

Серия «Библиотека студента»

Г. З. Гарбер

Основы программирования на Visual Basic и VBA в Excel 2007

*Допущено Научно-методическим советом по информатике
при Министерстве образования и науки РФ в качестве
учебного пособия по дисциплине «Информатика»
для студентов высших учебных заведений*

**Москва
СОЛОН-ПРЕСС
2008**

УДК 681.3.06
ББК 32.973-01
Г20

Рецензенты:

М. И. Коробочкин, заведующий кафедрой информатики ГУЗа,
профессор

В. Б. Глаголев, профессор кафедры прикладной математики
МЭИ (ТУ)

Сведения об авторе

Геннадий Зеликович Гарбер — доктор технических наук, главный научный сотрудник ФГУП «НПП Пульсар», профессор кафедры информатики Государственного университета по землеустройству, участник международных конференций по информатике, включен в биографические справочники «Who's Who in the World» и «Who's Who in Science and Engineering».

Область интересов: математическое моделирование полупроводниковых приборов, интегральных схем и их межсоединений, а также разработка методики преподавания численных методов и программирования в Microsoft Office.

Гарбер Г. З.

Г20 Основы программирования на Visual Basic и VBA в Excel 2007. — М.: СОЛОН-ПРЕСС, 2008. — 192 с.: ил. — (Серия «Библиотека студента»)

ISBN 978-5-91359-003-9

Рассматриваются основные элементы среды разработки программ, а также базовые конструкции алгоритмических языков Visual Basic и VBA (Visual Basic для приложений). В качестве пользовательского интерфейса программы (макроса) используются стандартные окна Windows, текстовые файлы, формы, таблица Excel. Рассматривается также вопрос разработки пользовательских функций Excel. Продемонстрированы приемы работы с отладчиком программ, справочными системами, макрорекордером и личной книгой макросов. Опыт программирования не является обязательным.

Книга ориентирована на преподавателей информатики, студентов вузов и колледжей, а также на широкий круг читателей, имеющих желание научиться программировать в табличном процессоре Excel 2007 и тем самым существенно расширить круг решаемых задач.

В идеальном случае подход к разработке программы делится на три части: вначале получить ясное понимание задачи, потом выделить ключевые идеи решения и, наконец, записать решение в виде программы.

Бьярн Страуструп — автор C++

О книге

Примеры программ и описание способов их использования позволяют постепенно осваивать технологию программирования практически с нуля.

Автору удалось на конкретных примерах с минимальной порцией теоретического введения показать первые (и, надо сказать, основные) шаги технологии объектно-ориентированного программирования в среде Visual Basic.

*Из заключения Научно-методического
совета по информатике*

В настоящее время в курсах информатики для вузов предусматривается углубленное изучение системы Microsoft Office, в частности табличного процессора Excel, и изучение основ программирования на алгоритмических языках. Автор считает, что эти два вопроса можно объединить, если изучать программирование на Visual Basic, а в последней трети курса изучать разработку макросов Excel средствами VBA. Опыт, накопленный в ГУЗе за 6 лет, говорит о плодотворности такого подхода.

Книга написана кратко, хорошим языком.

М. И. Коробочкин, рецензент

Автора отличает оригинальность изложения материала: на первых же страницах он описывает методику использования отладчика программ среды Visual Basic, и далее этот отладчик используется как средство изучения конструкций алгоритмического языка.

В подборе материала книги сказался опыт автора как специалиста по математическому моделированию.

В. Б. Глаголев, рецензент

Содержание

Введение	6
1. Элементы интегрированной среды разработки программ VB	9
2. Основные команды отладки программ	15
3. Переменные. Типы данных	19
4. Две основные функции преобразования типов данных .	23
5. Константы.	25
6. Получение справки	27
7. Синтаксис оператора присваивания	31
8. Арифметические выражения	32
9. Математические функции	38
10. Логические выражения	41
11. Оператор перехода.	45
12. Конструкции принятия решений	46
13. Циклы	51
14. Проявление погрешности представления вещественных чисел	58
15. Массивы	61
16. Пользовательские процедуры	69
17. Использование стандартных окон операционной системы Windows	76
18. Записи	80
19. Работа со строками	84
20. Работа с текстовыми файлами	90
21. Формулировка демонстрационных задач.	96

22. Программа транспонирования матрицы относительно неглавной диагонали	98
23. Пользовательские формы	102
24. Отступление. Разработка программы с формой в автономной среде VB	115
25. Загрузка формы из окна Excel	117
26. Расположение элементов управления на листе Excel. .	119
27. Пользовательские функции Excel	122
28. Два метода разработки макросов	127
29. Макрорекодер Excel	128
30. Текст макроса на VBA и его редактирование	131
31. Объекты и события	133
32. Объект Application	137
33. Объекты Workbook, Workbooks и ActiveWorkbook . . .	144
34. Объекты Worksheet, Worksheets и ActiveSheet	149
35. Объекты Range, Selection и ActiveCell	152
36. Просмотр объектов	156
37. Использование таблицы Excel в качестве пользовательского интерфейса программ	159
38. Еще два макроса Excel. Личная книга макросов . . .	162
39. Еще две пользовательские функции Excel.	167
Заключение	172
Приложение 1. Типы данных Visual Basic	173
Приложение 2. Греческий алфавит	175
Приложение 3. Основные математические функции. . . .	176
Приложение 4. Рекомендации по темам контрольных работ .	178
Библиографический список	182
Предметный указатель	183

Введение

Решение какой-либо задачи с помощью компьютера требует ввода в него последовательности команд, которая называется программой. Программирование (т. е. написание программ) уже давно стало составной частью деятельности «продвинутых» пользователей компьютеров.

Программы пишутся на том или ином языке, понятном компьютеру. Такой язык называется алгоритмическим. К настоящему времени разработано множество алгоритмических языков: С, С++, Pascal, Object Pascal и др.

Мы будем изучать Visual Basic (VB). Выбор языка объясняется тем, что VB используется в популярной системе Microsoft Office, в частности, в ее последней версии 2007 года, сильно отличающейся от предыдущих версий.

Visual Basic — штатный язык программирования системы Microsoft Office, в состав которой входит табличный процессор Excel.

Макросом (или макрокомандой) Excel называется последовательность операций, снабженная именем, которая может быть выполнена автоматически. Программа, написанная на VB в Excel, удовлетворяет этому определению, поэтому ее можно назвать макросом.

Изучение данной книги возможно лишь в процессе работы на компьютере, оснащенном табличным процессором Excel 2007. Книга рассчитана на читателя, уже имеющего опыт работы с Excel, а также с программой Проводник (Windows Explorer) и текстовым процессором Word, причем читатель должен уметь пользоваться буфером обмена операционной системы Windows. Опыт программирования не является обязательным.

В книге рассмотрены базовые конструкции, имеющиеся во всех алгоритмических языках, — оператор присваивания, условные операторы, циклы, массивы и др. Кроме того, рассмотрены вопросы разработки макросов и пользовательских функций Excel средствами макрорекордера и алгоритмического языка VBA (Visual Basic

for Applications — VB для приложений), являющегося расширением Visual Basic: помимо всех основных конструкций VB, язык VBA включает ряд специфических конструкций.

В качестве пользовательского интерфейса программы (макроса) используются стандартные окна операционной системы Windows, текстовые файлы, формы, таблица Excel.

Пользовательским интерфейсом программы называется средство общения пользователя с программой. Форма — это прямоугольник с текстовыми полями, кнопками, переключателями и другими элементами управления, знакомыми пользователям Microsoft Office.

При изучении VB с помощью данной книги рекомендуем тщательно прорабатывать на компьютере приведенные в ней программы. Рекомендуем также выполнять все задания и чаще использовать такие мощные средства обучения, как отладчик программ VB, макрорекордер Excel и справочные системы Excel и VB.

При изучении материала данной книги одна и та же программа может использоваться несколько раз. Поэтому советуем сохранять в дисковой памяти компьютера тексты программ, набираемых на клавиатуре в так называемом окне кода. Для этого надо:

- в окне кода выделить текст программы, как в текстовом процессоре Word;
- скопировать его в буфер обмена, например, нажав клавиши **Ctrl + C**, т. е. одновременно нажав клавиши **Ctrl** и **C**;
- вставить текст программы из буфера обмена в документ Word, например, нажав клавиши **Ctrl + V**;
- сохранить документ в дисковой памяти компьютера.

Аналогичным образом текст программы переносится обратно, из документа Word в окно кода.

Для сохранения программы достаточно также сохранить книгу Excel, в которой эта программа содержится, как это описано на стр. 101.

Для лучшего запоминания обозначений VB в книге приведен перевод отдельных английских слов на русский язык (при этом автор в основном руководствовался терминологией Excel 97, в котором Visual Basic частично русифицирован). При работе со справочными системами Excel и VB советуем использовать сайт www.translate.ru или англо-русский электронный словарь ABBYY Lingvo.

При обозначении последовательности операций иногда используются многоточие и символ > в смысле «далее», например, **Office > Печать > ...** Это выражение означает, что надо щелкнуть (кликнуть) левой кнопкой мышки на кнопке **Office**, далее на пункте **Печать** открывшегося списка и затем выполнить очевидные операции.

Вместо «кликнуть левой кнопкой мышки» часто будем использовать один глагол «кликнуть».

В приложении 1 приведены типы данных VB.

В приложении 2 приведен греческий алфавит с русскими и английскими названиями букв. Наличие этого приложения оправдано тем, что в VB нет символов (литер) греческих букв.

В приложении 3 приведены основные математические функции VB и операторы, позволяющие рассчитывать значения ряда функций, которых нет в VB.

В приложении 4 приведен ряд заданий, которые можно рассматривать как рекомендации для преподавателей по темам контрольных работ. Эти задания можно также использовать для самопроверки при самостоятельном изучении материала книги.

В разделе «Библиографический список» перечислены справочные системы и книги, использованные автором.

В разделе «Предметный указатель» приведены основные термины и обозначения с указанием номеров страниц, на которых раскрывается их смысл. Это позволяет использовать книгу в качестве справочника.

Как следует из названия книги, автор не стремился к полноте изложения материала по программированию на Visual Basic и VBA в Excel 2007. Однако материала книги достаточно для разработки пользовательских функций и макросов Excel, позволяющих производить сложные вычисления.

При написании книги использовался компьютер, работающий под управлением операционной системы Microsoft Windows XP.

Связаться с автором можно с помощью электронной почты: g.garber@g23.relcom.ru; ggarber@mail.ru.

1. Элементы интегрированной среды разработки программ VB

Написание и отладка программ производится в интегрированной среде разработки программ Visual Basic, входящей в состав системы Microsoft Office, которую в дальнейшем будем называть средой Visual Basic или просто Visual Basic.

Под отладкой программы понимается выявление и исправление ошибок, которые, как правило, присутствуют в только что написанном тексте (коде) программы.

Для входа в среду VB используется вкладка **Разработчик** окна Excel, которая находится в одном ряду с вкладками **Главная**, **Вставка** и др. При ее отсутствии следует выполнить такие операции:

- нажать кнопку **Office** (в левом верхнем углу), кликнув на ней;
- нажать кнопку **Параметры Excel**;
- в открывшемся окне *Параметры Excel* нажать кнопку **Основные**;
- в разделе *Основные параметры работы с Excel* включить опцию **Показывать вкладку «Разработчик»** на ленте, поставив соответствующую галочку;
- нажать кнопку **ОК**.

При этом в окне Excel появится вкладка **Разработчик**.

В среду Visual Basic войдем следующим образом:

- в окне Excel (с книгой) активизируем вкладку **Разработчик**, кликнув на ней;
- выполним команду **Visual Basic**, кликнув на соответствующей пиктограмме.

Эту последовательность операций можно записать так: **Разработчик > Visual Basic**.

В результате откроется окно *Microsoft Visual Basic* (рис. 1.1), в котором можно выполнять различные действия: ввод и редактирование текста программы, а также ее отладку и выполнение. (Вы-

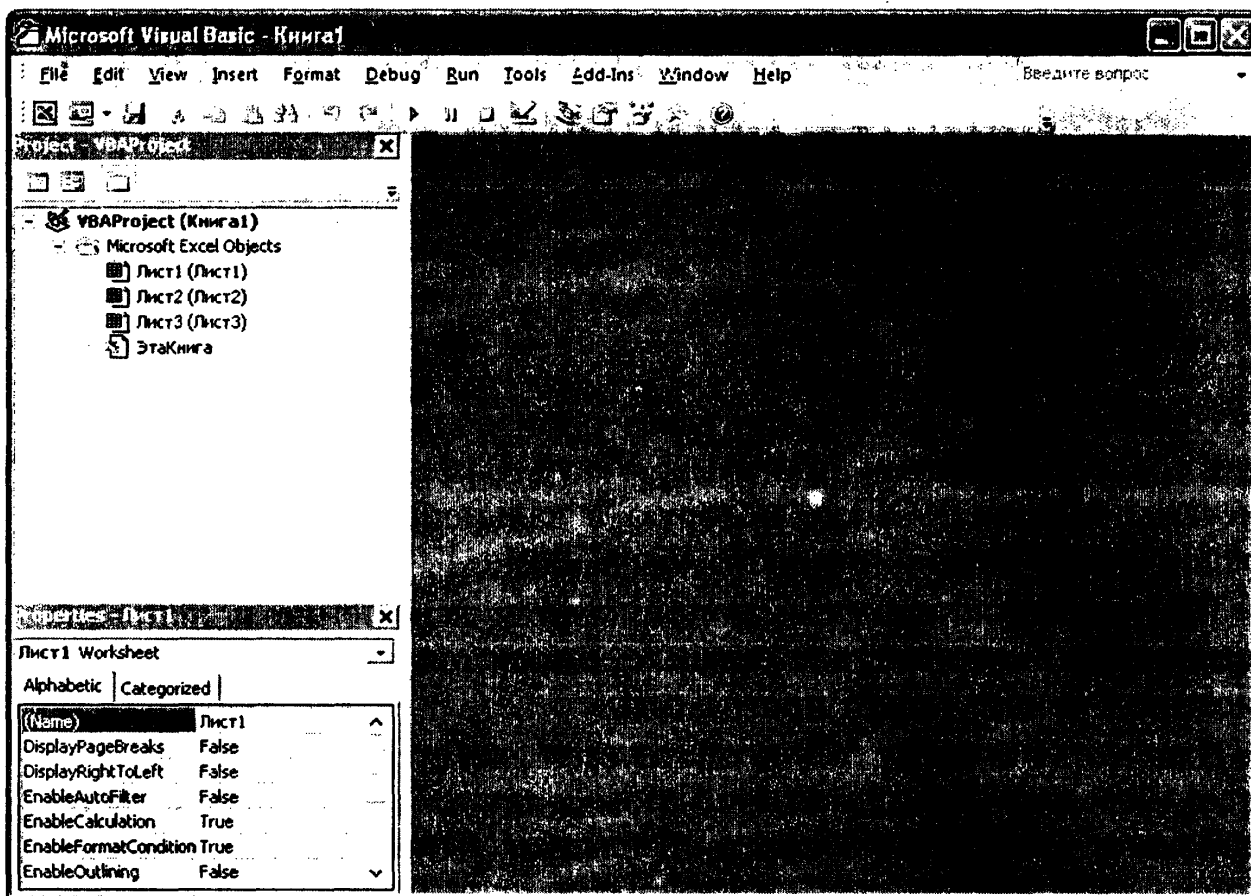


Рис. 1.1. Окно VB со стандартной панелью инструментов и окнами проекта и свойств

полнять программу можно и находясь в окне Excel, но об этом позже.)

Программу еще называют приложением, проектом или макросом. Она находится в книге Excel.

Рассмотрим основные элементы среды Visual Basic.

1. Строка меню. В ее состав входят все меню, которые используются при создании программы. Среди меню есть стандартные, присущие многим окнам Windows: **File** (Файл), **Edit** (Правка), **View** (Вид), **Tools** (Сервис), **Help** (Помощь) и др. Имеются также меню, при помощи которых пользователь компьютера может создавать, запускать и отлаживать программу: **Insert** (Вставка), **Run** (Запуск), **Debug** (Отладка) и др.

2. Контекстное меню. Служит для удобства выполнения действий, относящихся к той области окна VB, в которой находится указатель мышки. Чтобы вызвать контекстное меню, надо поместить указатель мышки в нужную область экрана и кликнуть пра-

вой кнопкой мышки. После этого следует кликнуть левой кнопкой мышки на нужной команде открывшегося списка.

3. Панели инструментов: *Standard* (Стандарт), *Edit* (Правка), *Debug* (Отладка) и др. По умолчанию отображается только стандартная панель инструментов *Standard* (Стандарт). Для того чтобы добавить или удалить какую-либо панель, следует выполнить **View (Вид) > Toolbars (Панели инструментов)** и кликнуть на нужной команде открывшегося списка. Галочка свидетельствует о наличии панели на экране.

Рассмотрим панели инструментов среды VB.

1. Панель *Standard* (Стандарт) является основной. С ее помощью можно выполнять широкий спектр действий. Она обычно расположена под строкой меню, однако при помощи мышки ее можно перетаскивать в другие области окна VB.

2. Панель *Edit* (Правка) предназначена для работы с текстом программы. Эта панель реализует возможности простейшего текстового редактора: копирование и перемещение текста в буфер обмена, вставку текста из буфера, поиск и/или замену слов и фраз в тексте программы и т. д.

3. Панель *Debug* (Отладка) предназначена для отладки программы. Предусмотрены широкие возможности для отладки: отслеживание текущих значений переменных программы, пошаговое выполнение программы, при котором на каждом шаге выполняется один оператор (statement, инструкция, команда, конструкция) или его часть, и т. д.

В книге в основном рассматриваются программы без формы в качестве пользовательского интерфейса. Разработка такой программы начинается со вставки модуля VB в активную книгу Excel. Для этого выполним следующую последовательность операций.

1. В окне Excel выполним **Разработчик > Visual Basic**. В результате появится окно VB (рис. 1.1), включающее в себя окна проекта (*Project Explorer*) и свойств (*Properties Window*).

Если окно проекта отсутствует, то для его вывода на экран следует кликнуть на соответствующем пункте меню **View (Вид)**. Окно свойств нам потребуется только в главе 23.

2. В окне проекта кликнем на строке **VBAProject (Книга1)**, где Книга1 — имя активной книги. При этом названная строка выделится.

3. Выполним **Insert (Вставка) > Module (Модуль)**. При этом изменится содержимое окна проекта — появится строка, соответ-

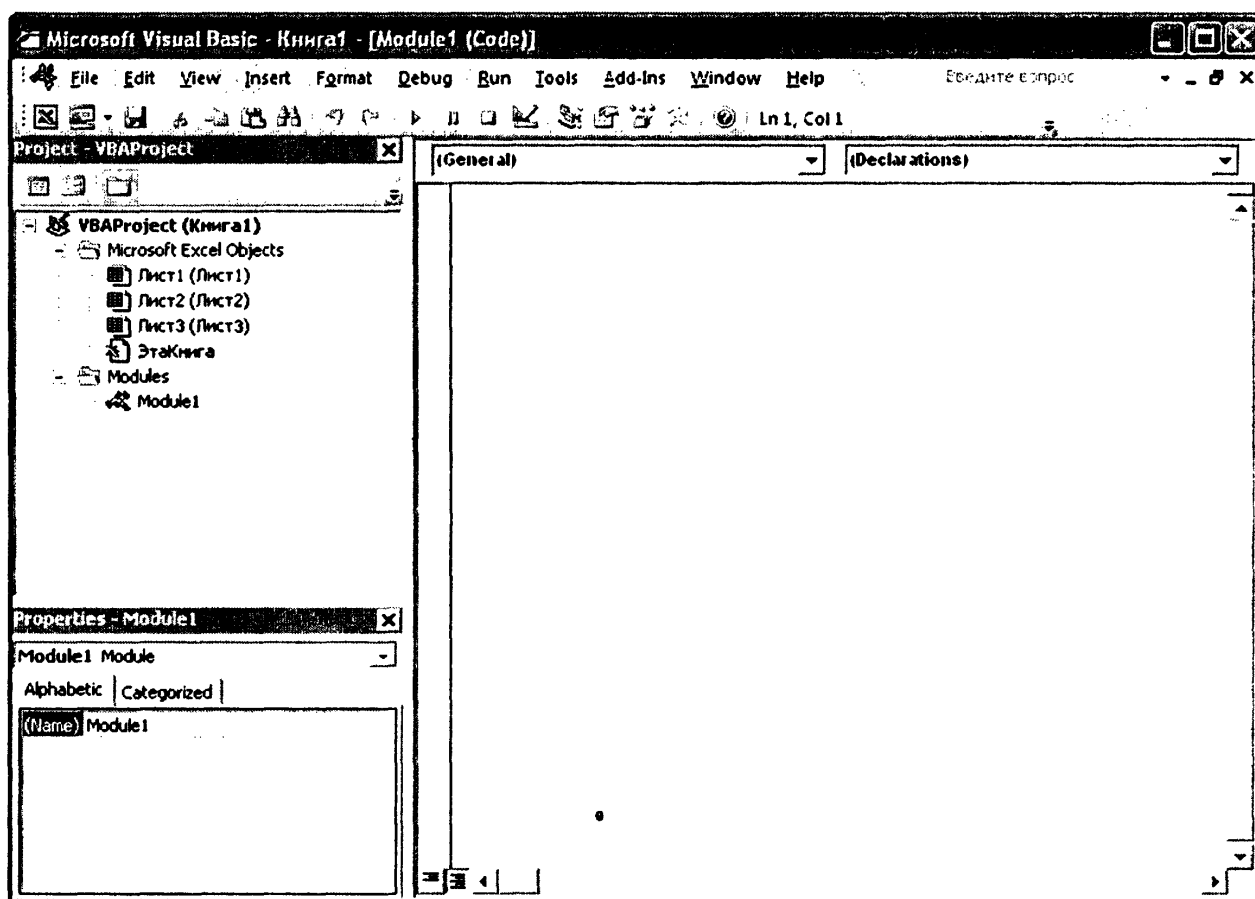


Рис. 1.2. Состояние окна VB после вставки модуля в книгу Excel

вующая вставленному модулю Module1. Кроме того, откроется пустое окно, которое называется окном кода, соответствующим модулю Module1 (рис. 1.2). В окне набирается текст (код) программы.

Для открытия окна кода, соответствующего модулю, вставленному ранее, надо дважды кликнуть на имени этого модуля в окне проекта.

Для удаления модуля следует выполнить такие операции:

1) кликнуть правой кнопкой мышки на имени этого модуля в окне проекта, например, на **Module1**;

2) в открывшемся контекстном меню выполнить команду **Remove (Удалить)**;

3) кликнуть на кнопке **No (Нет)** в открывшемся окне с вопросом, экспортировать модуль перед удалением или нет.

Как уже говорилось, в окне кода набирается текст программы. Первая и последняя строки (операторы) программы стандартные:

```
Sub имя ()
End Sub
```

Имя программы *имя* назначается ее разработчиком. Оно должно удовлетворять следующим условиям:

- первый символ имени должен быть буквой;
- имя может содержать только буквы, цифры и символ соединения `_`;
- имя не должно содержать более 255 символов.

Подчеркнем, что в имени программы нельзя использовать пробел. Если мы хотим, чтобы имя программы состояло из нескольких слов, то вместо пробела надо использовать знак соединения или каждое слово начинать с заглавной буквы. Примеры имени программы:

```
моя_программа
МояПрограмма
MyProgram
My_Program
```

Между первой и последней строками набираются остальные строки (операторы) программы. При этом можно пользоваться привычными командами редактирования (как в текстовом процессоре Word), а также буфером обмена. Ввод строки оканчивается нажатием клавиши **Enter**.

В качестве примера рассмотрим программу расчета длины гипотенузы прямоугольного треугольника с катетами $a = 3$ и $b = 4$, основанную на теореме Пифагора.

В окно кода введем (с клавиатуры) следующий текст программы:

```
Sub Пифагор()
    a = 3
    b = 4
    c = Sqr(a ^ 2 + b ^ 2)
End Sub
```

Слово Пифагор набрано русскими буквами; все остальные буквы — английские (рис. 1.3).

Видно, что операция возведения в степень выглядит, как в Excel; `Sqr` — функция извлечения квадратного корня, как в английской версии Excel (в русской версии Excel функция извлечения квадратного корня называется **КОРЕНЬ**).

Если для наглядности оператор следует разместить на нескольких строках, то для переноса следует использовать символ пробела

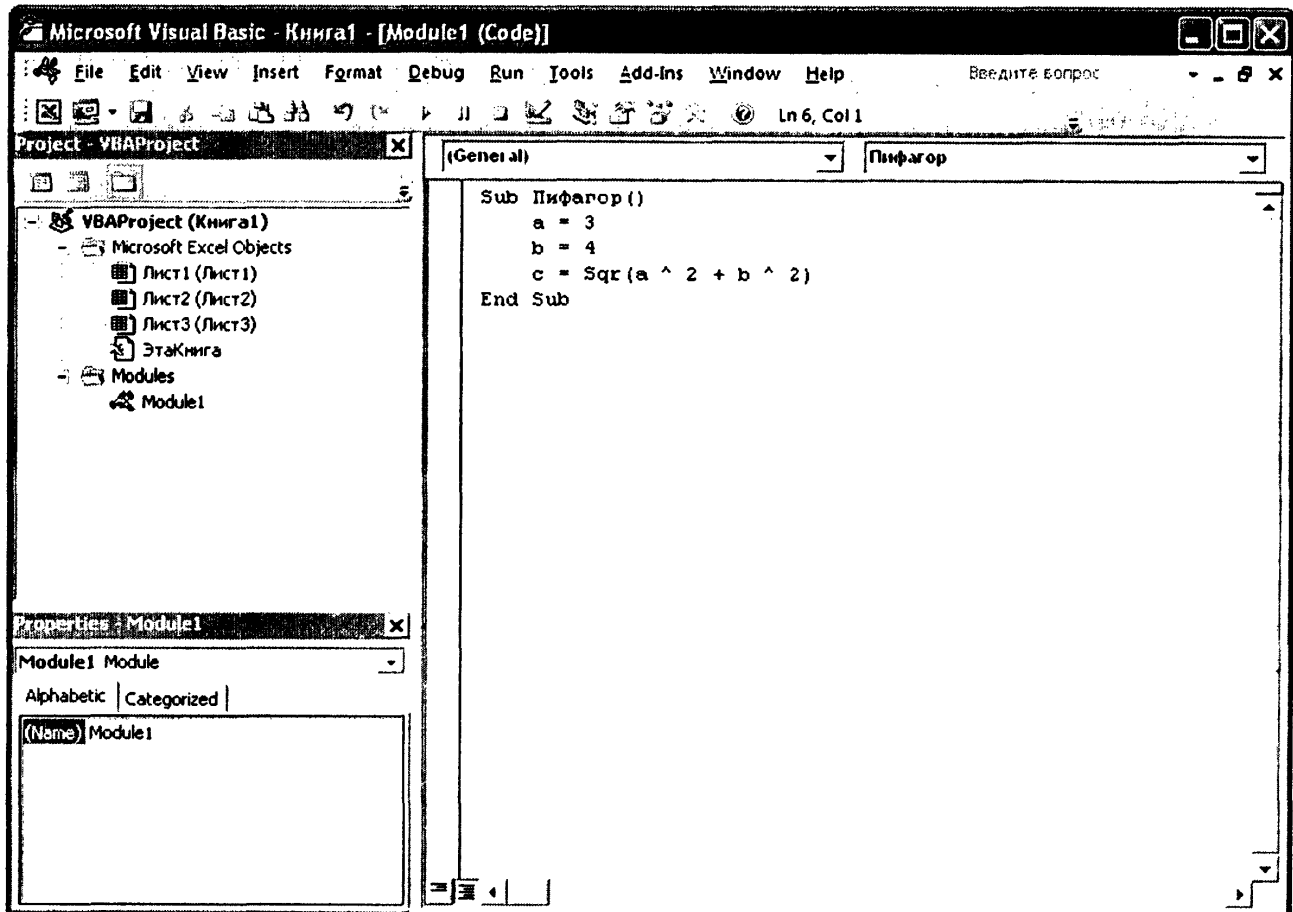


Рис. 1.3. Окно VB с программой Пифагор в окне кода

с последующим символом соединения `_` (после ввода этих символов надо нажать клавишу **Enter**). Если несколько операторов следует разместить на одной строке, то между ними надо поставить двоеточие и символ пробела.

Апостроф (запятая в верхней части строки) означает, что следующая за ним информация (до конца строки) является комментарием, т. е. набором символов, который никак не влияет на выполнение программы.

Таким образом, нашу программу можно записать в виде

```
Sub Пифагор()
    a = 3: b = 4
    c = _
        Sqr(a ^ 2 + b ^ 2) 'согласно теореме Пифагора
End Sub
```

В следующей главе мы произведем расчет по программе Пифагор.

2. Основные команды отладки программ

После набора текста программы следует этап отладки, т. е. выявления и исправления ошибок в программе. При этом используется отладчик.

Команды отладчика можно увидеть, если в среде VB открыть меню **Debug (Отладка)**. Рассмотрим основные команды отладчика.

1. **Step Into (Шаг с заходом)** — выполнение одного оператора программы или его части. Эта команда также выполняется при нажатии клавиши **F8**. Ее можно использовать для пошагового выполнения программы.

2. **Run To Cursor (Выполнить до текущей позиции)** — выполнение программы до мигающего курсора. (Для установки мигающего курсора нужно кликнуть в нужном месте программы.) Рассматриваемая команда выполняется также при нажатии клавиш **Ctrl + F8**.

3. **Toggle Breakpoint (Точка останова)** — установка или ликвидация точки останова. Такой точкой отмечается строка в программе, перед которой выполнение программы временно прекращается. Установка или ликвидация точки останова производится в том месте, где находится мигающий курсор. Эта команда также выполняется при нажатии клавиши **F9**.

Для установки или ликвидации точки останова можно кликнуть на серой левой границе окна кода напротив нужной строки.

4. **Clear All Breakpoints (Снять все точки останова)** — ликвидация всех точек останова.

5. **Add Watch (Добавить контрольное значение)** — визуализация текущих значений выбранных переменных. Методика использования этой команды приведена на стр. 64 и 65.

Следующие две команды входят в состав меню **Run (Запуск)**.

1. **Run (Запуск)** — запуск программы и переход от одной точки останова к другой. Если точек останова нет, то программа выполняется полностью. Эта команда изображена стрелкой на панелях инструментов, в частности на стандартной панели.

2. **Reset (Сброс)** — прекращение выполнения программы. Эта команда изображена квадратом на панелях инструментов.

При остановках во время выполнения программы стрелкой слева и желтым цветом выделяется оператор, который еще не выполнен. Значение той или иной переменной можно увидеть, подведя к ней указатель мышки.

Рассчитанное значение переменной *c* в программе Пифагор можно увидеть с помощью отладчика. Для этого выполним следующие операции.

1. Отметим точкой останова последнюю строку программы, например, кликнув на серой левой границе окна кода напротив этой строки (рис. 2.1).

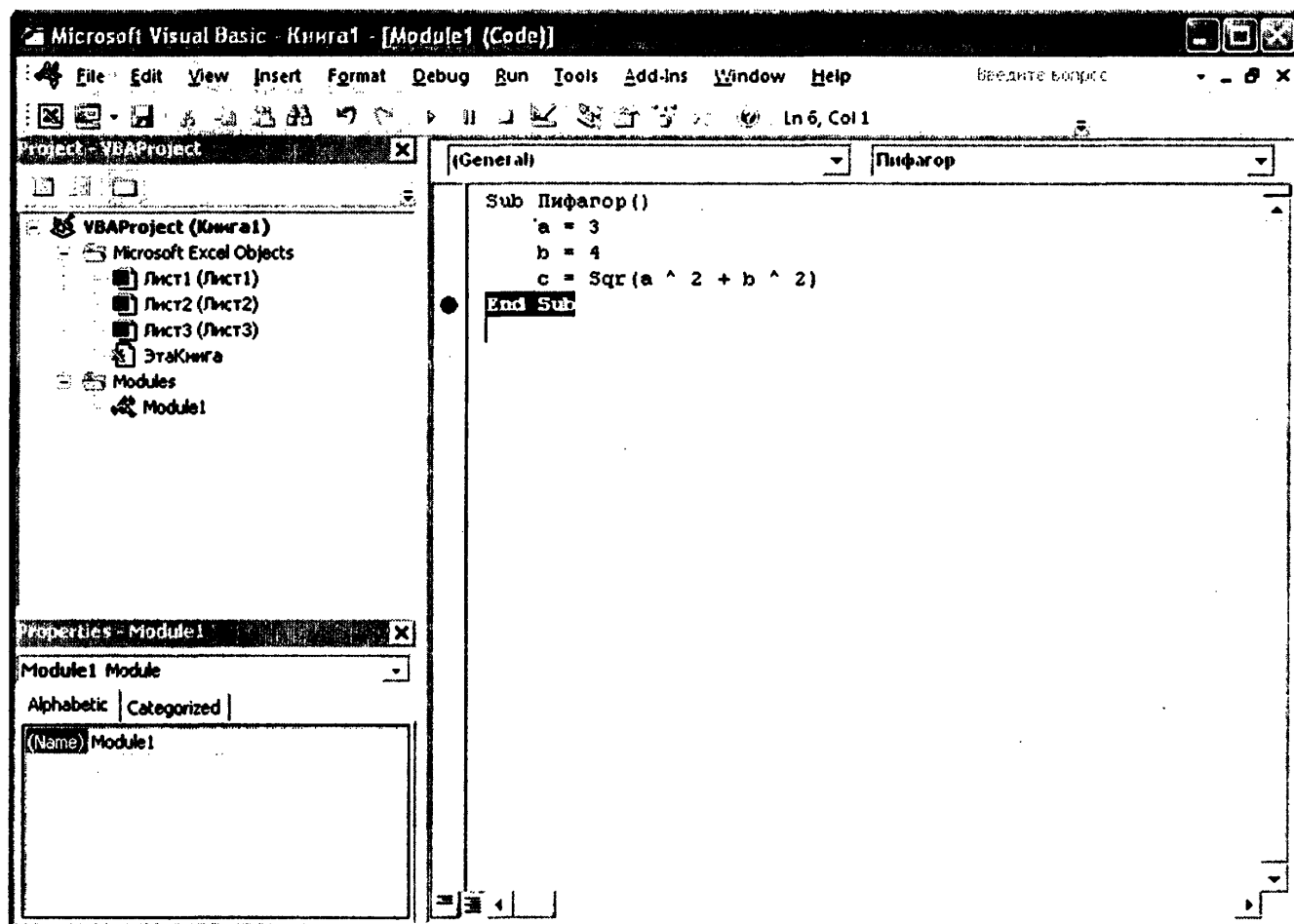


Рис. 2.1. Окно VB с программой Пифагор и точкой останова

2. Кликнем на стрелке на панели инструментов для выполнения программы до точки останова.

3. В открывшемся окне *Macros (Макрос)* нажмем кнопку **Run (Выполнить)**.

2. Основные команды отладки программ

4. Подведем указатель мышки к переменной *c* в тексте программы; при этом высветится $c = 5$ (рис. 2.2).

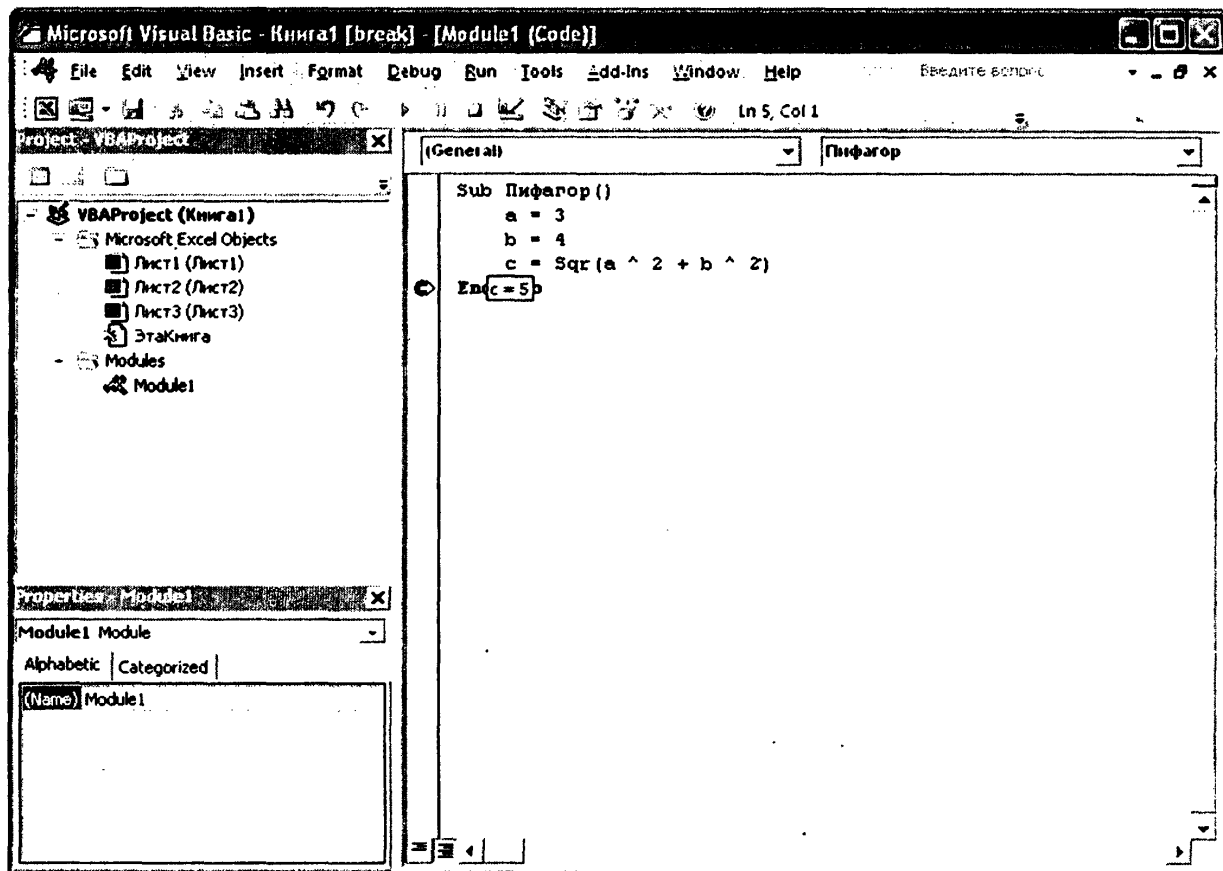


Рис. 2.2. Визуализация значения переменной *c* во время останова

5. Кликнем на стрелке на панели инструментов для окончания выполнения программы.

Выполнение программы означает последовательное выполнение ее операторов: сначала компьютер задает значения *a* и *b*, а затем рассчитывает значение *c*. В этом легко убедиться, если выполнить следующее.

Задание: уберите точку останова (кликнув на ней) и произведите пошаговое выполнение программы Пифагор с помощью клавиши **F8**, наблюдая за изменением значений переменных *a*, *b* и *c*.

Перед первым нажатием клавиши **F8** мигающий курсор должен находиться в тексте программы.

Выполняемая программа находится в оперативной памяти компьютера. По-английски оперативная память называется «main memory», что переводится как «основная память». Такое название объясняется тем, что оперативное запоминающее устройство не-

посредственно связано с процессором и содержит данные, непосредственно участвующие в операциях. Процессор выполняет операции (в частности арифметические) и управляет работой компьютера (в частности считыванием информации из оперативной памяти и записью информации в оперативную память).

Теперь перейдем к основным конструкциям языка Visual Basic.

3. Переменные. Типы данных

Переменные в программировании имеют такой же смысл, как в математике. Перед тем, как использовать переменную, ее рекомендуется описать (объявить).

Синтаксис оператора описания переменной:

`Dim переменная [As тип]`

В этой конструкции:

- `Dim` — ключевое слово, свидетельствующее о том, что объявляется переменная (*dimension* — размер);
- *переменная* — имя объявляемой переменной;
- `As` — ключевое слово, используемое при задании типа данных (*as* — как);
- *тип* — тип данных для объявляемой переменной или (что то же самое) тип переменной (см. приложение 1).

Здесь и далее квадратными скобками выделяется необязательная часть конструкции (которая может отсутствовать). Такое использование квадратных скобок допустимо, т. к. в конструкциях VB они не используются.

Другими словами, имеются две конструкции оператора описания переменной:

`Dim переменная`

`Dim переменная As тип`

Первую конструкцию мы рассмотрим позже.

Когда при выполнении программы компьютер встречает оператор `Dim`, он выделяет переменной *переменная* часть своей оперативной памяти, которую в программировании принято называть ячейкой. Размер выделенной ячейки, исчисляемый в байтах, определяется типом переменной *тип*. Размеры ячеек, соответствующих разным переменным, могут сильно отличаться (см. второй столбец таблицы в приложении 1).

Напомним, что байт является одной из основных единиц измерения объема информации. Чтобы читатель получил представле-

ние об объеме информации в один байт, отметим, что для хранения информации об одном пикселе (цветной точке на экране дисплея) обычно достаточно трех байт, по байту для интенсивностей красного, зеленого и синего цветов.

Одним оператором `Dim` можно описать несколько переменных, перечислив их через запятую. Пример:

```
Dim i As Byte, j As Integer, k As Integer
```

В языках программирования ключевыми (служебными) словами называются такие слова, которые используются только в конструкциях языка. Ключевые слова нельзя использовать в качестве имен программ и переменных в программах. По умолчанию среда VB настроена таким образом, что при наборе текста программы в окне кода все ключевые слова выделяются синим цветом (комментарии выделяются зеленым цветом, синтаксические ошибки — красным).

В VB используются не только служебные слова, но и служебные словосочетания. Они встретятся позже.

Ограничения на имена переменных такие же, как на имена программ (стр. 13), причем регистр букв не имеет значения. Например, если объявлена переменная `Alpha`, а затем мы пытаемся объявить переменную `alpha`, то слово «Alpha» будет исправлено на «alpha».

`Alpha` — английское название одной из букв греческого алфавита. Так как в VB нет символов (литер) греческих букв, то рекомендуем в именах переменных и программ использовать английские или русские названия этих букв согласно приложению 2.

При использовании русских букв следует иметь в виду следующее: хотя некоторые из них (`а`, `е`, `о`, `р`, `с`, `у`, `х`) имеют такое же написание, как английские буквы, для компьютера буквы с одинаковым русским и английским написанием — разные символы (литеры). Поэтому, например, переменные `оса`, набранные русскими и английскими буквами, являются разными переменными, и при выполнении программы им отводятся разные ячейки оперативной памяти.

Ошибки, связанные с одинаковым написанием русских и английских букв, можно исключить, если среду VB настроить так, чтобы она требовала описания переменных. Для этого надо:

- 1) в среде VB открыть меню **Tools (Сервис)**;
- 2) выбрать пункт **Options (Параметры)**;
- 3) активизировать вкладку **Editor (Редактор)**;

4) включить опцию **Require Variable Declaration** (Явное описание переменных);

5) нажать кнопку **ОК**.

В результате в окне кода, соответствующем новому модулю, первой будет строка вида

```
Option Explicit
```

Эту строку можно ввести в окно кода или удалить вручную, как обычную строку программы.

При наличии строки `Option Explicit` компьютер диагностирует использование в тексте программы необъявленной переменной: при выполнении программы компьютером будет выдана ошибка *Variable not defined* (*Переменная не определена*).

В приложении 1 приведена таблица типов данных VB.

Типы данных `Byte`, `Integer`, `Long`, `Currency`, `Single` и `Double` называются числовыми типами данных.

Согласно третьему столбцу рассматриваемой таблицы:

- в ячейке оперативной памяти компьютера, соответствующей переменной типа `Byte`, могут храниться только натуральные числа и нуль;
- в ячейке, соответствующей переменной типа `Integer` или `Long`, могут храниться целые числа;
- в ячейке, соответствующей переменной типа `Currency`, `Single` или `Double`, могут храниться числа с дробной частью.

Если при описании переменной программист не указывает тип данных (например, `Dim W`), то переменной (`W`) автоматически будет присвоен тип `Variant`. Это означает, что в ячейке, соответствующей этой переменной, может храниться информация любого вида, т. е. тип данных `Variant` аналогичен формату «Общий» табличного процессора Excel.

Рассмотрим оператор

```
Dim i, j As Integer
```

Этот оператор эквивалентен следующему:

```
Dim i As Variant, j As Integer
```

Если мы хотим, чтобы обе переменные (`i` и `j`) имели тип `Integer`, то должны их описать так:

```
Dim i As Integer, j As Integer
```

или

```
Dim i As Integer  
Dim j As Integer
```

Существуют способы описания переменных, не использующие ключевое слово Dim, однако нами они не рассматриваются.

Пример. Программа расчета количества дней в XX веке и определения текущих даты и времени имеет вид

```
Sub Век_20()  
    Dim D1 As Date  
    Dim D2 As Date  
    Dim N As Long  
    D1 = #1 Jan 1900#           'начало века  
    D2 = #31 Dec 1999#         'конец века  
    N = D2 - D1 + 1            'количество дней  
    D1 = Time                  'текущее время  
    D2 = Now                   'текущие дата и время  
End Sub
```

В программе Век_20 выражения #1 Jan 1900# и #31 Dec 1999# означают даты 1 января 1900 года и 31 декабря 1999 года соответственно. Вычитание первой даты из второй дает количество дней между этими датами.

Слово Time (которое переводится с английского как «время») означает обращение к функции VB, значением которой является текущее время. Говорят также, что эта функция возвращает в программу (или просто «возвращает») текущее время. Слово Now (которое переводится как «сейчас») означает обращение к функции VB, возвращающей текущие дату и время.

Более подробно о функциях VB мы поговорим позже, в частности в главах 4 и 9.

Задание:

- 1) вставьте модуль в активную книгу Excel, как описано на стр. 11;
- 2) в окно кода нового модуля введите текст программы Век_20;
- 3) выполните эту программу пошагово с помощью клавиши F8, наблюдая за изменением значений переменных D1, D2 и N.

4. Две основные функции преобразования типов данных

Строкой называется последовательность символов (текст), взятая в кавычки.

Примеры строк:

"Миру мир!"

"37"

"37 рублей"

"X=128.12"

В главе 19 определение строки будет расширено, но пока нам достаточно данного определения.

Часто бывает необходимо преобразовать строку в число. Для этого используется функция `Val`. Она преобразует числовое начало строки в число. Если функция `Val` этого сделать не может, то она возвращает 0. Аргументом функции `Val` является строка; возвращает эта функция число.

Для обратного преобразования (т. е. числа в строку) используется функция `Str`. Аргументом этой функции является число, переменная какого-либо из числовых типов или арифметическое выражение (см. главу 8). Возвращает функция `Str` строку.

Задание: путем пошагового выполнения проверьте правильность работы следующей программы:

```
Sub StrVal()  
    Dim strA As String  
    Dim curB As Currency  
    strA = "45.77"  
    curB = Val(strA)           'Результат: curB = 45.77  
    strA = Str(curB)           'Результат: strA = " 45.77"  
    curB = Val("4.7 = X")     'Результат: curB = 4.7  
    curB = Val("X = 4.7")     'Результат: curB = 0  
End Sub
```

Первый комментарий соответствует случаю, когда при пошаговом выполнении программы желтым выделен оператор

```
strA = Str(curB)
```

Этот комментарий означает следующее: если указатель мышки подвести к `curB`, то высветится `curB = 45.77` — результат выполнения оператора, находящегося в одной строке с комментарием.

