'Scale Form1
End Sub
Private Sub Form_Load()
Text1.Text = "0"
n = Text1.Text
Command1.Caption = "Graphic Krivoy"
Command2.Caption = "Cls"
Command3.Caption = "Numer Function" + Str(n)
Command4.Caption = "Scale"
End Sub
```

Литература

1. Абрамов С. А., Гнездилова Г. Г., Капустина Е. А., Селюн М. И. Задачи по программированию. М.: Наука, 1988
2. Дудкин Г. А., Егоров А. Н., Крупенина Н. В., Неклюдов С. Ю. Практикум по информатике. Санкт-Петербург. Практикум по информатике, 2001
3. Виктор Зиборов "Visual Basic 2010 на примерах" Издательство: БХВ-Петербург Год издания: 2010
4. В. Долженков, М. Мозговой "Visual Basic .NET. Учебный курс "Издательство: СПб.: Питер Год издания: 2003
5. Зеньковский В.А. Программирование на Visual Basic 6.5 и Visual Basic.Net. Москва: Солон-Пресс 2006
6. Никита Культин "Visual Basic освой на примерах " Издательство:BHV Год издания: 2004
7. Трусов М. А. "Visual Basic.NET Практическое руководство для начинающего программиста" Издательство: НТ Пресс Год издания: 2006
8. Учебно-образовательные интернет-ресурсы. method-kopilka.ru
9. Яндекс.Словари›Издательский словарь-справочник

СОДЕРЖАНИЕ

Салпагаров Ханафи Магометович,

Бостанова Мариям Магометовна,

Практикум решения задач на ЭВМ в среде Visual Basic 6.0

Учебно-методическое пособие

План университета 2016, поз. 6

Редактор Н.В. Ефрюкова
Корректор М.М. Бостанова
Компьютерная вёрстка и набор М.М. Бостанова

Подписано в печать
Бумага офисная
Формат 60x84/16.
Объем: 10,5 уч-изд. л.
Тираж 100 экз.

Издательство Карачаево-Черкесского государственного университета:
369202, г. Карачаевск, ул. Ленина, 29.
ЛР №040310 от 21. 10. 1997.