Второй комментарий соответствует случаю, когда желтым выделен оператор

```
curB = Val("4.7 = X")
```

Этот комментарий означает: если указатель мышки подвести к `strA`, то высветится `strA = " 45.77"`. И т. д.

Аналогичный смысл имеют все комментарии в программах, начинающиеся со слова «Результат».

Подчеркнем, что при остановках во время выполнения программы желтым цветом выделяется первый из невыполненных операторов.

Комментарии в программе `StrVal` соответствуют случаю, когда операционная система Windows настроена на использование десятичной точки. Однако настройка Windows может быть такой, что вместо десятичной точки следует использовать десятичную запятую. При этом комментарии в программе `StrVal` должны иметь вид

```
'Результат: curB = 45,77  
'Результат: strA = " 45.77"  
'Результат: curB = 4,7  
'Результат: curB = 0
```

В дальнейшем будем считать, что операционная система Windows настроена на использование десятичной точки.

Помимо функций `Val` и `Str`, имеются и другие функции преобразования типов данных. Они будут рассмотрены в главе 8.

5. Константы

Наряду с переменными в VB используются константы. Как и переменной, константе соответствует ячейка оперативной памяти. Однако, в отличие от переменной, содержимое ячейки, соответствующей константе, в программе изменить нельзя.

Существует две разновидности констант — пользовательские и встроенные.

Пользовательские константы требуют объявления. Для этого используется оператор вида

```
Const константа [As тип] = значение
```

В этом операторе:

- *Const* — ключевое слово, которое показывает, что объявляется константа;
- *As* — ключевое слово, с которого начинается задание типа данных;
- *константа* — имя объявляемой константы;
- *тип* — тип данных для объявляемой константы или (что то же самое) тип константы (см. приложение 1);
- *значение* — значение, присваиваемое константе.

Ограничения на имена констант такие же, как на имена переменных (стр. 20).

Примеры объявления констант:

```
Const pi As Double = 3.141592654
Const e As Double = 2.718281828
Const Message = "Завершение работы"
Const Millennium As Date = #1 Jan 2000#
Const beta As Currency = 1/3
```

При выполнении последнего оператора константе *beta* будет присвоен результат деления 1 на 3, то есть 0.3333.

При помощи одного оператора *Const* можно объявить несколько констант, перечислив их через запятую.

Пример:

```
Const Min = 0, Max = 1000, Flag As Boolean = False
```

В качестве примера использования констант рассмотрим следующую программу перевода угла (angle) из градусной меры в радианную:

```
Sub deg2rad()  
    Dim angleD As Currency  
    Dim angleR As Currency  
    Const pi As Double = 3.141592654  
    angleD = 270          'Угол равен 270 градусам  
    angleR = angleD * pi / 180  
                          'Результат: угол в радианах  
End Sub
```

Обратите внимание, что константа `pi` объявляется оператором

```
Const pi As Double = 3.141592654
```

перед использованием в операторе

```
angleR = angleD * pi / 180
```

Задание: разработайте программу `rad2deg`, производящую перевод угла из радианной меры в градусную.

Встроенные константы не требуют объявления.

Имена встроенных констант Visual Basic начинаются с префикса `vb`, например, `vbFriday` (число 6; Friday — пятница). Слово «префикс» переводится с латинского как «стоящий перед корнем», «приставка».

Для имен (в частности имен констант) разработчиками Windows принято следующее соглашение: имена данных, близких по смыслу, начинаются с одного и того же короткого префикса. В частности, встроенные константы Visual Basic имеют префикс `vb`, встроенные константы Excel имеют префикс `xl`.

Помимо `vbFriday`, нам встретятся следующие встроенные константы: `vbCr`, `vbLf`, `vbCrLf`, `xlR1C1`, `xlCalculationAutomatic`, `xlCalculationManual` и др.

6. Получение справки

Для уточнения значения и смысла той или иной встроенной константы следует, находясь в среде VB, нажать клавишу F2. При этом появится так называемое окно просмотра объектов (рис. 6.1). В верхнем текстовом поле этого окна должно быть установлено **All Libraries** (Все библиотеки). В следующем текстовом поле набираем то, что нас интересует, например, *vbFriday*. Кликаем на пиктограмме бинокля. Ответ — в разделе *Search Results* (Результаты поиска), см. рис. 6.2.

Для закрытия окна просмотра объектов надо кликнуть на крестике, находящемся на правом конце строки меню.

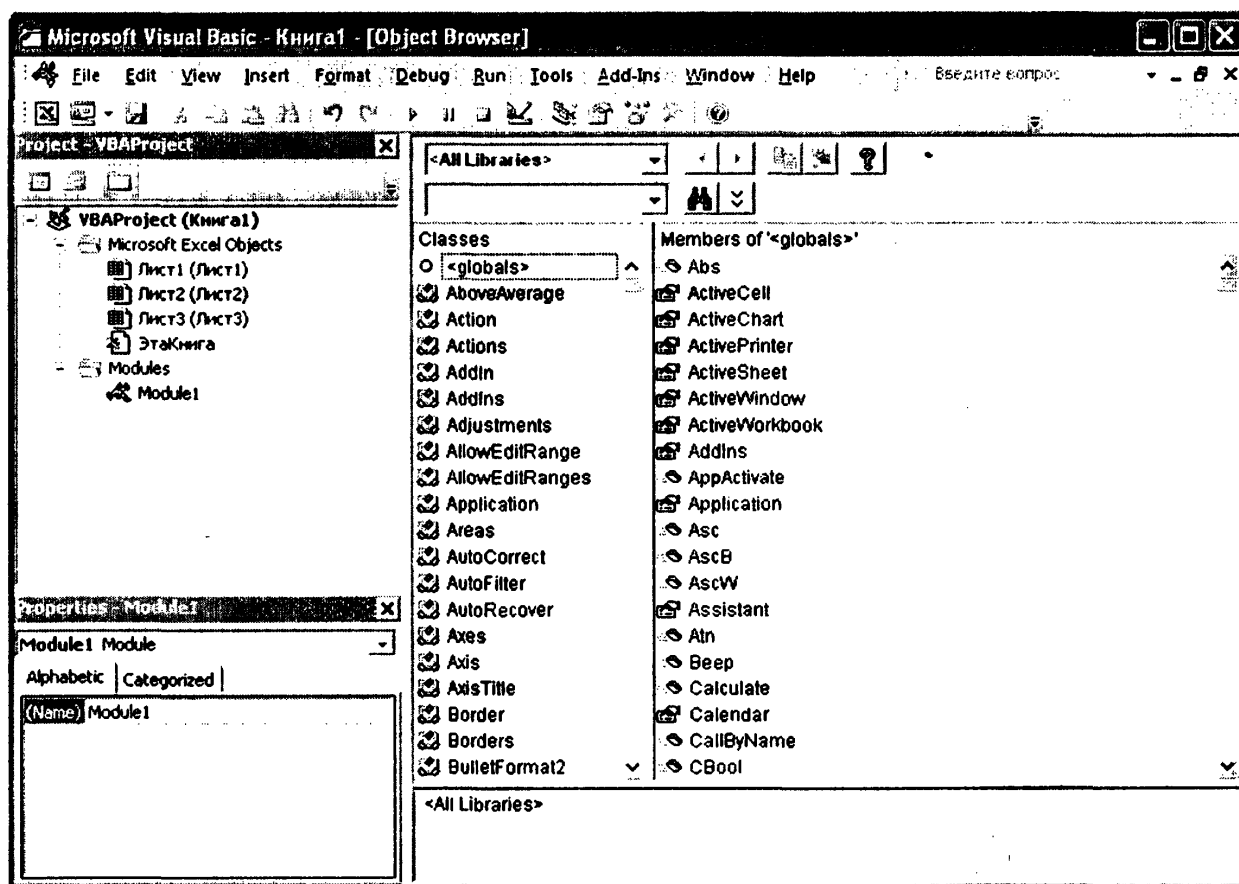


Рис. 6.1. Окно VB, включающее в себя окно просмотра объектов

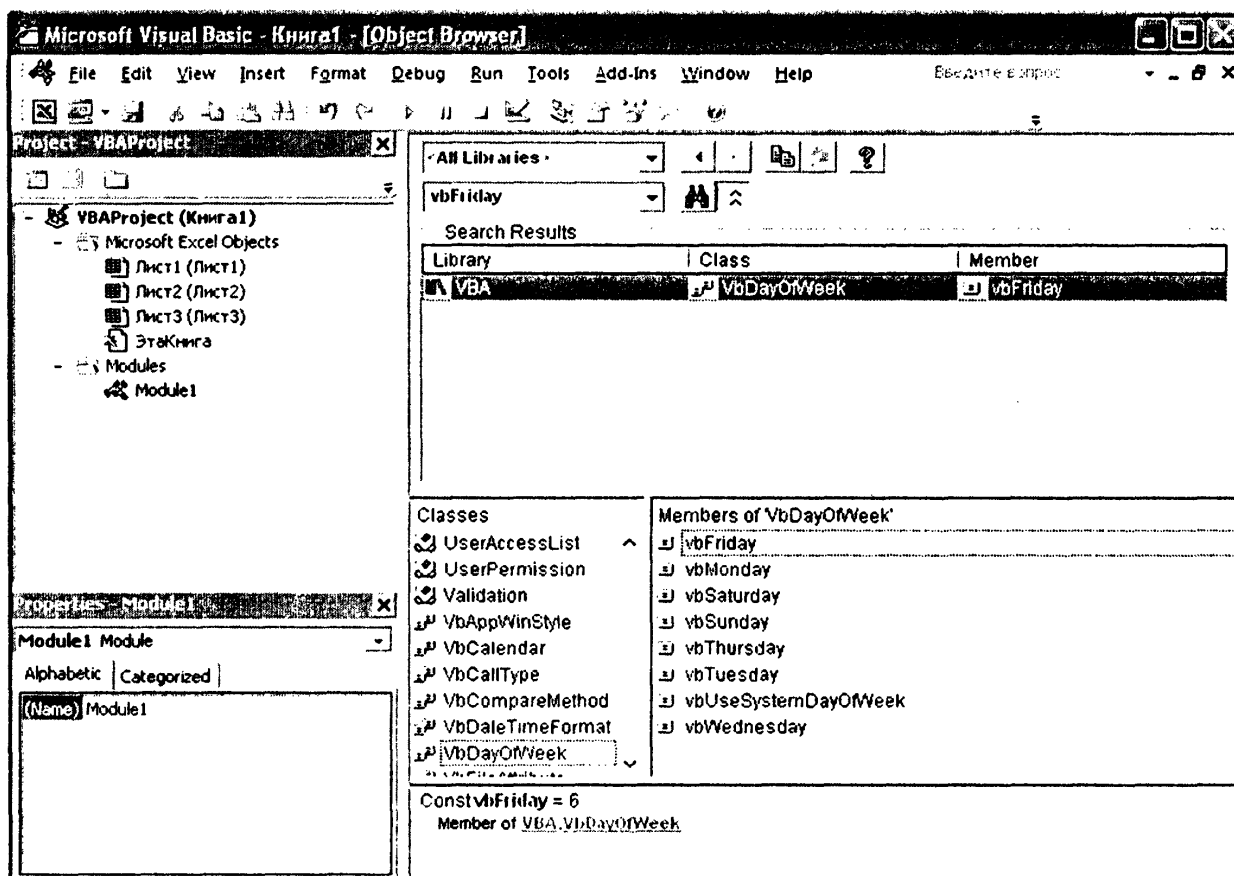


Рис. 6.2. Информация о встроенной константе vbFriday

Задание: с помощью рассмотренной справочной системы, вызываемой нажатием клавиши **F2**, убедитесь в том, что константа `xlR1C1` равна -4150.

Часто необходима справочная система, вызываемая нажатием клавиши **F1**. Мигающий курсор рекомендуем предварительно установить на слове, смысл которого представляет интерес. Это может ускорить процесс нахождения нужной информации.

Задание:

1) в окно кода введите программу `StrVal`, приведенную на стр. 23;

2) установите мигающий курсор на слове `Val`, кликнув на нём, и нажмите клавишу **F1**; при этом на экране дисплея появится окно *Справка: Excel*, содержащее полную информацию о функции `Val` (рис. 6.3);

3) изучите фрагмент программы

```
Dim MyValue
MyValue = Val("2457")           ' Returns 2457.
```

6. Получение справки

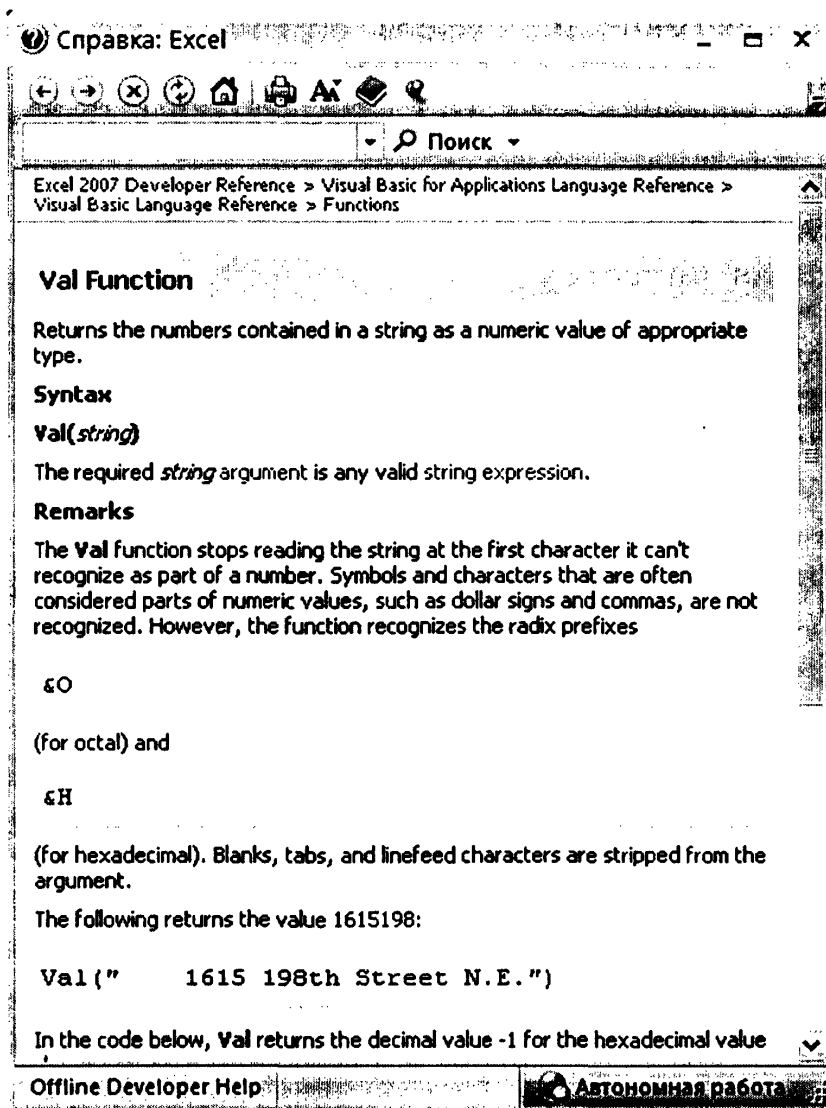


Рис. 6.3. Окно справочной системы Excel с информацией о функции Val

```
MyValue = Val(" 2 45 7")      ' Returns 2457.  
MyValue = Val("24 and 57")    ' Returns 24.
```

приведенный в качестве примера использования функции Val в нижней части окна справочной системы (returns — возвращает);

4) вставьте этот фрагмент в программу StrVal следующим образом:

- выделите фрагмент (с помощью мышки, как в текстовом процессоре Word);
- скопируйте его в буфер обмена, нажав клавиши **Ctrl + C**;
- поместите мигающий курсор напротив последней строки программы в окне кода;

- вставьте фрагмент из буфера обмена в текст программы StrVal, нажав клавиши **Ctrl + V**.

В результате программа StrVal примет следующий вид:

```
Sub StrVal()  
    Dim strA As String  
    Dim curB As Currency  
    strA = "45.77"  
    curB = Val(strA)           'Результат: curB = 45.77  
    strA = Str(curB)          'Результат: strA = " 45.77"  
    curB = Val("4.7 = X")     'Результат: curB = 4.7  
    curB = Val("X = 4.7")     'Результат: curB = 0  
  
    Dim MyValue  
    MyValue = Val("2457")      ' Returns 2457.  
    MyValue = Val(" 2 45 7")   ' Returns 2457.  
    MyValue = Val("24 and 57") ' Returns 24.  
End Sub
```

Задание: выполните полученную программу пошагово (с помощью клавиши **F8**), наблюдая за изменением значения переменной MyValue.

7. Синтаксис оператора присваивания

Оператор присваивания имеет следующий синтаксис:

переменная = выражение

В нем *переменная* — имя переменной, *выражение* — арифметическое или логическое выражение (см. главы 8 и 10) или строка, которую можно рассматривать как выражение (см. главу 19).

Знак = в операторе присваивания называется знаком присваивания.

Оператор присваивания работает следующим образом:

- 1) компьютер рассчитывает значение выражения *выражение*;
- 2) полученное значение записывается в ячейку оперативной памяти компьютера, соответствующую переменной *переменная*, т. е. присваивается этой переменной.

При выполнении оператора присваивания может происходить преобразование типа данных. Этот вопрос рассмотрен в следующей главе.

8. Арифметические выражения

Целое число в VB представляется последовательностью цифр со знаком «минус» или без знака. Примеры целых чисел:

-18 32 0

Если в числе имеется дробная часть, то она отделяется от целой части точкой. При этом целую часть можно опустить, если она равна нулю. Примеры чисел с дробной частью:

0.5 -5.68 -.12 3. .035

В последнем примере цифра 3 с точкой означает число с нулевой дробной частью, т. е. по сути целое число.

Рассмотренная форма записи чисел с дробной частью называется основной.

Возможно также представление чисел с дробной частью в экспоненциальной форме. Например, заряд электрона, равный $1.6 \cdot 10^{-19}$ кулона, запишется в виде $1.6E-19$

Вместо английской буквы E при указании порядка можно использовать букву D, т. е. заряд электрона можно записать в виде $1.6D-19$

Одной из основных конструкций любого алгоритмического языка является арифметическое выражение, очень похожее на математическое выражение. Оно содержит постоянные (числа, константы), переменные и/или функции, соединенные между собой знаками арифметических операций. Отдельно взятое число, константа, переменная или функция также является арифметическим выражением. Ниже приведена таблица с эквивалентными математическими и арифметическими выражениями, не содержащими функции (арифметические выражения с функциями мы рассмотрим в следующей главе).

Согласно синтаксису оператора присваивания, арифметическое выражение находится справа от знака присваивания =.

Пример:

$z = 5 * x + 12 * y$

8. Арифметические выражения

ПРИМЕРЫ ВЫРАЖЕНИЙ

Математическое выражение	Арифметическое выражение VB
$5x + 12y$	<code>5 * x + 12 * y</code>
$\frac{x}{y}$	<code>x / y</code>
y^x	<code>y ^ x</code>
x	<code>x</code>
$19.55 \cdot 10^{-6}$	<code>19.55E-6</code> или <code>19.55D-6</code>

Арифметические операции обозначаются так: + (сложение), - (вычитание, изменение знака), * (умножение), / (деление), ^ (возведение в степень), \ (целочисленное деление, т. е. деление целых чисел с отбрасыванием остатка), Mod (определение остатка от деления целых чисел, являющегося целым числом).

Пример программы:

```
Sub Арифметика1()  
    Dim m As Integer  
    Dim n As Integer  
    Dim x As Double  
    m = 5: n = 2  
    x = m / n           'Результат: x = 2.5  
    x = m \ n           'Результат: x = 2  
    x = m Mod n         'Результат: x = 1  
End Sub
```

Задание: произведите пошаговое выполнение последней программы с помощью клавиши **F8** и проследите за изменением значения переменной `x`.

При наличии в выражении нескольких арифметических операций порядок их выполнения определяется правилом приоритетов арифметических операций:

- 1) в первую очередь выполняется возведение в степень (^);
- 2) далее выполняются умножение и деление (*, /) в порядке следования;
- 3) целочисленное деление (\);
- 4) операция определения остатка от деления целых чисел (Mod);
- 5) операция изменения знака числа (-);

б) в последнюю очередь выполняются сложение и вычитание (+, -) в порядке следования.

Легко заметить, что правило приоритетов арифметических операций в VB такое же, как в Excel (в части операций возведения в степень, умножения, деления, изменения знака числа, сложения и вычитания, имеющих в Excel).

Как и в Excel, для изменения последовательности операций используются круглые скобки: сначала рассчитываются значения арифметических выражений, заключенных в круглые скобки. Квадратные и фигурные скобки в конструкциях VB не используются.

ПРИМЕРЫ ВЫРАЖЕНИЙ

Математическое выражение	Арифметическое выражение VB
$-a^b$	$-a^b$ или $-a^{(b)}$
a^{-b}	$a^{(-b)}$
a^{b+c}	$a^{(b+c)}$
$10^{-4.7}$	$10^{(-4.7)}$
$10^{4.7}$	$10^{4.7}$
$A \cdot B$	$A * B$
$A \cdot (-B)$	$A * (-B)$
a^{b^c}	$a^{(b^c)}$
$(a^b)^c$	a^b^c или $(a^b)^c$
$\frac{a \cdot b}{c \cdot d}$	$a * b / (c * d)$ или $(a * b) / (c * d)$
$a \cdot 10^4$	$a * 1E4$, $a * 1D4$ или $a * 10000$

При расчете значения арифметического выражения результаты выполнения промежуточных арифметических операций либо остаются в процессоре, либо записываются в ячейки сверхоперативной памяти (откуда считываются в нужный момент).

Сверхоперативная (кэш) память — это запоминающее устройство с малым временем доступа (т. е. записи и считывания информации), в несколько раз меньшим, чем время доступа к оперативной памяти. Используется для временного хранения промежуточных результатов и содержимого часто используемых ячеек оперативной памяти. В сверхоперативной памяти может также находиться выполняемый фрагмент программы.

8. Арифметические выражения

Рассмотрим следующий оператор присваивания с арифметическим выражением в правой части:

$$z = 5 * x + 12 * y$$

Процессор выполняет этот оператор примерно так:

1) производит умножение 12 на содержимое ячейки y , результат записывает в ячейку сверхоперативной памяти, которую обозначим $w1$;

2) производит умножение 5 на содержимое ячейки x , к результату прибавляет содержимое ячейки $w1$;

3) полученный результат записывает в ячейку оперативной памяти z .

В математике и программировании «участники» операций называются операндами, как в случае арифметических операций, так и в случае логических операций, рассматриваемых в главе 10. Например, в операции умножения $12 * y$ операндами являются число 12 и переменная y .

Задание: составив программу, аналогичную программе Арифметика1, убедитесь в том, что $(m \setminus n) * n + m \text{ Mod } n$ равняется m .

Пример:

```
Sub Арифметика2()  
    Dim m As Integer, n As Integer  
    Dim x As Single, y As Single  
    x = 3: m = 2: n = -1  
    y = (-3) ^ m           'Результат: y = 9  
    y = -(3 ^ m)           'Результат: y = -9  
    y = -3 ^ m             'Результат: y = -9  
    y = 10 + (x + 7) ^ (m + n) 'Результат: y = 20  
    y = 10 + x + 7 ^ m + n   'Результат: y = 61  
End Sub
```

В арифметических выражениях могут присутствовать величины (переменные и постоянные) разных типов. Если тип, полученный при вычислении значения арифметического выражения в правой части оператора присваивания (справа от знака присваивания $=$), не совпадает с типом переменной в левой части оператора присваивания (слева от $=$), то при выполнении оператора присваивания производится преобразование типа.

Типичной является ситуация, когда значение арифметического выражения справа от знака присваивания имеет дробную часть, а

переменная слева имеет тип Integer или Long. При этом в ходе выполнения присваивания происходит преобразование значения согласно следующему правилу округления:

- если дробная часть значения равна 0.5, то это значение округляется до четного числа из двух ближайших целых чисел;
- в противном случае значение округляется до ближайшего целого числа.

При выполнении операций \ и Mod над дробными числами происходит округление операндов до целых чисел, т. к. операции \ и Mod применимы только к целым числам. Округление производится согласно сформулированному правилу. Результаты выполнения операций \ и Mod являются целыми числами.

Операции, в которых фигурирует один операнд, называются унарными операциями. Из арифметических операций унарной является только операция изменения знака числа (-). Операции над двумя операндами (^, *, /, \, Mod, +, -) называются бинарными.

В VB предусмотрена возможность «насильственного» преобразования одного типа данных в другой при помощи специальных функций. Две из них (Str и Val) мы рассмотрели в главе 4, остальные приведены в следующей таблице.

ФУНКЦИИ ПРЕОБРАЗОВАНИЯ ТИПОВ ДАННЫХ

Название функции	Получаемый тип данных
CBool	Boolean
CByte	Byte
CCur	Currency
CDate	Date
CDbl	Double
CInt	Integer
CLng	Long
CSng	Single
CStr	String
CVar	Variant

Требования к аргументам этих функций, а также примеры их использования можно посмотреть в справочной системе, вызываемой нажатием клавиши F1.

8. Арифметические выражения

Задание: путем пошагового выполнения проверьте правильность работы следующей программы:

```
Sub Функции()  
    Dim intN As Integer  
    Dim strN As String  
    Dim sngN As Single  
    intN = -15  
    strN = Str(intN)           'Результат: strN = "-15"  
    strN = CStr(intN)         'Результат: strN = "-15"  
    intN = 15                  '8-ая строка  
    strN = Str(intN)           'Результат: strN = " 15"  
    strN = CStr(intN)         'Результат: strN = "15"  
    sngN = 25.5                '11-ая строка  
    intN = 1 + CInt(sngN)      'Результат: intN = 27  
    intN = CInt(1 + sngN)      'Результат: intN = 26  
End Sub
```

Обратите внимание на следующее:

- если аргументами функций Str и CStr является неотрицательное число, то эти функции возвращают разные строки (см. девятую и десятую строки программы);
- функция CInt округляет значение аргумента согласно правилу округления (см. строки с одиннадцатой по тринадцатую).

9. Математические функции

Помимо ранее рассмотренных функций, в VB имеются и другие функции, называемые математическими (см. приложение 3). Аргументами этих функций являются арифметические выражения.

В тексте программы аргумент функции заключается в круглые скобки.

Аргумент тригонометрических функций (косинуса, синуса и тангенса) задается в радианах, а не в градусах. Обратная тригонометрическая функция $\text{Atn}(x)$ возвращает в программу значение угла в радианах от $-\pi/2$ до $\pi/2$, тангенс которого равен значению арифметического выражения x .

Функция $\text{Sgn}(x)$ возвращает 1, 0 и -1 соответственно при $x > 0$, $x = 0$ и $x < 0$.

Отметим, что в VB функция $\text{Log}(x)$ рассчитывает натуральный логарифм.

В приложении 3 приведены также операторы VB, позволяющие рассчитывать значения тригонометрической функции $\text{ctg } x$, обратных тригонометрических функций $\arcsin x$, $\arccos x$ и $\text{arcctg } x$ и десятичного логарифма $\lg x$.

Вдобавок к функциям, приведенным в приложении 3, рассмотрим функцию $\text{Round}(x[, n])$, предназначенную для округления чисел с дробной частью. Напомним, что квадратными скобками выделяется та часть конструкции, которая может отсутствовать. Другими словами, имеются две функции округления — $\text{Round}(x, n)$ и $\text{Round}(x)$.

Функция $\text{Round}(x, n)$ возвращает в программу значение арифметического выражения x , округленное до n знаков после десятичной точки. Функция $\text{Round}(x)$ возвращает целое число согласно правилу округления (стр. 36) и $\text{Round}(x) = \text{CInt}(x)$.

Математические функции используются в арифметических выражениях. Примером этого является оператор присваивания

```
c = Sqr(a ^ 2 + b ^ 2)
```

программы Пифагор (стр. 13).

9. Математические функции

При расчете значения арифметического выражения функции имеют приоритет перед арифметическими операциями. Так, например, перед выполнением присваивания в операторе

$$e = b / c * \cos(a)$$

процессор сначала рассчитает значение косинуса (причем в процессе расчета он использует ячейки сверхоперативной памяти), а затем выполнит операции деления и умножения.

Рассмотренные функции имели один или два аргумента. Существуют также функции, не имеющие аргументов. К ним относятся функция Time, возвращающая текущее время, и функция Now, возвращающая текущие дату и время. Эти функции фигурировали в программе Век_20 на стр. 22.

Примеры функций можно посмотреть также в справочной системе, вызываемой нажатием клавиши F1.

Задание:

1) вставьте операторы

```
Dim V As Single, W As Single
V = intN + Round(Sqr(2*intN), 2)
W = Round(V)
```

перед последней строкой программы функции (стр. 37);

2) последнюю строку программы отметьте точкой останова;

3) выполните программу до точки останова и убедитесь в том, что полученные значения переменных V и W равны 33.21 и 33 соответственно;

4) объясните этот результат.

Задание N: из четырех значений $x = -1.8, -1.25, 1.27, 1.68$ выберите такие значения, для которых выполняется N-ое условие из следующих десяти:

- 1) $\text{Fix}(x) = \text{CInt}(x)$ и $\text{Fix}(x) \neq \text{Int}(x)$;
- 2) $\text{Int}(x) = \text{CInt}(x)$ и $\text{Int}(x) \neq \text{Fix}(x)$;
- 3) $\text{Fix}(x) = \text{Int}(x) = \text{CInt}(x) = \text{Round}(x)$;
- 4) $\text{Fix}(x) = \text{Int}(x)$ или $\text{Fix}(x) \neq \text{CInt}(x)$;
- 5) $\text{Fix}(x) = \text{CInt}(x)$ или $\text{Fix}(x) \neq \text{Int}(x)$;
- 6) $\text{Int}(x) = \text{CInt}(x)$ или $\text{Int}(x) \neq \text{Fix}(x)$;
- 7) $\text{Fix}(x) = \text{Int}(x)$ или $\text{Fix}(x) = \text{CInt}(x)$;
- 8) $\text{Int}(x) = \text{CInt}(x)$ или $\text{Int}(x) = \text{Fix}(x)$;
- 9) $\text{CInt}(x) = \text{Fix}(x)$ или $\text{CInt}(x) = \text{Int}(x)$;
- 10) $\text{Fix}(x) = \text{Int}(x)$ и $\text{Fix}(x) \neq \text{CInt}(x)$.

Полезной является функция `Rnd`, которая предназначена для расчета случайных чисел, используемых для моделирования случайных явлений. Простейшая модель случайных явлений связана с подбрасыванием монеты и наблюдением за выпадением «орла» или «решки».

Идея моделирования случайных явлений известна очень давно. В связи с появлением электронных вычислительных машин эта идея получила интенсивное развитие в 1950-х годах под именем метода Монте-Карло.

Методом Монте-Карло рассчитываются финансовые риски, а также моделируются микроэлектронные приборы и эволюция звезд. Это — лишь часть задач, требующих расчета случайных чисел.

Задание: ознакомьтесь с описанием функции `Rnd`, которое приведено в справочной системе, вызываемой нажатием клавиши **F1**.

Пример использования функции `Rnd` приведен на стр. 68.

10. Логические выражения

Помимо арифметических выражений, в VB можно использовать логические выражения (утверждения), принимающие одно из двух значений типа Boolean — True (истина, логическая единица) или False (ложь, логический нуль). Отдельно взятое значение True или False также является логическим выражением.

Примеры логических выражений:

```
5 >= 3
5 < 3
False
```

Знаки >= и < являются известными знаками сравнения «больше или равно» и «меньше». Первое логическое выражение имеет значение True, второе — False, третье — False.

Согласно синтаксису оператора присваивания, логическое выражение находится справа от знака присваивания =.

Пример. Программа с четырьмя операторами присваивания:

```
Sub Логика1()
    Dim x As Integer
    Dim y As Integer
    Dim blnA As Boolean
    x = 5
    y = 2
    blnA = x > y           'Результат: blnA = True
    blnA = x = y           'Результат: blnA = False
End Sub
```

В программе Логика1 два логических выражения: $x > y$ и $x = y$. Их значения при $x = 5$ и $y = 2$ приведены в комментариях.

Более сложные логические выражения составляются с помощью логических операций. Рассмотрим три из них — Not, And и Or.

Операция Not определяется следующим образом:

- если A равно True, то Not A равно False;
- если A равно False, то Not A равно True.

Операция **Not** имеет один операнд — **A**, т. е. является унарной операцией. Она называется логическим НЕ или логическим отрицанием.

Определения логических операций **And** и **Or** даются следующими таблицами.

Определение операции And

A	B	A And B
True	True	True
True	False	False
False	True	False
False	False	False

Определение операции Or

A	B	A Or B
True	True	True
True	False	True
False	True	True
False	False	False

Как видно из приведенных таблиц, операции **And** и **Or** имеют два операнда — **A** и **B**, т. е. являются бинарными операциями.

Выражение **A And B** имеет значение **True** только в том случае, когда оба операнда (и **A**, и **B**) имеют значение **True**. Во всех остальных случаях выражение **A And B** имеет значение **False**. Операция **And** называется конъюнкцией, логическим И или логическим умножением.

Выражение **A Or B** имеет значение **False** только в том случае, когда оба операнда имеют значение **False**. Во всех остальных случаях выражение **A Or B** имеет значение **True**. Операция **Or** называется дизъюнкцией, логическим ИЛИ или логическим сложением.

При наличии в выражении нескольких логических операций порядок их выполнения определяется следующим правилом приоритетов:

- 1) в первую очередь выполняется операция **Not**;
- 2) далее выполняется операция **And**;
- 3) в последнюю очередь выполняется операция **Or**.

Для изменения последовательности логических операций используются круглые скобки (как в арифметических выражениях). Скобки можно использовать и для большей наглядности логических выражений.

Пример:

```
Sub Логика2()  
    Dim x As Double  
    Dim y As Double  
    Dim z As Double  
    Dim blnA As Boolean  
    x = 1  
    y = 2.87  
    z = 3.12  
    blnA = (x > y) And (y < z) 'Результат: blnA = False  
    blnA = x < y And y < z      'Результат: blnA = True  
    blnA = x > y Or y > z        'Результат: blnA = False  
    blnA = Not (x < y Or Not y < z)  
                                'Результат: blnA = False  
    blnA = Not x > y And x > y  
                                'Результат: blnA = False  
    blnA = Not (x > y And x > y)  
                                'Результат: blnA = True  
End Sub
```

В операторе `blnA = (x > y) And (y < z)` скобки введены для выделения логических выражений `x > y` и `y < z`; эти скобки можно убрать.

Задание: путем пошагового выполнения программы Логика2 проверьте правильность ее работы.

В логических выражениях используются следующие знаки сравнения: `<` (меньше), `>` (больше), `=` (равно), `<=` (меньше или равно), `>=` (больше или равно) и `<>` (не равно).

Задание N:

- напишите программу, которая из четырех значений `x = -1.8, -1.25, 1.27, 1.68` выбирает такие значения, для которых выполняется N-ое условие из следующих десяти:
 - 1) `Fix(x) = CInt(x)` и `Fix(x) ≠ Int(x)`;
 - 2) `Int(x) = CInt(x)` и `Int(x) ≠ Fix(x)`;
 - 3) `Fix(x) = Int(x) = CInt(x) = Round(x)`;
 - 4) `Fix(x) = Int(x)` или `Fix(x) ≠ CInt(x)`;

- 5) $\text{Fix}(x) = \text{CInt}(x)$ или $\text{Fix}(x) \neq \text{Int}(x)$;
 - 6) $\text{Int}(x) = \text{CInt}(x)$ или $\text{Int}(x) \neq \text{Fix}(x)$;
 - 7) $\text{Fix}(x) = \text{Int}(x)$ или $\text{Fix}(x) = \text{CInt}(x)$;
 - 8) $\text{Int}(x) = \text{CInt}(x)$ или $\text{Int}(x) = \text{Fix}(x)$;
 - 9) $\text{CInt}(x) = \text{Fix}(x)$ или $\text{CInt}(x) = \text{Int}(x)$;
 - 10) $\text{Fix}(x) = \text{Int}(x)$ и $\text{Fix}(x) \neq \text{CInt}(x)$;
- убедитесь в правильности работы программы, сравнив результат расчета с результатом выполнения задания N на стр. 39.

11. Оператор перехода

Операторы рассмотренных программ выполняются компьютером поочередно. Такие программы называются линейными.

Для изменения последовательности выполнения операторов (т. е. для ветвления программы) используется оператор перехода `GoTo`, имеющий следующий синтаксис:

`GoTo метка`

В этом операторе *метка* — это целое неотрицательное число без знака (0, 1, 2, 3, ...) или последовательность букв и цифр, начинающаяся с буквы (например, `start53a`).

Перед оператором, на который должен быть осуществлен переход (или, что то же самое, должно быть передано управление), ставится метка с двоеточием. После выполнения оператора с меткой выполняется оператор, следующий за ним (если помеченный оператор не `GoTo`).

Примеры использования оператора `GoTo` приведены в следующей главе (см. программу IT2 на стр. 47).

Если меткой является целое неотрицательное число, то это число еще называют номером оператора (строки).

12. Конструкции принятия решений

Типичной является ситуация, когда в определенном месте программы необходимо выполнить те или иные операторы в зависимости от некоторых условий. Выбор операторов осуществляется с помощью одной из двух конструкций принятия решений — If...Then и Select Case.

Первая конструкция принятия решений — If...Then, называемая условным оператором. На русский язык «If...Then» переводится как «если...то». Существует несколько разновидностей этой конструкции.

Простейший условный оператор:

If *условие* Then *оператор*

В этой конструкции *условие* представляет собой логическое выражение. Компьютер рассчитывает значение этого логического выражения.

Если значением логического выражения *условие* является True, то будем говорить, что условие истинно. Если значением является False, то будем говорить, что условие ложно.

При истинности условия выполняется оператор *оператор*, находящийся после ключевого слова Then, при ложности *оператор* не выполняется.

Далее выполняется оператор, следующий за конструкцией If...Then, вне зависимости от того, выполнен *оператор* или нет (если *оператор* не является оператором перехода GoTo).

Рассмотрим следующую программу:

```
Sub IT1()  
    Dim X As Byte  
    X = 12                                'начальное значение X  
    If (X > 9 And X < 12) Then X = X + 1  
    X = X + 2  
    X = X * 2                             'конечное значение X  
End Sub
```

Задание:

1) поставьте точку останова напротив строки End Sub;

12. Конструкции принятия решений

2) запустите программу, кликнув на стрелке на панели инструментов;

3) убедитесь в том, что в момент останова X равняется 28;

4) объясните этот результат;

5) кликните на стрелке на панели инструментов для окончания выполнения программы;

6) в конструкции If...Then измените условие так, чтобы в момент останова переменная X имела значение 30;

7) текст программы IT1 верните в исходное состояние.

Частным случаем рассмотренной конструкции If...Then является оператор условного перехода, включающий в себя оператор перехода GoTo. Синтаксис оператора условного перехода:

If *условие* Then GoTo *метка*

Согласно этому оператору, при истинности условия происходит переход на метку *метка* и выполняется помеченный оператор.

Пример программы:

```
Sub IT2()  
    Dim X As Integer  
    X = 12  
2:   If X > 9 And X < 12 Then GoTo LastLine  
        X = X - 2  
        GoTo 2  
LastLine:  
End Sub
```

Задание:

1) выполните программу IT2 пошагово, следя за последовательностью выполнения операторов и изменением значения X;

2) объясните, почему значение X меняется так, а не иначе.

Далее оператор перехода GoTo не рассматривается.

Если при истинности условия требуется выполнить несколько операторов, то следует использовать конструкцию вида

```
If условие Then  
    операторы  
End If
```

Этот условный оператор работает таким образом.

Если условие истинно, то выполняется блок операторов *операторы*, находящийся после ключевого слова Then. Если же условие ложно, то блок *операторы* не выполняется.

Далее выполняется оператор, следующий за служебным словосочетанием `End If`, вне зависимости от того, выполнен блок операторы или нет.

Рассмотрим следующую конструкцию `If...Then`:

```
If условие Then
    операторы1
Else
    операторы2
End If
```

Эта конструкция условного оператора работает следующим образом.

Если условие истинно, то выполняется блок *операторы1*, находящийся после ключевого слова `Then`. Если условие ложно, то выполняется блок *операторы2*, находящийся после ключевого слова `Else`.

После выполнения блока *операторы1* или *операторы2* выполняется оператор, следующий за строкой `End If`.

В качестве примера использования последней конструкции приведем следующую программу:

```
Sub IT3()
    Dim X As Byte
    X = 12                                'начальное значение X
    If (X > 9 And X < 12) Then
        X = X + 3
    Else
        X = X + 2
    End If
    X = X * 2                             'конечное значение X
End Sub
```

При любом начальном значении переменной `X` эта программа дает такое же конечное значение `X`, как и программа `IT1`.

В том случае, когда блок операторов нужно выполнять после проверки не одного, а нескольких условий, следует использовать такую конструкцию `If...Then`:

```
If условие1 Then
    операторы1
ElseIf условие2 Then
    операторы2
```

12. Конструкции принятия решений

```
ElseIf условие3 Then
    операторы3
.
.
.
[Else
    операторыN]
End If
```

Если первое условие истинно (т. е. логическое выражение *условие1* имеет значение True), то выполняется блок *операторы1*. Если оно ложно, то проверяется второе условие, находящееся после ключевого слова *ElseIf*. При истинности второго условия (т. е. если логическое выражение *условие2* имеет значение True) выполняется блок *операторы2*, и т. д. Если ни одно из условий не является истинным, то выполняется блок *операторыN*, расположенный после ключевого слова *Else*.

Далее выполняется оператор, следующий за служебным словосочетанием *End If*, вне зависимости от того, какой из блоков (*операторы1*, *операторы2*, ... или *операторыN*) выполнен.

Мы рассмотрели конструкции условного оператора. В дополнение к ним рассмотрим функцию

IIf(условие, выражение1, выражение2)

В ней *условие* представляет собой логическое выражение, а *выражение1* и *выражение2* — арифметические или логические выражения (см. главы 8 и 10) или строки, которые можно рассматривать как выражения (см. главу 19).

Функция *IIf* возвращает в программу значение выражения *выражение1* или *выражение2* в зависимости от того, какое значение принимает логическое выражение — True или False.

Пример:

```
Sub IT4 ()
    Dim intA As Integer
    Dim strA As String
    intA = 6
    strA = IIf(intA Mod 2 = 0, "Четное", "Нечетное")
End Sub
```

Вторая конструкция принятия решений — *Select Case*, называемая оператором выбора. Эта конструкция выглядит так:

```
Select Case выражение
    Case значение1
```

```
        операторы1
    Case значение2
        операторы2
    .
    .
    .
    [Case Else
        операторыN]
End Select
```

В ней *выражение* — арифметическое или логическое выражение или строка. Работает оператор выбора следующим образом.

Если выражение *выражение* принимает значение *значение1*, то выполняется блок операторов *операторы1*. Если выражение *выражение* принимает значение *значение2*, то выполняется блок *операторы2*, и т. д. Если выражение не равно ни одному из приведенных значений, то выполняется блок *операторыN*, находящийся после служебного словосочетания *Case Else*.

Далее выполняется оператор, следующий за строкой *End Select*, вне зависимости от того, какой из блоков (*операторы1*, *операторы2*, ... или *операторыN*) выполнен.

В том случае, когда при нескольких значениях выражения *выражение* необходимо выполнить один и тот же блок операторов, эти значения нужно перечислить после ключевого слова *Case*, разделяя их запятыми.

Пример:

```
Sub Выбор()
    Dim x As Integer
    x = 1
    Select Case 2 * x + 1
        Case 1
            x = x + 1
        Case 2, 3, 4
            x = 10
        Case Else
            x = 20
    End Select
    x = x Mod 3
End Sub
```

Задание: выполните программу *Выбор* пошагово, следя за последовательностью выполнения операторов и изменением значения переменной *x*.

13. Циклы

Для многократного выполнения блока операторов используется один из трех операторов цикла — *For...Next*, *While...Wend* или *Do...Loop*.

Цикл *For...Next* используется в том случае, когда количество выполнений заданного блока операторов известно заранее. Данная конструкция имеет следующий синтаксис:

```
For счетчик = начало To конец [Step шаг]  
    операторы  
Next [счетчик]
```

При положительном значении величины *шаг* цикл *For...Next* работает следующим образом.

Переменной *счетчик* сначала присваивается значение *начало*. Далее проверяется условие *счетчик* > *конец*. Если результатом проверки этого условия является значение *True*, то происходит окончание работы цикла. При этом операторы цикла *операторы* ни разу не выполняются.

Если результатом проверки условия *счетчик* > *конец* является *False*, то в этом случае операторы цикла выполняются первый раз. После этого происходит переход на начало цикла. Далее значение переменной *счетчик* увеличивается на величину шага *шаг*, после чего снова проверяется условие *счетчик* > *конец*. Если результатом является *False*, то операторы цикла выполняются второй раз, и т. д.

Выполнение цикла оканчивается, когда результатом проверки условия *счетчик* > *конец* становится *True*. По окончании цикла выполняется оператор, следующий за циклом, т. е. за ключевым словом *Next*.

Пример. Программа расчета факториал числа 6 имеет следующий вид:

```
Sub Факториал1()  
    Dim I As Byte  
    Dim F As Long
```

```
F = 1
For I = 1 To 6 Step 1
    F = F * I
Next I
End Sub
```

Напомним, что факториал целого положительного числа n обозначается $n!$ и (согласно определению) $n!$ равняется произведению целых чисел от 1 до n : $n! = 1 \times 2 \times \dots \times n$.

Задание:

1) произведите пошаговое выполнение программы Факториал1 и посмотрите, как меняется значение переменной F;

2) в программе Факториал1 замените 6 на 13 и попробуйте выполнить эту программу;

3) руководствуясь описанием типа данных Long (см. приложение 1), объясните причину аварийного останова во время выполнения последней программы.

При отрицательном значении величины *шаг* цикл For...Next работает, как при положительном значении, но проверяется условие *счетчик* < *конец*.

Пример. Второй вариант программы расчета 6!:

```
Sub Факториал2 ()
    Dim I As Byte
    Dim F As Long
    F = 1
    For I = 6 To 1 Step -1
        F = F * I
    Next I
End Sub
```

В случае отсутствия ключевого слова Step шаг изменения переменной *счетчик* полагается равным единице.

В качестве параметров цикла *начало*, *конец* и *шаг* можно использовать не только числа, но и арифметические выражения. Важно, чтобы к моменту выполнения цикла For...Next все переменные в этих арифметических выражениях имели числовые значения (были определены).

Пример. Третий вариант программы расчета 6!:

```
Sub Факториал3 ()
    Const e As Double = 2.718281828
```

```

Dim J As Byte
Dim N As Byte
Dim F As Long
N = Round(e)
F = 1
For J = 1 To N ^ 2 - 3
    F = F * J
Next J
End Sub

```

В программе Факториал3 арифметическим выражением является параметр цикла *конец* (равен $N^2 - 3$). К моменту выполнения цикла переменная *N* имеет значение 3, а арифметическое выражение $N^2 - 3$ имеет значение 6.

Отсутствие ключевого слова *Step* говорит о том, что шаг изменения переменной *J* равен 1; меняется *J* от 1 до 6.

Зачастую требуется досрочно выйти из цикла. В этом случае цикл *For...Next* записывается в виде

```

For счетчик = начало To конец [Step шаг]
    операторы1
    If условие Then Exit For
    операторы2
Next [счетчик]

```

Досрочное окончание цикла обеспечивается оператором, обозначаемым служебным словосочетанием *Exit For*. Работает цикл следующим образом.

При каждом значении переменной *счетчик* после выполнения блока *операторы1* компьютер рассчитывает значение логического выражения *условие*. Если это значение равно *False*, то работа цикла продолжается. В противном случае работа цикла оканчивается: блок *операторы2* не выполняется, а осуществляется переход на оператор, следующий за циклом.

Пример. Четвертый вариант программы расчета 6!:

```

Sub Факториал4()
    Dim I As Byte
    Dim F As Long
    F = 1
    For I = 1 To 13 Step 1
        F = F * I
        If I = 6 Then Exit For
    Next I
End Sub

```

```
Next I
End Sub
```

Цикл *While...Wend* применяется в том случае, когда число выполнений операторов цикла заранее неизвестно. Синтаксис этого цикла:

```
While условие
    операторы
Wend
```

Работа цикла *while...wend* начинается с расчета значения логического выражения *условие*. Если *условие* = False, то работа цикла оканчивается, т. е. осуществляется переход на оператор, расположенный после ключевого слова *wend*. Если *условие* = True, то выполняются операторы цикла *операторы*. После этого опять рассчитывается значение логического выражения *условие*, и т. д.

Пример. Пятый вариант программы расчета 6!:

```
Sub Факториал5 ()
    Dim I As Byte
    Dim F As Long
    F = 1
    I = 1
    While I <= 6
        F = F * I
        I = I + 1
    Wend
End Sub
```

Цикл *Do...Loop*, как и цикл *While...Wend*, применяется, когда число выполнений операторов цикла заранее неизвестно. Существует четыре разновидности данной конструкции.

Первой разновидностью является цикл *Do While...Loop*, имеющий следующий синтаксис:

```
Do While условие
    операторы
Loop
```

Работа цикла *Do While...Loop* начинается с расчета значения логического выражения *условие*. Если *условие* = False, то работа цикла оканчивается, т. е. осуществляется переход на оператор, расположенный после ключевого слова *Loop*. Если усло-

13. Циклы

вие = True, то выполняются операторы цикла *операторы*. После этого опять рассчитывается значение логического выражения *условие*, и т. д.

Цикл Do While...Loop эквивалентен рассмотренному ранее циклу While...Wend.

Пример. Шестой вариант программы расчета 6!:

```
Sub Факториал6()  
    Dim I As Byte  
    Dim F As Long  
    F = 1  
    I = 1  
    Do While I <= 6  
        F = F * I  
        I = I + 1  
    Loop  
End Sub
```

Пример. Программа «шагания» по оси координат:

```
Sub Steps1()  
    Dim x As Single  
    Dim h As Single  
    h = 0.5          'величина шага равна 0.5  
    x = 44           'начальное значение x равно 44  
    Do While x < 55  'конечное значение x равно 55  
        x = x + h    'значение x увеличивается на h  
    Loop            'do while ... - делай пока выполняется ...  
End Sub
```

Задание: произведите пошаговое выполнение программы Steps1 и посмотрите, как меняется значение переменной x.

Вторая разновидность конструкции Do...Loop — цикл Do Until...Loop. Синтаксис этого цикла:

```
Do Until условие  
    операторы  
Loop
```

Работа цикла Do Until...Loop начинается с расчета значения логического выражения *условие*. Если *условие* = True, то работа цикла оканчивается, т. е. осуществляется переход на оператор, расположенный после ключевого слова Loop. Если *условие* = False,

то выполняются операторы цикла *операторы*. После этого опять рассчитывается значение логического выражения *условие*, и т. д.

Пример. Второй вариант программы «шагания» по оси координат:

```
Sub Steps2()  
    Dim x As Single  
    Dim h As Single  
    h = 0.5           'величина шага равна 0.5  
    x = 44            'начальное значение x равно 44  
    Do Until x >= 55  'конечное значение x равно 55  
        x = x + h     'значение x увеличивается на h  
    Loop             'do until ... - делай до выполнения ...  
End Sub
```

Видно, что в циклах `Do While...Loop` и `Do Until...Loop` существует ситуация, когда операторы цикла не выполняются ни разу, т. к. условие окончания цикла проверяется перед выполнением этих операторов.

Третья разновидность конструкции `Do...Loop` — цикл `Do...Loop While` — имеет синтаксис

```
Do  
    операторы  
Loop While условие
```

Работа цикла `Do...Loop While` начинается с выполнения операторов цикла *операторы*. После этого рассчитывается значение логического выражения *условие*. Если *условие* = `False`, то работа цикла оканчивается. В противном случае опять выполняются операторы цикла, и т. д.

Пример. Третий вариант программы «шагания» по оси координат:

```
Sub Steps3()  
    Dim x As Single  
    Dim h As Single  
    h = 0.5           'величина шага равна 0.5  
    x = 44            'начальное значение x равно 44  
    Do  
        x = x + h     'значение x увеличивается на h  
    Loop While x < 55 'конечное значение x равно 55  
End Sub
```

Четвертая разновидность конструкции `Do...Loop` — цикл `Do...Loop Until` — имеет синтаксис

```
Do
    операторы
Loop Until условие
```

Работа цикла `Do...Loop Until` начинается с выполнения операторов цикла *операторы*. После этого рассчитывается значение логического выражения *условие*. Если *условие* = True, то работа цикла оканчивается. В противном случае опять выполняются операторы цикла, и т. д.

Пример. Четвертый вариант программы «шагания» по оси координат:

```
Sub Steps4()
    Dim x As Single
    Dim h As Single
    h = 0.5           'величина шага равна 0.5
    x = 44            'начальное значение x равно 44
    Do
        x = x + h     'значение x увеличивается на h
    Loop Until x >= 55 'конечное значение x равно 55
End Sub
```

Видно, что в последних двух разновидностях цикла `Do...Loop` операторы цикла *операторы* хотя бы один раз обязательно выполняются, т. к. условие окончания цикла проверяется после выполнения этих операторов.

Внутри цикла `Do...Loop` может находиться оператор `Exit Do`, предназначенный для досрочного выхода из цикла. Употребляется этот оператор так же, как `Exit For` в цикле `For...Next`, — он входит в состав условного оператора.

Отметим, что для цикла `While...Wend` не существует оператора досрочного выхода из цикла.

14. Проявление погрешности представления вещественных чисел

В последней программе уменьшим величину шага h в пять раз. При этом можно ожидать, что количество выполнений оператора цикла $x = x + h$ увеличится в пять раз, а конечное значение x останется равным 55. Однако это не так: конечное значение x равно 55.09983. Чтобы в этом убедиться, выполним следующее.

1. В окно кода введем программу

```
Sub Steps5()  
    Dim x As Single  
    Dim h As Single  
    h = 0.1 'этим оператором Steps5 отличается от Steps4  
    x = 44  
    Do  
        x = x + h  
    Loop Until x >= 55  
End Sub
```

2. Установим точку останова на строке End Sub.

3. Кликнув на стрелке на панели инструментов, выполним программу до точки останова.

4. Подведем указатель мышки к переменной x в тексте программы; при этом высветится $x = 55.09983$ (рис. 14.1).

Описанный «феномен» изменения конечного значения x при уменьшении шага h в целое число раз объясняется тем, что в компьютере «почти все» вещественные (действительные, реальные) числа представляются с погрешностью.

В данном случае шаг h , являющийся величиной типа Single, несколько меньше 0.1. Поэтому в процессе работы программы оператор цикла выполняется «лишний» раз. Этим и

14. Проявление погрешности представления вещественных чисел

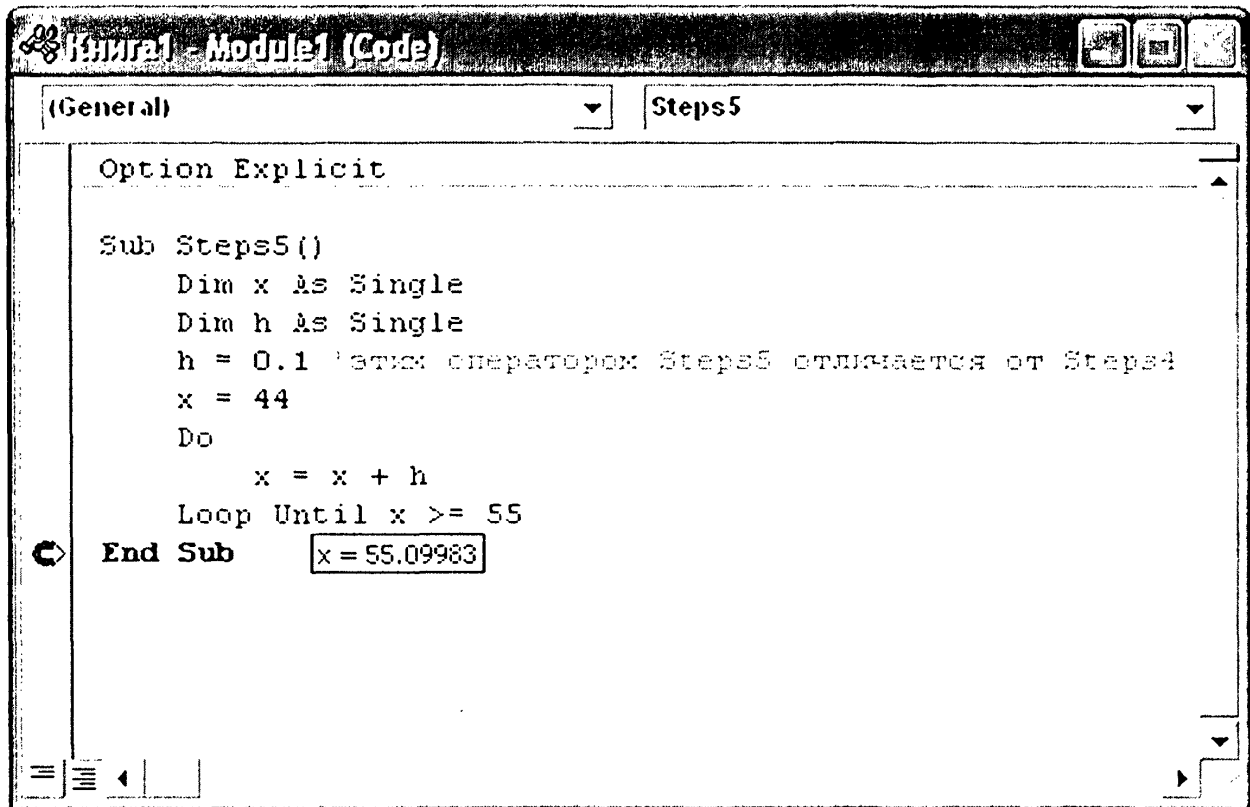


Рис. 14.1. Окно кода с текстом программы Steps5 и конечным значением переменной `x`

объясняется заметное отличие конечного значения `x` от ожидаемого значения 55.

В операторе описания переменной `h` ключевое слово `Single` заменим ключевым словом `Currency`:

```
Sub Steps6()
    Dim x As Single
    Dim h As Currency
    h = 0.1
    x = 44
    Do
        x = x + h
    Loop Until x >= 55
End Sub
```

При выполнении программы Steps6 конечное значение `x` получается равным 55 (рис. 14.2). Объясняется это тем, что в программе Steps6 шаг `h` точно равен 0.1, поэтому в процессе работы программы оператор цикла выполняется нужное число раз.

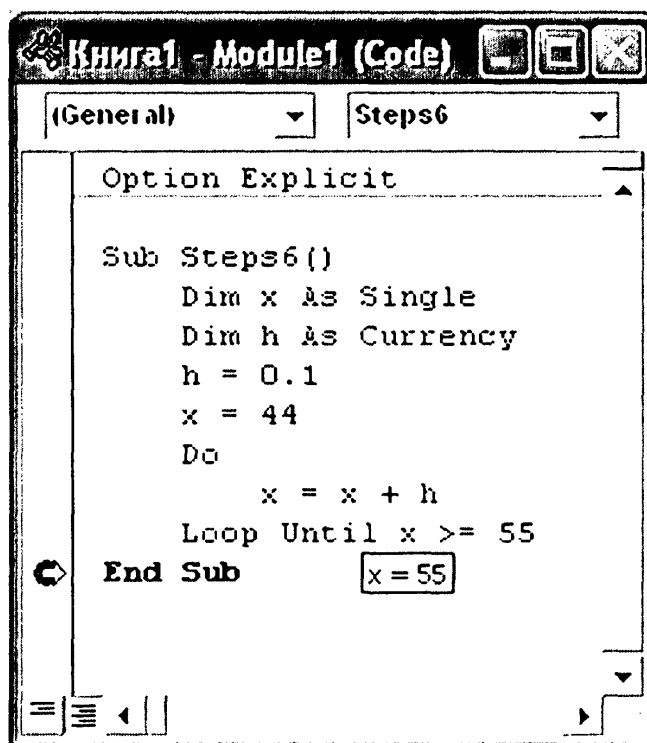


Рис. 14.2. Окно кода с текстом программы Steps6 и конечным значением переменной x

Из сказанного следует, что при работе с вещественными числами обязательно надо учитывать возможную неточность их представления, т. к. она может привести к неожиданным результатам.

15. Массивы

Массивом называется последовательность или таблица переменных одного типа, называемых элементами массива. В обращении к элементу указывается имя массива и один или несколько индексов.

Прежде чем использовать массив, его следует описать (объявить). Кроме того, для каждого индекса должны быть определены нижняя и верхняя границы, в пределах которых индекс может меняться.

Массив занимает $\mathcal{E} \times \text{КЭ} + 4 \times \text{КИ} + 20$ байтов оперативной памяти, где $\mathcal{E}$ — количество байтов для одного элемента массива, КЭ, КИ — количества элементов и индексов.

Существует два вида массивов — статические и динамические. Для каждого вида массивов изложение будем вести по такой схеме: сначала рассмотрим одномерные массивы, имеющие один индекс, а затем — многомерные, имеющие несколько индексов, и одномерные как частный случай многомерных.

Статические массивы. При описании статического одномерного массива задаются нижняя и верхняя границы для индекса, определяющие количество элементов массива, причем заданные границы не могут быть изменены в программе.

Статические массивы описываются в программе так же, как и переменные, т. е. при помощи ключевых слов Dim и As. Границами являются целые числа в круглых скобках. Между нижней и верхней границами ставится ключевое слово To.

Примеры:

```
Dim arrB(1 To 10) As Integer
Dim A(-10 To 10) As String
```

Если в скобках указано только одно целое число, то это — верхняя граница. При этом нижняя граница полагается равной нулю.

Пример:

```
Dim arrA(9) As Byte
```

Этот оператор эквивалентен оператору

```
Dim arrA(0 To 9) As Byte
```

Если нужно, чтобы нижней границей массивов была единица, то перед первой строкой программы следует набрать строку

```
Option Base 1
```

При этом оператор последнего примера будет эквивалентен следующему:

```
Dim arrA(1 To 9) As Byte
```

Значения границ не должны выходить за пределы диапазона значений для данных типа Long (см. приложение 1).

Рассмотрим следующую программу:

```
Sub СтатическиеМассивы()  
    Dim B1(1 To 6) As Byte, S1 As Byte  
    Dim B2(1 To 6) As Integer, S2 As Integer  
    Dim B3(1 To 6) As Long, S3 As Long  
    Dim I As Byte  
    'Определение первых пяти элементов массивов:  
    For I = 1 To 5  
        B1(I) = I  
        B2(I) = I ^ 2  
        B3(I) = I ^ 3  
    Next  
    'Определение шестых элементов массивов:  
    S1 = 0  
    S2 = 0  
    S3 = 0  
    For I = 1 To 5  
        S1 = S1 + B1(I)  
        S2 = S2 + B2(I)  
        S3 = S3 + B3(I)  
    Next  
    B1(6) = S1  
    B2(6) = S2  
    B3(6) = S3  
End Sub
```

В программе СтатическиеМассивы:

- первый оператор цикла используется для определения первых пяти элементов массивов B1, B2 и B3; по окончании это-

го цикла первые пять элементов массивов имеют числовые значения, равные 1-ой, 2-ой и 3-ей степеням значений I: $B1(1) = 1$, $B1(2) = 2$, ..., $B1(5) = 5$; $B2(1) = 1$, $B2(2) = 4$, ..., $B2(5) = 25$; $B3(1) = 1$, $B3(2) = 8$, ..., $B3(5) = 125$;

- второй оператор цикла используется для суммирования определенных ранее элементов массивов; по окончании этого цикла суммы первых пяти элементов массивов B1, B2 и B3 соответственно равны $S1 = 15$, $S2 = 55$ и $S3 = 225$;
- в конце программы шестые элементы массивов полагаются равными найденным значениям сумм: $B1(6) = 15$, $B2(6) = 55$ и $B3(6) = 225$.

Как уже говорилось, массивы бывают одномерными и многомерными. Выше рассмотрены одномерные массивы.

Одномерный массив аналогичен строке или столбцу таблицы Excel (при стиле ссылок R1C1) и вектору в математике.

Для описания многомерных массивов (имеющих несколько индексов) используются конструкции, аналогичные рассмотренной. Отличие состоит в том, что указывается несколько границ, через запятую.

Примеры:

```
Dim A(4, 4) As Byte
Dim B(1 To 5, -5 To -1) As Byte
```

В данных примерах объявленные массивы A и B являются двумерными и содержат одинаковое количество элементов, равное $5 \times 5 = 25$.

Двумерный массив аналогичен прямоугольному диапазону ячеек таблицы Excel (при стиле ссылок R1C1) и матрице в математике.

Обращение к элементу многомерного массива осуществляется при помощи имени массива и индексов, перечисленных через запятую.

Примеры обращений:

```
A(i, j + 1) = 17
B(2, -3) = 25.5
b = C(K, 25)
```

Пример описания трехмерного массива, содержащего $5 \times 5 \times 5 = 125$ элементов:

```
Dim C(1 To 5, -5 To -1, 4) As Byte
```

Максимальное число «измерений» массива очень велико — 60.

Динамические массивы используются в том случае, когда количество элементов массива заранее неизвестно, а определяется в процессе выполнения программы. По окончании работы с динамическим массивом можно освободить память, которую он занимает. Это важно для задач, требующих большого объема оперативной памяти.

Описание динамического массива осуществляется в два этапа.

1. Объявляется массив с использованием оператора Dim, но без указания размерности. Признаком массива являются скобки после его имени.

2. В нужном месте программы описывается данный массив с указанием размерности при помощи оператора ReDim, причем в качестве границ можно использовать не только целые числа, но и арифметические выражения. Важно, чтобы к моменту выполнения оператора ReDim все переменные в этих арифметических выражениях имели числовые значения.

Пример:

```
Sub ДинамическийМассив()  
    Dim A() As Byte           'объявление массива  
    Dim M As Integer  
    Dim N As Integer  
    M = 3  
    ReDim A(-5 To M ^ 2)      'указание размерности  
    For N = -5 To M ^ 2  
        A(N) = N + 30  
    Next  
    ReDim A(5)                'указание размерности  
    N = 0  
    Do  
        A(N) = N ^ 3  
        N = N + 1  
    Loop Until N ^ 2 > 10  
End Sub
```

Задание: после ввода последней программы в окно кода выполните следующие операции.

1. Установите мигающий курсор на переменной A (точнее, слева или справа от A) и выполните **Debug (Отладка) > Add Watch (Добавить контрольное значение)**. Появится окно *Add Watch (Добавление контрольного значения)*, содержащее A в текстовом поле *Expression (Выражение)*.

2. Нажмите кнопку ОК. Появится окно *Watches* (Контрольные значения) со строкой, соответствующей массиву А.

3. Установите мигающий курсор в любом месте текста программы.

4. Произведите пошаговое выполнение программы (рис. 15.1), наблюдая за изменением значений элементов массива А с помощью окна *Watches* (Контрольные значения).

Чтобы в окне *Watches* (Контрольные значения) появились элементы массива А, надо кликнуть на знаке «плюс» слева от А.

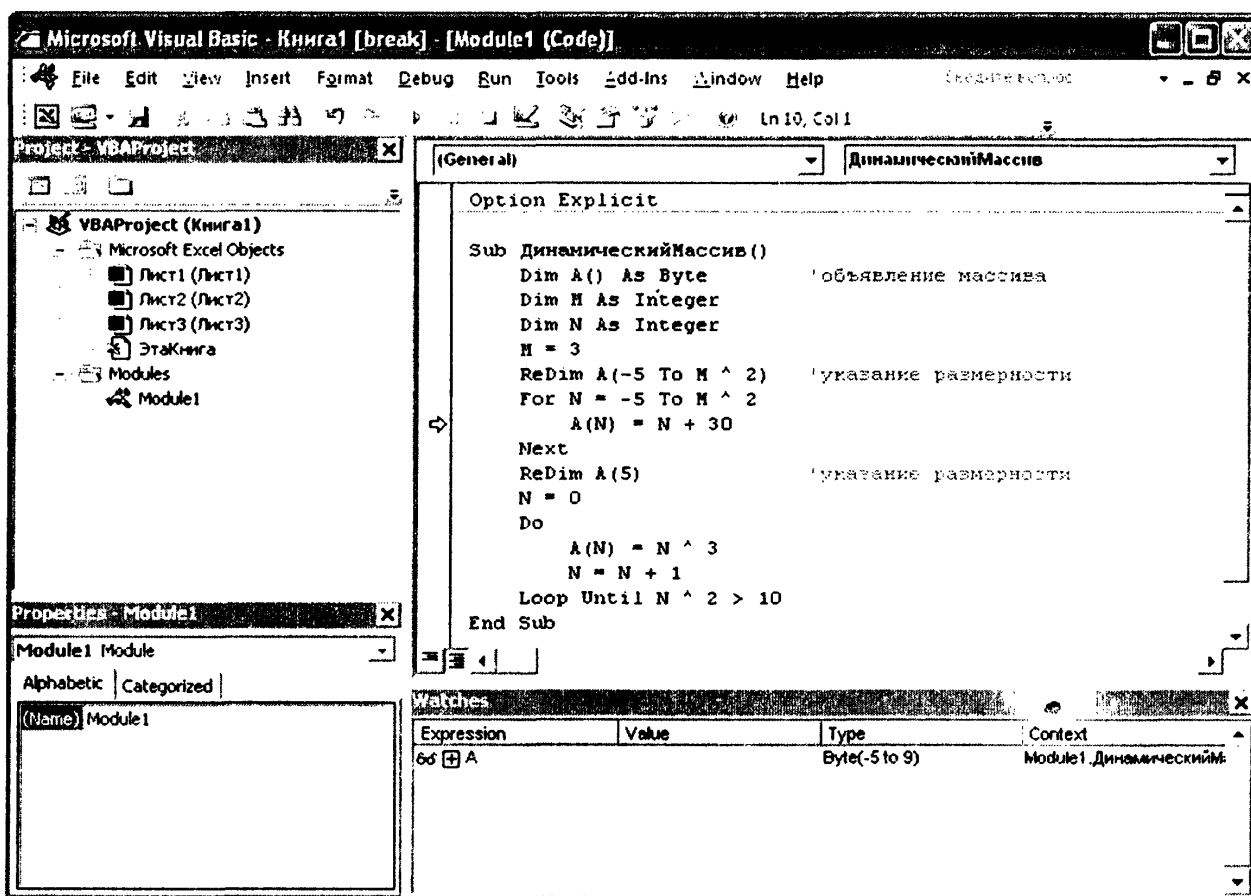


Рис. 15.1. Пошаговое выполнение программы ДинамическийМассив

5. Удалите строку из окна *Watches* (Контрольные значения), кликнув на ней (для выделения) и нажав клавишу **Delete**.

При помощи оператора `ReDim` можно задавать любые размерности массива.

Пример:

```
Dim arrA() As Byte
```

```
ReDim arrA(1, 1)
```

3 зак. 129

'двумерный массив

```
arrA(0, 0) = 13  
arrA(1, 1) = 14  
ReDim arrA(3, 3, 1 To 3)      'трехмерный массив
```

Следует иметь в виду, что при каждом выполнении оператора ReDim (т. е. при каждом переопределении массива) значения элементов массива будут потеряны, т. к. оператор ReDim обнуляет все элементы массива.

Для того чтобы при переопределении массива значения элементов не пропали, следует использовать ключевое слово Preserve.

Пример:

```
Sub Сохранение1()  
    Dim J As Integer  
    Dim arrA() As Integer  
    ReDim arrA(-5 To 1)          'указание размерности  
    For J = -5 To 1  
        arrA(J) = J ^ 2  
    Next J  
    ReDim Preserve arrA(-5 To 4) 'указание размерности  
    For J = 2 To 4  
        arrA(J) = J ^ 3  
    Next J  
End Sub
```

Задание: произведите пошаговое выполнение программы Сохранение1, наблюдая за значениями элементов массива arrA с помощью окна *Watches* (Контрольные значения).

Отметим, что в последней программе ключевое слово Preserve используется при изменении верхней границы одномерного массива. При изменении нижней границы Preserve использовать нельзя.

В случае многомерного массива ключевое слово Preserve можно использовать только при изменении верхней границы последнего «измерения» массива.

Пример:

```
Sub Сохранение2()  
    Dim I As Integer  
    Dim J As Integer  
    Dim arrA() As Integer          'объявление массива  
    ReDim arrA(2, -5 To 1)         'указание размерности
```

```

For I = 0 To 2
    For J = -5 To 1
        arrA(I, J) = (I + 1) * J ^ 2
    Next J
Next I
ReDim Preserve arrA(2, -5 To 4) 'указание размерности
For I = 0 To 2
    For J = 2 To 4
        arrA(I, J) = (I + 1) * J ^ 3
    Next J
Next I
End Sub

```

Задание: произведите пошаговое выполнение программы Сохранение2, наблюдая за значениями элементов массива arrA с помощью окна *Watches (Контрольные значения)*.

Существует возможность определения значений нижней и верхней границ массива любой размерности. Для этого следует использовать функции LBound и UBound соответственно. Описания этих функций можно найти в справочной системе, вызываемой нажатием клавиши F1.

Пример. Использование функций LBound и UBound для определения границ одномерного массива arrA:

```

Dim Low As Long
Dim Up As Long
Low = LBound(arrA)    'Low - нижняя граница массива
Up = UBound(arrA)     'Up - верхняя граница массива

```

Функция UBound необходима, например, когда значение верхней границы одномерного массива неизвестно и при этом необходимо увеличить это значение на определенное число.

Как говорилось выше, преимуществом динамического массива перед статическим является то, что оперативная память, занимаемая динамическим массивом, может быть освобождена. Для этого используется оператор Erase.

Пример:

```

Sub Память()
    Dim A() As Byte
    Dim B() As Byte
    ReDim A(8)          'Память для A: 9 + 4 + 20 = 33 байта
    Erase A             'Память для A: 0 байтов

```

```
ReDim B(2, 2) 'Память для B: 9 + 8 + 20 = 37 байтов
End Sub
```

Задание: после ввода программы Память в окно кода нового модуля выполните следующее.

1. Дважды выполнив **Debug (Отладка) > Add Watch (Добавить контрольное значение)**, в окне *Watches (Контрольные значения)* сгенерируйте строки, соответствующие массивам A и B.

2. Кликните в любом месте текста программы для установки там мигающего курсора.

3. Произведите пошаговое выполнение программы Память, наблюдая за распределением памяти с помощью окна *Watches (Контрольные значения)*.

Для решения многих математических задач требуются массивы случайных чисел.

Пример. Программа, рассчитывающая 10 случайных чисел от 0 до 1, имеет вид

```
Sub СлучайныеЧисла()
    Dim N As Long
    Dim I As Long
    Dim S() As Single
    N = 10
    ReDim S(1 To N)
    Randomize
    For I = 1 To N
        S(I) = Rnd
    Next I
End Sub
```

Случайные числа рассчитываются с помощью функции Rnd (стр. 40). В данном случае обращение к ней производится в цикле For...Next.

Перед серией обращений к функции Rnd должен находиться оператор Randomize.

Задание: выполните последнюю программу пошагово, наблюдая за значениями элементов массива S с помощью окна *Watches (Контрольные значения)*.

16. Пользовательские процедуры

Блок операторов, предназначенный для многократного выполнения в разных точках программы, целесообразно оформить как процедуру. При этом блок записывается один раз и ему присваивается имя с параметрами (формальными). Эта запись блока операторов называется описанием процедуры.

В общем случае процедура имеет:

- входные параметры, которые считаются заданными;
- выходные параметры, рассчитываемые в ходе выполнения блока операторов.

После того, как произведено описание процедуры, в программу помещаются обращения к этой процедуре с нужными параметрами (фактическими). Эти обращения помещаются в те точки программы, в которых по смыслу должен присутствовать блок операторов, оформленный как процедура.

Процедуры делятся на функции и подпрограммы. Функции можно использовать в арифметических и логических выражениях и строках (т. к. функция принимает значение); подпрограммы в выражениях и строках использовать нельзя. В этом состоит основное отличие функций от подпрограмм.

Описание пользовательской *функции* имеет следующий синтаксис:

```
Function название (формальные_параметры) [As тип]  
    операторы  
End Function
```

где *название* — имя функции; *формальные_параметры* — имена параметров, перечисленные через запятую; *тип* — тип значения функции; *операторы* — блок операторов.

В блоке *операторы* обязательно должен присутствовать хотя бы один оператор присваивания, в левой части которого (слева от знака присваивания =) находится имя функции *название*.

Обращение к функции (находящееся в программе) имеет вид

название (*фактические_параметры*)

где *фактические_параметры* — массивы, переменные, константы, числа и/или строки, перечисленные через запятую. В числе фактических параметров могут быть записи, которые мы рассмотрим позже.

В результате обращения к функции *название* в программу возвращается значение этой функции, соответствующее заданным параметрам.

Рассмотрим следующий код программы и описания функции, находящихся в одном модуле:

```
Sub Program1()  
    Dim L As Long  
    Dim W As Double  
    L = Fact(12)  
    W = 4.2 + Fact(10) / 2  
End Sub  
  
Function Fact(N) As Long           'N < 13  
    Dim I As Byte  
    Dim J As Long  
    J = 1  
    For I = 1 To N  
        J = J * I  
    Next I  
    Fact = J  
End Function
```

Первая группа операторов — это программа Program1. Вторая группа операторов — описание функции Fact, которая рассчитывает факториал целого положительного числа N, являющегося формальным параметром.

В программе два обращения к функции Fact, с фактическими параметрами 12 и 10.

1. Обращение к функции Fact фигурирует в правой части оператора присваивания $L = \text{Fact}(12)$. В результате выполнения этого оператора значение функции Fact при $N = 12$ (т. е. значение, возвращаемое функцией Fact в программу) присваивается переменной L.

2. Обращение к функции Fact фигурирует в арифметическом выражении $4.2 + \text{Fact}(10) / 2$. Значение этого арифметического выражения присваивается переменной W.

Описание пользовательской *подпрограммы* имеет следующий синтаксис:

```
Sub название (формальные_параметры)
    операторы
End Sub
```

где *название* — имя подпрограммы; *формальные_параметры* — имена параметров, перечисленные через запятую; *операторы* — блок операторов.

Имеется два эквивалентных оператора обращения к подпрограмме:

```
Call название (фактические_параметры)
название фактические_параметры
```

где *фактические_параметры* — список фактических параметров, как в обращении к функции. При наличии ключевого слова `Call` список *фактические_параметры* заключается в скобки, в отсутствие `Call` скобки не ставятся.

Рассмотрим следующий код программы и описания подпрограммы, находящихся в одном модуле:

```
Sub Program2 ()
    Dim aa As Single
    Dim bb As Single
    Dim cc1 As Single
    Dim cc2 As Single
    Dim cc3 As Single
    aa = 3
    bb = 4
    Call Hypotenuse(aa, bb, cc1)
                                '1-е обращение к подпрограмме
    Call Hypotenuse(3, 4, cc2)
                                '2-е обращение к подпрограмме
    Hypotenuse aa, bb, cc3
                                '3-е обращение к подпрограмме
End Sub

Sub Hypotenuse(ByVal A, ByVal B, ByRef C)
    C = Sqr(A ^ 2 + B ^ 2)
End Sub
```

Первая группа операторов — это программа Program2, вторая — описание подпрограммы Hypotenuse, рассчитывающей длину гипотенузы прямоугольного треугольника.

В программе Program2 имеется три оператора обращения к подпрограмме Hypotenuse, причем два из них содержат ключевое слово Call, а одно не содержит.

Формальные параметры А и В (в описании подпрограммы) являются входными. Это — длины катетов. Ключевое слово ByVal перед А и В в первой строке подпрограммы означает, что эти параметры вызываются по значению (value — значение). В этом случае при обращении к подпрограмме ей передаются значения А и В. Это — соответственно 3 и 4 при всех трех обращениях к подпрограмме Hypotenuse.

Формальный параметр С является выходным. Это — длина гипотенузы. Ключевое слово ByVal перед С в первой строке подпрограммы означает, что параметр С вызывается по ссылке (reference — ссылка). В этом случае при обращении к подпрограмме ей передается адрес ячейки оперативной памяти, соответствующей переменной cc1 (при первом обращении к подпрограмме Hypotenuse), cc2 (при втором обращении) или cc3 (при третьем обращении).

Ключевое слово ByVal можно опускать.

В рассмотренном коде параметры А, В и С — простые переменные (не массивы). Если параметрами подпрограммы являются массивы, то все они (и входные, и выходные) вызываются по ссылке.

Рассмотрим следующий код:

```
Dim N1 As Integer
Sub Program3()
    Dim xx(50) As Double, yy(50) As Double
    Dim i As Integer
    N1 = 3
    For i = N1 To 30
        xx(i) = 0.1 * i
    Next i
    Call XSINX(30, xx, yy) 'обращение к подпрограмме
End Sub

Sub XSINX(ByVal N2, ByVal X() As Double, _
    ByVal F() As Double)
```

```
Dim j As Integer
For j = N1 To N2
    F(j) = X(j) * Sin(X(j))
Next j
End Sub
```

Кроме использования массивов в качестве параметров подпрограммы, в последнем коде есть еще нечто новое: переменная N1 объявлена перед первой строкой программы. Это сделано для того, чтобы переменная N1 была «видима» (в плане возможности использования) и в программе, и в подпрограмме. Вернемся к вопросу объявления переменных.

До сих пор мы говорили о том, как объявлять переменные, но не говорили, где их объявлять. Их можно объявлять в двух местах:

- внутри программы или процедуры;
- в верхней части окна кода, которая называется областью общих объявлений модуля.

Место объявления переменной определяет область ее использования. Если, например, переменная объявлена в процедуре (как переменная j в последнем коде), то только эта процедура ее видит. Другие процедуры (если они есть) и программа не могут использовать значение этой переменной и менять его.

Такую переменную называют *локальной*. Говорят также, что переменная видима на уровне процедуры.

Чтобы значение переменной было доступно всем процедурам данного модуля, ее надо объявить в области общих объявлений модуля (как объявлена переменная N1 в последнем коде). Тогда программа и все процедуры, определенные в данном модуле, могут использовать значение этой переменной и менять его.

Такую переменную называют *глобальной*. Говорят также, что переменная видима на уровне модуля.

Сказанное относительно переменных относится и к пользовательским константам, но значение константы, естественно, нельзя менять.

Задание: произведите выполнение программ Program1, Program2 и Program3 с помощью клавиш F8 и Ctrl + F8.

Перед первым нажатием клавиши F8 мигающий курсор должен находиться в тексте программы, а не процедуры.

Среди параметров процедуры (функции или подпрограммы), кроме обязательных параметров, могут быть и необязательные, которые могут отсутствовать в обращении к процедуре.

Объявление необязательности параметра осуществляется при помощи ключевого слова `Optional`, которое ставится перед именем этого параметра в первой строке описания процедуры. Необязательный параметр должен иметь тип `Variant`.

Пример:

```
Sub Program4()  
    Dim vntA As Byte  
    Dim vntB As Byte  
    Dim C As Integer  
    vntA = 5  
    vntB = 10  
    C = Apt(vntA, vntB) 'Результат: vntB = 6, C = 625  
    vntB = 10  
    C = Apt(vntA)       'Результат: vntB = 10, C = 625  
End Sub  
  
Function Apt(ByVal a, Optional b As Variant)  
    If Not IsMissing(b) Then b = a + 1  
    Apt = a ^ 4  
End Function
```

В этом примере параметр `b` функции `Apt` является необязательным, о чем говорит `Optional` перед `b`. Словосочетание `As Variant` после `b` можно опустить.

В описании функции `Apt`:

- `Not` — логическая операция;
- `IsMissing` — функция.

Значение `IsMissing(b)` равно `True` при отсутствии второго параметра в обращении к `Apt` и `False` при наличии этого параметра.

Задание: произведите пошаговое выполнение программы `Program4`, наблюдая за изменением значений переменных `vntB` и `C`.

Существует возможность указывать значение, которое должен принимать необязательный параметр при его отсутствии в обращении к процедуре.

Пример:

```
Sub Program5()  
    Dim vntA As Byte  
    Dim vntB As Byte
```

```
Dim vntC As Byte
vntA = 5
vntB = 10
Call Opt(vntA, vntC, vntB)      'Результат: vntC = 15
vntB = 10
Call Opt(vntA, vntC)           'Результат: vntC = 8
End Sub

Sub Opt(ByVal a, c, Optional b = 3)
    c = a + b
End Sub
```

В приведенном примере параметр *b* подпрограммы *Opt* является необязательным, причем если в обращении к *Opt* присутствуют только два фактических параметра, то при выполнении оператора *c = a + b* полагается *b = 3*.

Задание: произведите пошаговое выполнение программы *Program5*, наблюдая за изменением значения переменной *vntC*.

Рассмотренные программы использовали по одной пользовательской процедуре. Так, например, программа *Program1* использовала только функцию *Fact* (с помощью этой функции рассчитывались значения *L* и *W*). Однако пользовательских процедур может быть несколько.

Программу и описания пользовательских процедур, ею используемых, также можно назвать программой. Так, например, программу *Program1* и описание функции *Fact* можно назвать программой *Program1*.

17. Использование стандартных окон операционной системы Windows

В VB имеется большое количество встроенных процедур, которые отличаются от пользовательских процедур тем, что их описания были запрограммированы разработчиками Visual Basic. От пользователя компьютера тексты встроенных процедур скрыты.

Встроенные процедуры, как и пользовательские, делятся на функции и подпрограммы.

Со встроенными функциями мы уже неоднократно встречались. К ним относятся:

- функции, рассмотренные в главе 9;
- функция `IIf` в главе 12;
- функция `IsMissing` в главе 16 и др.

Ниже рассмотрим встроенную функцию `InputBox`, предназначенную для ввода информации (в программу) с помощью стандартных окон Windows.

Примером встроенной подпрограммы является подпрограмма `MsgBox`, предназначенная для вывода информации (из программы) в стандартные окна Windows.

Используем встроенные процедуры `InputBox` и `MsgBox` в программе расчета длины гипотенузы прямоугольного треугольника по длинам катетов.

1. В окно кода введем программу

```
Sub Pythagoras()  
    Dim a As Single  
    Dim b As Single  
    Dim c As Single  
    Dim s As String  
1:  s = _  
    InputBox("Введите длину первого катета и нажмите OK")  
2:  a = Val(s)  
3:  s = _  
    InputBox("Введите длину второго катета и нажмите OK")
```

```
4: b = Val(s)
5: c = Sqr(a ^ 2 + b ^ 2)
6: s = Str(c)
7: MsgBox s
End Sub
```

2. Запустим программу Pythagoras. Появится окно, предлагающее ввести длину первого катета (рис. 17.1). С помощью клавиатуры введем, например, значение 400 (без кавычек) в текстовое поле этого окна и нажмем кнопку **OK**.

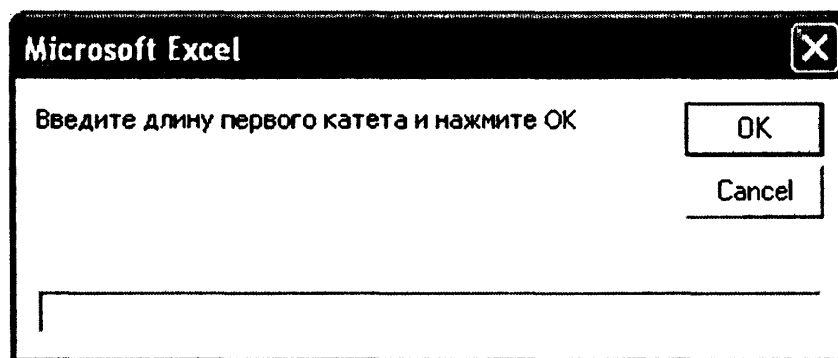


Рис. 17.1. Первое окно для ввода исходных данных

3. Появится окно, предлагающее ввести длину второго катета (рис. 17.2). Введем, например, значение 300; после этого нажмем кнопку **OK**.

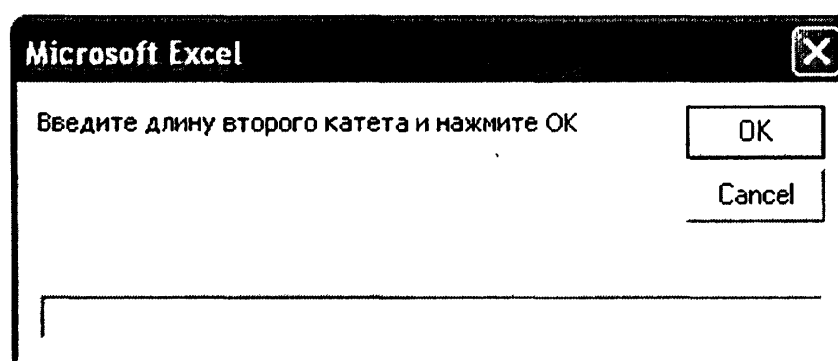


Рис. 17.2. Второе окно для ввода исходных данных

4. Появится окно, содержащее рассчитанное значение длины гипотенузы (рис. 17.3).

5. Кликнем на кнопке **OK** для закрытия окна и завершения выполнения программы Pythagoras.

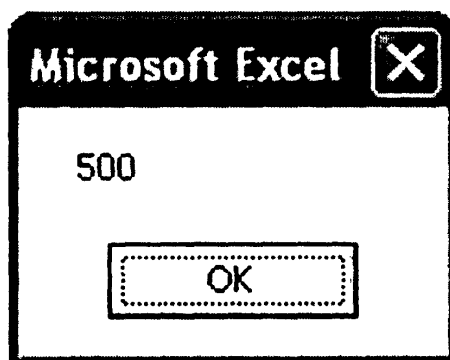


Рис. 17.3. Окно с результатом расчета

Первый оператор программы (с меткой 1) содержит функцию `InputBox`. Она используется для ввода информации с клавиатуры. Возвращает эта функция строку, введенную в текстовое поле окна на рис. 17.1, т. е. "400". Эта строка присваивается переменной `s` строкового типа.

Второй оператор преобразует значение строки `s` в число 400 и присваивает это значение переменной числового типа `a`.

Третий оператор присваивает переменной `s` строку "300", введенную в текстовое поле окна на рис. 17.2.

Четвертый оператор преобразует значение строки `s` в число 300 и присваивает это значение переменной `b`.

Пятый оператор рассчитывает длину гипотенузы по теореме Пифагора. Полученное значение 500 присваивается переменной числового типа `c`.

Шестой оператор преобразует значение 500 в строку. В результате выполнения этого оператора переменная `s` принимает значение "500".

Седьмой оператор, представляющий собой обращение к подпрограмме `MsgBox`, выводит на экран окно с рассчитанным значением 500 (рис. 17.3).

В обращениях к процедурам `InputBox` и `MsgBox` фигурирует один параметр (строкового типа), который является обязательным. Однако параметров может быть несколько. Назначение необязательных параметров можно посмотреть в справочной системе, вызываемой нажатием клавиши **F1**. Перед нажатием этой клавиши рекомендуем мигающий курсор установить на нужном слове (`InputBox` или `MsgBox`) в окне кода с программой `Pythagoras`, кликнув на этом слове.

Отметим, что в справочной системе подпрограмма `MsgBox` называется функцией, т. к. ее можно использовать в арифметиче-

ских выражениях. При этом в программу возвращается значение типа Integer, соответствующее кнопке, на которой кликнул пользователь. Приведем пример использования MsgBox в качестве функции.

В программу Pythagoras введем расчет площади прямоугольного треугольника. Для этого седьмой оператор заменим тремя операторами:

```
Sub Pythagoras()  
    Dim a As Single  
    Dim b As Single  
    Dim c As Single  
    Dim s As String  
1:   s = _  
        InputBox("Введите длину первого катета и нажмите OK")  
2:   a = Val(s)  
3:   s = _  
        InputBox("Введите длину второго катета и нажмите OK")  
4:   b = Val(s)  
5:   c = Sqr(a ^ 2 + b ^ 2)  
6:   s = Str(c)  
    Dim Ret As Integer  
    Ret = MsgBox(s, vbYesNo, "Рассчитать площадь?")  
    If Ret = vbYes Then MsgBox Str(a * b / 2)  
End Sub
```

При выполнении полученной программы изменится содержимое окна с результатом расчета длины гипотенузы (рис. 17.4). При клике на кнопке Yes появится окно с результатом расчета площади треугольника (рис. 17.5).

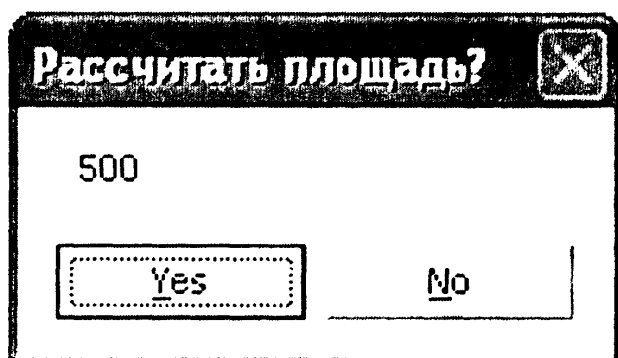


Рис. 17.4. Первое окно с результатом расчета

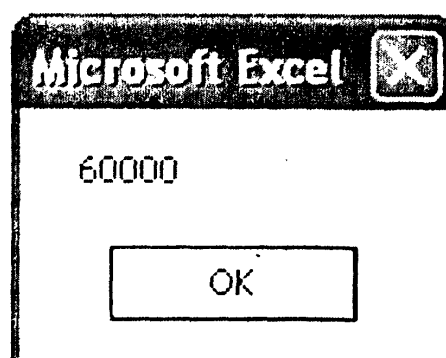


Рис. 17.5. Второе окно с результатом расчета

18. Записи

В том случае, когда несколько переменных различных типов по смыслу связаны между собой, их целесообразно объединить в так называемую запись, которую также называют пользовательским типом данных. Каждая переменная, являющаяся составной частью записи, называется полем.

Прежде чем использовать запись, ее необходимо создать (определить) с помощью оператора `Type`, имеющего следующий синтаксис:

```
Type название
    поле1 As тип1
    поле2 As тип2
    .
    .
    .
    полеN As типN
End Type
```

где *название* — название пользовательского типа данных; *поле1*, *поле2*, ..., *полеN* — имена полей; *тип1*, *тип2*, ..., *типN* — типы данных для соответствующих полей.

Рассмотренный оператор создания (определения) записи помещается перед первой строкой программы.

Пусть, например, необходимо обрабатывать информацию о результатах сессии в университете, в которой оценки ставились по трем предметам — физике, математике и информатике. В этом случае понадобятся такие поля:

- 1) фамилия, имя и отчество студента;
- 2) номер зачетной книжки;
- 3) оценка по физике;
- 4) оценка по математике;
- 5) оценка по информатике.

Пусть в рассматриваемом примере название создаваемой записи будет `Session`, имена полей — `Name`, `Number`, `Physics`, `Math` и `Inform`. Тогда оператор создания записи имеет вид

```
Type Session
  Name As String
  Number As Long
  Physics As Byte
  Math As Byte
  Inform As Byte
End Type
```

Для того чтобы получить доступ к созданной записи, необходимо одну или несколько переменных в программе объявить (описать) с помощью оператора `Dim`, как объявляли переменные в главе 3:

```
Dim переменная As название
```

где *переменная* — имя переменной; *название* — название пользовательского типа данных.

Пример:

```
Dim Sess As Session, BestSess As Session
```

В примере описаны переменные `Sess` и `BestSess`, тип этих переменных — `Session`. Говорят также, что `Sess` и `BestSess` — записи типа `Session`.

Обращение к полю записи осуществляется при помощи имени переменной и имени поля (после точки).

Пример:

```
Sess.Name = "Захаркин Максим Сергеевич"
```

В результате выполнения этого оператора полю `Name` переменной `Sess` (типа `Session`) присваивается значение "Захаркин Максим Сергеевич".

Для заполнения полей удобно использовать оператор `With`, который имеет следующий синтаксис:

```
With переменная
  .поле1 = выражение1
  .поле2 = выражение2
  .
  .
  .полеN = выражениеN
End With
```

Пример:

```
With Sess
    .Name = "Захаркин Максим Сергеевич"
    .Number = 02237
    .Physics = 4
    .Math = 5
    .Inform = 5
End With
```

Оператор присваивания можно применять как к полям, так и к записям целиком.

Примеры:

```
BestSess.Number = Sess.Number
BestSess = Sess
```

Часто удобно использовать массивы записей. Так, например, результаты сессии студентов группы Э14 естественно хранить в массиве SessЭ14. Если этот массив записей является статическим, то объявляется он следующим образом:

```
Dim SessЭ14(1 To 31) As Session
```

Пример. Программа расчета расстояния между точками, включающая в себя оператор создания записи Point, программу Расстояние и описание подпрограммы Distance, имеет вид

```
Type Point                                'point - точка
    Name As String                        'name - имя
    x As Single
    y As Single
End Type

Sub Расстояние()
    Dim P() As Point
    Dim L As Single
    ReDim P(0 To 1)
    With P(0)
        .Name = "Тушино"
        .x = 17.6
        .y = 29.7
    End With
    With P(1)
        .Name = "Останкино"
```

```
.x = 44.1
.y = 37.5
End With
Call Distance(P(0), P(1), L) 'distance - расстояние
MsgBox Str(L)                'вывод значения L
End Sub

Sub Distance(A As Point, B As Point, S As Single)
    S = Sqr((A.x - B.x) ^ 2 + (A.y - B.y) ^ 2)
End Sub
```

Задание: разберитесь в работе этой программы, отобразив массив P и переменную L в окне визуализации текущих значений переменных *Watches* (*Контрольные значения*).

Особенности программы Расстояние:

- массив записей P является динамическим;
- первые два параметра подпрограммы Distance — записи типа Point.

19. Работа со строками

Строкой будем называть не только последовательность символов, заключенную в кавычки, но и переменную строкового типа, объявленную с помощью ключевого слова `String`.

Пример:

```
Dim A As String
Dim B As String * 15
A = "Информатика"
B = "Информатика"
```

В примере `A`, `B` и `"Информатика"` являются строками.

Строка как переменная может иметь переменную или постоянную длину (см. приложение 1).

- Строка переменной длины характеризуется тем, что занимаемый ею объем оперативной памяти может меняться в процессе выполнения программы. В примере `A` — строка переменной длины. Согласно приложению 1, после выполнения оператора `A = "Информатика"` строка `A` занимает 21 байт оперативной памяти.
- Строка постоянной длины занимает фиксированный объем оперативной памяти. При ее объявлении после ключевого слова `String` ставится звездочка и указывается объем оперативной памяти (в байтах), выделяемый этой строке. В примере `B` — строка постоянной длины, занимающая 15 байтов оперативной памяти.

Количество символов значения, присваиваемого строке постоянной длины, может быть как меньше указанного в описании (пятнадцати), так и больше. В первом случае в конец строки вместо недостающих символов автоматически будут добавлены пробелы, а во втором случае лишние символы будут автоматически удалены.

Для объединения двух и более строк используется символ `&` (амперсанд) или знак «плюс». Результатом объединения строк является строка.

Пример:

```
Sub Строки1()  
    Dim strA As String  
    Dim strB As String  
    Dim strC As String  
    strA = "Строковая "  
    strB = "переменная"  
    strC = strA & strB  
        'Результат: strC = "Строковая переменная"  
    strC = strA + strB  
        'Результат: strC = "Строковая переменная"  
    strC = strA & "переменная"  
        'Результат: strC = "Строковая переменная"  
    strC = "Строковая " + strB  
        'Результат: strC = "Строковая переменная"  
End Sub
```

Благодаря операции объединения строк, строку можно рассматривать как выражение, аналогичное арифметическим и логическим выражениям.

При работе со строками используются три функции удаления пробелов:

- LTrim удаляет все пробелы в начале строки (слева — left);
- RTrim удаляет все пробелы в конце строки (справа — right);
- Trim удаляет все пробелы в начале и конце строки.

Пример:

```
Sub Строки2()  
    Dim strA As String  
    Dim strB As String  
    strA = "    Строковая переменная    "  
    strB = LTrim(strA)  
        'Результат: strB = "Строковая переменная    "  
    strB = RTrim(strA)  
        'Результат: strB = "    Строковая переменная"  
    strB = Trim(strA)  
        'Результат: strB = "Строковая переменная"  
End Sub
```

Часто бывает необходимо преобразовать число в строку. Для этого используется функция CStr или Str. Для обратного преоб-

разования используется функция Val. Эти функции рассмотрены в главах 4 и 8.

Функция Space возвращает в программу строку, состоящую из пробелов; их количество определяется значением аргумента.

Пример:

```
Sub Строки3()  
    Dim strA As String  
    Dim curB As Currency  
    Dim strC As String  
    strA = "X = "  
    curB = 45.77  
    strC = strA & CStr(curB)  
    'Результат: strC = "X = 45.77"  
    curB = Val("45.77 = X")  
    'Результат: curB = 45.77  
    curB = Val(strC)          'Результат: curB = 0  
    strC = "Строковая" & Space(3) & "переменная"  
    'Результат: strC = "Строковая   переменная"  
End Sub
```

Существует возможность преобразования строки таким образом, чтобы все используемые в ней буквы стали заглавными (т. е. буквами верхнего регистра — upper case) или строчными (т. е. буквами нижнего регистра — lower case). Для этого используются функции UCase и LCase соответственно.

Пример:

```
Sub Строки4()  
    Dim strA As String  
    Dim strB As String  
    strA = "Павел Иванов"  
    strB = UCase(strA) 'Результат: strB = "ПАВЕЛ ИВАНОВ"  
    strB = LCase(strA) 'Результат: strB = "павел иванов"  
End Sub
```

Для того чтобы заменить какую-либо часть строки или определенные символы, следует использовать функцию Replace.

Пример:

```
Sub Строки5()  
    Dim strA As String  
    Dim strB As String
```

```
strA = "Павел Иванов"
strB = Replace(strA, "Иванов", "Гусев")
      'Результат: strB = "Павел Гусев"
End Sub
```

Для определения количества символов в строке (не считая кавычек) применяется функция `Len`, причем аргументом является строка.

Пример:

```
Sub Строки6()
    Dim strA As String
    Dim intA As Integer
    strA = "Строковая переменная"
    intA = Len(strA)          'Результат: intA = 20
End Sub
```

Функции `Replace` и `Len` имеют и другие возможности, которые можно изучить с помощью справочной системы, вызываемой нажатием клавиши **F1**.

Нередко возникает необходимость выделения из строки той или иной ее части. В этом случае следует использовать одну из функций `Left`, `Right` и `Mid` (*middle* — середина).

Обращения к функциям `Left` и `Right` имеют следующий вид:

```
Left(строка, количество)
Right(строка, количество)
```

Эти функции возвращают строку, содержащую указанное количество символов *количество* начала и конца строки *строка* соответственно.

Обращение к функции `Mid` имеет вид

```
Mid(строка, номер[, количество])
```

Эта функция возвращает строку, содержащую указанное количество символов строки *строка*, начиная с символа с заданным порядковым номером *номер*. Если количество символов не указывается, то функция `Mid` возвращает все символы до конца строки.

Пример:

```
Sub Строки7()
    Dim strA As String
    Dim strB As String
    strA = "Моя строковая переменная"
```

```
strB = Left("Моя строковая переменная", 3)
        'Результат: strB = "Моя"
strB = Right(strA, 10)
        'Результат: strB = "переменная"
strB = Mid(strA, 5, 9)
        'Результат: strB = "строковая"
strB = Mid(strA, 15)
        'Результат: strB = "переменная"
End Sub
```

Рассмотрим пример использования сложных строк.

В программе расчета расстояния между точками Расстояние оператор MsgBox Str(L) на стр. 83 заменим операторами

```
Dim S As String
Dim M As Single
S = InputBox("Введите масштаб и нажмите OK")
        'ввод масштаба, рис. 19.1
M = Val(S)
L = M * L
MsgBox "Расстояние от " & P(0).Name & " до " & _
        P(1).Name & vbCrLf & "составляет" & Str(L) & " км"
        'вывод расстояния, рис. 19.2
```

Запустим полученную программу, кликнув на стрелке на панели инструментов.

В текстовое поле появившегося окна (рис. 19.1) введем 0.4 и нажмем кнопку **ОК** для продолжения выполнения программы.

Для окончания выполнения программы нажмем кнопку **ОК** в появившемся окне с результатом расчета (рис. 19.2).

В приведенном фрагменте программы фигурируют обращения к процедурам InputBox и MsgBox, входящим в состав VB (см. гла-

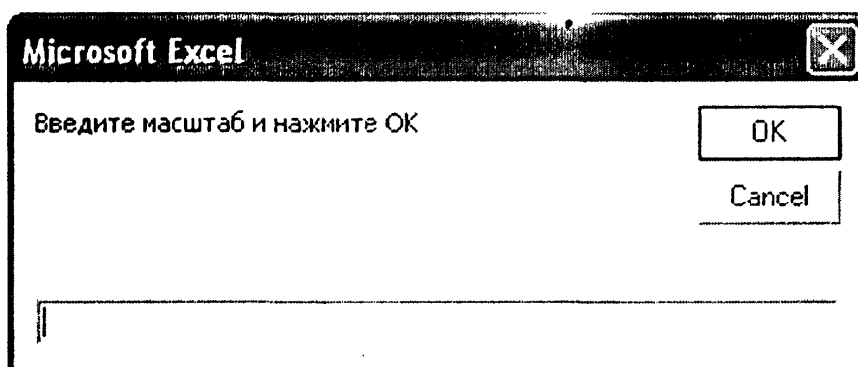


Рис. 19.1. Окно для ввода исходных данных

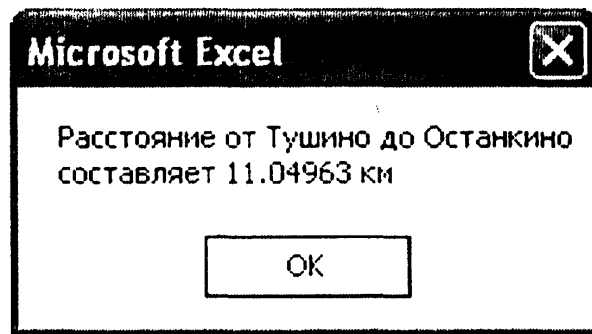


Рис. 19.2. Окно с результатом расчета

ву 17). Встроенная функция `InputBox` используется для ввода информации с клавиатуры; эта функция возвращает строку, введенную в текстовое поле. Встроенная подпрограмма `MsgBox` используется для вывода информации на экран дисплея. Параметром этой подпрограммы является сложная строка (выражение), в состав которой входит константа `vbCrLf`.

Благодаря использованию константы `vbCrLf`, информация в окне с результатом расчета (рис. 19.2) занимает две строки. Константу `vbCrLf` можно заменить объединением констант «возврат» и «новая строка» `vbCr & vbLf`.

Полезной является функция `TimeValue`, преобразующая время из формата строки в формат даты. В качестве примера использования этой функции рассмотрим следующую программу.

```
Sub TV()  
    Dim dTime1 As Date  
    Dim dTime2 As Date  
    Dim dTime3 As Date  
    dTime1 = #2:30:45# + TimeValue("00:15:00")  
                'Результат: dTime1 = 2:45:45  
    dTime2 = Now + TimeValue("00:25:00")  
    dTime3 = Time + TimeValue("00:00:10")  
End Sub
```

В результате выполнения второго оператора присваивания переменная `dTime2` будет иметь значение, равное текущим дате и времени плюс 25 минут. В результате выполнения третьего оператора присваивания переменная `dTime3` будет иметь значение, равное текущему времени плюс 10 секунд.

20. Работа с текстовыми файлами

Файлом называется область на жестком диске (винчестере), диске, компакт-диске или каком-либо другом носителе информации, содержащая однотипную информацию и имеющая название. При работе программы с файлами основными являются операции считывания информации из файла и записи информации в файл.

Существует несколько разновидностей файлов. Мы рассмотрим текстовые файлы. Их содержимым является последовательность строк произвольной длины (без кавычек). В конце каждой строки находится сочетание символов «возврат» и «новая строка», означающее окончание этой строки.

Текстовые файлы мы будем изучать с помощью редактора Блокнот (Notepad), входящего в состав операционной системы Windows. При этом символы «возврат» и «новая строка» мы не увидим.

Для начала работы с текстовым файлом используется оператор открытия файла `Open`, имеющий следующий синтаксис:

`Open имя For назначение As номер`

Здесь *имя* — полное имя файла (строка, содержащая имя файла с путем и расширением), *номер* — номер файла, *назначение* — это ключевое слово `Input`, `Output` или `Append`.

- `Input` означает, что файл открывается для считывания из него информации.
- `Output` — файл открывается для записи в него информации.
- `Append` — файл открывается для добавления информации.

В качестве номера файла *номер* рекомендуем использовать переменную типа `Integer`, значение которой задается с помощью оператора присваивания

`номер = FreeFile`

Этот оператор должен находиться перед оператором `Open`; `FreeFile` — функция, возвращающая в программу незанятый номер файла.

По окончании работы с файлом его следует закрыть с помощью оператора Close. Синтаксис:

Close *номер*

Для добавления в файл новых строк используется оператор Print, имеющий следующий синтаксис:

Print #*номер*, *строка*

Пусть файл с указанным номером *номер* открыт при помощи ключевого слова Output. При выполнении оператора Print в начало файла записывается строка *строка*. При повторном выполнении оператора Print в файл добавляется вторая строка и т. д.

Пусть файл с указанным номером открыт при помощи ключевого слова Append. При выполнении оператора Print строка *строка* добавляется в конец файла.

Рассмотрим два способа извлечения информации из текстового файла, открытого при помощи ключевого слова Input.

Первый способ извлечения информации — с помощью оператора Line Input, имеющего синтаксис

Line Input #*номер*, *переменная*

Этот оператор производит считывание из файла (с заданным номером *номер*) очередной строки, которая присваивается переменной строкового типа *переменная*.

Второй способ извлечения информации из файла — с помощью функции

Input (*количество*, *номер*)

Эта функция возвращает в программу строку, содержащую указанное количество символов *количество* из файла с указанным номером *номер*.

В программах Создание и Добавление, приведенных ниже, используются рассмотренные операторы и функции работы с файлами. На примере этих программ рассмотрим и другие полезные операторы и функции.

Sub Создание ()

Dim FName1 As String

Dim FName2 As String

Dim FNum1 As Integer

```
Dim FNum2 As Integer
Dim n1 As Long
Dim n2 As Long
Dim strA As String          'вспомогательная строка
MkDir("c:\Documents and Settings\usr\texts")
                                'создание папки texts
FName1 = "c:\Documents and Settings\usr\texts\a.txt"
                                'полное имя 1-го файла
FName2 = "c:\Documents and Settings\usr\texts\b.txt"
                                'полное имя 2-го файла

'Создание файла a.txt:
FNum1 = FreeFile
Open FName1 For Output As FNum1
                                'открытие файла a.txt
strA = "Создан текстовый файл,"
Print #FNum1, strA
Print #FNum1, "который состоит из нескольких строк."
'Определение количества символов в файле a.txt:
n1 = LOF(FNum1)
Close FNum1                    'заккрытие файла a.txt
MsgBox "В файле a.txt" & Str(n1) & " символов"
'Копирование информации из файла a.txt в файл b.txt:
FNum1 = FreeFile
Open FName1 For Input As FNum1
                                'открытие файла a.txt

FNum2 = FreeFile
Open FName2 For Output As FNum2
                                'открытие файла b.txt
Do Until EOF(FNum1)            'цикл считывания-записи
    Line Input #FNum1, strA
    Print #FNum2, strA
Loop
Close FNum1                    'заккрытие файла a.txt
'Добавление новой строки в файл b.txt:
strA = "Добавлена новая строка."
Print #FNum2, strA
'Определение количества символов в файле b.txt:
n2 = LOF(FNum2)
Close FNum2                    'заккрытие файла b.txt
FNum2 = FreeFile
Open FName2 For Input As FNum2
                                'открытие файла b.txt
```

```
strA = Input(1, FNum2)      'считывание символа
Close FNum2                 'заккрытие файла b.txt
MsgBox "В файле b.txt" & Str(n2) & " символов, " _
    & "причем первый символ - " & strA
End Sub
```

Перед выполнением программы Создание в ее текст вместо `usr` следует ввести конкретное имя пользователя.

Оператор `MkDir("c:\Documents and Settings\usr\texts")` создает папку `texts` внутри папки `usr`. (При повторном запуске программы Создание оператор создания папки `MkDir` следует убрать, например, пометив его апострофом.)

Функция `LOF` возвращает в программу количество символов в файле, причем символы «возврат» и «новая строка» (которыми оканчивается каждая строка) принимаются в расчет. Аргументом этой функции является номер файла.

Функция `EOF`, используемая в условии окончания цикла `Do Until...Loop`, возвращает `True` при достижении конца файла. Аргументом этой функции является номер файла.

Подпрограмма `MsgBox` выводит строку, являющуюся ее параметром, на экран дисплея. В программе Создание два обращения к `MsgBox`, поэтому по ходу выполнения программы на экране по очереди появляются окна, изображенные на рис. 20.1 и 20.2.

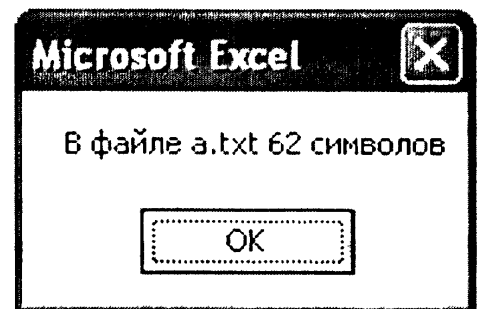


Рис. 20.1

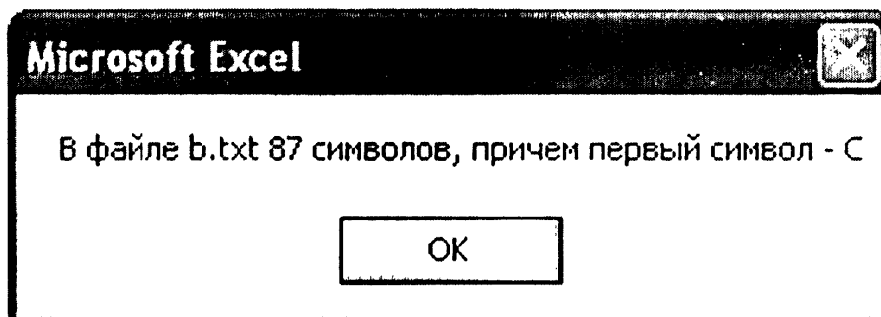


Рис. 20.2

По окончании выполнения программы Создание файлы `a.txt` и `b.txt` имеют следующее содержание.

Файл a.txt:

Создан текстовый файл,
который состоит из нескольких строк.

Файл b.txt:

Создан текстовый файл,
который состоит из нескольких строк.
Добавлена новая строка.

Задание:

1) запустите программу Создание, кликнув на стрелке на панели инструментов среды VB;

2) по окончании выполнения программы просмотрите содержимое файлов a.txt и b.txt с помощью текстового редактора Блокнот (Notepad) и убедитесь в правильности работы программы.

Может потребоваться уничтожение файлов a.txt и b.txt и папки texts по окончании выполнения программы Создание. Для этого перед строкой End Sub следует вставить следующие операторы ликвидации файлов и папки:

```
Kill(FName1)
Kill(FName2)
Rmdir("c:\Documents and Settings\usr\texts")
```

Рассмотрим программу добавления строки в файл b.txt и запуска редактора Блокнот (Notepad)

```
Sub Добавление()
    Dim FNum As Integer
    Dim n As Long
    Dim RetVal          'для функции Shell
    FNum = FreeFile
    'Открытие файла b.txt и ввод новой строки:
    Open "c:\Documents and Settings\usr\texts\b.txt" _
        For Append As FNum
    Print #FNum, "Добавлена вторая новая строка."
    'Определение количества символов в файле b.txt:
    n = LOF(FNum)
    Close FNum          'заккрытие файла b.txt
    MsgBox "В файле b.txt" & Str(n) & " символов"
    'Запуск редактора Блокнот:
    RetVal = Shell("c:\Windows\notepad.exe", 1)
End Sub
```

Редактор Блокнот (Notepad) запускается с помощью функции Shell.

Отметим, что функция Shell позволяет запустить исполняемый файл любой программы, имя которого имеет расширение exe, в частности созданный в автономной среде VB (см. главу 24).

Программу Добавление запустим по окончании выполнения программы Создание (без операторов ликвидации файлов и папки). По ходу выполнения программы Добавление на экране дисплея появляется окно, изображенное на рис. 20.3.

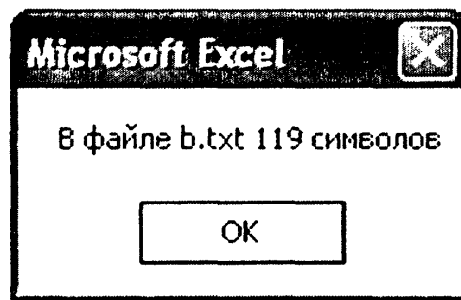


Рис. 20.3

По окончании выполнения программы Добавление файл b.txt принимает вид

Создан текстовый файл,
который состоит из нескольких строк.
Добавлена новая строка.
Добавлена вторая новая строка.

Одним из результатов выполнения программы Добавление является окно редактора Блокнот (Notepad), запускаемого функцией Shell. С помощью этого редактора можно увидеть содержимое файла, в частности b.txt, и при необходимости создать новый файл или отредактировать существующий.

При открытии файла на добавление информации (с помощью ключевого слова Append) его может не существовать. В этом случае файл создается. При открытии файла на считывание информации (с помощью ключевого слова Input) файл, естественно, должен существовать.

Рассмотренные конструкции Visual Basic позволяют решать достаточно сложные задачи. Две из них сформулированы в следующей главе.

21. Формулировка демонстрационных задач

В нескольких главах книги мы использовали задачу расчета длины гипотенузы прямоугольного треугольника по длинам катетов. Однако эта задача очень простая и не может продемонстрировать всех возможностей языка программирования. Поэтому будем также использовать две задачи транспонирования матрицы (таблицы) чисел — относительно главной и неглавной диагоналей.

Пусть A — матрица чисел, содержащая m строк и n столбцов:

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1j} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2j} & \dots & a_{2n} \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ a_{i1} & a_{i2} & \dots & a_{ij} & \dots & a_{in} \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ a_{m1} & a_{m2} & \dots & a_{mj} & \dots & a_{mn} \end{pmatrix}$$

Матрицу можно A представить в компактном виде: $A = \{a_{ij}\}$, где $1 \leq i \leq m$, $1 \leq j \leq n$.

Пусть B — матрица, содержащая n строк и m столбцов:

$$B = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1i} & \dots & b_{1m} \\ b_{21} & b_{22} & \dots & b_{2i} & \dots & b_{2m} \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ b_{j1} & b_{j2} & \dots & b_{ji} & \dots & b_{jm} \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ b_{n1} & b_{n2} & \dots & b_{ni} & \dots & b_{nm} \end{pmatrix}$$

В компактном виде: $B = \{b_{ji}\}$, где $1 \leq j \leq n$, $1 \leq i \leq m$.

Если элементы матрицы B рассчитаны по формуле $b_{ji} = a_{ij}$, то говорят, что матрица B получена транспонированием матрицы A *относительно главной диагонали* (или просто транспонированием матрицы A).

Если элементы матрицы B рассчитаны по формуле $b_{ji} = a_{m+1-i, n+1-j}$, то говорят, что матрица B получена транспонированием матрицы A *относительно неглавной диагонали*.

Отметим, что главная диагональ матрицы A — воображаемая прямая, проходящая из левого верхнего угла матрицы в правый нижний угол, а неглавная диагональ — прямая, проходящая из правого верхнего угла в левый нижний угол.

22. Программа транспонирования матрицы относительно неглавной диагонали

Рассмотрим программу решения одной из демонстрационных задач, которая использует текстовый файл `a.txt` с исходными данными (содержащий значения m и n и матрицу A) и создает текстовый файл `b.txt` с результатом расчета (содержащий матрицу B):

```
Sub ТРАНСПН()  
Dim FNameA As String, FNameB As String, FNum As Integer  
Dim strC As String, strD As String, strE As String  
Dim m As Integer, n As Integer  
Dim i As Integer, j As Integer  
Dim k As Integer, l As Integer  
Dim A() As Double, B() As Double  
1: FNameA = "c:\Documents and Settings\usr\texts\a.txt"  
2: FNameB = "c:\Documents and Settings\usr\texts\b.txt"  
3: FNum = FreeFile  
'Открытие файла a.txt:  
4: Open FNameA For Input As FNum  
'Считывание значений m и n из файла a.txt:  
5: Line Input #FNum, strC  
6: strC = Mid(strC, 3)  
7: m = Val(strC)  
8: Line Input #FNum, strC  
9: strC = Mid(strC, 3)  
10: n = Val(strC)  
'Задание размерностей матриц:  
11: ReDim A(1 To m, 1 To n)  
12: ReDim B(1 To n, 1 To m)  
'Считывание матрицы A из файла a.txt:  
13: For i = 1 To m  
14:     Line Input #FNum, strC  
15:     j = 0
```

22. Программа транспонирования матрицы

```
16:      strD = ""          'строка "" не равна строке " "
17:      l = Len(strC)
18:      For k = 1 To l
19:          strE = Mid(strC, k, 1)
20:          If strE <> " " Then strD = strD & strE
21:          If strE = " " Or k = l Then
22:              j = j + 1
23:              A(i, j) = Val(strD)
24:              strD = ""
25:          End If
26:      Next k
27: Next i
'Заккрытие файла a.txt:
28: Close FNum
29: FNum = FreeFile
'Открытие файла b.txt:
30: Open FNameB For Output As FNum
'Формирование матрицы B и ее запись в файл b.txt:
31: For j = 1 To n
32:     strC = ""
33:     For i = 1 To m
34:         B(j, i) = A(m + 1 - i, n + 1 - j)
35:         strC = strC & Str(B(j, i)) & " "
36:     Next i
37:     Print #FNum, strC
38: Next j
'Заккрытие файла b.txt:
39: Close FNum
End Sub
```

В 16-ой, 24-ой и 32-ой строках программы (с метками 16, 24 и 32) фигурирует пустая строка "", не содержащая ни одного символа. Строка " ", фигурирующая в 20-ой, 21-ой и 35-ой строках, содержит только символ пробела.

Файл исходных данных c:\Documents and Settings\usr\texts\a.txt, содержащий значения *m* и *n* и матрицу *A*, можно создать с помощью редактора Блокнот (Notepad). Окно этого редактора приведено на рис. 22.1.

Запуск программы ТРАНСПН можно осуществить любым из следующих двух методов:

- кликнув на стрелке на панели инструментов окна VB;
- выполнив в окне Excel Разработчик > Макросы > строка ТРАНСПН > Выполнить.

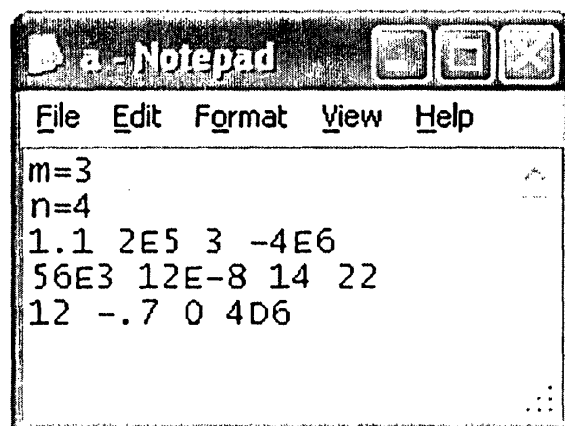


Рис. 22.1. Исходные данные: элементы строк матрицы отделяются друг от друга одним пробелом

Можно назначить сочетание клавиш, запускающее программу. Для этого в окне Excel выполним **Разработчик > Макросы > строка ТРАНСПН > Параметры**. В открывшемся окне *Параметры макроса* назначим сочетание клавиш **Ctrl + t**, нажмем кнопку **ОК** и закроем окно *Макрос*, кликнув на крестике в правом верхнем углу. После этого при одновременном нажатии клавиш **Ctrl** и **t** будет производиться запуск программы ТРАНСПН.

Рассмотрим методы перехода из окна VB в окно Excel и обратно.

Перейти из окна VB в окно Excel можно одним из трех способов:

- свернув окно VB;
- кликнув на кнопке Excel, находящейся на панели задач рабочего стола Windows;
- кликнув на пиктограмме Excel на левом конце панели инструментов окна VB.

Вернуться в окно VB можно, выполнив **Разработчик > Макросы > строка ТРАНСПН > Изменить**. При этом появится окно VB с текстом программы ТРАНСПН в окне кода. Далее в программу можно внести исправления, например, можно изменить имена файлов исходных данных и результатов расчета.

Если вместо кнопки **Изменить** мы нажмем кнопку **Войти**, то также попадем в окно VB. Далее программу ТРАНСПН можно, например, выполнить пошагово.

Результатом расчета по программе ТРАНСПН является матрица *B*, которая содержится в файле `c:\Documents and Settings\usr\texts\b.txt`. На рис. 22.2 изображено окно редактора Блокнот (Notepad) с матрицей *B*.

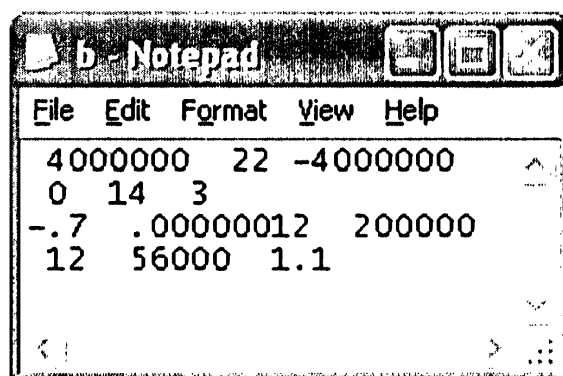


Рис. 22.2. Результаты расчета

Текст программы ТРАНСПН почти совпадает с текстом программы, рассматриваемой в следующей главе. Поэтому программа ТРАНСПН еще может потребоваться.

Для дальнейшего использования программы ТРАНСПН ее надо сохранить. Для этого в окне VB выполним **File (Файл) > Save (Сохранить)**. В открывшемся окне *Сохранение документа*:

- выберем папку, например, `c:\Documents and Settings\usr`;
- в текстовое поле *Имя файла* с помощью клавиатуры введем *КнигаТРАНСПН*;
- в текстовое поле *Тип файла* с помощью скрытого списка введем *Книга Excel с поддержкой макросов*;
- кликнем на кнопке **Сохранить**.

Программа ТРАНСПН будет сохранена в составе книги Excel с именем КнигаТРАНСПН.

Как мы видим, пользовательским интерфейсом программы ТРАНСПН являются текстовые файлы.

23. Пользовательские формы

Часто в качестве пользовательского интерфейса программы целесообразно использовать форму — прямоугольник с текстовыми полями, кнопками и другими элементами управления. Для создания этого вида пользовательского интерфейса в среде VB имеется конструктор форм.

Создадим форму для программы расчета длины гипотенузы прямоугольного треугольника по длинам катетов, содержащую (сверху вниз):

- 1) место для вывода (из программы) результата расчета;
- 2) текстовое поле для ввода (в программу) длины первого катета;
- 3) текстовое поле для ввода длины второго катета;
- 4) кнопку для запуска программы.

Пусть активная книга Excel имеет имя Книга1. Разработка программы (проекта) с формой начинается с выполнения следующей последовательности операций, производящей вставку формы в активную книгу Excel.

1. В окне Excel выполним **Разработчик > Visual Basic**. В результате появится окно VB (рис. 1.1), включающее в себя окно проекта (*Project Explorer*).

2. В окне проекта выделим строку **VBAProject (Книга1)**, кликнув на ней.

3. Выполним **Insert (Вставка) > UserForm (Пользовательская форма)**.

Появится форма UserForm1 (прямоугольник с точками) и изменится содержимое окна проекта — появится строка UserForm1 (рис. 23.1).

Появится также панель элементов управления (*Toolbox*). При отсутствии этой панели она выводится кликом на форме и далее на соответствующем пункте меню **View (Вид)**.

При отсутствии окна свойств (*Properties Window*) его следует открыть, выполнив **View (Вид) > Properties Window (Окно свойств)**.

Окно свойств «привязано» к форме, т. к. она выделена рамкой с 8 маркерами (рис. 23.1).

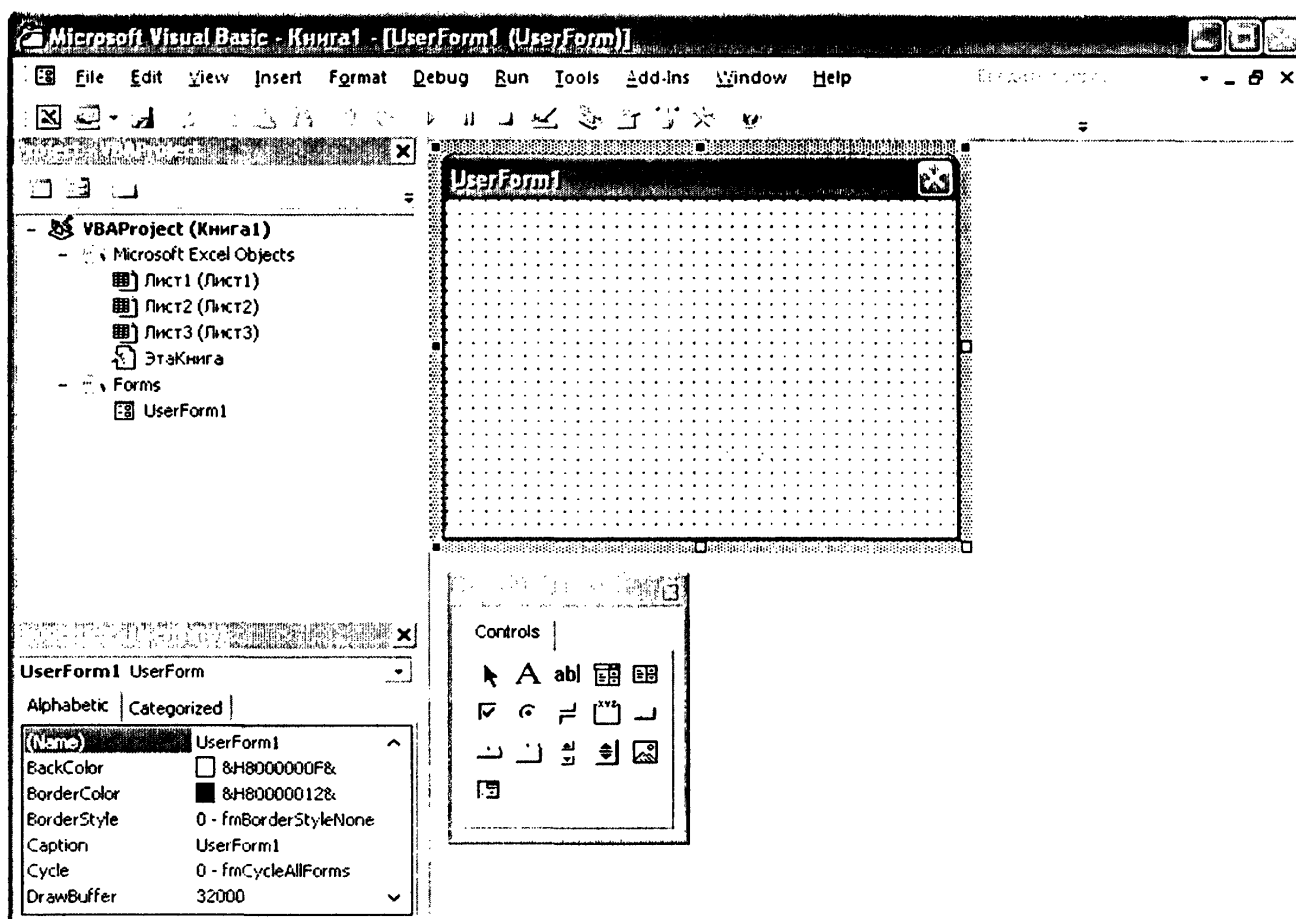


Рис. 23.1. Состояние окна VB после вставки формы в книгу Excel

Для изменения ширины формы UserForm1 выполним следующее.

Перетащим правую границу формы, «взявшись» за белый маркер в центре этой границы. При этом значение свойства Width (в окне свойств) изменится.

Термин «взяться» означает:

- 1) подвести указатель мышки к нужной точке;
- 2) нажать левую кнопку мышки.

Значение свойства Width можно также задать в окне свойств.

Аналогичным образом можно изменить высоту формы UserForm1 (свойство Height формы).

После клика (в окне свойств) на свойстве Caption (Заголовок) изменим его на следующее: *Моя первая форма*.

Панель элементов управления (*Toolbox*), которую мы видим на рис. 23.1, содержит набор пиктограмм элементов управления, которые можно просто включить в проект. Нам потребуются следующие три элемента: Label (Надпись), TextBox (Поле) и CommandButton (Кнопка).

Для «высвечивания» названия элемента надо подвести к нему указатель мышки.

Чтобы добавить в форму элемент и его отредактировать, надо выполнить следующее.

1. Выделить форму, кликнув на ней.

2. Кликнуть на требуемом элементе панели элементов управления.

3. Переместить указатель мышки в форму. Указатель примет вид крестика.

4. Поместить крестик в нужную часть формы и нажать левую кнопку мышки. После этого можно отпустить кнопку мышки.

5. Можно также, не отпуская левую кнопку мышки, перетаскивать указатель мышки, например, вниз и вправо. После этого отпустить кнопку мышки.

6. Если надо изменить положение элемента в форме или его размеры, следует с помощью мышки передвинуть элемент («взявшись» за центр) или его границу («взявшись» за один из белых маркеров).

7. Установить нужные свойства элемента с помощью окна свойств, которое «привязано» к этому элементу (т. к. он выделен рамкой с 8 маркерами).

По окончании редактирования элемента надо снять выделение этого элемента, кликнув вне него. Чтобы вернуться к редактированию элемента, надо его выделить, кликнув на изображении этого элемента в форме.

Для удаления выделенного элемента из формы надо нажать клавишу **Delete**.

В форме требуется отвести *место для надписи*, куда будет выводиться результат расчета (при эксплуатации программы расчета длины гипотенузы). Применим для этой цели элемент Label (Надпись).

Чтобы добавить в форму элемент Label (Надпись) и изменить некоторые его свойства, выполним следующее.

1. Кликнем на элементе Label (Надпись) панели элементов управления.

2. Переместим указатель мышки в форму. При этом он примет вид крестика.

3. Поместим крестик в левую верхнюю часть формы и нажмем левую кнопку мышки. Не отпуская ее, перетащим указатель мышки вниз и вправо. После этого отпустим кнопку мышки.

4. Если надо изменить положение или размеры будущей надписи, следует с помощью мышки передвинуть отведенное ей место («взявшись» за центр) или его границу («взявшись» за один из белых маркеров).

5. Для установки вида и размера шрифта используем свойство Font (Шрифт) в окне свойств. Выберем это свойство, кликнув на нем. При этом появится кнопка с изображением трюеточия, которая показывает, что для установки свойства Font (Шрифт) используется диалоговое окно.

6. Кликнем на кнопке с изображением трюеточия. При этом откроется окно *Font* (рис. 23.2), предназначенное для установки параметров надписи *Label1*, находящейся в форме.

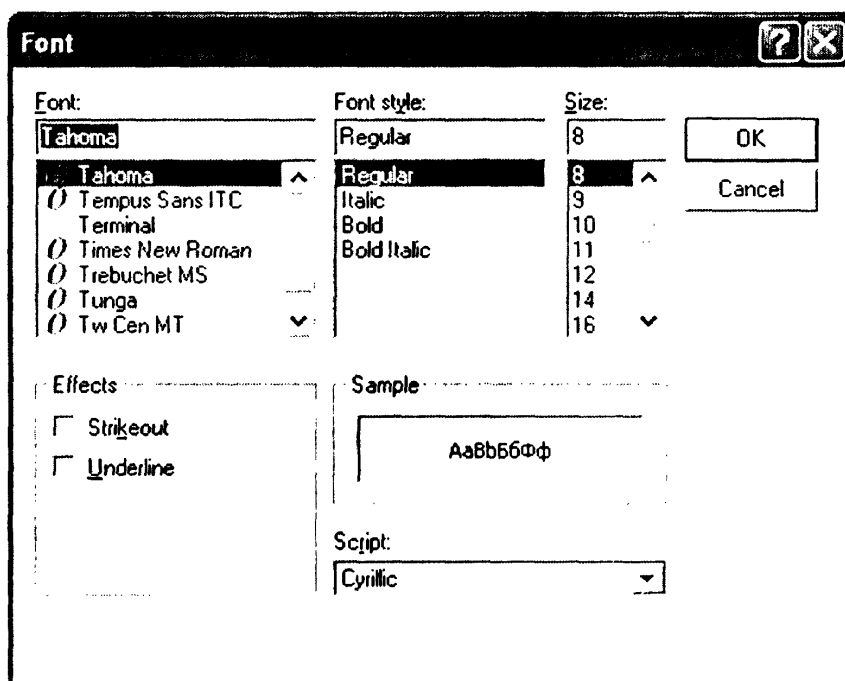


Рис. 23.2. Окно для задания формата надписи

7. Кликнем на **Cambria Math**, чтобы установить этот шрифт. В разделе *Sample* (Образец) — образец текста.

8. Установим размер шрифта 10.

9. Кликнем на кнопке **ОК**.

10. Установим следующие значения других свойств элемента *Label1*.

TextAlign (Выравнивание текста): 2

Caption (Заголовок): *Введите исходные данные и нажмите кнопку «Счет»*

Name (Имя): *LabelMessage*

11. При необходимости изменим (с помощью мышки) высоту и ширину места для надписи, а также высоту и ширину формы, чтобы надпись выглядела, как на рис. 23.3.

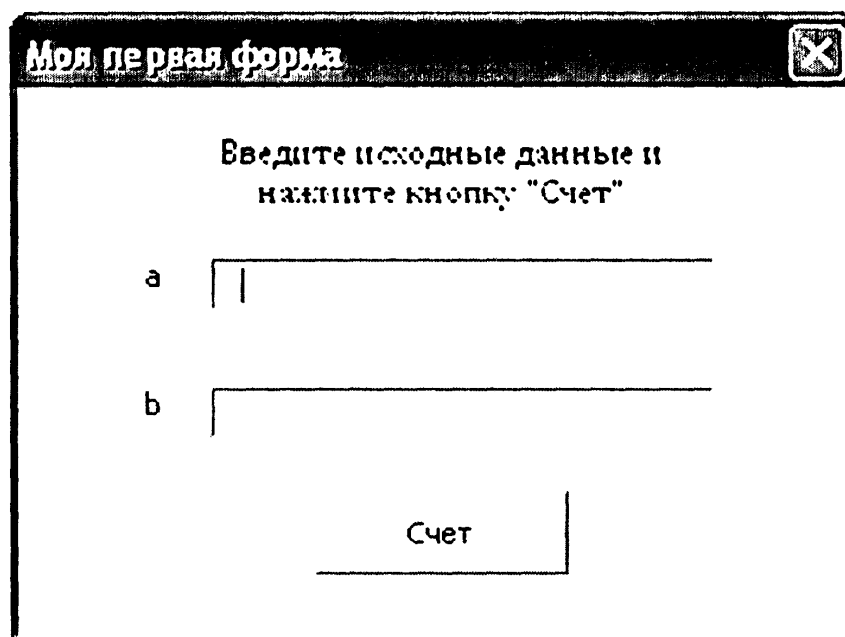


Рис. 23.3. Вид формы после запуска программы

Аналогично в форму вставляются *текстовые поля*. Для этого надо применить элемент TextBox (Поле). Вставим два текстовых поля (рис. 23.3), присвоив им имена TextBoxA и TextBoxB. Слева от текстовых полей создадим надписи «а» и «б» с помощью элемента Label (Надпись).

Чтобы добавить в форму *кнопку*, выполним следующее.

1. Кликнем на элементе CommandButton (Кнопка) панели элементов управления.

2. Изобразим кнопку в форме.

3. Установим следующие свойства кнопки.

Caption: *Счет*

Name: *ComButCalc*

4. Снимем выделение кнопки, кликнув вне нее.

Мы закончили разработку формы. Далее следует *разработка программы*.

Дважды кликнем на кнопке Счет, находящейся в разработанной форме. В появившемся окне кода имеется заготовка программы, связанной с кнопкой. Она начинается со строки

```
Private Sub ComButCalc_Click()
```

и заканчивается строкой

```
End Sub
```

Служебное слово `Private` (которое появилось в первой строке программы автоматически) подчеркивает тот факт, что обращаться к программе можно только в рамках того модуля, в котором находится ее текст. (Впрочем, мы не собираемся обращаться к программе `ComButCalc_Click` из какого-либо модуля.) Если первую строку программы набирать вручную, то слово `Private` можно опустить.

Программа расчета длины гипотенузы прямоугольного треугольника `ComButCalc_Click` похожа на программу `Pythagoras` со стандартными окнами Windows в качестве пользовательского интерфейса (см. главу 17):

```
Private Sub ComButCalc_Click()  
    Dim a As Single  
    Dim b As Single  
    Dim c As Single  
    Dim s As String  
1:  s = TextBoxA.Text  
2:  a = Val(s)  
3:  s = TextBoxB.Text  
4:  b = Val(s)  
5:  c = Sqr(a ^ 2 + b ^ 2)  
6:  s = Str(c)  
7:  LabelMessage.Caption = "c =" & s  
End Sub
```

В отличие от программы `Pythagoras`, программа `ComButCalc_Click` содержит три конструкции, похожие на записи, — `TextBoxA`, `TextBoxB` и `LabelMessage`.

Поле `Text` «записи» `TextBoxA` (или свойство `Text` элемента `TextBoxA`) — переменная строкового типа, значение которой совпадает со строкой, введенной в текстовое поле *a* формы при ее эксплуатации. Свойство `Text` элемента `TextBoxB` совпадает со строкой, введенной в текстовое поле *b*. Длины катетов задаются при выполнении операторов с 1-го по 4-й (с метками 1, 2, 3 и 4).

При выполнении 5-го и 6-го операторов рассчитывается и преобразуется в строку *s* длина гипотенузы.

При выполнении 7-го оператора свойству Caption элемента LabelMessage присваивается строка, в состав которой входит строка s. Значение этого свойства выводится в форму.

Разработанную программу можно запустить, кликнув на стрелке на панели инструментов окна VB. В результате этого произойдет загрузка формы (рис. 23.3). На рис. 23.4 изображено состояние формы по окончании очередного расчета, т. е. после ввода длин катетов в текстовые поля *a* и *b* и клика на кнопке **Счет**.

Рис. 23.4. Вид формы по окончании расчета длины гипотенузы

Далее расчет можно повторить при других исходных данных, т. е. длинах катетов. По окончании серии расчетов следует закрыть форму (кликнув на крестике в правом верхнем углу) и тем самым завершить выполнение программы.

Программу ComButCalc_Click можно упростить:

```
Private Sub ComButCalc_Click()  
    Dim a As Single  
    Dim b As Single  
    Dim c As Single  
1:  a = TextBoxA.Value  
2:  b = TextBoxB.Value  
3:  c = Sqr(a ^ 2 + b ^ 2)  
4:  LabelMessage.Caption = "c =" & Str(c)  
End Sub
```

Мы использовали свойство Value элемента TextBox. Поэтому при выполнении первого оператора присваивания (с меткой 1) информация, находящаяся в текстовом поле формы, интерпрети-

руется согласно типу переменной, находящейся слева от =, т. е. как число. То же можно сказать и про второй оператор присваивания (с меткой 2).

Программой транспонирования матрицы относительно неглавной диагонали ТРАНСПН из предыдущей главы неудобно пользоваться, т. к. имена файлов задаются в тексте программы. Для устранения этого недостатка программу ТРАНСПН дополним формой, которую мы уже построили. В этом случае элементы формы имеют следующий смысл (рис. 23.3):

- 1) место для вывода (из программы) сообщений;
- 2) текстовое поле для ввода (в программу) полного имени файла исходных данных;
- 3) текстовое поле для ввода полного имени файла результатов расчета;
- 4) кнопка для запуска программы.

Программа получается путем редактирования текста программы ТРАНСПН, находящегося в книге Excel с именем КнигаТРАНСПН (стр. 101). При открытии этой книги может быть выдано предупреждение системы безопасности *Запуск макросов отключен* с кнопкой **Параметры**. В этом случае:

- кликнем на кнопке **Параметры**;
- в появившемся окне *Параметры безопасности Microsoft Office* включим опцию **Включить это содержимое** и кликнем на кнопке **ОК**.

Далее войдем в среду VB стандартным образом и откроем окно кода с текстом программы ТРАНСПН, дважды кликнув на имени соответствующего модуля. Используя буфер обмена, программу ComButCalc_Click (разработанную ранее) приведем к следующему виду:

```
Private Sub ComButCalc_Click()  
Dim FNameA As String, FNameB As String, FNum As Integer  
Dim strC As String, strD As String, strE As String  
Dim m As Integer, n As Integer  
Dim i As Integer, j As Integer  
Dim k As Integer, l As Integer  
Dim A() As Double, B() As Double  
1: FNameA = TextBoxA.Text  
2: FNameB = TextBoxB.Text  
3: FNum = FreeFile  
'Открытие файла a.txt:
```

```
4: Open FNameA For Input As FNum
'Считывание значений m и n из файла a.txt:
5: Line Input #FNum, strC
6: strC = Mid(strC, 3)
7: m = Val(strC)
8: Line Input #FNum, strC
9: strC = Mid(strC, 3)
10: n = Val(strC)
'Задание размерностей матриц:
11: ReDim A(1 To m, 1 To n)
12: ReDim B(1 To n, 1 To m)
'Считывание матрицы A из файла a.txt:
13: For i = 1 To m
14:     Line Input #FNum, strC
15:     j = 0
16:     strD = ""          'строка "" не равна строке " "
17:     l = Len(strC)
18:     For k = 1 To l
19:         strE = Mid(strC, k, 1)
20:         If strE <> " " Then strD = strD & strE
21:         If strE = " " Or k = l Then
22:             j = j + 1
23:             A(i, j) = Val(strD)
24:             strD = ""
25:         End If
26:     Next k
27: Next i
'Заккрытие файла a.txt:
28: Close FNum
29: FNum = FreeFile
'Открытие файла b.txt:
30: Open FNameB For Output As FNum
'Формирование матрицы B и ее запись в файл b.txt:
31: For j = 1 To n
32:     strC = ""
33:     For i = 1 To m
34:         B(j, i) = A(m + 1 - i, n + 1 - j)
35:         strC = strC & Str(B(j, i)) & " "
36:     Next i
37:     Print #FNum, strC
38: Next j
'Заккрытие файла b.txt:
```

```
39: Close FNum
40: LabelMessage.Caption = "Расчет окончен. Введите..."
End Sub
```

Свойство `Text` элемента `TextBoxA` — переменная строкового типа, значение которой совпадает с полным именем файла, содержащего значения m и n и матрицу A ; это полное имя вводится в текстовое поле a формы при эксплуатации программы транспонирования матрицы. Свойство `Text` элемента `TextBoxB` — переменная строкового типа, значение которой совпадает с полным именем файла, предназначенного для хранения матрицы B ; это полное имя вводится в текстовое поле b . Полные имена открываемых файлов задаются при выполнении 1-го и 2-го операторов (с метками 1 и 2).

При выполнении 40-го оператора свойству `Caption` элемента `LabelMessage` присваивается строка "Расчет окончен. Введите...". Содержимое этого свойства выводится в форму.

Разработанную программу можно запустить, кликнув на стрелке на панели инструментов окна VB. В результате этого произойдет загрузка формы (рис. 23.3). На рис. 23.5 изображено состояние формы по окончании очередного расчета, т. е. после ввода полных имен файлов

```
c:\Documents and Settings\usr\texts\a.txt
c:\Documents and Settings\usr\texts\b.txt
```

соответственно в текстовые поля a и b и клика на кнопке **Счет**.

Далее расчет можно повторить при других именах файлов или другом содержании файла исходных данных. По окончании серии расчетов следует закрыть форму (кликнув на крестике в правом верхнем углу) и тем самым завершить выполнение программы.

Файл исходных данных, содержащий значения m и n и матрицу A , можно создать или отредактировать с помощью редактора Блокнот (Notepad). Результат расчета по программе (матрицу B) можно увидеть с помощью того же редактора. Окно редактора Блокнот (Notepad) с исходными данными и результатами расчета изображено на рис. 22.1 и 22.2.

Элемент `TextBox` (Поле) можно использовать не только для ввода информации, но и для вывода. При этом свойство `Text` соответствующего элемента `TextBox` (Поле) должно присутствовать в левой части оператора присваивания (слева от знака `=`).

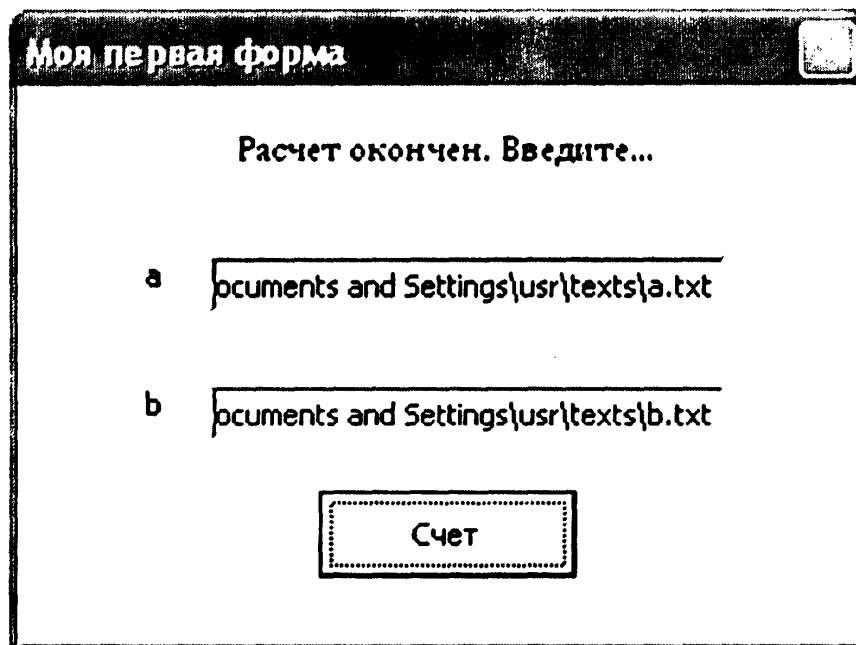


Рис. 23.5. Вид формы по окончании транспонирования матрицы

Задание: используя элемент TextBox (Поле), модифицируйте форму и текст последней программы так, чтобы сообщения выводились в текстовое поле.

Мы научились пользоваться тремя элементами панели элементов управления — Label (Надпись), TextBox (Поле) и CommandButton (Кнопка). Рассмотрим также элемент CheckBox (Флажок), который используется, когда некую «возможность» надо либо включать, либо выключать.

Элемент CheckBox (Флажок) изображается в виде квадратика. При клике на квадратике в нем появляется галочка, при повторном клике галочка исчезает. Свойство Value элемента CheckBox (Флажок) имеет значение True при наличии галочки в квадратике и False при отсутствии галочки.

Вернемся к началу главы. Форму для программы расчета длины гипотенузы прямоугольного треугольника по длинам катетов, изображенную на рис. 23.3, дополним элементом CheckBox (Флажок), который расположим слева от кнопки. Программу ComButCalc_Click() на стр. 108 модифицируем так, чтобы при наличии галочки, помимо длины гипотенузы, рассчитывалась еще и площадь треугольника:

```
Private Sub ComButCalc_Click()  
    Dim a As Single  
    Dim b As Single
```

```
Dim c As Single
Dim s As String
Dim out As String
a = TextBoxA.Value
b = TextBoxB.Value
c = Sqr(a ^ 2 + b ^ 2)
out = "c =" & Str(c)
If CheckBox1.Value Then
    s = Str(a * b / 2)
    out = out & vbCrLf & "s =" & s
End If
LabelMessage.Caption = out
End Sub
```

В этой программе `CheckBox1` — имя элемента `CheckBox` (Флажок).

На рис. 23.6 и 23.7 приведены состояния формы для программы расчета длины гипотенузы и площади прямоугольного треугольника.

Моя первая форма

Введите исходные данные и нажмите кнопку "Счет"

a

b

s

Счет

Рис. 23.6. Вид формы после запуска программы

Выполнив **File (Файл) > Save (Сохранить) > ...** в окне VB (стр. 101), сохраним книгу Excel в дисковой памяти компьютера под именем КнигаФорма. При этом сохранится и проект, входящий в состав книги Excel.

Рис. 23.7. Вид формы по окончании расчета

При возвращении к работе над проектом может потребоваться двойной клик на строке **UserForm1** в окне проекта. Удаляется форма так же, как модуль, с помощью контекстного меню (стр. 12).

Задание: используя форму с элементом **CheckBox** (Флажок), модифицируйте текст программы на стр. 109—111 так, чтобы при установке галочки производилось транспонирование матрицы относительно неглавной диагонали, а при удалении галочки — транспонирование матрицы относительно главной диагонали.

Задача транспонирования матрицы относительно главной диагонали сформулирована в главе 21.

24. Отступление. Разработка программы с формой в автономной среде VB

Программы, рассмотренные ранее и рассматриваемые после этой главы, имеют общий недостаток: для проведения расчетов по этим программам компьютер пользователя должен быть оснащен табличным процессором Excel. Если требуется выполнять программу в отсутствие Excel, то ее следует разработать в среде Visual Basic, не входящей в состав Microsoft Office. Рассмотрим особенности процесса разработки программы (проекта) с формой в среде Visual Basic, входящей в состав пакета Microsoft Visual Studio 6.0.

Для запуска среды VB надо в окне Windows выполнить **Пуск (Start) > Все программы (All Programs) > Microsoft Visual Studio 6.0 > Microsoft Visual Basic 6.0**. При этом возможны следующие варианты.

- Появится окно *Project1* с формой *Form1*, которая выглядит, как прямоугольник с точками.
- Появится окно *New Project*. В этом случае для начала разработки проекта следует активизировать вкладку **New** и дважды кликнуть на пиктограмме **Standard EXE**. При этом появится окно *Project1* с формой *Form1*.
- Окно среды VB не содержит ни окна *Project1*, ни окна *New Project*. В этом случае для начала разработки проекта надо выполнить **File > New Project**, а затем в появившемся окне *New Project* дважды кликнуть на пиктограмме **Standard EXE**. При этом появится окно *Project1* с формой *Form1*.

В результате окно среды VB примет вид, приведенный на рис. 24.1. Разработка формы и программы производится, как описано в предыдущих главах.

Проект содержит несколько файлов. При запоминании проекта (**File > Save Project**) следует указать путь в заранее организованную папку, например, vb, полное имя которой c:\Documents and Settings\usr\vb, где usr — имя пользователя. Если к проекту

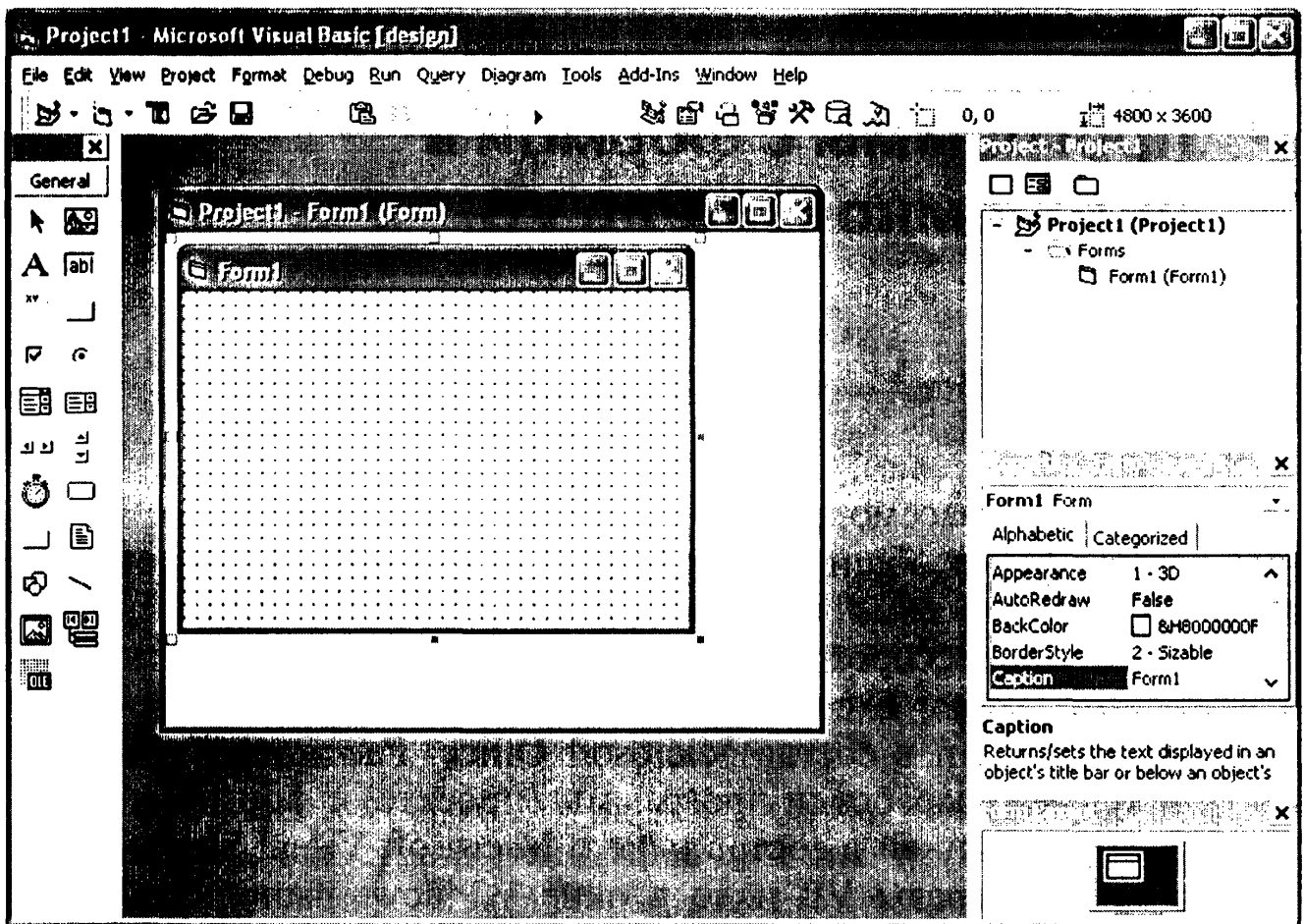


Рис. 24.1. Окно автономной среды Visual Basic с формой

надо вернуться, то следует выполнить **File > Open Project** и дважды кликнуть на файле **Project1.vbp** в этой папке.

По окончании разработки проекта следует получить исполняемый файл программы, имя которого имеет расширение **exe**. Для этого надо выполнить следующие операции:

1) **File > Make Project1.exe**;

2) в появившемся окне *Make Project* указать местоположение исполняемого файла и имя, которое следует ему присвоить;

3) нажать кнопку **ОК**.

Для загрузки формы надо выполнить следующее:

1) найти пиктограмму исполняемого файла программы, например, в окне программы Проводник (Windows Explorer);

2) дважды кликнуть на этой пиктограмме.

Далее, пользуясь формой, можно эксплуатировать программу. Если мы хотим передать кому-либо программу с формой, то достаточно передать исполняемый файл.

Теперь вернемся к программированию на Visual Basic в табличном процессоре Excel.

25. Загрузка формы из окна Excel

Форма, разработанная в главе 23, загружалась из окна VB (кликом на стрелке на панели инструментов). Однако хотелось бы иметь возможность загружать форму из окна Excel. Для этого разработаем простую программу.

Откроем книгу Excel с именем КнигаФорма (стр. 113). В окне VB этой книги выполним **Insert (Вставка) > Module (Модуль)**.

В окне кода наберем программу

```
Sub ЗагрузкаФормы()  
    UserForm1.Show  
End Sub
```

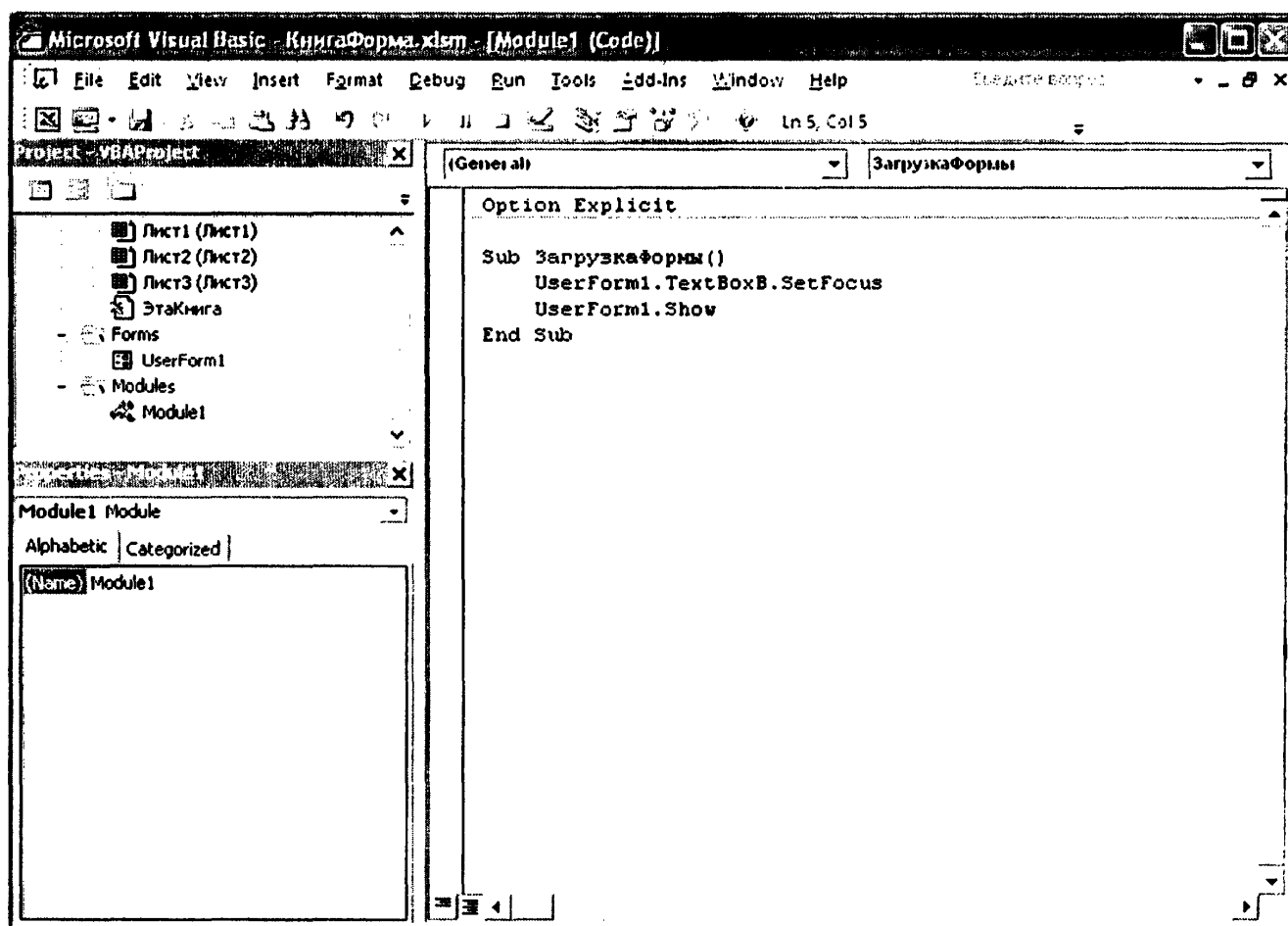


Рис. 25.1. Окно VB со строками UserForm1 и Module1 в окне проекта

В этой программе UserForm1 — имя формы, разработанной в главе 23.

Теперь форму, изображенную на рис. 23.6, можно загружать из окна Excel. Для этого надо выполнить следующее: **Разработчик > Макросы > строка ЗагрузкаФормы > Выполнить**.

Выполнив **Разработчик > Макросы > строка ЗагрузкаФормы > Параметры > ...**, можно назначить сочетание клавиш, запускающее программу ЗагрузкаФормы, т. е. загружающее форму UserForm1.

Если мы хотим, чтобы в загруженной форме мигающий курсор находился, например, во втором текстовом поле, то в программу ЗагрузкаФормы надо вставить оператор

```
UserForm1.TextBoxB.SetFocus
```

При этом программа примет следующий вид (рис. 25.1):

```
Sub ЗагрузкаФормы()  
    UserForm1.TextBoxB.SetFocus  
    UserForm1.Show  
End Sub
```

Запомним книгу с именем КнигаФорма, дополненную программой ЗагрузкаФормы.

26. Расположение элементов управления на листе Excel

Элементы управления, рассмотренные в главе 23, в частности кнопку, можно располагать на листе Excel.

Кнопка для загрузки формы создается следующим образом.

1. В окне Excel с книгой КнигаФорма выполним **Разработчик > Вставить**. При этом появится панель, содержащая элементы управления формы (рис. 26.1).

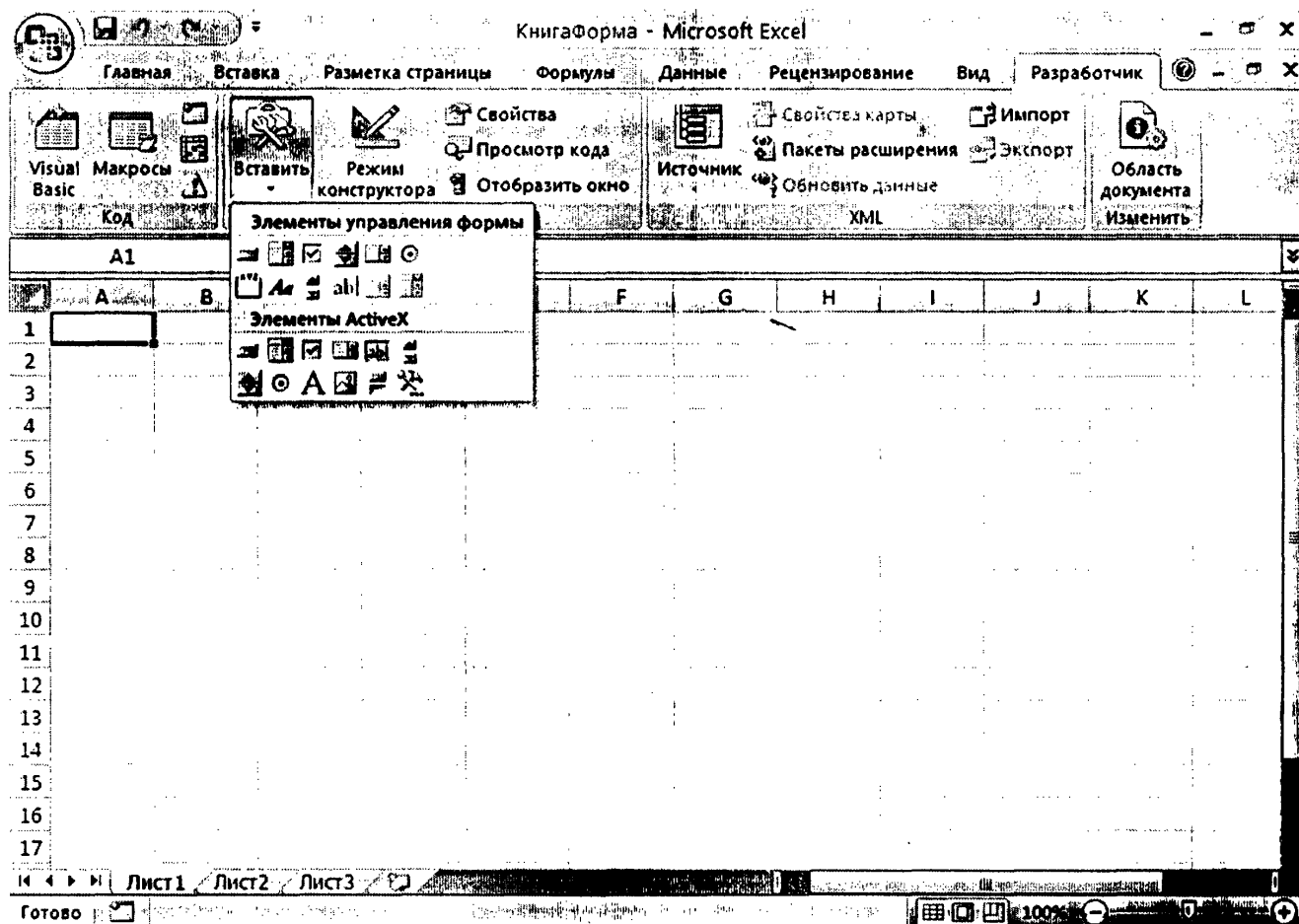


Рис. 26.1. Окно Excel с панелью элементов

2. Кликнув на элементе Кнопка раздела *Элементы управления формы*, изобразим кнопку на листе Excel так, как делали это в главе 23. При этом на созданной кнопке автоматически появится

надпись *Кнопка1* и откроется окно *Назначить макрос объекту* (рис. 26.2).

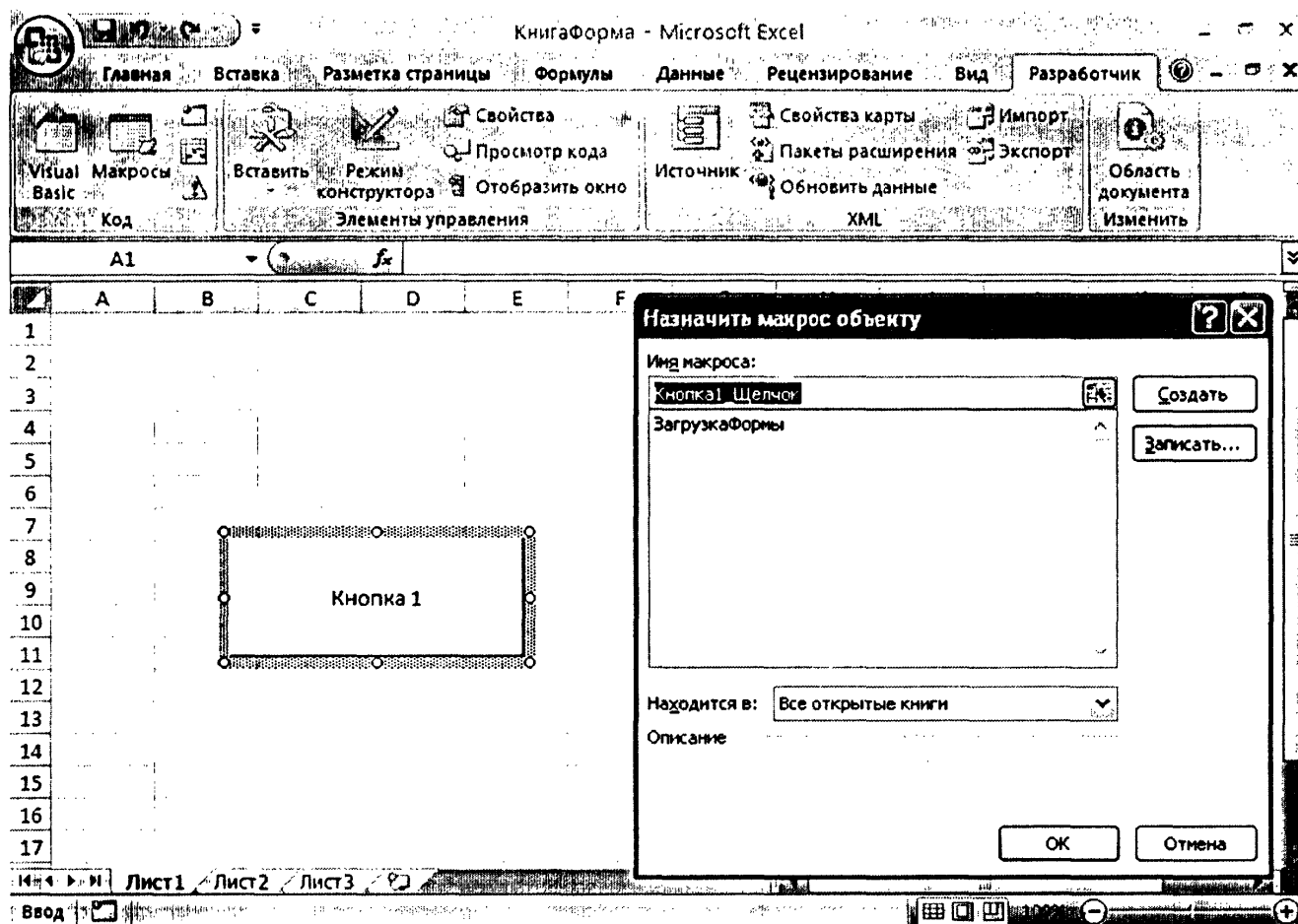


Рис. 26.2. Окно Excel с созданной кнопкой и окном *Назначить макрос объекту*

3. Кликнем на кнопке **Создать**. При этом откроется окно VB (рис. 26.3).

4. Между строками заготовки программы в окне кода вставим оператор `UserForm1.Show`. Программа примет вид

```
Sub Кнопка1_Щелчок()  
    UserForm1.Show  
End Sub
```

5. Вернемся в окно Excel и кликнем вне кнопки для снятия выделения.

Теперь для загрузки формы, изображенной на рис. 23.6, достаточно кликнуть на кнопке **Кнопка1**.

Для изменения надписи на кнопке надо сделать на ней правый клик и выполнить команду **Изменить текст** открывшегося контек-

26. Расположение элементов управления на листе Excel

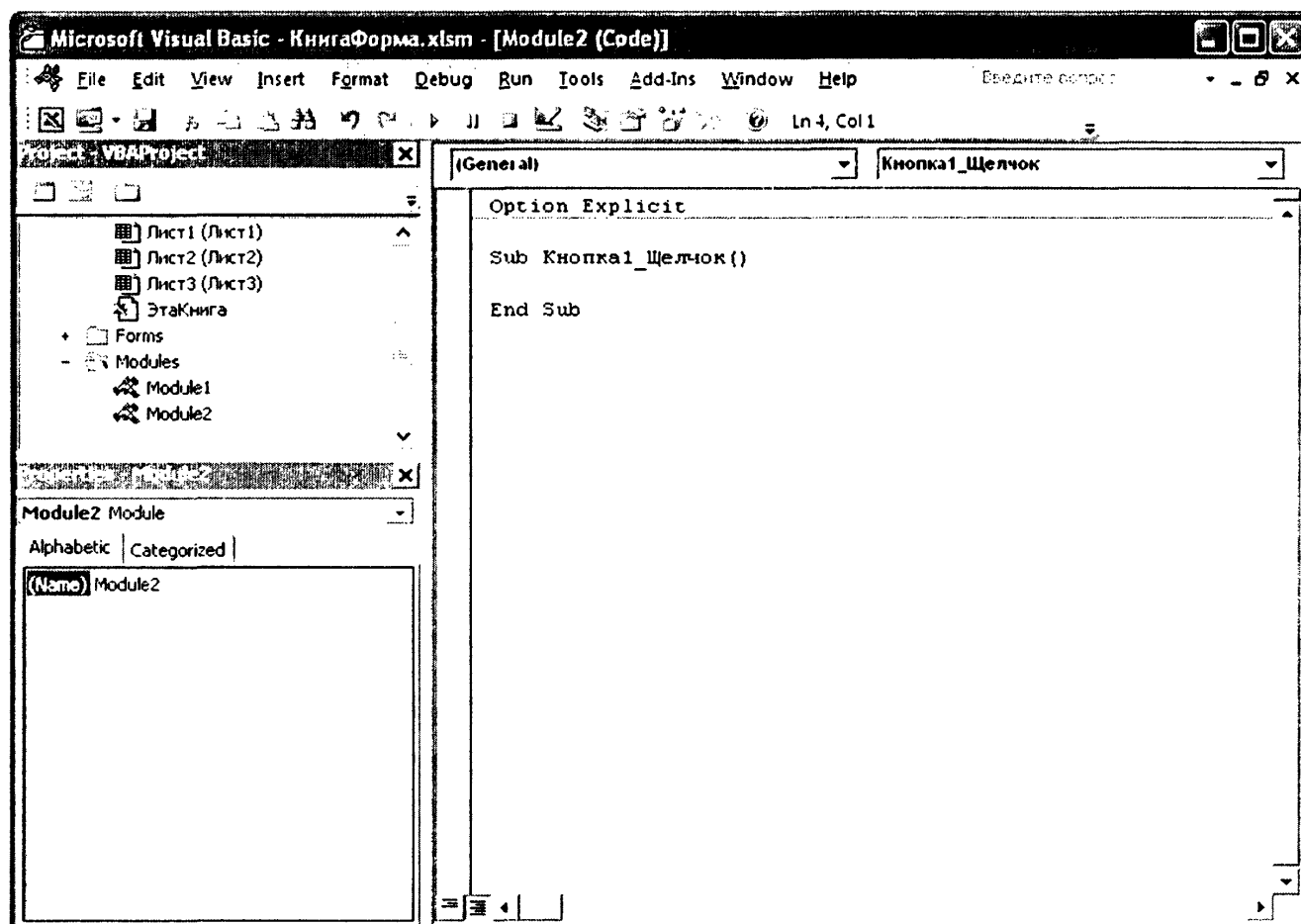


Рис. 26.3. Окно VB со строками **Module1** и **Module2** в окне проекта

стного меню. Для удаления кнопки надо ее выделить, сделав на ней правый клик, и далее нажать клавишу **Delete**.

Лист Excel с элементами управления играет роль формы. Чтобы эта «форма» стала красивой, можно убрать сетку с листа Excel. Для этого в Excel надо выполнить **Вид > Показать или скрыть > выключить опцию Сетка**.

27. Пользовательские функции Excel

Нашей первой программой на VB была программа Пифагор, предназначенная для расчета длины гипотенузы прямоугольного треугольника по длинам катетов. Рассмотрим функцию Excel для решения этой простой задачи:

```
Function Гипотенуза(a, b)
    Гипотенуза = Sqr(a ^ 2 + b ^ 2)
End Function
```

Этот текст вводится в окно кода среды VB (после вставки модуля в активную книгу). Результатом этого является функция Гипотенуза в категории *Определенные пользователем* библиотеки функций Excel. Чтобы в этом убедиться:

- кликнем на кнопке **fx** строки формул Excel;
- в текстовое поле *Категория* открывшегося окна мастера функций введем *Определенные пользователем* с помощью скрытого списка.

В списке *Выберите функцию* (рис. 27.1) увидим строку **Гипотенуза**, что говорит о включении разработанной функции в библиотеку функций Excel. Нажмем кнопку **Отмена**.

Созданная нами функция используется, например, следующим образом.

1. В две ячейки окна Excel вводим значения длин катетов, например, 30.02 и 40 соответственно в ячейки A2 и B2.

2. Выделяем ячейку для длины гипотенузы, например, C2.

3. В поле формул (после =) вводим функцию Гипотенуза, аргументами которой являются адреса ячеек с длинами катетов, причем между аргументами ставим символ «точка с запятой», а не запятую, как это принято в VB.

4. Производим расчет одним из следующих способов:

- кликнув на галочке в строке формул;
- нажав клавишу **Enter**.

В выделенной ячейке — результат расчета длины гипотенузы (рис. 27.2).

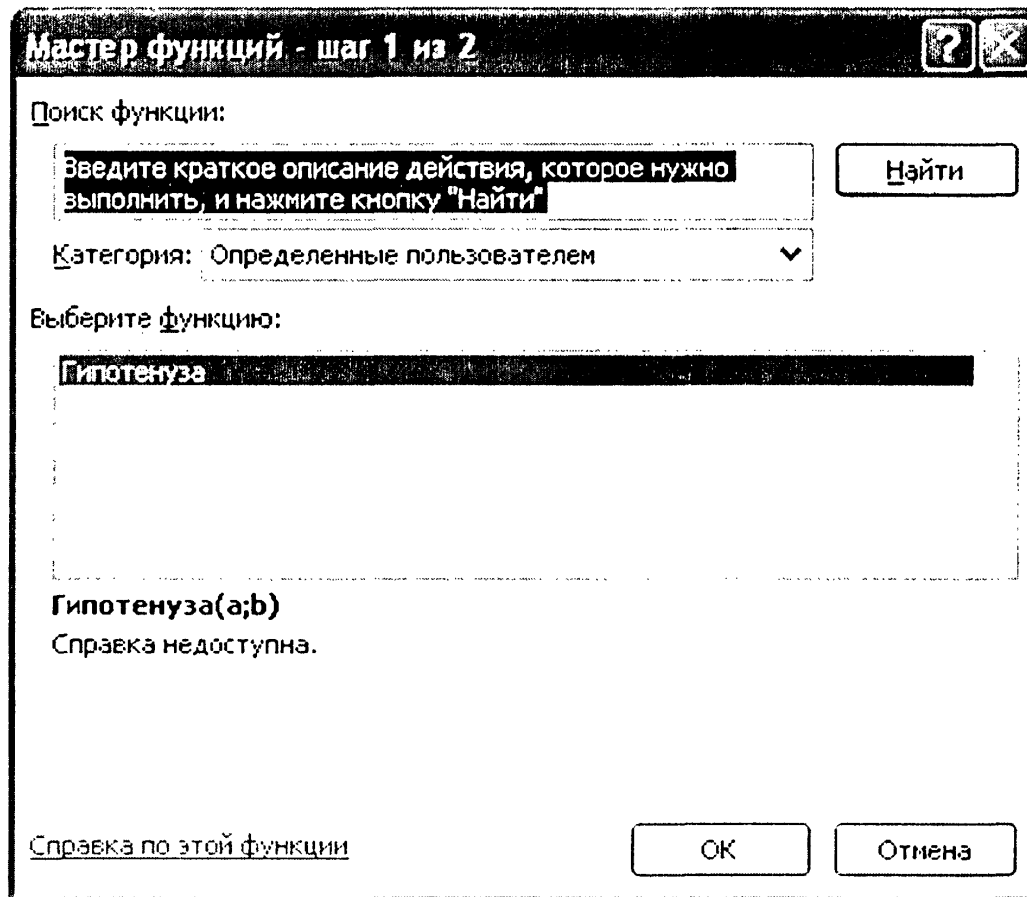


Рис. 27.1. Первое окно мастера функций Excel

При использовании мастера функций для выполнения расчета надо нажать кнопку **ОК** в окне *Аргументы функции*.

Пользовательскую функцию Гипотенуза можно использовать в формулах Excel, как встроенные функции Excel.

В Excel при решении задачи транспонирования матрицы (диапазона ячеек) относительно главной диагонали, сформулированной в главе 21, содержимое строк и столбцов меняется местами.

Для такого транспонирования матрицы в Excel имеется функция **ТРАНСП**, находящаяся в категории *Ссылки и массивы* библиотеки функций. Она является примером функции, возвращающей массив чисел (а не одно число, как многие другие функции).

Используется функция **ТРАНСП** следующим образом.

1. В ячейки окна Excel вводим транспонируемую матрицу (содержащую, например, 3 строки и 4 столбца).

2. Выделяем диапазон ячеек для результата транспонирования матрицы (содержащий 4 строки и 3 столбца).

3. В поле формул (после =) вводим функцию **ТРАНСП**, аргументом которой является исходный диапазон ячеек (содержащий 3 строки и 4 столбца).

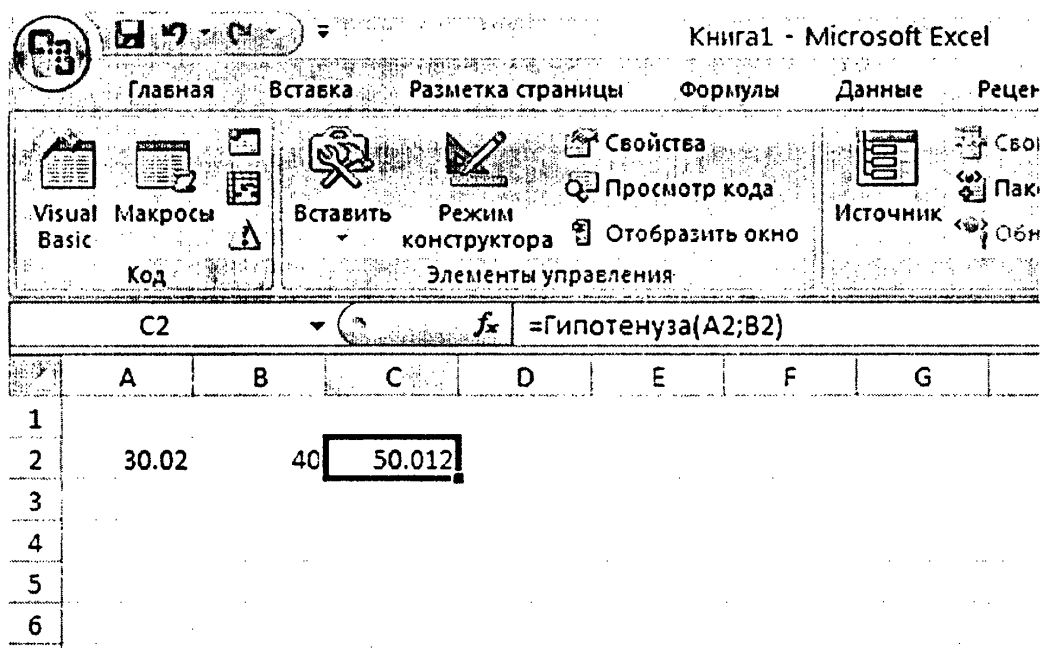


Рис. 27.2. Использование функции Гипотенуза

4. Производим расчет одним из следующих способов:

- кликнув на галочке в строке формул на фоне нажатых клавиш **Ctrl** и **Shift**;
- нажав клавиши **Ctrl + Shift + Enter**.

В выделенном диапазоне ячеек появится результат транспонирования матрицы.

При использовании мастера функций для выполнения транспонирования надо нажать кнопку **ОК** в окне *Аргументы функции* на фоне нажатых клавиш **Ctrl** и **Shift**.

Интерес представляет функция Транспн, производящая транспонирование матрицы относительно неглавной диагонали (см. главу 21). Ниже приведен текст этой функции, который вводится в окно кода среды VB после вставки модуля в активную книгу Excel.

```
Function Транспн(massiv As Variant) As Variant
    Dim m As Integer, n As Integer
    Dim i As Integer, j As Integer
    Dim R() As Variant          'описание матрицы результата
    m = massiv.Rows.Count      'количество строк
    n = massiv.Columns.Count    'количество столбцов
    ReDim R(1 To n, 1 To m)     'задание размерности
    For j = 1 To n
        For i = 1 To m
            R(j, i) = massiv(m + 1 - i, n + 1 - j)
        Next i
    Next j
End Function
```

```
Next j
  Транспн = R
End Function
```

Параметр `massiv` можно рассматривать как встроенную запись, т. е. запись, определение которой скрыто от пользователя. Точнее, `massiv` — переменная типа `Range` (Диапазон), но об этом говорится позже.

Результатом ввода приведенного текста в окно кода является функция `Транспн` в категории *Определенные пользователем* библиотеки функций Excel.

Используется функция `Транспн` так же, как встроенная функция `ТРАНСП`.

1. В ячейки окна Excel (например, A1:D3) вводим матрицу, предназначенную для транспонирования относительно неглавной диагонали.

2. Выделяем диапазон ячеек для результата транспонирования (например, A5:C8).

3. В поле формул вводим формулу `=Транспн(A1:D3)`

4. Производим расчет одним из двух способов:

- кликнув на галочке в строке формул на фоне нажатых клавиш **Ctrl** и **Shift**;

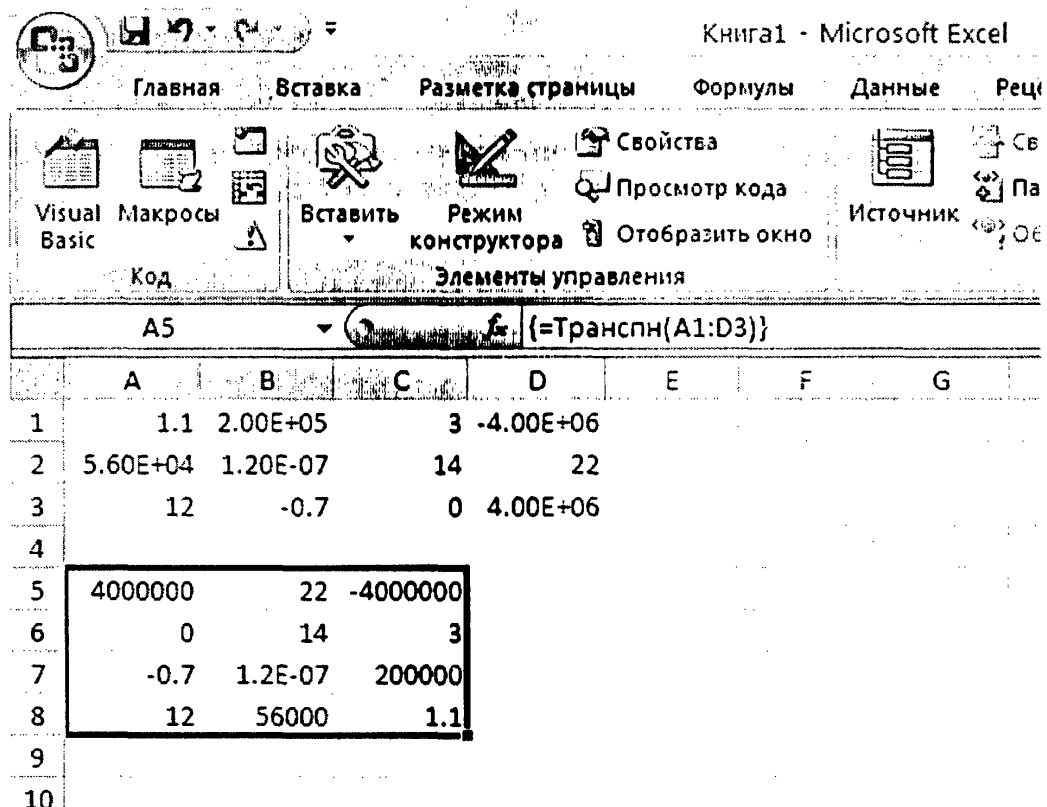


Рис. 27.3. Использование функции `Транспн`

- нажав клавиши **Ctrl + Shift + Enter**.

В выделенном диапазоне ячеек A5:C8 появится результат транспонирования матрицы относительно неглавной диагонали (рис. 27.3).

При использовании мастера функций для выполнения транспонирования надо нажать кнопку **ОК** в окне *Аргументы функции* на фоне нажатых клавиш **Ctrl** и **Shift**.

Интересно сравнить функцию Транспн с программой ТРАНСПН из главы 22. По количеству операторов функция в 4 раза короче программы. Кроме того, функцией удобней пользоваться. В частности, ее можно включать в состав макросов Excel, разрабатываемых, например, с помощью макрорекордера (см. главу 29).

28. Два метода разработки макросов

Макросом (или макрокомандой) Excel называется последовательность операций, снабженная именем, которая может быть выполнена автоматически.

Макрос записывается на алгоритмическом языке VBA (Visual Basic for Applications — VB для приложений), включающем рассмотренные нами основные конструкции Visual Basic. Язык VBA является расширением Visual Basic.

Используются два метода разработки макросов:

- с помощью среды VB;
- с помощью макрорекордера Excel.

Первым методом мы пользовались в предыдущих главах, в частности при разработке двух программ (макросов) загрузки формы UserForm1 (см. главы 25 и 26).

Второй метод более простой, но менее универсальный по сравнению с первым; его обычно рассматривают при изучении табличного процессора Excel. В нашем курсе этот метод будет рассмотрен, т. к. макрорекордер является отличным средством обучения первому методу разработки макросов. Если мы не знаем, как на VBA записать какое-либо действие (или последовательность действий), надо с помощью макрорекордера создать макрос, выполняющий это действие (или последовательность действий), а затем изучить получившийся текст макроса.

Кроме того, макрорекордер облегчает использование первого метода. Дело в том, что программу на VBA часто удобнее разрабатывать не «с нуля», а начиная с прототипа, разработанного с помощью макрорекордера.

Таким образом, нужную программу можно получить путем редактирования текста макроса на VBA, являющегося результатом работы макрорекордера.

29. Макрорекордер Excel

Работа макрорекордера во многом напоминает запись с помощью обычного магнитофона (recorder — магнитофон). Макрорекордер:

- протоколирует операции, выполняемые пользователем;
- преобразует их в последовательность операторов VBA.

Создадим с помощью макрорекордера макрос для выполнения следующей последовательности операций:

- 1) удаление сетки с листа Excel и задание стиля ссылок R1C1;
- 2) присвоение всем ячейкам листа денежного формата;
- 3) выделение ячейки R14C5, то есть E14.

Для создания макроса выполним следующее.

1. Выполним **Разработчик > Запись макроса** (в разделе *Код*).

2. В текстовые поля открывшегося окна *Запись макроса* введем имя макроса и другую информацию.

Имя макроса: *MR*

Сочетание клавиш: *Ctrl+m* (это сочетание клавиш будет использоваться для запуска макроса)

Сохранить в: *Эта книга*

Описание: *Результат работы макрорекордера*

3. В окне *Запись макроса* с заполненными текстовыми полями (рис. 29.1) кликнем на кнопке **ОК**. Запись макроса началась. При этом вместо кнопки **Запись макроса** (в разделе *Код*) появилась кнопка **Остановить запись**.

4. Отменим отображение сетки в окне Excel: **Вид > Показать или скрыть >** выключим опцию **Сетка**.

5. Установим стиль ссылок R1C1:

- кликнем на кнопке **Office** (в левом верхнем углу);
- выполним **Параметры Excel > Формулы**;
- в разделе *Работа с формулами* включим опцию **Стиль ссылок R1C1**;
- кликнем на кнопке **ОК**.

6. Зададим денежный формат для всех ячеек активного листа; для этого:

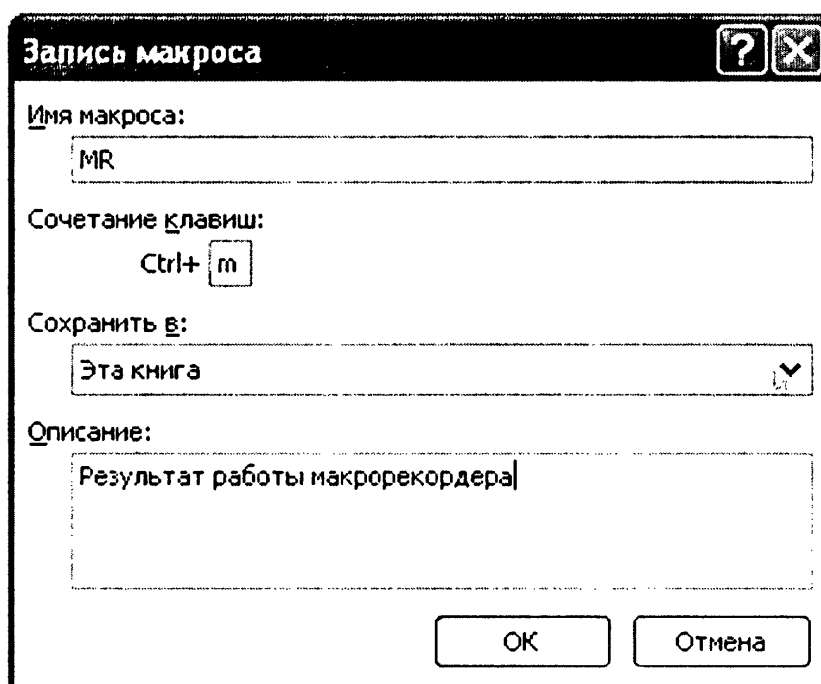


Рис. 29.1

- выделим все ячейки листа, кликнув на пересечении верхней строки таблицы (с номерами столбцов) и левого столбца (с номерами строк);
- активизируем вкладку **Главная**;
- в разделе *Число* с помощью скрытого списка установим **Денежный**.

7. Кликнем на ячейке R14C5, находящейся в 14-й строке и 5-м столбце.

8. Выполним **Разработчик > Остановить запись**.

Запись макроса окончилась. Результатом работы макрорекордера является макрос MR.

Перед проверкой работоспособности созданного макроса восстановим стиль ссылок A1:

- кликнем на кнопке **Office**;
- выполним **Параметры Excel > Формулы**;
- в разделе *Работа с формулами* выключим опцию **Стиль ссылок R1C1**;
- кликнем на кнопке **ОК**.

Перед запуском макроса активизируем лист, для которого следует выполнить записанные в макросе операции, например, Лист3.

Рассмотрим основные способы запуска макроса.

- Запуск с помощью комбинации клавиш — путем одновременного нажатия клавиш **Ctrl** и **m**.

- Запуск макроса путем выбора имени: **Разработчик > Макросы > строка MR > Выполнить**.

При выполнении макроса компьютер повторяет действия, перечисленные в пунктах 4 — 7.

После восстановления стиля ссылок A1 книгу Excel с макросом MR, являющимся результатом работы макрорекордера, сохраним под именем КнигаМакрорекордер.

Для удаления макроса надо выполнить **Разработчик > Макросы > строка MR > Удалить > Да**. Но мы не будем удалять макрос, т. к. он нам потребуется в следующих главах.

30. Текст макроса на VBA и его редактирование

Макросу соответствует программа на VBA, причем имя программы — MR, как у макроса.

Для вывода текста программы (макроса) на экран дисплея надо в окне Excel с открытой книгой КнигаМакрорекодер выполнить **Разработчик > Макросы > строка MR > Изменить**.

Появится окно VB, причем в окне кода будет присутствовать следующий текст макроса:

```
Sub MR()  
' MR Макрос  
' Результат работы макрорекодера  
' Сочетание клавиш: Ctrl+m  
    ActiveWindow.DisplayGridlines = False  
    Application.ReferenceStyle = xlR1C1  
    Cells.Select  
    Selection.NumberFormat = "#,##0.00$"  
    Range("E14").Select  
End Sub
```

Имеется возможность редактирования текста макроса, полученного с помощью макрорекодера.

Предположим, что мы хотим задавать содержимое ячейки R14C5 (т. е. E14), причем с помощью стандартного окна Windows. Для этого в тексте макроса перед последней строкой напечатаем оператор вида

```
ActiveCell.Formula = _  
InputBox("Введите цену штуки в рублях" _  
& vbCrLf & "в активную ячейку")
```

Полученный текст макроса приведен на рис. 30.1.

В результате запуска макроса MR, содержащего дополнительные строки, появится окно, изображенное на рис. 30.2.

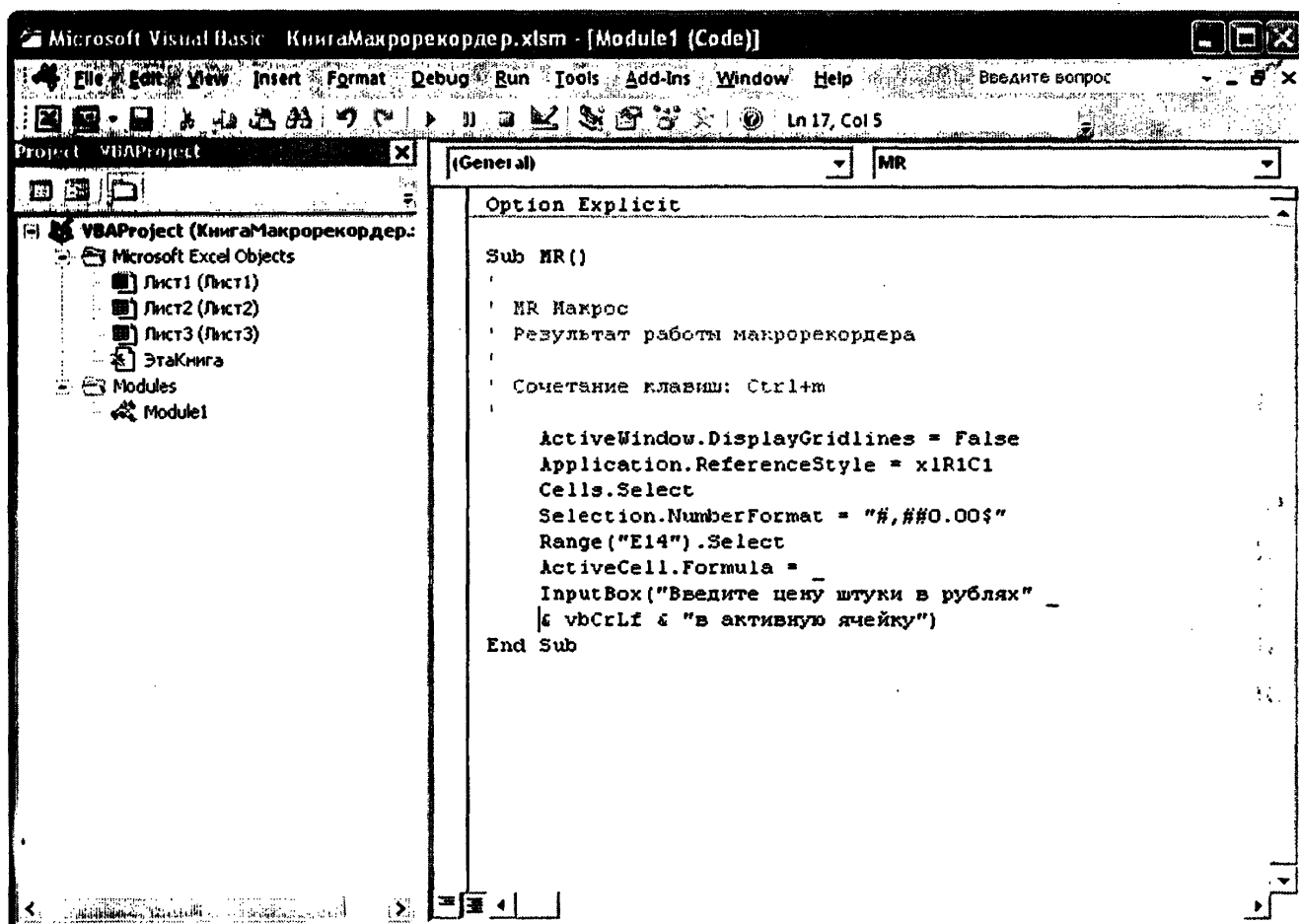


Рис. 30.1. Окно VB с текстом макроса MR после ввода дополнительного оператора

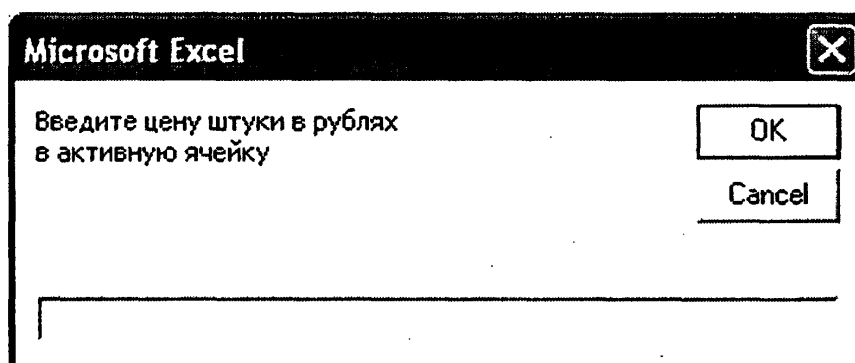


Рис. 30.2

В текстовое поле *Введите цену штуки в рублях в активную ячейку* следует ввести нужное значение, например, 12. После клика на кнопке **ОК** содержимое текстового поля попадет в ячейку R14C5; в ней высветится *12.00р.*

Далее рассмотрим основные конструкции языка VBA, не входящие в язык Visual Basic.

31. Объекты и события

Все современные алгоритмические языки, в частности Visual Basic и VBA, являются объектно- и событийно-ориентированными.

Объектно-ориентированный подход подразумевает выделение обособленной части предметной области (для которой разрабатывается программа), обладающей некоторыми характеристиками (свойствами) и способами изменения этих характеристик (методами). Программная реализация этой части предметной области называется объектом.

Примерами предметных областей являются задачи расчета финансового риска, моделирования микроэлектронных приборов и эволюции звезд (стр. 40), а также сами алгоритмические языки Visual Basic и VBA.

Событийно-ориентированный подход опирается на следующие два понятия:

- «событие» — качественное изменение состояния объекта, возникающее в результате действий пользователя или работы компьютера;
- «обработчик события» — набор команд, выполняемых компьютером при возникновении события.

Мы уже сталкивались с объектами Visual Basic — это UserForm (Пользовательская форма), Label (Надпись), TextBox (Поле), CommandButton (Кнопка) и CheckBox (Флажок) в главе 23. Примером события является клик на кнопке **Счет**, находящейся в форме. Результат обработки этого события — запуск программы ComButCalc_Click. Другой пример события приведен в конце главы 32.

Алгоритмический язык VBA предназначен для создания программ в приложениях Microsoft Office, таких как текстовый процессор Word, табличный процессор Excel, графический редактор Visio и др. VBA отличается от Visual Basic наличием специфических объектов Microsoft Office и приложений.

Специфические объекты VBA первой группы (Microsoft Office) собраны в библиотеке Office, общей для всех приложений системы

Microsoft Office. Специфические объекты второй группы (приложений) собраны в библиотеках приложений Microsoft Office.

Нам интересны объекты VBA, собранные в библиотеке Excel и называемые объектами Excel. Примерами таких объектов служат диапазон ячеек Excel, книга (или, что то же самое, рабочая книга) и лист (рабочий лист).

Для изучения объектов VBA необходимо использовать справочные системы, вызываемые нажатием клавиш **F1** и **F2** в среде VB (см. главы 6 и 36).

Операторы работы с объектом производят следующее:

- возвращение в программу свойств объекта;
- установку (задание) свойств объекта;
- выполнение методов объекта.

Синтаксис возвращения свойства объекта:

переменная = объект.свойство

где *переменная* — имя переменной.

Синтаксис установки свойства объекта:

объект.свойство = выражение

В последнем операторе присваивания *выражение* представляет собой арифметическое или логическое выражение или строку. Выполняется этот оператор в два этапа:

- 1) производится расчет значения выражения *выражение*;
- 2) полученное значение присваивается свойству объекта *объект.свойство*.

Отметим, что не все свойства объекта возвращаются и устанавливаются. Имеются свойства, которые можно только вернуть или только установить. Для уточнения возможностей того или иного свойства следует использовать справочные системы, вызываемые нажатием клавиш **F1** и **F2**.

Синтаксис выполнения метода объекта:

[переменная =] объект.метод

Как правило, при выполнении метода никакое значение в программу не возвращается. В этом случае часть последнего оператора, находящаяся в квадратных скобках, должна отсутствовать. Однако есть методы, при выполнении которых в программу возвращается некое значение. Но и в этом случае часть последнего

оператора, находящаяся в квадратных скобках, может отсутствовать, если возвращаемое значение не представляет интерес (т. е. не используется в программе).

Методы можно рассматривать как встроенные процедуры. Их использование аналогично использованию встроенной процедуры MsgBox, которая является и подпрограммой, и функцией (см. главу 17).

Существует иерархия объектов. Высшим в иерархии объектов приложения (в нашем случае Excel) является объект Application (Приложение), все остальные объекты Excel «вложены» в него.

Схема вложения объектов друг в друга напоминает русскую матрешку, но (в отличие от матрешки) в каждый объект может быть вложено несколько объектов.

Полная ссылка на объект состоит из ряда имен, разделенных точками. Ряд начинается с Application и заканчивается именем самого объекта. Например, полная ссылка на ячейку A1 листа Код книги Архив имеет вид

```
Application.Workbooks("Архив").Worksheets("Код"). _  
Range("A1")
```

Приводить каждый раз полную ссылку на объект не обязательно. Часто достаточно ограничиться неполной (неявной) ссылкой, в которой активные в данный момент объекты опускаются.

Пусть в примере книга Архив является активной. Тогда ссылку можно сократить. Неполная ссылка имеет вид

```
Worksheets("Код").Range("A1")
```

В последних двух выражениях в скобках стоят строки, которые могут быть как угодно сложными. Например, выражение

```
Range("G" & CStr(CInt(Now - #1 Jan 2000#))).Select
```

означает выделение в столбце G активного листа Excel ячейки, номер которой равен количеству дней, прошедших с начала века.

Задание:

1) вставьте последний оператор перед строкой End Sub программы Век_20 (стр. 22);

2) выполните полученную программу и обратите внимание на положение активной ячейки Excel.

Видно, что конструкция объекта похожа на запись (см. главу 18). Поэтому в первом приближении объекты можно рассматривать как встроенные записи VBA, т. е. как записи, не определяемые пользователем компьютера.

Далее рассмотрим основные объекты Excel: Application (Приложение), Workbook (Книга), Workbooks (Книги), Worksheet (Лист), Worksheets (Листы), Range (Диапазон), Selection (Выделение) и др. Знание их свойств и методов полезно для разработки пользовательских функций и макросов Excel.

32. Объект Application

Как уже говорилось, объект Application (Приложение) занимает верхний уровень в иерархии объектов Excel. Это значит, что он управляет установками уровня приложения, т. е. такими, которые можно найти в окне *Параметры Excel* (это окно открывается при выполнении **Office > Параметры Excel**). Об этом говорит и оператор

```
Application.ReferenceStyle = xlR1C1
```

в тексте макроса MR, полученного с помощью макрорекордера (см. главу 30).

Однако роль объекта Application не ограничивается заданием параметров Excel. Если мы хотим в программе на VBA использовать встроенные функции Excel, то нам также необходим объект Application.

Используем функции СРЕДНЕЕ (AVERAGE) и СУММА (SUM) следующим образом.

1. Вставим модуль в активную книгу Excel. В окно кода введем программу

```
Sub ВстроенныеФункции()  
    Dim W As Single  
    W = Application. _  
        Average(Worksheets("Лист1").Range("A1:A4"))  
    MsgBox "Среднее значение диапазона = " & CStr(W)  
    W = Application. _  
        Sum(Worksheets("Лист1").Range("A1:A4"))  
    MsgBox "Сумма значений диапазона = " & CStr(W)  
End Sub
```

2. Перейдем на лист Лист1 активной книги.

3. Введем в ячейку A1 число 100, в ячейку A2 — 200, в A3 — 300, в A4 — 400.

4. Запустим макрос ВстроенныеФункции. Появится первое окно, показывающее среднее значение введенных чисел (рис. 32.1).

5. Для закрытия окна и продолжения выполнения макроса нажмем кнопку **ОК**. Появится второе окно, которое показывает сумму введенных чисел (рис. 32.2).

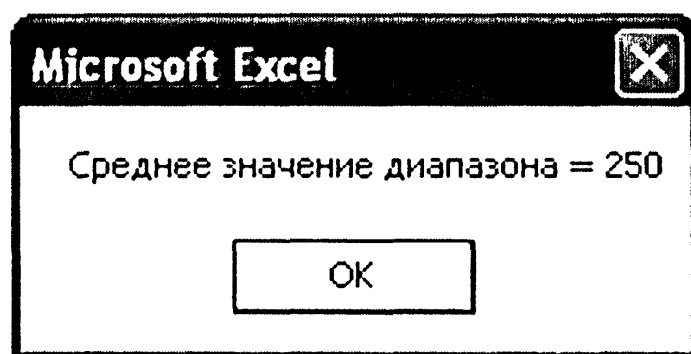


Рис. 32.1

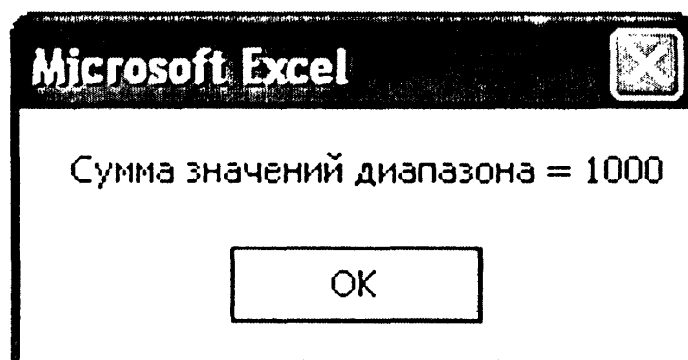


Рис. 32.2

6. Нажмем кнопку **ОК** для закрытия окна и завершения выполнения макроса.

Как видно из текста последней программы, встроенные функции Excel являются *свойствами* объекта Application.

Помимо встроенных функций Excel, объект Application имеет и другие свойства. Из длинного списка свойств выделим следующие.

- **ActiveWorkbook** — активная книга.
- **ActiveSheet** — активный лист активной книги.
- **ActiveCell** — активная ячейка на активном листе активной книги.

В следующем примере в активной ячейке устанавливается шрифт «курсив» и в нее вводится текст *Отчет за май*:

```
Sub Курсив()  
    With Application.ActiveCell  
        .Font.Italic = True  
        .Value = "Отчет за май"  
    End With  
End Sub
```

Использованный оператор `With` был определен на стр. 81.

Подробнее свойства `ActiveWorkbook`, `ActiveSheet` и `ActiveCell` (являющиеся объектами) рассматриваются в следующих трех главах.

- `Calculation` — режим вычислений (см. окно *Параметры Excel* после выполнения **Office > Параметры Excel > Формулы**).

Основные значения этого свойства:

`xlCalculationAutomatic` — автоматический расчет (при изменении данных в ячейках Excel пересчет по формулам производится автоматически — привычный режим работы Excel);

`xlCalculationManual` — расчет вручную (например, при нажатии клавиши **F9**).

В качестве примера использования свойства `Calculation` рассмотрим следующий оператор, устанавливающий режим автоматического расчета:

```
Application.Calculation = xlCalculationAutomatic
```

- `Dialogs` — семейство (коллекция) диалоговых окон Excel.

Часто встречающиеся в VBA термины «семейство» и «коллекция» означают одно и то же — группу однотипных объектов. Мы будем использовать первый термин — «семейство».

С помощью свойства `Dialogs` объекта `Application` можно производить открытие диалоговых окон Excel. Так, например, выполнение оператора

```
Application.Dialogs(xlDialogOpen).Show
```

приводит к открытию известного нам диалогового окна *Открытие документа*.

В этом операторе `Application.Dialogs(xlDialogOpen)` — объект (типа `Balloon`); `Show` — метод этого объекта.

С методом `Show` мы уже встречались в главах 25 и 26. Имеется в виду оператор открытия пользовательской формы

```
UserForm1.Show
```

Задание:

1) перед последней строкой программы *ВстроенныеФункции* (стр. 137) вставьте оператор открытия диалогового окна сохранения документа

```
Application.Dialogs(xlDialogSaveAs).Show
```

при этом программа примет вид

```
Sub ВстроенныеФункции()  
    Dim W As Single  
    W = Application. _  
        Average(Worksheets("Лист1").Range("A1:A4"))  
    MsgBox "Среднее значение диапазона = " & CStr(W)  
    W = Application. _  
        Sum(Worksheets("Лист1").Range("A1:A4"))  
    MsgBox "Сумма значений диапазона = " & CStr(W)  
    Application.Dialogs(xlDialogSaveAs).Show  
End Sub
```

2) перейдите в окно Excel;

3) введите произвольные числа в ячейки диапазона A1:A4 (например, 100, 200, 300 и 400);

4) выполните полученный макрос ВстроенныеФункции;

5) в открывшемся окне *Сохранение документа* задайте имя файла *КнигаApplication*, тип файла *Книга Excel с поддержкой макросов*;

6) нажмите кнопку **Сохранить**;

7) убедитесь, что полученный файл КнигаApplication содержит макрос ВстроенныеФункции.

В данной главе оператор With использовался нами для установки свойств объекта. Однако его можно использовать и для выполнения методов объекта. Так, например, оператор

```
Application.Dialogs(xlDialogSaveAs).Show
```

можно заменить конструкцией вида

```
With Application.Dialogs(xlDialogSaveAs)  
    .Show  
End With
```

В конструкции With могут быть свойства и методы одновременно. Усложним конец программы ВстроенныеФункции такой конструкцией:

```
Sub ВстроенныеФункции()  
    Dim W As Single  
    W = Application. _  
        Average(Worksheets("Лист1").Range("A1:A4"))  
    MsgBox "Среднее значение диапазона = " & CStr(W)  
    W = Application. _
```

```
Sum(Worksheets("Лист1").Range("A1:A4"))
MsgBox "Сумма значений диапазона = " & CStr(W)
With Application
    .ActiveCell.Value = "Отчет за май"
    .Dialogs(xlDialogSaveAs).Show
End With
End Sub
```

При этом по окончании выполнения макроса и сохранения книги (например, под именем КнигаApplication) в активной ячейке появится текст *Отчет за май*.

Из **методов** объекта Application назовем следующие.

- Quit — выход из Excel.

Оператор применения метода Quit имеет вид

```
Application.Quit
```

- Calculate — принудительные вычисления.

В качестве примеров использования метода Calculate рассмотрим следующие операторы:

```
Application.Calculate
Worksheets("Отчет7").Calculate
Worksheets("Отчет7").Range("A1:C10").Calculate
```

Если в Excel установлен режим «расчет вручную», то:

1) выполнение первого оператора приводит к расчету по формулам во всех открытых книгах Excel (изображенных кнопками на панели задач рабочего стола Windows), что эквивалентно нажатию клавиши F9;

2) выполнение второго оператора приводит к расчету на листе Отчет7 активной книги;

3) выполнение третьего оператора приводит к расчету в ячейках диапазона A1:C10 на листе Отчет7 активной книги.

Напомним, что настройка Excel на расчет вручную производится с помощью оператора

```
Application.Calculation = xlCalculationManual
```

Можно также в окне Excel выполнить следующую последовательность операций: **Office > Параметры Excel > Формулы > включить опцию Вручную**.

- OnTime — запуск макроса в заданный момент времени; время и имя макроса являются параметрами метода.

В качестве примера использования метода OnTime рассмотрим следующие две программы:

```
Sub Auto_Open()  
    Application.OnTime TimeValue("15:20:00"), "MyMacro"  
End Sub
```

```
Sub MyMacro()  
    Range("A1") = 13.13  
End Sub
```

Они находятся в одном модуле или разных модулях одной и той же книги Excel.

Сохраним книгу, содержащую эти программы, например, под именем КнигаOnTime. Далее выполним следующее:

- активизируем книгу КнигаOnTime;
- нажмем кнопку **Параметры** на панели *Предупреждение системы безопасности*;
- в появившемся окне *Параметры безопасности Microsoft Office* включим опцию **Включить это содержимое** и нажмем кнопку **ОК**.

При этом автоматически запустится программа Auto_Open (благодаря ее имени). Далее происходит выполнение оператора

```
Application.OnTime TimeValue("15:20:00"), "MyMacro"
```

через малые временные интервалы (так работает метод OnTime). Событием является совпадение текущего времени с заданным временем 15 часов 20 минут. Результат обработки этого события — запуск программы MyMacro, которая записывает число 13.13 в ячейку A1 активного листа. По окончании выполнения программы MyMacro управление передается в программу Auto_Open, т. е. продолжается выполнение оператора

```
Application.OnTime TimeValue("15:20:00"), "MyMacro"
```

через малые временные интервалы.

Программу MyMacro не трудно видоизменить так, чтобы она выдавала в ячейку A1 текущее время, начиная с 15 часов 20 минут:

```
Sub MyMacro()  
1: Range("A1") = Time  
2: Application.OnTime Now + TimeValue("00:00:01"), _  
    "MyMacro"  
End Sub
```

Первый оператор (с меткой 1) записывает время в ячейку A1 активного листа. В результате выполнения второго оператора через малые временные интервалы производится запуск программы MyMacro каждую секунду, поэтому каждую секунду выполняется первый оператор. В программу Auto_Open управление не передается.

Если удалить программу Auto_Open, то текущее время будет выдаваться после запуска программы MyMacro: **Разработчик > Макросы > строка MyMacro > Выполнить.**

33. Объекты Workbook, Workbooks и ActiveWorkbook

Объект Workbook (Книга) в иерархии объектов идет сразу после объекта Application. Легко понять следующие *свойства* объекта Workbook:

- Name — имя книги Excel с расширением (xls или xlsm, если в книге нет макросов, и xlsm, если макросы есть);
- FullName — имя книги с путем и расширением (полное имя книги).

Рассмотрим некоторые *методы* объекта Workbook.

- Close — закрытие книги Excel.
- Save — сохранение книги.
- SaveAs — сохранение книги.

Отличие этого метода от предыдущего состоит в том, что он имеет ряд необязательных параметров. К ним относятся FileName (Имя файла), FileFormat (Формат файла), Password (Пароль).

Примеры использования свойств и методов объекта Workbook приводятся ниже, при рассмотрении объектов типа Workbook. Слово «тип» здесь употребляется в том же смысле, что и в выражениях «переменная типа Integer» и «запись типа Session».

Объекты типа Workbook обладают свойствами и методами объекта Workbook.

Объект Workbooks (Книги) представляет собой объект, содержащий все открытые книги Excel. Этот объект называют еще семейством Workbooks.

Выражение

```
Workbooks ("Отдел_кадров")
```

означает объект типа Workbook, соответствующий открытой книге Excel с именем Отдел_кадров.

Рассмотрим основные *методы* объекта Workbooks.

- Activate — активизация указанной книги (из открытых), при которой ее первый лист становится активным.

Примером использования метода Activate является оператор активизации книги Отдел_кадров

```
Workbooks("Отдел_кадров").Activate
```

- Add — создание новой книги, которая сразу становится активной.

Если метод Add фигурирует в правой части оператора присваивания, то этот метод возвращает в программу созданную книгу Excel, являющуюся объектом типа Workbook.

Примером использования метода Add является второй оператор (с меткой 2) в программе НоваяКнига, приведенной в конце главы.

Отметим, что методом Add обладают все семейства. Он добавляет в семейство новый объект соответствующего типа. Так, в семейство Workbooks добавляется объект типа Workbook.

Как говорилось в предыдущей главе, ActiveWorkbook (Активная книга) является одним из свойств объекта Application. Оно представляет собой объект типа Workbook, соответствующий активной книге Excel. Как объект типа Workbook, свойство ActiveWorkbook обладает свойствами и методами объекта Workbook.

Чтобы получить пример использования свойств Name и FullName и метода Save объекта Workbook, выполним следующее.

1. В окне Excel с открытой книгой КнигаМакрорекордер (стр. 130) нажмем кнопку **Параметры** панели *Предупреждение системы безопасности*, включим опцию **Включить это содержимое** и нажмем кнопку **ОК**.

2. Выполним **Разработчик > Макросы > строка MR > Изменить**.

3. Перед последней строкой текста макроса MR вставим следующие операторы (рис 33.1):

```
1: Dim S As String
2: S = ActiveWorkbook.Name
3: MsgBox S
4: MsgBox ActiveWorkbook.FullName
5: ActiveWorkbook.Save
```

4. Запустим макрос MR, например, кликнув на стрелке на панели инструментов VB.

В результате выполнения второго оператора (с меткой 2) получается строка S, значение которой равно имени активной книги.

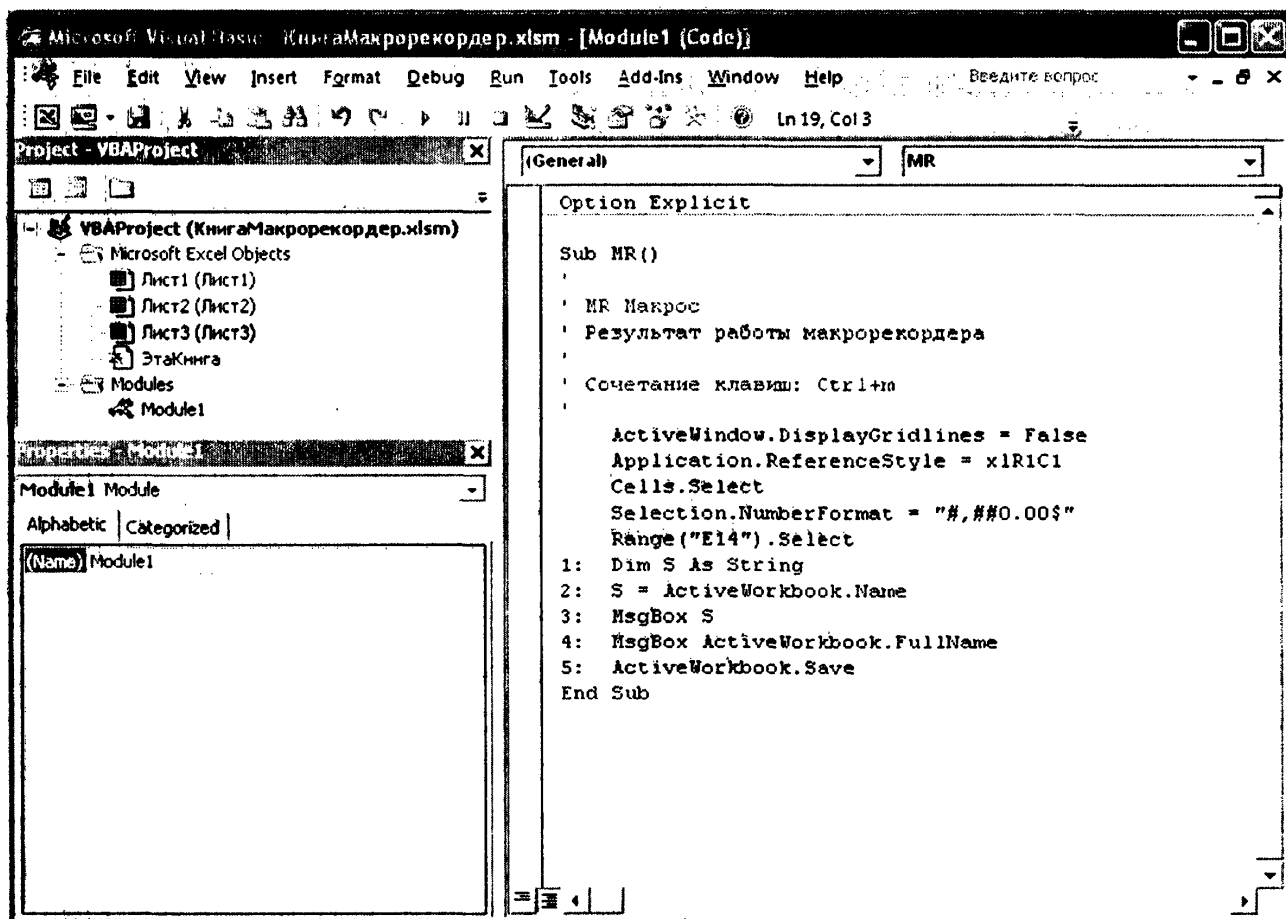


Рис. 33.1. Окно VB с текстом макроса MR после ввода дополнительных операторов

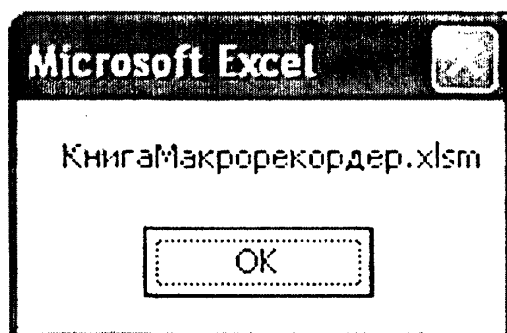


Рис. 33.2

Третий оператор выводит эту строку на экран дисплея (рис. 33.2). Для продолжения выполнения макроса MR следует нажать кнопку ОК. Четвертый оператор выводит на экран полное имя активной книги (рис. 33.3). После нажатия кнопки ОК выполняется пятый оператор, который сохраняет активную книгу.

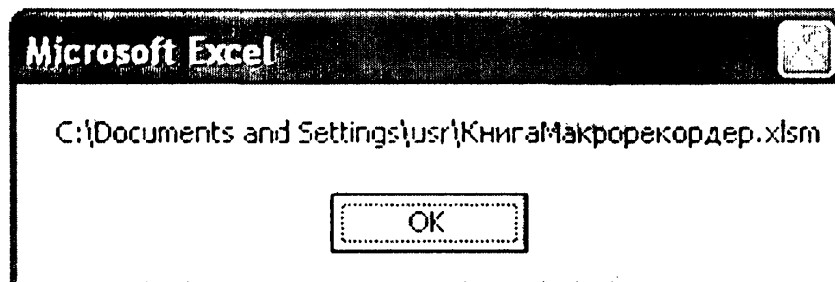


Рис. 33.3

33. Объекты Workbook, Workbooks и ActiveWorkbook

В следующем примере будет создана новая книга, в нее будет введено числовое значение, затем новая книга будет сохранена и закрыта, закрыто будет также окно Excel.

1. В новую активную книгу Excel вставим модуль. В окно кода введем следующую программу (рис. 33.4):

```
Sub НоваяКнига()  
1: Dim wbNewWorkbook As Workbook  
2: Set wbNewWorkbook = Workbooks.Add  
3: wbNewWorkbook.Worksheets("Лист1").Range("A1").Value _  
    = 100  
4: wbNewWorkbook.SaveAs _  
    "c:\Documents and Settings\usr\Час0.xlsx"  
5: wbNewWorkbook.Close  
6: MsgBox "Книга закрыта"  
7: Application.Quit  
End Sub
```

2. Запустим макрос НоваяКнига.

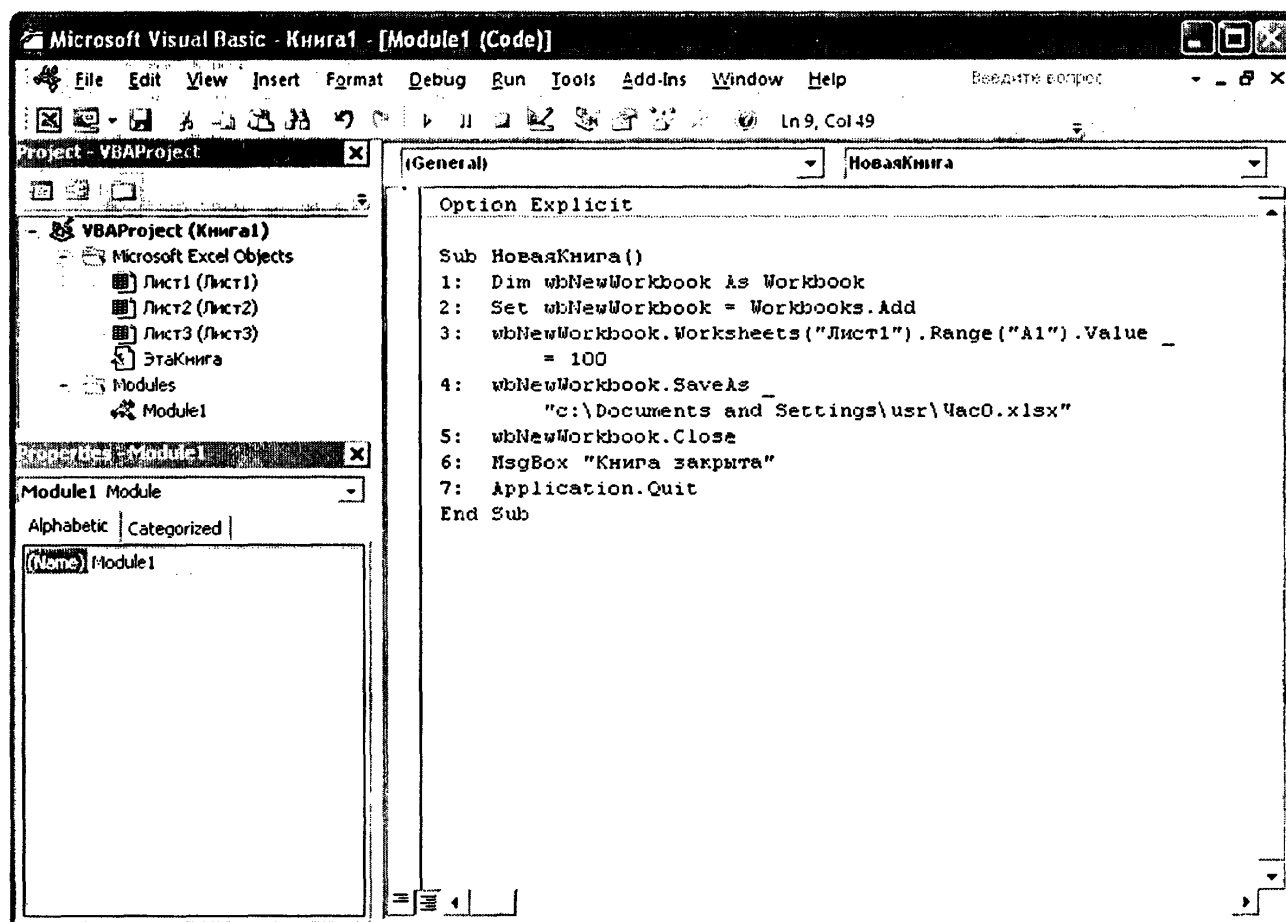


Рис. 33.4. Окно VB с текстом макроса НоваяКнига в окне кода

3. Когда появится окно с сообщением, что книга закрыта, кликнем на кнопке **ОК**. Макрос завершит работу, и произойдет выход из Excel.

4. Откроем книгу Час0, находящуюся в папке c:\Documents and Settings\usr, и увидим число 100 в ячейке A1 листа Лист1.

Программа НоваяКнига начинается с оператора объявления переменной `wbNewWorkbook` (первый оператор, с меткой 1). В результате выполнения второго оператора создается новая книга, которая присваивается переменной `wbNewWorkbook`. В ячейку A1 листа Лист1 этой книги вводится число 100 (третий оператор).

При создании книги табличный процессор Excel дает ей имя по умолчанию, например, Книга2. Так как имя точно не известно, то в программе используется переменная `wbNewWorkbook` вместо `Workbooks("Книга2")`. Сохраняется книга под полным именем c:\Documents and Settings\usr\Час0.xlsx (четвертый оператор).

Пятый оператор закрывает книгу Час0, а шестой оператор выводит на экран дисплея окно с сообщением об этом. Седьмой оператор осуществляет выход из Excel.

Вернемся ко второму оператору. Он начинается с ключевого слова `Set`, являющегося признаком того, что в операторе присваивания (следующем за `Set`) фигурируют объекты: `wbNewWorkbook` и `Workbooks.Add` — объекты типа `Workbook`.

Отметим, что в первом операторе тип `Workbook` можно заменить типом `Object` (см. приложение 1), аналогичным известным нам типу `Variant`.

34. Объекты Worksheet, Worksheets и ActiveSheet

Объект Worksheet (Лист) в иерархии объектов идет сразу после объекта Workbook. Из *свойств* объекта Worksheet рассмотрим следующие.

- Name — имя листа.
- Cells — семейство всех ячеек листа или конкретная ячейка, если в скобках указаны номера строки и столбца, на пересечении которых ячейка расположена.

Первый индекс в скобках задает номер строки, а второй индекс задает номер столбца. Между индексами ставится запятая.

Отметим, что свойство Cells будет рассматриваться и в следующей главе, применительно к объекту Range (Диапазон).

Из *методов* объекта Worksheet рассмотрим два:

- Activate — активизация листа;
- Delete — удаление листа из книги.

Примеры использования свойств и методов объекта Worksheet приводятся ниже, при рассмотрении объектов типа Worksheet. Последние объекты обладают свойствами и методами объекта Worksheet.

Worksheets (Листы) представляет собой объект, содержащий в себе все листы книги Excel. Этот объект называют еще семейством Worksheets.

Каждый лист объекта Worksheets может быть идентифицирован либо по номеру, либо по имени. Например, Worksheets(1) обозначает первый лист книги, а Worksheets(«Лист1») — лист с именем Лист1. И Worksheets(1), и Worksheets(«Лист1») являются объектами типа Worksheet.

В качестве примеров использования методов Activate и Delete объекта Worksheet рассмотрим следующие операторы:

```
Worksheets(3).Activate
```

```
Worksheets("Лист2").Delete
```

Первый оператор активизирует третий лист активной книги, а второй оператор удаляется лист Лист2.

При работе с объектом Worksheets часто применяется *метод* Add, добавляющий в семейство новый лист, который сразу становится активным. Если этот метод фигурирует в правой части оператора присваивания, то он возвращает в программу созданный лист, являющийся объектом типа Worksheet.

Примером использования метода Add является второй оператор (с меткой 2) в программе НовыйЛист, приведенной в конце главы.

Семейства, в частности Worksheets, обладают полезным *свойством* Count (Счет). Это свойство позволяет определить количество объектов, составляющих семейство.

Примером использования свойства Count является следующая программа:

```
Sub КоличествоЛистов()  
    MsgBox Str(Worksheets.Count)  
End Sub
```

При выполнении последней программы на экране дисплея появляется окно, содержащее количество листов в активной книге и кнопку **ОК**. Для окончания выполнения программы следует нажать кнопку **ОК**.

Как говорилось в главе 32, ActiveSheet (Активный лист) является одним из свойств объекта Application. Это свойство представляет собой объект типа Worksheet, соответствующий активному листу активной книги. Как объект типа Worksheet, свойство ActiveSheet обладает свойствами и методами объекта Worksheet.

Примером использования свойства Name объекта Worksheet является следующий оператор, при выполнении которого активному листу присваивается название месяца:

```
ActiveSheet.Name = "Июль"
```

В качестве примера использования свойства Cells объекта Worksheet рассмотрим программу, которая проверяет значения в ячейках диапазона A1:E4 активного листа и при отрицательном значении окрашивает ячейку в зеленый цвет и назначается шрифт «курсив»:

```
Sub Цвет()  
    Dim i As Integer, j As Integer  
    For i = 1 To 4  
        For j = 1 To 5  
            With ActiveSheet.Cells(i, j)  
                If .Value < 0 Then  
                    .Font.Color = QBColor(10)  
                    .Font.Italic = True  
                End If  
            End With  
        Next j  
    Next i  
End Sub
```

В качестве примера использования метода Add объекта Worksheets рассмотрим программу вставки листа в активную книгу.

1. В окно кода введем текст

```
Sub НовыйЛист()  
1: Dim wsNewWorksheet As Worksheet  
2: Set wsNewWorksheet = Worksheets.Add  
3: wsNewWorksheet.Name = Format(Date, "d mmmm yyyy")  
End Sub
```

2. Выполним макрос НовыйЛист.

3. Если мы находимся в окне Excel, то увидим новый лист, именем которого является текущая дата.

Программа НовыйЛист (похожая на программу НоваяКнига из предыдущей главы) начинается с оператора объявления переменной wsNewWorksheet (первый оператор, с меткой 1). Этой переменной присваивается создаваемый лист (второй оператор). Далее используется свойство Name для задания имени листа (третий оператор).

Функция Format применяется для управления внешним видом даты. Первый аргумент — функция Date, возвращающая текущую дату. Второй аргумент — строка, задающая следующий формат даты: число, полное название месяца, год полностью.

35. Объекты Range, Selection и ActiveCell

Объект Range (Диапазон) в иерархии объектов идет сразу после объекта Worksheet. Он позволяет работать со следующими элементами Excel:

- 1) диапазон ячеек;
- 2) диапазон столбцов;
- 3) диапазон строк;
- 4) отдельная ячейка.

Рассмотрим следующие *свойства* объекта Range.

- Formula — формула в ячейках диапазона при стиле адресации A1.

Например, операторы

```
Range("C2:C8").Formula = "=$A$4+COS($A$10) "  
Range("D:E").Formula = "=$A$4+COS($A$10) "  
Range("2:2").Formula = "=$A$4+COS($A$10) "  
Range("B3").Formula = "=$A$4+COS($A$10) "
```

используются для ввода формулы

```
=$A$4+COS($A$10)
```

в ячейки диапазона C2:C8, в ячейки столбцов D и E, в ячейки второй строки и в ячейку B3 активного листа.

- FormulaR1C1 — формула в ячейках диапазона при стиле адресации R1C1.

Например, для ввода формулы

```
=SQRT(R5C8)
```

в ячейку B1 листа Л1 используется оператор

```
Worksheets("Л1").Range("B1").FormulaR1C1 = "=SQRT(R5C8) "
```

- Address — адрес ячейки.
- Offset — диапазон, смещенный относительно активного (выделенного) диапазона согласно значениям, заключенным в скобки.

Первый индекс в скобках задает смещение по вертикали, а второй — по горизонтали. Между индексами ставится запятая.

- Value — массив значений в ячейках диапазона (или одно значение, если диапазон содержит одну ячейку).
- Columns — семейство столбцов, из которых состоит диапазон.
- Rows — семейство строк, из которых состоит диапазон.
- Cells — семейство ячеек диапазона.

Отметим, что свойство Cells рассматривалось в предыдущей главе, применительно к объекту Worksheet (Лист).

Назовем следующие *методы* объекта Range.

- Clear — удаление содержимого ячеек диапазона.

Например, следующий оператор очищает ячейки A1:G37 на листе Лист1:

```
Worksheets("Лист1").Range("A1:G37").Clear
```

- Select — активизация (выделение) ячеек диапазона.

Свойствами и методами объекта Range обладают объекты типа Range.

При работе с объектами типа Range удобно использовать цикл For Each...Next, аналогичный циклу For...Next. Синтаксис этого оператора можно изучить с помощью справочной системы, вызываемой нажатием клавиши F1.

В следующем примере оператор цикла For Each...Next используется для возведения в квадрат значений всех ячеек диапазона A1:A6 на листе Лист1 активной книги:

```
Sub Квадрат()  
    Dim x As Range  
    For Each x In Worksheets("Лист1").Range("A1:A6")  
        x.Value = x.Value ^ 2  
    Next  
End Sub
```

Полезным является объект Selection (Выделение), который позволяет работать с выделенными (активными) ячейками. Являясь объектом типа Range, объект Selection обладает свойствами Columns, Rows и Cells. В данном случае перечисленные свойства являются семействами соответственно столбцов, строк и ячеек выделенного (активного) диапазона.

Пример. Следующая программа вставляет таблицу умножения в ячейки выделенного диапазона активного листа:

```
Sub ТаблицаУмножения()  
    Dim m As Integer, n As Integer  
    Dim i As Integer, j As Integer  
    m = Selection.Rows.Count      'количество строк  
    n = Selection.Columns.Count   'количество столбцов  
    For i = 1 To m                '6-ой оператор  
        For j = 1 To n           '7-ой оператор  
            Selection.Cells(i, j).Value = i * j  
        Next j  
    Next i  
End Sub
```

В четвертом и пятом операторах фигурирует свойство Count, которое позволяет определить количество объектов, составляющих семейство. В результате выполнения этих операторов переменным m и n присваиваются значения, равные соответственно количеству строк и столбцов в выделенном диапазоне.

Восьмой оператор может иметь вид

```
Selection.Cells(i, j) = i * j
```

В этом случае .Value присутствует неявно.

Задание:

- 1) введите программу ТаблицаУмножения в окно кода;
- 2) перейдите в окно Excel и выделите диапазон ячеек для таблицы умножения;
- 3) выполните макрос ТаблицаУмножения;
- 4) изучив результат, убедитесь в правильности работы программы.

Как говорилось в главе 32, ActiveCell (Активная ячейка) является одним из свойств объекта Application. Это свойство представляет собой объект типа Range, соответствующий активной ячейке на активном листе активной книги. Как объект типа Range, свойство ActiveCell обладает свойствами и методами объекта Range.

В программе СвойстваДиапазона, приведенной ниже, содержатся примеры использования рассмотренных свойств и методов объекта Range.

Будем считать, что активным является лист Лист1.

1. Введем число 100 в ячейку B1, 200 — в B2 и 300 — в B3.

2. В ячейку B4 введем формулу

```
=СУММ(B1:B3)
```

3. Войдем в среду VB и вставим модуль в активную книгу Excel.

4. В окно кода введем следующий текст:

```
Sub СвойстваДиапазона()  
1: Worksheets("Лист1").Range("A1").Select  
2: ActiveCell.Offset(2, 3).Select  
3: MsgBox "Текущая ячейка – " & ActiveCell.Address  
4: MsgBox "Значение ячейки B4 = " & Range("B4").Value  
5: MsgBox "Формула в ячейке B4: " & _  
    Range("B4").Formula  
End Sub
```

5. Перейдем на лист Лист1 и запустим макрос СвойстваДиапазона. Появится первое окно с сообщением *Текущая ячейка – \$D\$3* и кнопкой **ОК** (рис. 35.1).

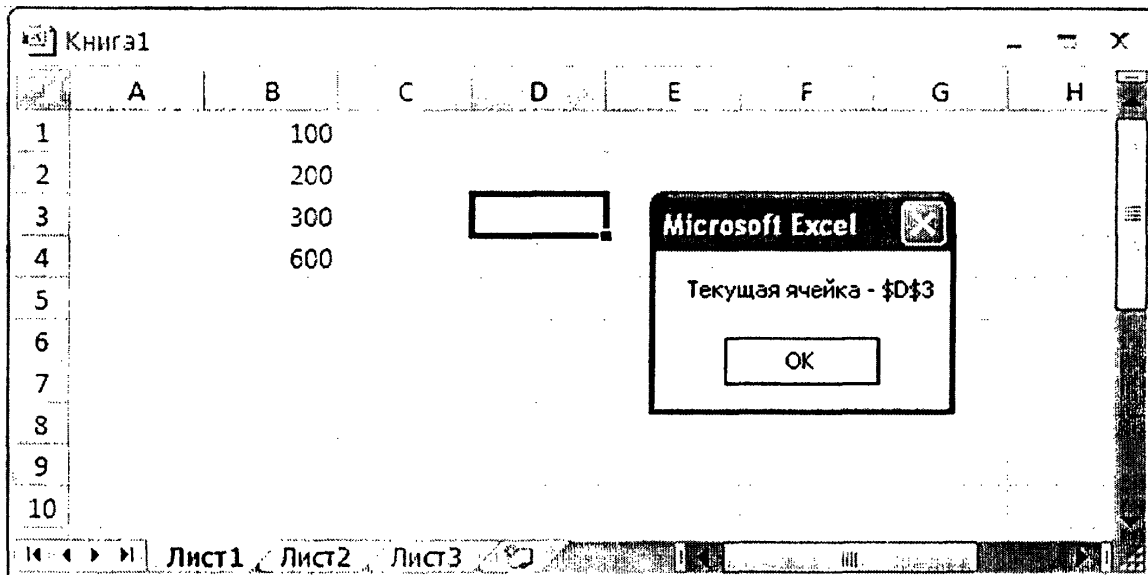


Рис. 35.1. Лист Excel с первым окном

6. Нажмем кнопку **ОК**. Появится окно с сообщением *Значение ячейки B4 = 600* и кнопкой **ОК**.

7. Нажмем кнопку **ОК**. Появится окно с сообщением *Формула в ячейке B4: =SUM(B1:B3)* и кнопкой **ОК**.

8. Нажмем кнопку **ОК** для закрытия окна с сообщением и завершения выполнения макроса.

Первый оператор программы СвойстваДиапазона (с меткой 1) производит активизацию ячейки A1 на листе Лист1. После этого с помощью свойства Offset активизируется ячейка D3, сдвинутая относительно ячейки A1 на 2 по вертикали и 3 по горизонтали (второй оператор). Далее последовательно выводятся три окна с сообщениями и кнопками **ОК**, первое из которых изображено на рис. 35.1.

36. Просмотр объектов

Информацию о любом объекте можно получить с помощью справочных систем, вызываемых нажатием клавиш **F1** и **F2**.

Допустим, что нас интересует объект **Range**.

Находясь в среде VB, нажмем клавишу **F2**. При этом откроется окно просмотра объектов.

В списке **Classes** (Классы) найдем строку **Range**, соответствующую объекту **Range**, и (кликнув) выделим ее. При этом появится список свойств и методов объекта **Range**, называемый *Members of 'Range'* (Компонент 'Range'), см. рис. 36.1.

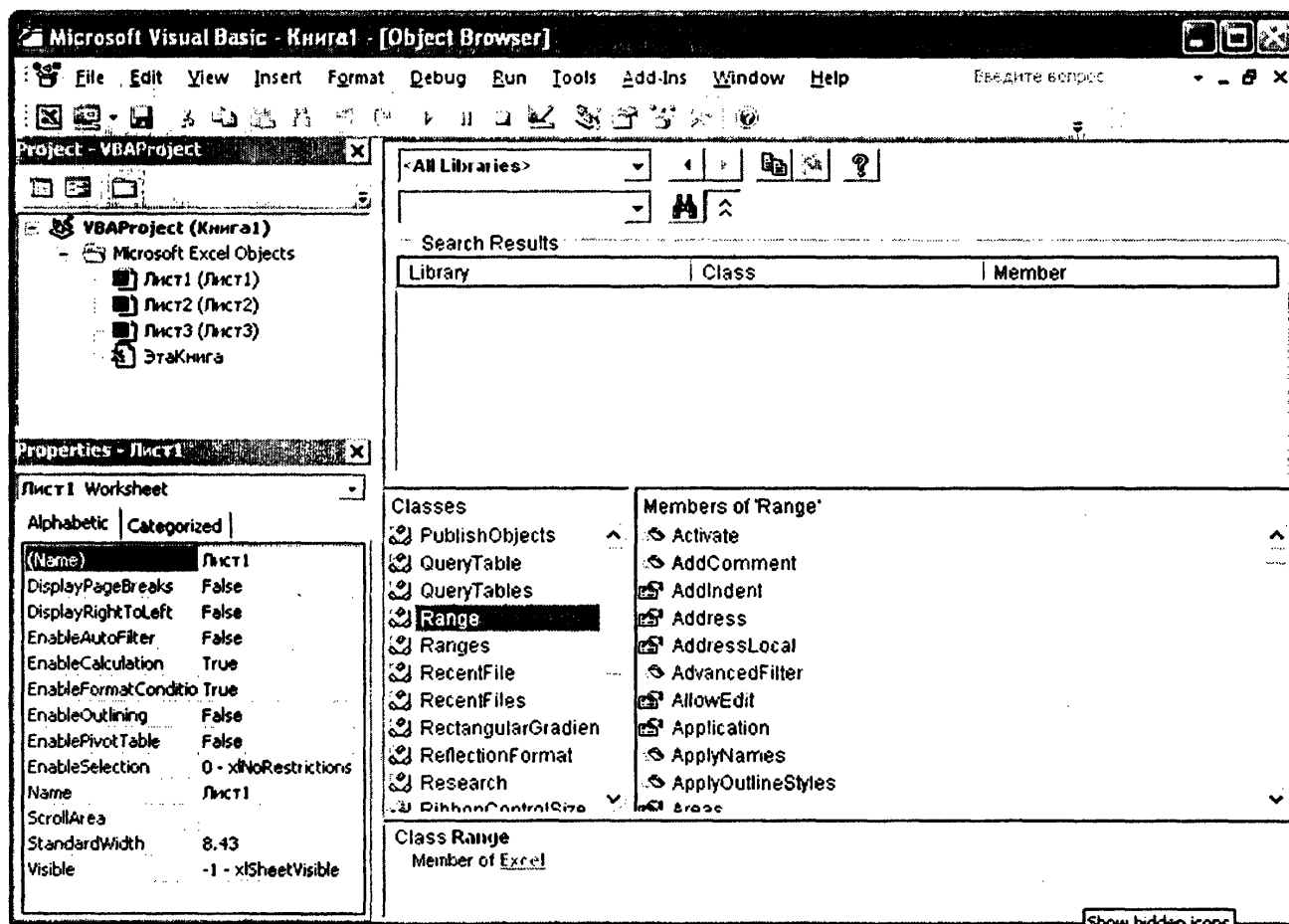


Рис. 36.1. Окно VB, включающее в себя окно просмотра объектов

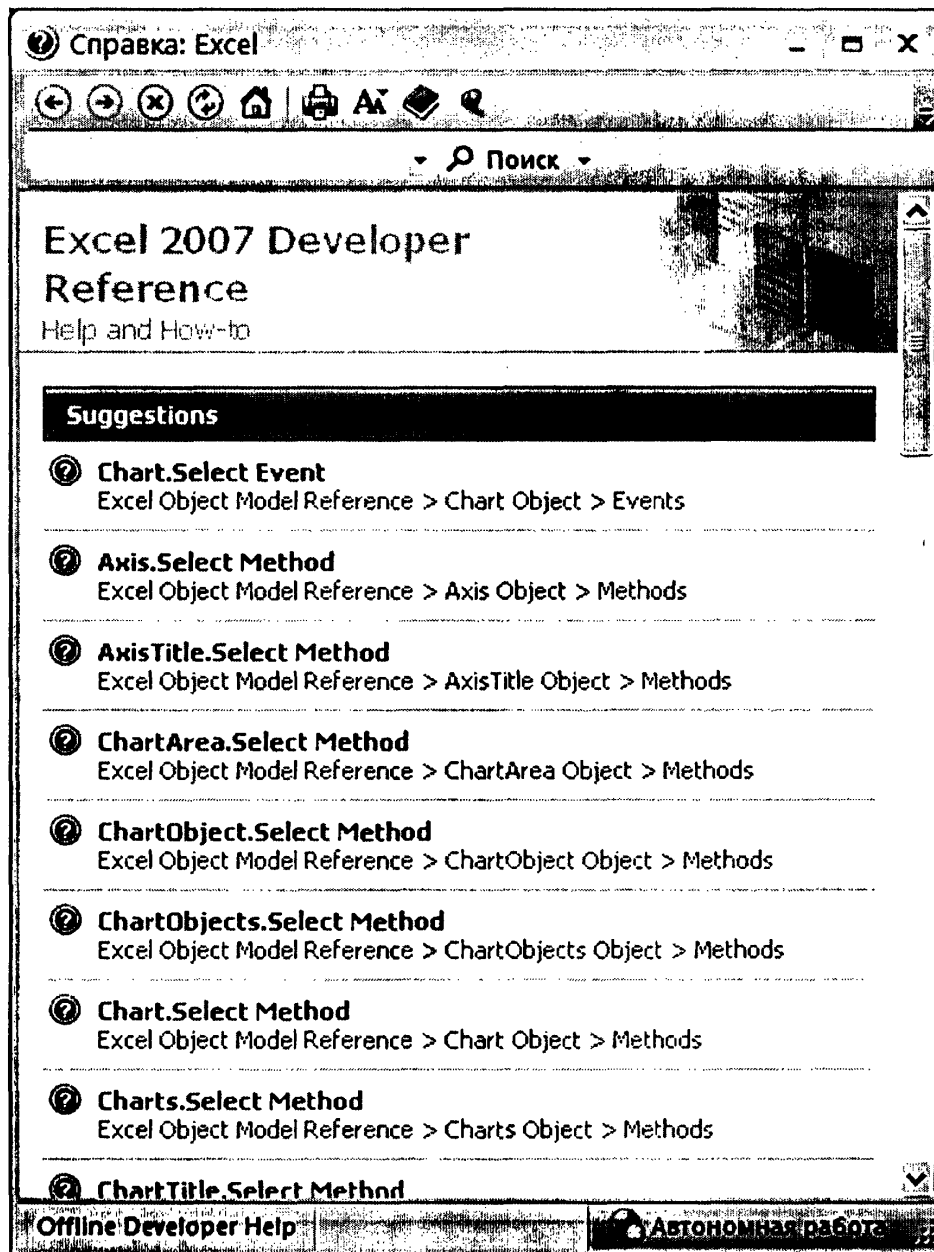


Рис. 36.2. Окно с гиперссылками и строкой поиска информации

Для получения справки о методе `Select` объекта `Range` выделим строку `Select` в последнем списке и нажмем клавишу `F1`. При этом откроется окно справочной системы Excel (рис. 36.2), с помощью которого легко найти необходимую информацию (рис. 36.3).

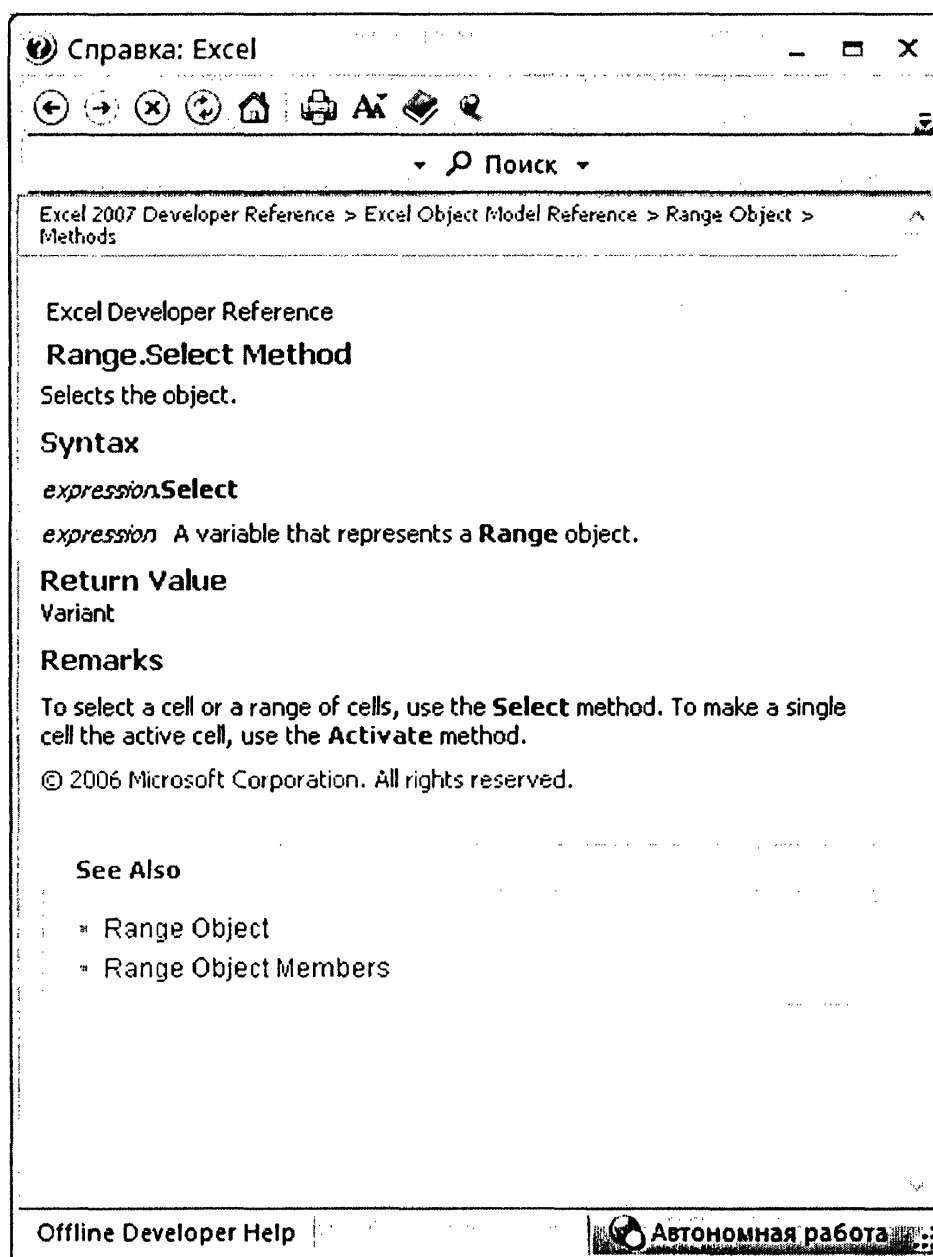


Рис. 36.3. Окно с описанием метода Range.Select

37. Использование таблицы Excel в качестве пользовательского интерфейса программ

Мы рассмотрели основные объекты Excel. Теперь можем использовать таблицу Excel в качестве пользовательского интерфейса программ.

Начнем с программы расчета длины катета прямоугольного треугольника по длинам гипотенузы и другого катета.

Пусть в ячейке A4 задана известная длина катета, в ячейке C4 — известная длина гипотенузы, ячейка B5 — для результата расчета.

Программа имеет вид:

```
Sub Катет()  
    Dim a As Single  
    Dim b As Single  
    Dim c As Single  
    a = Range("A4").Value  
    c = Range("C4").Value  
    If c <= a Then  
        Range("B5").Value = "Error" 'сообщение об ошибке  
    End If  
    b = Sqr(c ^ 2 - a ^ 2)  
    Range("B5").Value = b  
End Sub
```

Задание:

- 1) в окно кода введите текст программы Катет;
- 2) перейдите в окно Excel, в ячейки A4 и C4 любого листа введите значения 4.3 и 5.1 соответственно;
- 3) запустите макрос Катет;
- 4) убедитесь, что в ячейке B5 активного листа появилось значение 2.742261.

Известно, что длина гипотенузы c должна быть больше длины катета a . Если число, введенное нами в ячейку C4, меньше или равно числу, введенному в ячейку A4, то при выполнении программы Катет логическое выражение $c \leq a$ примет значение True. В результате этого в ячейку B5 будет выдано сообщение об ошибке *Error*, после чего выполнится оператор окончания выполнения программы End (этот оператор нам встретился впервые).

Нетрудно понять, как работает следующий макрос решения задачи транспонирования матрицы относительно неглавной диагонали, сформулированной в главе 21:

```
Sub ТРАНСПН()
    Dim m As Integer, n As Integer
    Dim i As Integer, j As Integer
    m = Selection.Rows.Count      'количество строк
    n = Selection.Columns.Count    'количество столбцов
    For j = 1 To n
        For i = 1 To m
            Selection.Cells(j + m + 1, i) = _
                Selection.Cells(m + 1 - i, n + 1 - j)
        Next i
    Next j
End Sub
```

	A	B	C	D	E	F
1						
2		1.1	2.00E+05	3	-4.00E+06	
3		5.60E+04	1.20E-07	14	22	
4		12	-0.7	0	4.00E+06	
5						
6		4000000	22	-4000000		
7		0	14	3		
8		-0.7	1.2E-07	200000		
9		12	56000	1.1		
10						

Рис. 37.1. Лист Excel с исходной матрицей и результатом ее транспонирования относительно неглавной диагонали

Используется макрос следующим образом:

- 1) в таблице Excel выделяем матрицу, которую надо транспонировать относительно неглавной диагонали;
- 2) запускаем макрос ТРАНСПН.

Ниже исходной матрицы появляется результат выполнения макроса (рис. 37.1).

38. Еще два макроса Excel. Личная книга макросов

Создадим макрос, выполняющий следующие операции.

1. При первом запуске макроса значение ячейки A3 листа Лист1 записывается в ячейку B2 листа Лист2.

2. При втором запуске макроса значение ячейки A3 листа Лист1 записывается в ячейку B4 листа Лист2.

3. При третьем запуске макроса значение ячейки A3 листа Лист1 записывается в ячейку B6 листа Лист2 и т. д.

В ячейку Лист1!A3 значения вводятся вручную перед очередным запуском макроса.

Используем ячейку Лист1!A4 в качестве вспомогательной ячейки. В нее следует записывать нуль перед серией запусков макроса.

В активную книгу Excel вставим модуль, и далее в окно кода введем следующее:

```
Sub Macr1()  
    Dim i As Integer  
    Dim M As Range  
    Set M = Range("Лист2!B1:Лист2!B200")  
    i = Range("Лист1!A4").Value + 2  
    M(i) = Range("Лист1!A3").Value  
    Range("Лист1!A4").Value = i  
End Sub
```

В этой программе:

- i — вспомогательная переменная;
- M — массив, соответствующий диапазону ячеек B1:B200 листа Лист2;
- Range("Лист1!A4").Value и Range("Лист1!A3").Value — значения ячеек A4 и A3 листа Лист1.

При первом запуске макроса Macr1 будет происходить следующее:

- 1) переменной i будет присвоено значение 2;

2) переменной $m(2)$ (т. е. ячейке B2 листа Лист2) будет присвоено значение ячейки A3 листа Лист1, например, 100;

3) в ячейку A4 листа Лист1 будет записано значение i , т. е. 2.

При втором запуске макроса:

1) переменной i будет присвоено значение 4;

2) переменной $m(4)$ (т. е. ячейке B4 листа Лист2) будет присвоено значение ячейки A3 листа Лист1, например, 200;

3) в ячейку A4 листа Лист1 будет записано значение i , т. е. 4.

И так далее. Таким образом, макрос `Macr1` выполняет нужные операции (рис. 38.1).

Разработанный макрос `Macr1` имеет два недостатка:

- этим макросом можно пользоваться только в той книге, в которой он разрабатывался;
- количество запусков макроса (в одной серии) не должно превосходить 100, т. к. используется массив M , содержащий 200 элементов.

Разработаем макрос `Macr2`, выполняющий те же операции, что и макрос `Macr1`, но лишенный этих недостатков.

1. Откроем новую книгу Excel, например, Книга1.

2. Выполним **Разработчик** > **Записать макрос**, т. е. включим макрорекордер. Появится окно *Запись макроса*.

3. В текстовое поле *Сохранить в* с помощью скрытого списка введем *Личная книга макросов* (рис. 38.2) и кликнем на кнопке **ОК**.

4. Кликнем на кнопке **Остановить запись**.

5. Откроем окно VB, кликнем на строке **VBAProject (PERSONAL.XLSB)** в окне проекта и далее выполним **File** > **Save PERSONAL.XLSB**.

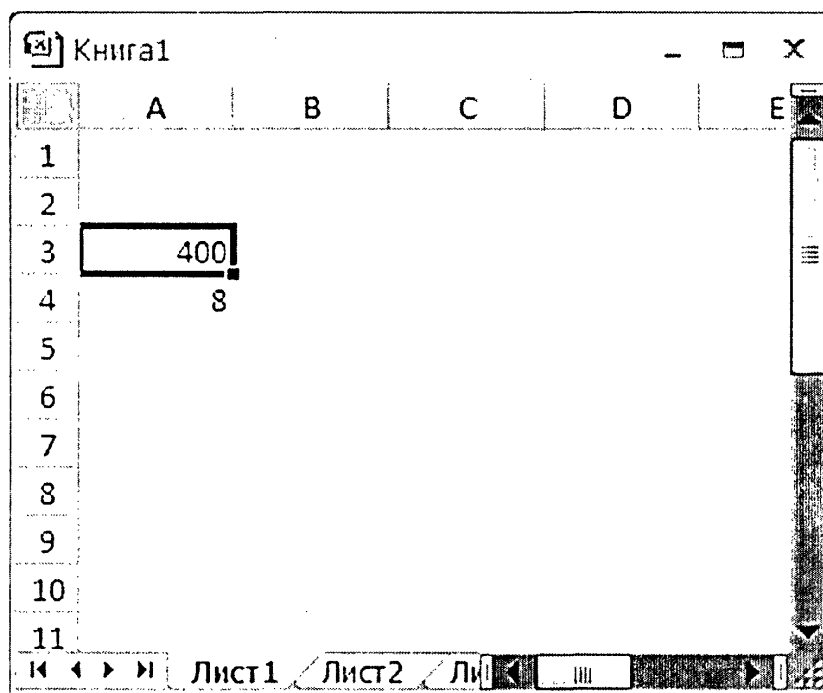
6. Закроем окно VB.

7. Попытаемся закрыть окно Excel, кликнув на крестике в правом верхнем углу. Появится окно с вопросом *Сохранить внесенные изменения в личной книге макросов?* Кликнем на кнопке **Да**, что приведет к закрытию окна Excel.

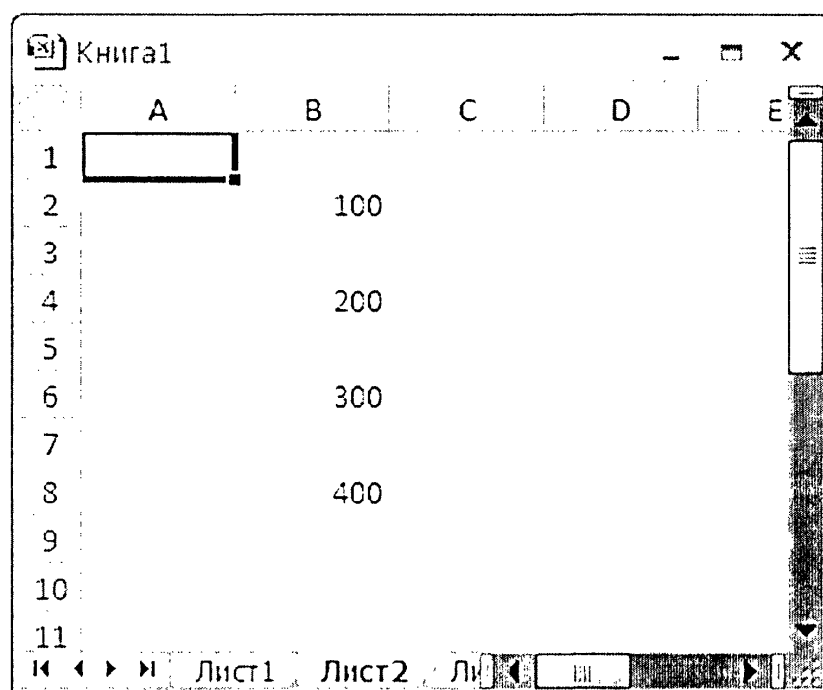
В результате будет создана личная книга макросов **PERSONAL.XLSB**.

Личная книга макросов — это книга, открываемая автоматически при запуске табличного процессора Excel. Макросами личной книги можно пользоваться в любой книге Excel данного пользователя.

Активизируем новую книгу, а затем откроем окно VB. В окне проекта VB кликнем на строке **VBAProject (PERSONAL.XLSB)** и



а)



б)

Рис. 38.1. Листы Лист1 (а) и Лист2 (б) после четвертого запуска макроса Macr1: в ячейку Лист1!А3 последовательно вводились значения 100, 200, 300 и 400

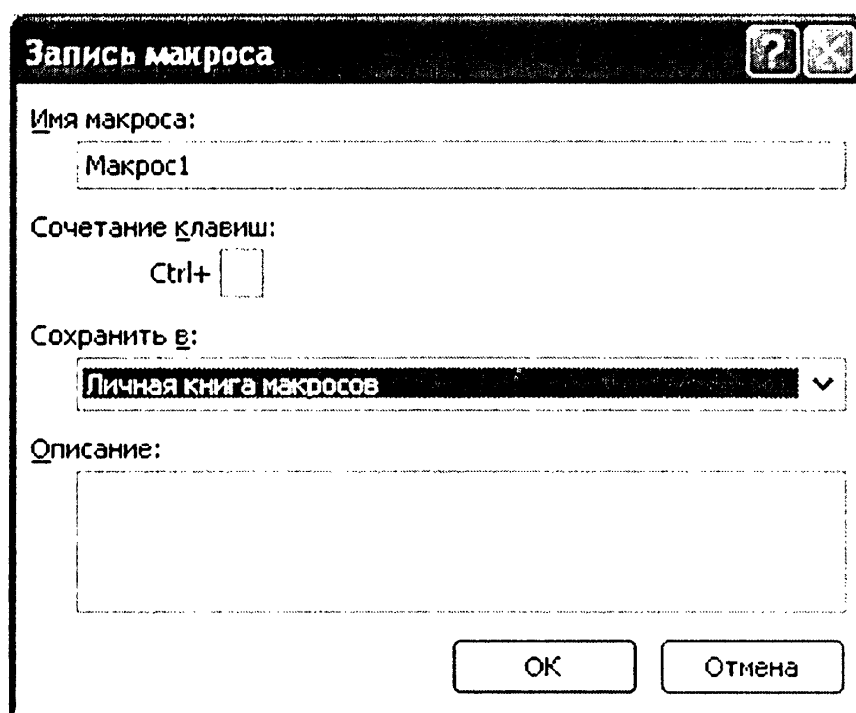


Рис. 38.2

далее выполним **Insert (Вставка) > Module (Модуль)**, что означает вставку модуля в личную книгу макросов.

В окно кода введем следующий текст программы, не содержащий массив **М** (рис. 38.3):

```
Sub Macr2()  
    Dim i As Integer  
    i = Range("Лист1!A4").Value + 2  
    Range("Лист2!B" & CStr(i)) = _  
        Range("Лист1!A3").Value  
    Range("Лист1!A4").Value = i  
End Sub
```

На рис. 38.3 модуль **Module1** в окне проекта — результат работы макрорекордера, который использовался для создания личной книги макросов **PERSONAL.XLSB** (этот модуль можно удалить).

Для сохранения изменений, внесенных в личную книгу макросов, надо в среде **VB** выполнить **File > Save PERSONAL.XLSB**.

Созданный макрос **Macr2** используется так же, как макрос **Macr1**, но им можно пользоваться в любой книге Excel данного пользователя и количество запусков макроса **Macr2** не ограничено.

Личной книгой макросов могут пользоваться и другие пользователи. Для этого файл **PERSONAL.XLSB** надо скопировать в соответствующую папку другого пользователя. Поговорим о месте

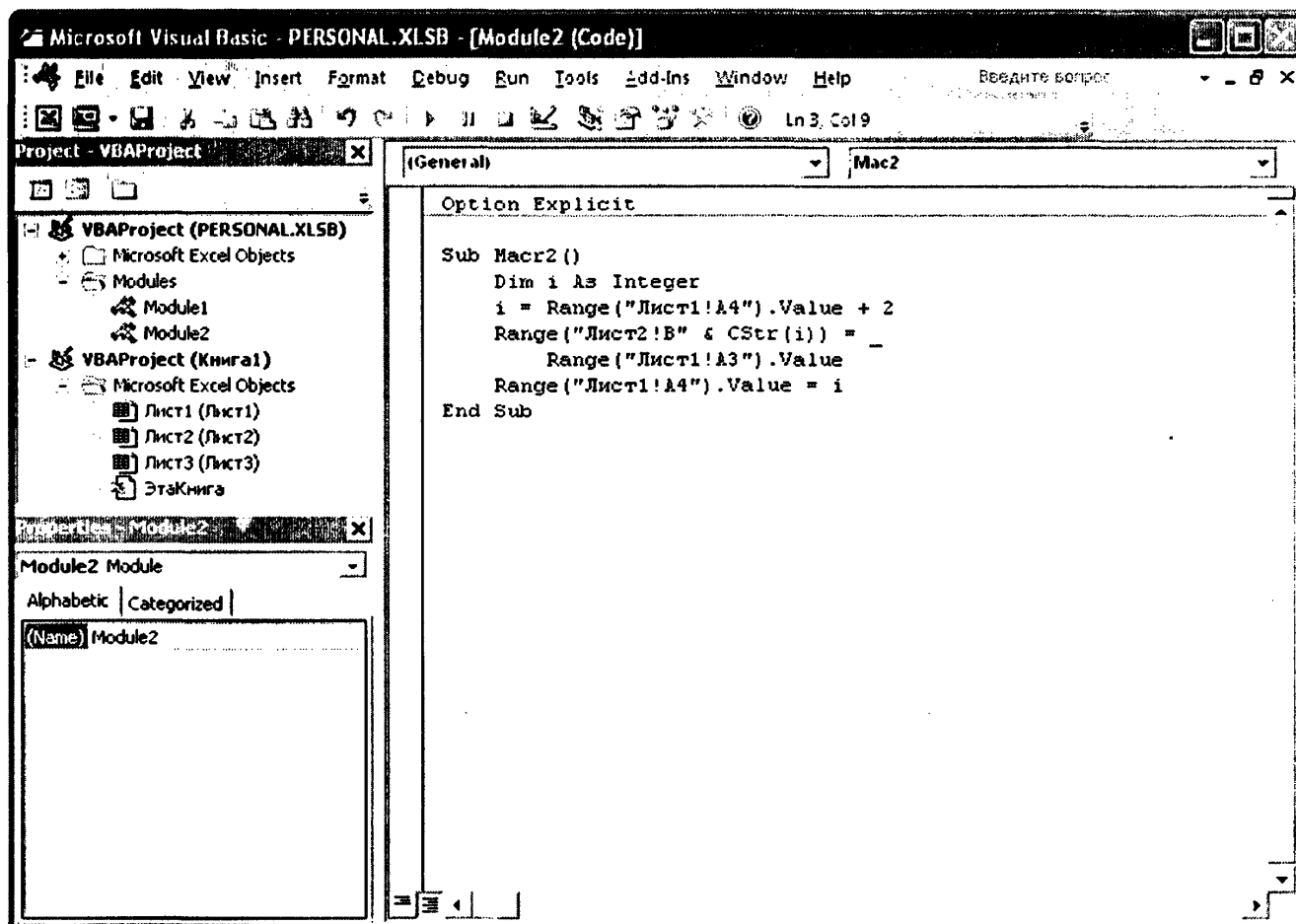


Рис. 38.3. Окно VB с текстом макроса Macr2 в окне кода модуля Module2

нахождения личной книги макросов в дисковой памяти компьютера.

При запуске программы Excel происходит автоматическое открытие книг, находящихся в папке XLSTART. Полное имя этой папки имеет следующий вид:

C:\Documents and Settings\usr\Application
Data\Microsoft\Excel\XLSTART

где usr — имя пользователя. Именно в этой папке находится файл PERSONAL.XLSB.

39. Еще две пользовательские функции Excel

Разработаем функцию, предназначенную для решения такой задачи.

Пусть заданы величина h и числовой массив M , элементы которого находятся в ячейках таблицы Excel.

Требуется определить, сколько элементов массива M больше h .

Функция должна возвращать количество элементов массива, превосходящих h .

Активизируем новую книгу, а затем откроем окно VB. Вставим модуль в личную книгу макросов. В окно кода введем следующий текст пользовательской функции:

```
Function Func8(M As Variant, h As Variant) As Integer
1:  Dim kol As Integer, i As Integer, j As Integer
2:  kol = 0
3:  If TypeName(M) = "Range" Then
4:      For i = 1 To M.Rows.Count
5:          For j = 1 To M.Columns.Count
6:              If M(i, j) > h Then
7:                  kol = kol + 1
8:              End If
9:          Next j
10:     Next i
11:     Func8 = kol
12: Else
13:     MsgBox "Func8: Аргумент — не диапазон"
14: End If
End Function
```

В третьей строке (с меткой 3) фигурирует функция `TypeName`, возвращающая (в виде строки) тип аргумента. С помощью этой функции проверяется тип переменной M .

При безошибочном использовании функции `Func8` переменная M имеет тип `Range` (Диапазон).

В четвертой и пятой строках значения `M.Rows.Count` и `M.Columns.Count` равны числу строк и столбцов в диапазоне `M`.

В шестой строке выражение `M(i, j)` означает ссылку на соответствующую ячейку диапазона `M`.

Пользовательская функция `Func8` используется следующим образом.

1. Вводим числа, например, в ячейки `D5:E8` листа `Лист4`.

2. Присваиваем этому диапазону ячеек, например, имя `Test8`; для этого:

- выделяем диапазон и кликаем на нем правой кнопкой мышки;
- в открывшемся контекстном меню выполняем команду **Имя диапазона**;
- в открывшемся окне *Создание имени* в текстовое поле *Имя* вводим `Test8`, в текстовое поле *Область* вводим `Лист4` с помощью скрытого списка (рис. 39.1) и кликаем на кнопке **ОК**.

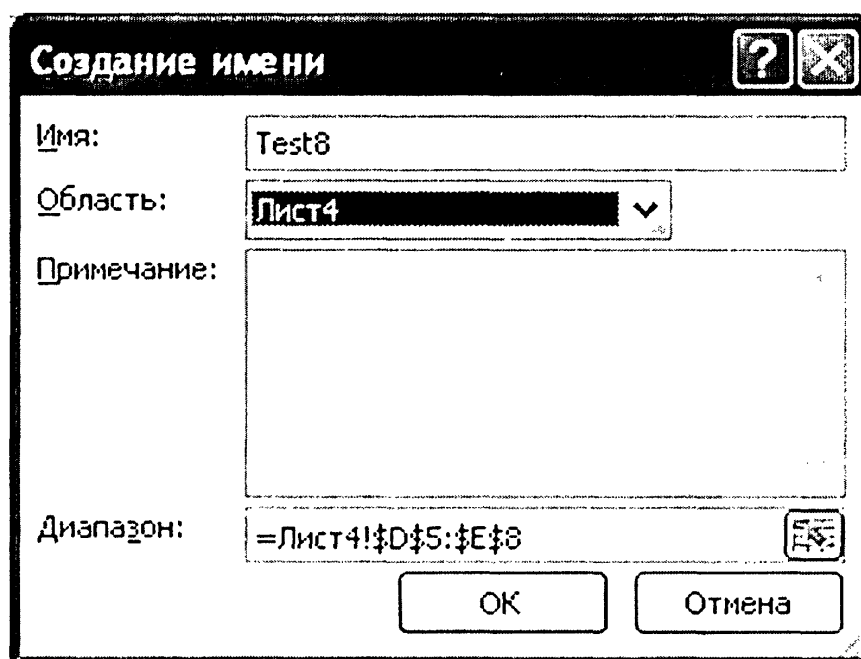


Рис. 39.1

3. Вводим, например, в ячейку `E11` формулу `=PERSONAL.XLSB!Func8(Test8;2)`

Мы уже отмечали, что в обращении к функции Excel аргументы отделяются друг от друга символом «точка с запятой», а не запятой, как это принято в Visual Basic.

4. Кликаем на галочке в строке формул.

39. Еще две пользовательские функция Excel

В ячейке E11 появится результат — количество чисел в ячейках D5:E8, превосходящих 2 (рис. 39.2).

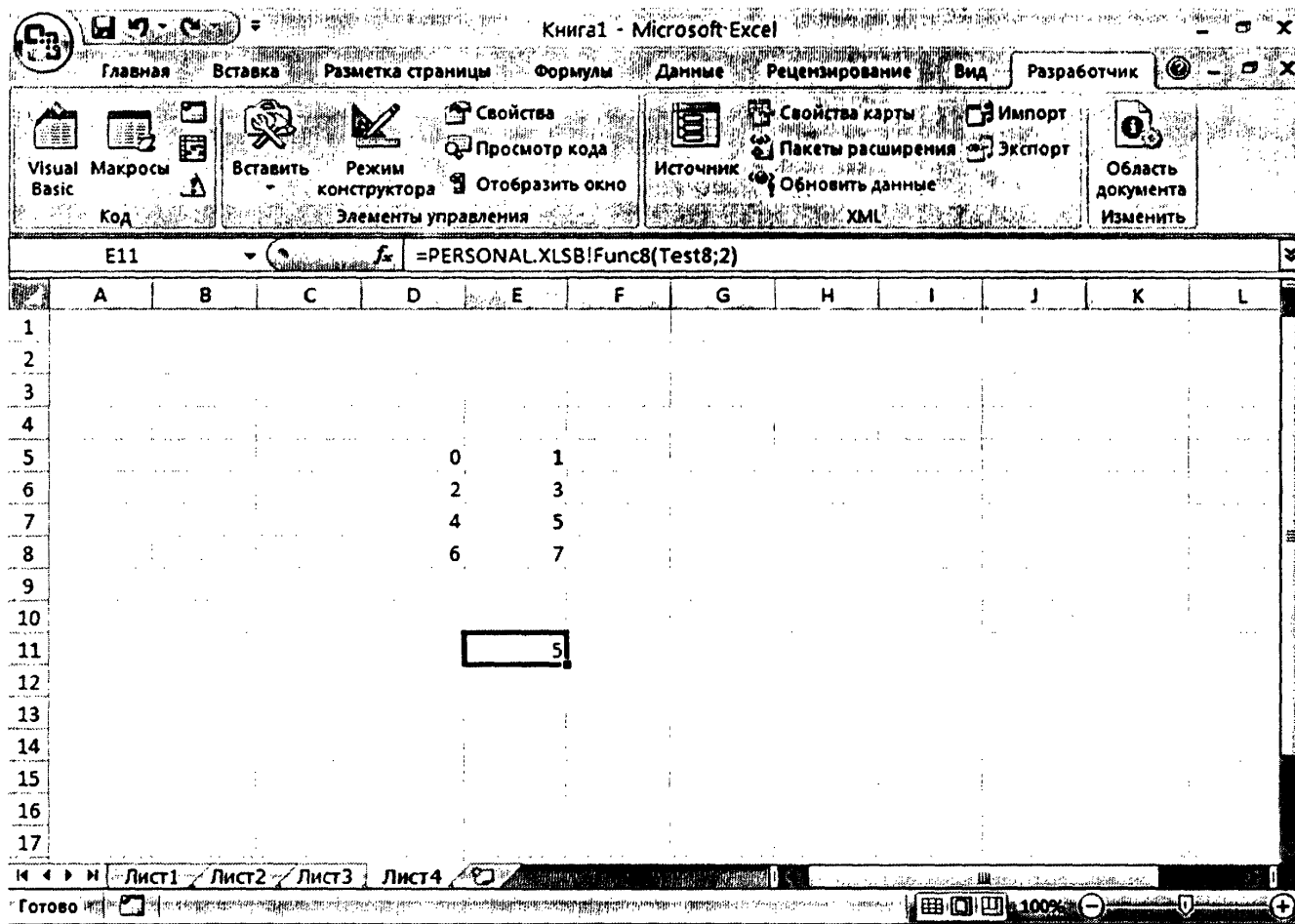


Рис. 39.2. Окно Excel с исходными данными и результатом расчета значения функции

На рис. 39.3 приведена модификация рассмотренной функции, не требующая присвоения диапазону имени.

В тексте функции `Func9` (рис. 39.3) вместо переменной `m` фигурирует переменная `massiv`. При этом функция `Func9` используется так:

- 1) выделяем ячейку для результата;
- 2) кликаем на `fx` в строке формул;
- 3) в открывшемся окне находим функцию `Func9` (рис. 39.4) и кликаем на кнопке **ОК**;

4) в соответствующие текстовые поля появившегося окна *Аргументы функции* (рис. 39.5) вводим исходный диапазон ячеек (например, D5:E8) и значение `h` (например, 2) и кликаем на кнопке **ОК**.

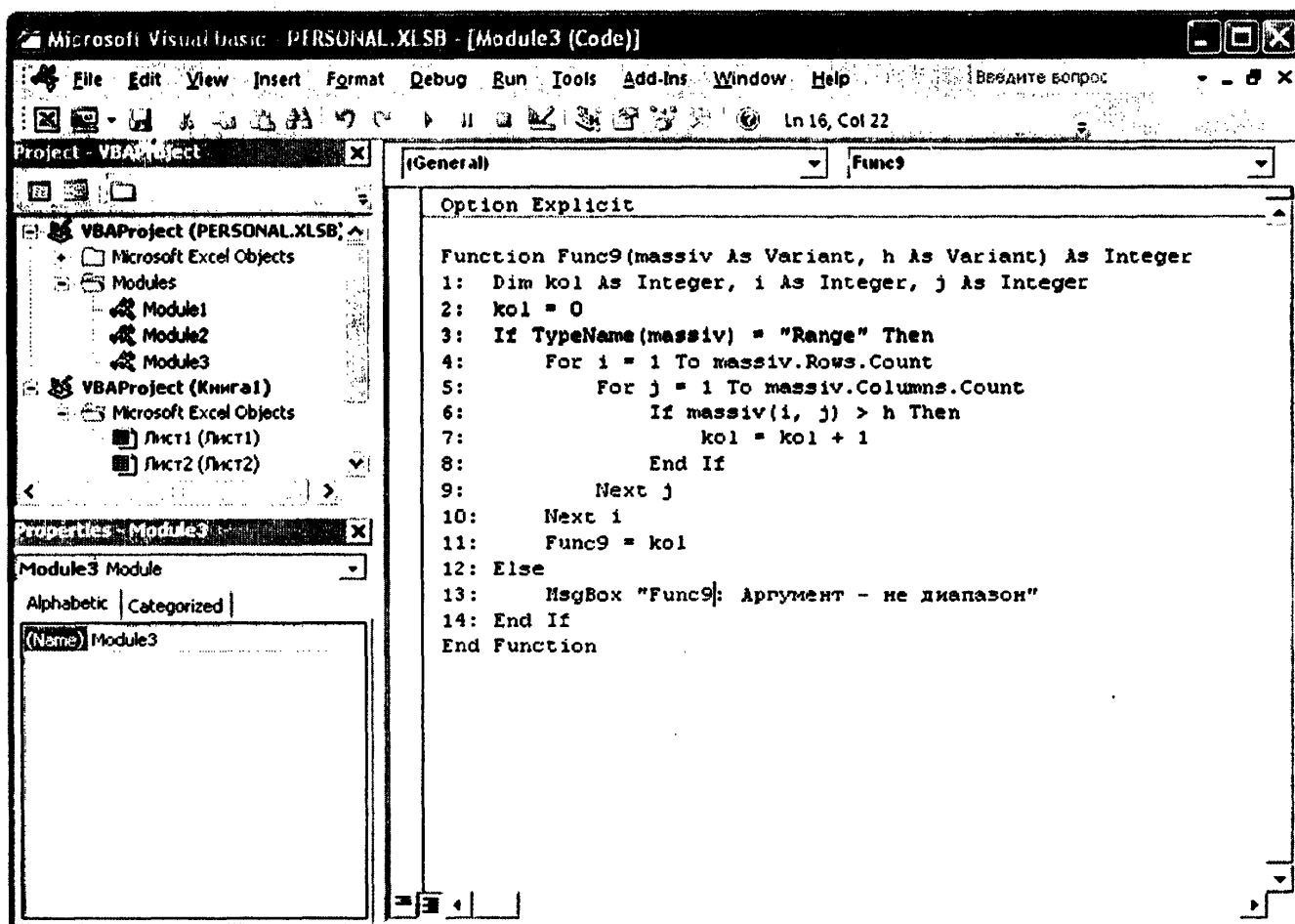


Рис. 39.3. Окно VB с текстом функции Func9 в окне кода

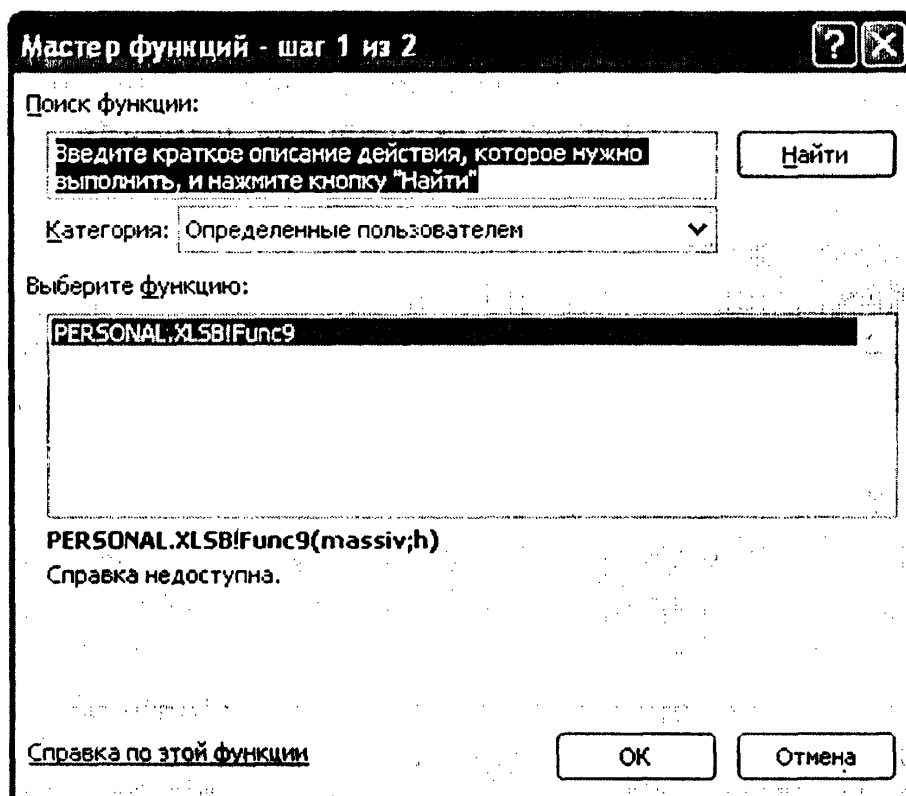


Рис. 39.4. Первое окно мастера функций Excel

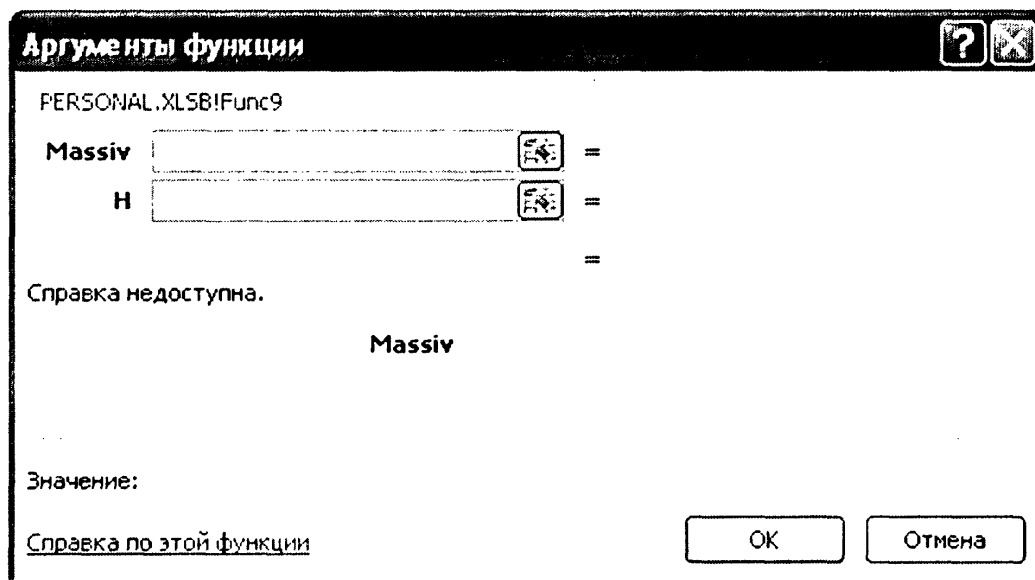


Рис. 39.5

В активной ячейке появится количество значений, превосходящих h .

Заключение

Данная книга дает лишь начальное представление о VBA. Чтобы стимулировать продолжение изучения этого алгоритмического языка, приведем цитату из книги [6].

«Примерно 90 % от общего числа пользователей работают только со средствами рабочего листа Excel, без применения языка программирования VBA, а значит, они используют лишь не более 10 % от реальных возможностей Excel. Таким образом, большая часть средств Excel остается невостребованной просто потому, что многие не знают о наличии VBA или не умеют им пользоваться».

Если при изучении VBA вы не все поймете, это не страшно. Дело в том, что сейчас программирование (в частности на VBA) достигло такого уровня сложности, что часто приходится работать «по образцу». Согласно этому методу, разработчик программы использует чужие программы или их части, не всегда до конца понимая, как они работают.

VBA является мощным средством увеличения возможностей табличного процессора Excel и всей системы Microsoft Office. Продолжить изучение этого алгоритмического языка можно по материалам справочных систем Microsoft Office. Рекомендуем также посетить сайт www.ozgrid.com авторов книги [11], а также использовать поисковую систему www.google.com для поиска готовых макросов.

Приложение 1

Типы данных Visual Basic

Тип данных	Размер ячейки (В — байт)	Значения переменной/константы или примечание
Boolean (логический)	2 В	True (истина, логическая единица) и False (ложь, логический нуль)
Byte (короткий целый беззнаковый)	1 В	Целые числа от 0 до 255
Integer (целый)	2 В	Целые числа от -32 768 до 32 767
Long (длинный целый)	4 В	Целые числа от -2 147 483 648 до 2 147 483 647
Currency (денежный)	8 В	Числа с четырьмя десятичными знаками от -922 337 203 685 477.5808 до 922 337 203 685 477.5807
Single (с плавающей точкой одинарной точ- ности)	4 В	Числа с дробной частью от $-3.402823 \cdot 10^{38}$ до $-1.401298 \cdot 10^{-45}$ для отрицательных чисел и от $1.401298 \cdot 10^{-45}$ до $3.402823 \cdot 10^{38}$ для положительных чисел
Double (с плавающей точкой двойной точно- сти)	8 В	Числа с дробной частью от $-1.79769313486231 \cdot 10^{308}$ до $-4.94065645841247 \cdot 10^{-324}$ для отрицательных чисел и от $4.94065645841247 \cdot 10^{-324}$ до $1.79769313486232 \cdot 10^{308}$ для положительных чисел
Date (дата)	8 В	От 1 января 100 года до 31 декабря 9999 года
String (строковый пе- ременной длины)	10 В + 1 В на символ	Длина строки от 0 до 2^{31} символов
String (строковый по- стоянной длины)	Задается при выполнении оператора Dim	Длина строки от 1 до 2^{16} символов

Тип данных	Размер ячейки (В — байт)	Значения переменной/константы или примечание
Variant (универсальный)	Для чисел 16 В	Значения соответствуют типу данных Boolean, Byte, Integer, Long, Currency, Single, Double или Date, определяемому автоматически
	Для строк 22 В + 1 В на символ	Длина строки от 0 до 2 ³¹ символов
Object (объект)	4 В	Используется при объявлении объектов; аналогичен типу Variant

Приложение 2

Греческий алфавит

Строчная буква	Заглавная буква	Русское название	Английское название
α	A	альфа	alpha
β	B	бета	beta
γ	Г	гамма	gamma
δ	Δ	дельта	delta
ε	E	эпсилон	epsilon
ζ	Z	дзета	zeta
η	H	эта	eta
θ	Θ	тета	theta
ι	I	йота	iota
κ	K	каппа	kappa
λ	Λ	ламбда	lambda
μ	M	мю	mu
ν	N	ню	nu
ξ	Ξ	кси	xi
$\omicron$	O	омикрон	omicron
π	P	пи	pi
ρ	P	ро	rho
σ	Σ	сигма	sigma
τ	T	тау	tau
υ	Y	ипсилон	upsilon
ϕ	Φ	фи	phi
χ	X	хи	chi
ψ	Ψ	пси	psi
ω	Ω	омега	omega

Приложение 3

Основные математические функции

Основные математические функции Visual Basic

Обращение к функции	Возвращаемое значение
Abs (x)	$ x $
Atn (x)	$\arctg x$
Cos (x)	$\cos x$
Exp (x)	e^x
Fix (x)	Результат отбрасывания дробной части x
Int (x)	Наибольшее целое число, не превосходящее x
Log (x)	Натуральный логарифм $\ln x$ при $x > 0$
Sgn (x)	1, 0 или -1 в зависимости от знака x
Sin (x)	$\sin x$
Sqr (x)	$\sqrt{x}$ при $x \geq 0$
Tan (x)	$\operatorname{tg} x$ при $x \neq \pm\pi/2$

Операторы пересчета

Ниже приведены операторы Visual Basic, позволяющие рассчитывать значения тригонометрической функции $\operatorname{ctg} x$, обратных тригонометрических функций $\arcsin x$, $\arccos x$ и $\operatorname{arcctg} x$ и десятичного логарифма $\lg x$.

```
ctg_x = Cos (x) / Sin (x)                                'если Sin(x) <> 0
```

```
Const pi As Double = 3.141592654
```

Приложение 3. Основные математические функции

$\arcsin_x = \text{Atn}(x / \text{Sqr}(1 - x^2))$	'если $\text{Abs}(x) <> 1$
$\arcsin_x = \text{Sgn}(x) * \pi / 2$	'если $\text{Abs}(x) = 1$
$\arccos_x = \text{Atn}(\text{Sqr}(1 - x^2) / x)$	'если $x <> 0$
$\arccos_x = \pi / 2$	'если $x = 0$
$\text{arcctg}_x = \text{Atn}(1 / x)$	'если $x <> 0$
$\text{arcctg}_x = \pi / 2$	'если $x = 0$
$\lg_x = \text{Log}(x) / 2.302585093$	'если $x > 0$

Приложение 4

Рекомендации по темам контрольных работ

В конце данного приложения приведена таблица, которая содержит функции $f(x)$ и соответствующие им отрезки $[a, b]$ из работы [7], обладающие следующими свойствами:

- функция $f(x)$ непрерывна и монотонна на отрезке $[a, b]$;
- значения функции на левой и правой границах отрезка $[a, b]$ имеют разные знаки.

На основе этой таблицы легко составить 31 и более вариантов заданий по программированию. Значения $f(a)$, приведенные в таблице, нужны для отладки программ.

Ниже приводятся примеры заданий.

1. Разработать программу, которая из четырех значений $y = f(a)$, $f(b)$, $f(0.5a + 0.5b)$, $0.5f(a) + 0.5f(b)$ выбирает такие значения, для которых выполняются следующие условия:

- 1) $\text{Fix}(y) = \text{CInt}(y)$ и $\text{Fix}(y) \neq \text{Int}(y)$;
- 2) $\text{Int}(y) = \text{CInt}(y)$ и $\text{Int}(y) \neq \text{Fix}(y)$;
- 3) $\text{Fix}(y) = \text{Int}(y) = \text{CInt}(y) = \text{Round}(y)$;
- 4) $\text{Fix}(y) = \text{Int}(y)$ или $\text{Fix}(y) \neq \text{CInt}(y)$;
- 5) $\text{Fix}(y) = \text{CInt}(y)$ или $\text{Fix}(y) \neq \text{Int}(y)$;
- 6) $\text{Int}(y) = \text{CInt}(y)$ или $\text{Int}(y) \neq \text{Fix}(y)$;
- 7) $\text{Fix}(y) = \text{Int}(y)$ или $\text{Fix}(y) = \text{CInt}(y)$;
- 8) $\text{Int}(y) = \text{CInt}(y)$ или $\text{Int}(y) = \text{Fix}(y)$;
- 9) $\text{CInt}(y) = \text{Fix}(y)$ или $\text{CInt}(y) = \text{Int}(y)$;
- 10) $\text{Fix}(y) = \text{Int}(y)$ и $\text{Fix}(y) \neq \text{CInt}(y)$.

В качестве пользовательского интерфейса программы использовать стандартные окна Windows.

2. Разработать программу создания текстового файла, состоящего не менее чем из 10 строк вида

$$f(x_i) = f_i$$

где x_i и f_i — числовые значения аргумента и функции;

i — номер числового значения, $0 \leq i \leq n$;

$x_0 = a$; $x_n = b$; $x_0 < x_1 < x_2 < \dots < x_{n-2} < x_{n-1} < x_n$.

3. Разработать программу расчета среднего значения функции $f(x)$ на отрезке $[a, b]$ по формуле

$$f_{cp} = (0.5f_0 + f_1 + f_2 + \dots + f_{n-2} + f_{n-1} + 0.5f_n)/n,$$

в которой фигурируют значения функции $f(x)$ в равностоящих точках отрезка $[a, b]$; значение n , превосходящее 100, задается. В качестве пользовательского интерфейса программы использовать стандартные окна Windows.

4. Разработать программу нахождения точки минимума функции $y(x) = f^2(x)$ на отрезке $[a, b]$ и расчета значения функции $y(x)$ в найденной точке. Точку минимума искать путем «шагания» по оси x с шагом h , не превосходящим 0.001. В качестве пользовательского интерфейса программы использовать форму, причем шаг h должен относиться к исходным данным программы; при $h > 0.001$ в форму должно выдаваться сообщение об ошибке.

5. Рассчитать значения функций $f(x)$, $g(x) = xf(x)$ и $h(x) = x^2 f(x)$ не менее чем в 10 точках отрезка $[a, b]$ (включая a и b) следующими методами:

- 1) разработав макрос с помощью макрорекордера;
- 2) разработав пользовательскую функцию Excel;
- 3) написав текст макроса на VBA.

Средствами Excel построить графики зависимостей $f(x)$, $g(x)$ и $h(x)$.

6. Разработать программу решения уравнения $f(x) = 0$ на отрезке $[a, b]$ методом деления отрезка пополам. Уравнение решать с заданной погрешностью h по x , не превосходящей 10^{-3} . В качестве пользовательского интерфейса программы использовать форму, причем погрешность h должна относиться к исходным данным программы; при $h > 10^{-3}$ должно выдаваться сообщение об ошибке.

Для читателя, не знакомого с методом деления отрезка пополам, приведем его описание.

Пусть fa и fb — значения функции f на соответствующих границах отрезка $[a, b]$; эти значения имеют разные знаки. Метод деления отрезка пополам работает следующим образом:

1) по формулам $c = (a + b)/2$ и $fc = f(c)$ рассчитываются координата середины отрезка $[a, b]$ и значение функции в середине отрезка;

2) если совпадают знаки fc и fa , то смещается левая граница отрезка $[a, b]$: полагается $a = c$ и $fa = fc$; далее выполняется четвертый пункт;

3) если совпадают знаки fc и fb , то смещается правая граница отрезка $[a, b]$: полагается $b = c$ и $fb = fc$;

4) повторяются три предыдущих пункта до тех пор, пока не выполнится условие $b - a < h$;

5) при выполнении последнего условия процесс решения уравнения $f(x) = 0$ оканчивается; конечное значение c считается решением этого уравнения.

Задания, приведенные в данном приложении, могут быть использованы преподавателями при подготовке тем контрольных работ. Эти задания можно также использовать для самопроверки при самостоятельном изучении материала книги.

Варианты заданий

№ варианта	Функция $f(x)$	Отрезок $[a, b]$	Значение $f(a)$
1	$(3.8 - 3 \sin \sqrt{x})/0.35 - x$	$[2; 3]$	0.3905776
2	$1/(3 + \sin(3.6x)) - x$	$[0; 0.85]$	0.3333333
3	$\cos \sqrt{1 - 0.3x^3} - x$	$[0; 1]$	0.5403023
4	$\sin \sqrt{1 - 0.4x^2} - x$	$[0; 1]$	0.841471
5	$0.25x^3 - x - 1.2502$	$[2; 3]$	-1.2502
6	$0.1x^2 - x \ln x$	$[1; 2]$	0.1
7	$3x - 4 \ln x - 5$	$[2; 4]^*$	-1.772589
8	$e^x - e^{-x} - 2$	$[0; 1]$	-2
9	$x + \sqrt{x} + \sqrt[3]{x} - 2.5$	$[0.4; 1]$	-0.7307382
10	$tg(x) - (tg^3 x)/3 + (tg^5 x)/5 - 0.3333333$	$[0; 0.8]$	-0.3333333
11	$\cos(2/x) - 2 \sin(1/x) + (1/x)$	$[1; 2]$	-1.099089
12	$\sin(\ln x) - \cos(\ln x) + 2 \ln x$	$[1; 3]$	-1

Приложение 4. Рекомендации по темам контрольных работ

№ варианта	Функция f(x)	Отрезок [a, b]	Значение f(a)
13	$\ln x - x + 1.8$	[2; 3]	0.4931472
14	$0.4 + \operatorname{arctg} \sqrt{x} - x$	[1; 2]	0.1853982
15	$x \cdot \operatorname{tg}(x) - 0.3333333$	[0.2; 1]	-0.2927913
16	$\operatorname{tg}(0.55x + 0.1) - x^2$	[0.4; 1]	0.171389
17	$2 - \sin(1/x) - x$	[1.2; 2]	0.0598231
18	$1 + \sin x - \ln(1 + x) - x$	[0; 1.5]	1
19	$-\cos(x^{0.52} + 2) - x$	[0.5; 1]	0.4029458
20	$\sqrt{\ln(1 + x) + 3} - x$	[2; 3]	0.024503
21	$e^x + \ln x - 10x$	[3; 4]	-8.815851
22	$3x - 14 + e^x - e^{-x}$	[1; 3]	-8.649598
23	$2 \ln^2 x + 6 \ln x - 5$	[1; 3]	-5
24	$2x \sin x - \cos x$	[0.4; 1]	-0.6095263
25	$\cos x - \exp(-x^2/2) + x - 1$	[1; 2]	-0.0662284
26	$\sqrt{1 - x} - \operatorname{tg}(x)$	[0; 0.9]	1
27	$\sin(x^2) + \cos(x^2) - 10x$	[0; 1]	1
28	$e^x + \sqrt{1 + e^{2x}} - 2$	[-1; 0]	-0.5665994
29	$\sqrt{1 - x} - \cos \sqrt{1 - x}$	[0; 0.9]	0.4596977
30	$\operatorname{tg}(x/2) - \operatorname{ctg}(x/2) + x$	[1; 2]	-0.2841852
31	$x - \cos x$	[0.5; 2.5]	-0.3775826

Библиографический список

1. Справочные системы Excel и Visual Basic.
2. Брудно А. Л. Алгол. — М.: Издательство «Наука», 1971. — 80 с.
3. Колесников А. Excel 2000 (русифицированная версия). — К.: Издательская группа BNV, 1999. — 496 с.
4. Глушаков С. В., Мельников В. В., Сурядный А. С. Программирование в среде Windows. — Харьков: Фолио; Ростов н/Д: Феникс; Киев: Абрис, 2000. — 487 с.
5. Подлин Ш. Освой самостоятельно программирование для Microsoft Excel 2000 за 24 часа. — М.: Издательский дом «Вильямс», 2000. — 304 с.
6. Гарнаев А. Ю. Microsoft Excel 2000: разработка приложений. — СПб.: «БХВ — Санкт-Петербург», 2000. — 576 с.
7. Глаголев В. Б. Visual Basic 6.0. Сборник заданий для лабораторно-практических занятий. — М.: Издательство МЭИ, 2000. — 116 с.
8. Волчёнков Н. Г. Программирование на Visual Basic 6: В 3-х ч. Часть 3. — М.: ИНФРА-М, 2000. — 238 с.
9. Глаголев В. Б. Excel 2000. Сборник заданий для лабораторных занятий. — М.: Издательство МЭИ, 2001. — 64 с.
10. Гарбер Г. З. Основы программирования на Visual Basic в Microsoft Excel. — М.: Торговый дом СПАРРК, 2004. — 224 с.
11. Холи Р., Холи Д. Excel. Трюки. — СПб.: Питер, 2006. — 287 с.

Предметный указатель

A

Abs 176
Activate 144, 145, 149
ActiveCell 131, 138, 139, 141, 154, 155
ActiveSheet 138, 139, 150, 151
ActiveWorkbook 138, 139, 145
Add 145, 147, 148, 150, 151
Add Watch 15, 64, 68
Address 152, 155
And 41 — 43, 46
Append 90, 91, 94, 95
Application 131, 135, 137 — 143, 147
As 19, 20, 25, 61, 69, 70, 72, 74, 80 — 84, 90, 92, 94, 147, 151, 153, 162
Atn 38, 176, 177
Auto_Open 142, 143
Average 137

B

Balloon 139
Boolean 26, 36, 41, 173
ByRef 71, 72
Byte 20, 21, 36, 62, 64, 173
ByVal 71, 72, 74, 75

C

Calculate 141
Calculation 139, 141
Call 71, 72, 75, 83

Caption 103, 105 — 108, 111, 113

Case 49, 50

Case Else 50

CBool 36

CByte 36

CCur 36

CDate 36

CDbl 36

Cells 131, 149, 151, 153, 154, 160

CheckBox 112, 113, 133

CInt 36 — 38, 135

CLng 36

Close 91, 92, 144, 147

Columns 124, 153, 154, 160, 167, 168

CommandButton 103, 106, 133

Const 25, 26, 52

Cos 39, 176

Count 124, 150, 154, 160, 167, 168

CSng 36

CStr 36, 37, 86, 135, 137, 165

Currency 21, 23, 25, 36, 59, 173

CVar 36

D

Date 22, 25, 36, 89, 151, 173

Delete 149

Dialogs 139 — 141

Dim 19 — 22, 61, 64, 81, 147,
151, 153, 162
DisplayGridlines 131
Do 54 — 57
Double 21, 25, 33, 36, 72, 173

E

Else 48, 49, 167
ElseIf 48, 49
End 159, 160
End Function 69, 70, 74, 122,
125, 167
End If 47 — 49, 151, 167
End Sub 12, 13, 71, 75, 107,
131
End With 81 — 83, 138, 140,
141, 151
EOF 92, 93
Erase 67
exe 94, 95, 116
Exit Do 57
Exit For 53
Exp 176

F

False 26, 41, 42, 173
FileFormat 144
FileName 144
Fix 176
Font 105, 138, 151
For 51, 52, 90, 92, 94,
98, 99
For Each 153
Format 151
Formula 131, 152, 155
FormulaR1C1 152
FreeFile 90, 92, 98, 99
FullName 144, 145
Function 69, 70, 74, 122, 124,
167

G

GoTo 45, 47

H

Height 103

I

If 46 — 48, 151, 167
IIf 49
In 153
Input 90 — 93, 98
InputBox 76, 78, 88, 131
Int 176
Integer 20, 21, 36, 173
IsMissing 74
Italic 138, 151

K

Kill 94

L

Label 103, 104, 133
LBound 67
LCase 86
Left 87, 88
Len 87, 99
Line Input 91, 92, 98
LOF 92 — 94
Log 38, 176, 177
Long 21, 22, 36, 62, 67, 173
Loop 54 — 57
LTrim 85

M

massiv 124, 125, 169
Mid 87, 88, 98, 99
MkDir 92, 93
Mod 33, 35, 36, 49, 50
MsgBox 76 — 79, 88, 150, 155

N

Name 105, 106, 144, 145,
149 — 151

Next 51, 52, 153
 Not 41 — 43, 74
 Now 22, 39, 89, 135, 143
 NumberFormat 131

O

Object 148, 174
 Offset 152, 155
 OnTime 142, 143
 Open 90, 92, 94, 98, 99
 Option 21, 62
 Optional 74, 75
 Or 41 — 43, 99
 Output 90 — 92, 99

P

Password 144
 PERSONAL.XLSB 163, 165,
 166, 168
 Preserve 66, 67
 Print 91, 92, 94, 99
 Private 106, 107

Q

QBColor 151
 Quit 141, 147

R

Randomize 68
 Range 125, 131, 135, 152, 153,
 155, 156
 ReDim 64, 66 — 68, 82, 98,
 124
 ReferenceStyle 131, 137
 Replace 86, 87
 Right 87, 88
 Rmdir 94
 Rnd 40, 68
 Round 38, 39, 53
 Rows 124, 153, 154, 160, 167,
 168
 RTrim 85

S

Save 144, 145
 SaveAs 144, 147
 Select 131, 135, 153, 155, 157
 Select Case 49, 50
 Selection 131, 153, 154, 160
 Set 147, 148, 151, 162
 SetFocus 118
 Sgn 38, 176, 177
 Shell 94, 95
 Show 117, 118, 120, 139 — 141
 Sin 73, 176
 Single 21, 35 — 37, 58, 59, 173
 Space 86
 Sqr 13, 39, 83, 159, 176, 177
 Step 51, 52
 Str 23, 24, 37, 77, 79, 88, 150
 String 23, 36, 84 — 88, 173
 Sub 12, 13, 71, 75, 107, 131
 Sum 137

T

Tan 176
 Text 107, 109, 111
 TextBox 103, 106, 108, 111,
 133
 Then 46 — 49, 151, 167
 Time 22, 39, 89, 143
 TimeValue 89
 To 51, 52, 61, 63
 Trim 85
 True 41, 42, 173
 Type 80 — 82
 TypeName 167

U

UBound 67
 UCase 86
 Until 55—57, 64, 92
 UserForm 133

- Катет 159, 160
- ТРАНСПН 98 — 100, 160, 161
- Массив (Array) 61
 - динамический (dynamic) 64 — 68
 - статический (static) 61 — 63
- Метка 45, 47
- Метод (Method) 133 — 135
 - Activate 144, 145, 149
 - Add 145, 147, 148, 150, 151
 - Calculate 141
 - Close 144, 147
 - Delete 149
 - OnTime 142, 143
 - Quit 141, 147
 - Save 144, 145
 - SaveAs 144, 147
 - Select 131, 135, 153, 155, 157
 - SetFocus 118
 - Show 117, 118, 120, 139 — 141
- О**
- Объект (Object) 133
 - ActiveCell 131, 138, 139, 141, 154, 155
 - ActiveSheet 138, 139, 150, 151
 - ActiveWorkbook 138, 139, 145
 - Application 131, 135, 137 — 143, 147
 - Balloon 139
 - Cells 131, 149, 151, 153, 154, 160
 - CheckBox 112, 113, 133
 - Columns 124, 153, 154, 160, 167, 168
 - CommandButton 103, 106, 133
 - Dialogs 139 — 141
 - Label 103, 104, 133
 - Object 148, 174
 - Range 125, 131, 135, 152, 153, 155, 156
 - Rows 124, 153, 154, 160, 167, 168
 - Selection 131, 153, 154, 160
 - TextBox 103, 106, 108, 111, 133
 - UserForm 133
 - Workbook 144, 145, 147, 148
 - Workbooks 135, 144, 145, 147, 148
 - Worksheet 149 — 151
 - Worksheets 135, 137, 141, 147, 149 — 153, 155
- Оператор (Statement) 11, 17
 - Call 71, 72, 75, 83
 - Close 91, 92
 - Const 25, 26, 52
 - Dim 19 — 22, 61, 64, 81, 147, 151, 153, 162
 - Do 54 — 57
 - End 159, 160
 - Erase 67
 - Exit Do 57
 - Exit For 53
 - For 51, 52, 90, 92, 94, 98, 99
 - For Each 153
 - GoTo 45, 47
 - If...Then 46 — 49, 151, 167
 - Kill 94
 - Line Input 91, 92, 98
 - MkDir 92, 93
 - Open 90, 92, 94, 98, 99
 - Print 91, 92, 94, 99

Randomize 68
ReDim 64, 66 — 68, 82, 98, 124
Rmdir 94
Select Case 49, 50
Set 147, 148, 151, 162
Type 80 — 82
With 81, 82, 138, 140, 141, 151
выбора 49, 50
перехода 45, 47
присваивания 31, 32, 35, 36, 41, 69, 82, 134
условного перехода 47
условный 46 — 49, 151, 167
цикла
 Do...Loop Until 57
 Do...Loop While 56
 Do Until...Loop 55, 56
 Do While...Loop 54, 55
 For Each...Next 153
 For...Next 51 — 54
 While...Wend 54

Операция (Operator)

арифметическая 32 — 36
 возведение в степень (^) 13, 33
 вычитание (—) 33, 34
 деление (/) 25, 33
 изменение знака (—) 33, 34
 определение остатка от деления (Mod) 33, 35, 36, 49, 50
 сложение (+) 13, 33, 34
 умножение (*) 26, 33
 целочисленное деление (\) 33, 35, 36

логическая 41 — 43
 отрицание (Not) 41 — 43, 74
 сложение (Or) 41 — 43, 99
 умножение (And) 41 — 43, 46

П

Параметр пользовательской процедуры
 необязательный 73 — 75
 фактический 69 — 71, 75
 формальный 69 — 72
Переменная (Variable) 19, 20
 глобальная 73
 локальная 73
Подпрограмма (Subprogram)
 встроенная MsgBox 76 — 79, 88, 150, 155
 пользовательская 71 — 73
Поле записи 80 — 83, 107
Префикс 26
Процедура (Procedure)
 встроенная 76
 пользовательская 76

С

Свойство (Property) 133, 134
 ActiveCell 131, 138, 139, 141, 154, 155
 ActiveSheet 138, 139, 150, 151
 ActiveWorkbook 138, 139, 145
 Address 152, 155
 Average 137
 Calculation 139, 141
 Caption 103, 105 — 108, 111, 113
 Cells 131, 149, 151, 153, 154, 160

- Columns 124, 153, 154, 160, 167, 168
- Count 124, 150, 154, 160, 167, 168
- Dialogs 139 — 141
- DisplayGridlines 131
- Font 105, 138, 151
- Formula 131, 152, 155
- FormulaR1C1 152
- FullName 144, 145
- Height 103
- Italic 138, 151
- Name 105, 106, 144, 145, 149 — 151
- NumberFormat 131
- Offset 152, 155
- Range 125, 131, 135, 152, 153, 155, 156
- ReferenceStyle 131, 137
- Rows 124, 153, 154, 160, 167, 168
- Selection 131, 153, 154, 160
- Sum 137
- Text 107, 109, 111
- Value 108, 112, 113, 138, 141, 147, 151, 153 — 155, 159, 162, 165
- Width 103
- Workbooks 135, 144, 145, 147, 148
- Worksheets 135, 137, 141, 147, 149 — 153, 155
- Семейство (Collection) 139
 - Cells 131, 149, 151, 153, 154, 160
 - Columns 124, 153, 154, 160, 167, 168
 - Dialogs 139 — 141
 - Rows 124, 153, 154, 160, 167, 168
 - Workbooks 135, 144, 145, 147, 148
 - Worksheets 135, 137, 141, 147, 149 — 153, 155
- Событие (Event) 133, 142
- Строка (String) 23, 84 — 89, 173
 - переменной длины 84, 173
 - постоянной длины 84, 173
- Т**
- Тип данных (Data type) 19, 173, 174
 - Boolean 26, 36, 41, 173
 - Byte 20, 21, 36, 62, 64, 173
 - Currency 21, 23, 25, 36, 59, 173
 - Date 22, 25, 36, 89, 173
 - Double 21, 25, 33, 36, 72, 173
 - Integer 20, 21, 36, 173
 - Long 21, 22, 36, 62, 67, 173
 - Object 148, 174
 - Single 21, 35 — 37, 58, 59, 173
 - String 23, 36, 84 — 88, 173
 - Variant 21, 36, 74, 124, 167, 174
- Точка останова 15 — 17
- Ф**
- Функция
 - встроенная (Function) 76
 - Abs 176
 - Atn 38, 176, 177
 - CBool 36
 - CByte 36
 - CCur 36
 - CDate 36
 - CDbl 36
 - CInt 36 — 38, 135
 - CLng 36

Cos 39, 176
CSng 36
CStr 36, 37, 86, 135, 137, 165
CVar 36
Date 151
EOF 92, 93
Exp 176
Fix 176
Format 151
FreeFile 90, 92, 98, 99
IIf 49
Input 91, 93
InputBox 76, 78, 88, 131
Int 176
IsMissing 74
LBound 67
LCase 86
Left 87, 88
Len 87, 99
LOF 92 — 94
Log 38, 176, 177
LTrim 85
Mid 87, 88, 98, 99
MsgBox 78, 79
Now 22, 39, 89, 135, 143
QBColor 151
Replace 86, 87
Right 87, 88
Rnd 40, 68
Round 38, 39, 53
RTrim 85
Sgn 38, 176, 177
Shell 94, 95

Sin 73, 176
Space 86
Sqr 13, 39, 83, 159, 176, 177
Str 23, 24, 37, 77, 79, 88, 150
Tan 176
Time 22, 39, 89, 143
TimeValue 89
Trim 85
TypeName 167
UBound 67
UCase 86
Val 23, 24, 28, 29, 98, 99
встроенная Excel
СРЕДНЕЕ 137
СУММА 137
ТРАНСП 123
пользовательская 69, 70
пользовательская Excel
Func8 167, 168
Func9 169
Гипотенуза 122, 123
Транспн 124 — 126

Э

Элемент управления

Кнопка (CommandButton) 103, 106, 119, 133
Надпись (Label) 103, 104, 133
Поле (TextBox) 103, 106, 108, 111, 133
Флажок (CheckBox) 112, 113, 133

Серия «Библиотека студента»

Геннадий Зеликович Гарбер

Основы программирования на Visual Basic и VBA в Excel 2007

Ответственный за выпуск

В. Митин

Макет и верстка

Н. Бармина

Обложка

Е. Холмский

ООО «СОЛОН—ПРЕСС»

123242, г. Москва, а/я 20

Телефоны:

(495) 254-44-10, (495) 252-36-96, (495) 252-25-21

Сайт издательства «СОЛОН-Пресс»: www.solon-press.ru

E-mail: solon-avtor@coba.ru

По вопросам приобретения обращаться:

ООО «АЛЪЯНС-КНИГА КТК»

Тел: (495) 258-91-94, 258-91-95, www.abook.ru

ООО «СОЛОН—ПРЕСС»

103050, г. Москва, Дегтярный пер., д. 5, стр. 2

Формат 60×88/16. Объем 12 п. л. Тираж **1000** экз.

Отпечатано в ООО «Арт-Диал»

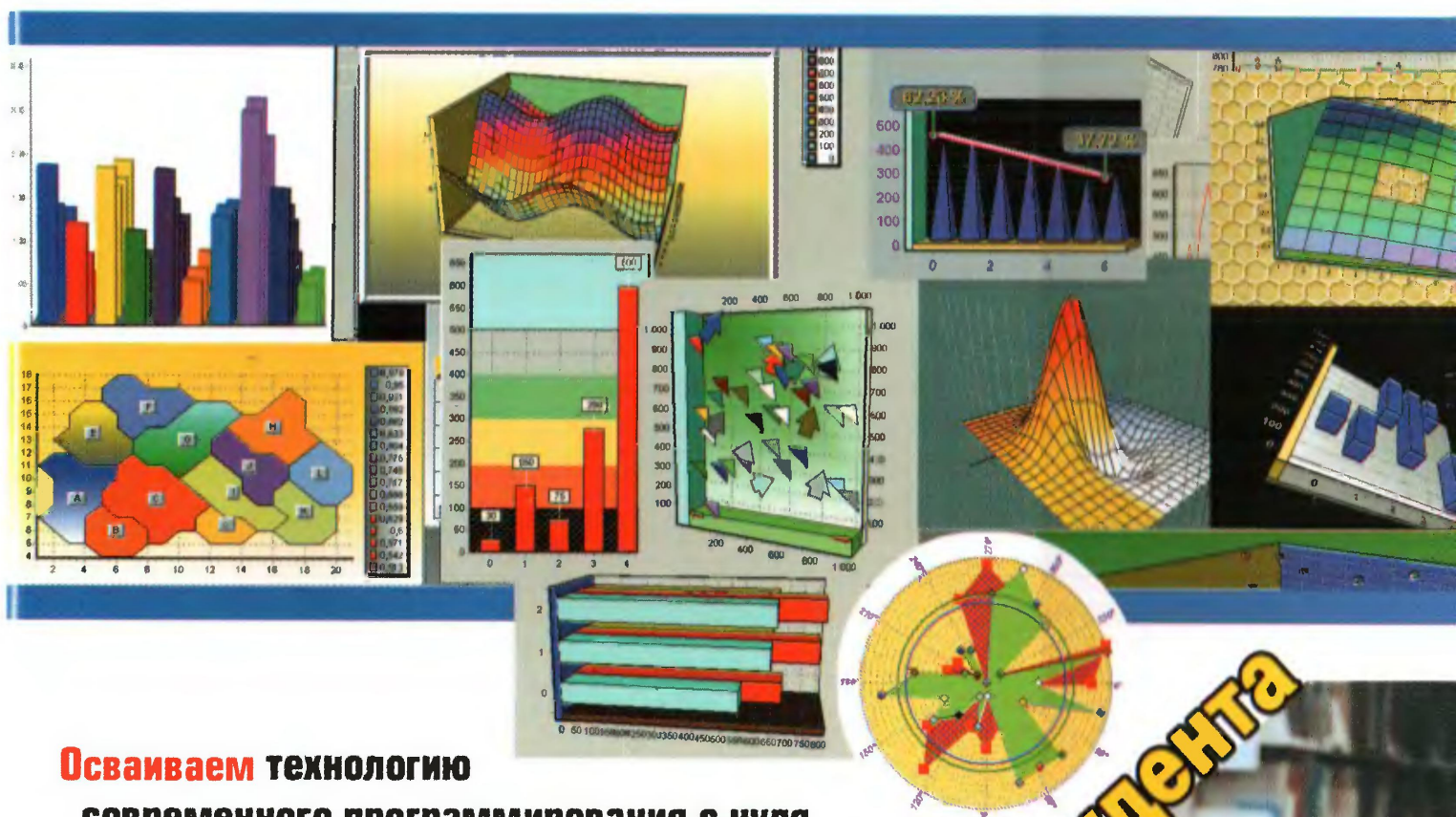
143983, МО, г. Железнодорожный, ул. Керамическая., д. 3

Заказ № 129

Г. З. Гарбер



ОСНОВЫ ПРОГРАММИРОВАНИЯ НА **Visual Basic** и **VBA** В **Excel 2007**



Осваиваем технологию

современного программирования с нуля

Движемся от Visual Basic к VBA - от простого
к сложному

Расширяем безгранично возможности Excel

Разрабатываем макросы - это актуально!

Оценка «Отлично»!

Библиотека студента

