

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНСТИТУТ ЭКОНОМИКИ И МЕНЕДЖМЕНТА

ОСНОВЫ ПРОГРАММИРОВАНИЯ в Microsoft Excel

Учебно-методическое пособие
по курсу «Информатика»
для студентов института экономики и менеджмента
направлений подготовки
38.03.01 – Экономика и 38.03.02 – Менеджмент

Томск
2017

РАССМОТРЕНО И УТВЕРЖДЕНО методической комиссией института экономики и менеджмента.

Протокол № 5 от 17 февраля 2017 г.

Председатель МК ИЭМ В.В. Маковеева

Пособие составлено в соответствии с тематикой практических занятий и программой курса «Информатика» для студентов института экономики и менеджмента направлений подготовки 38.03.01 – Экономика и 38.03.02 – Менеджмент.

В данном пособии изучаются основные принципы программирования в языке Visual Basic for Application (VBA) для автоматизации проведения расчетов в табличном процессоре Microsoft Excel. Подробно изучаются основные алгоритмические конструкции, понятия константы, переменной и массива. Рассматриваются основные операторы и конструкции VBA.

Предназначено для преподавателей, аспирантов, студентов и магистрантов дневной, очно-заочной и заочной формы обучения.

СОСТАВИТЕЛЬ: Лещинский Борис Семенович, к.т.н., доцент кафедры информационных технологий и бизнес-аналитики ИЭМ НИ ТГУ.

ПРЕДИСЛОВИЕ

Функционирование компьютера, любых его компонент осуществляется только по определенным алгоритмам. *Алгоритм* – это *точно определенный* способ решения задачи в виде *конечной* (по времени) последовательности действий (операций). Для описания алгоритмов используют различные способы (графические, текстовые, формульные).

В силу технических особенностей компьютера алгоритм для него должен быть представлен вполне определенным образом – в виде *программы*, которая представляет собой последовательность команд для процессора. Каждая команда – это описание какой-либо инструкции, которую должен исполнить процессор. Эти команды, должны быть выражены *в машинном коде*, т.е. на языке, понятном процессору. Программист, конечно, может, затрачивая значительные усилия, понимать и составлять программы на машинном языке, но это очень сложно. Поэтому для их создания принят следующий подход: программист пишет текст алгоритма на языке, который ему понятен и удобен, а затем с помощью специальных программ (систем программирования) переводит его на машинный язык и превращает в удобный для процессора вид. Разговорный язык для этой цели не подходит в связи с нечеткостью и многозначностью, поэтому разрабатывают специальные языки, которые называют *алгоритмическими* или *языками программирования*.

Итак, *языки программирования* – это языки описания алгоритмов, которые нужны человеку, а не компьютеру. Они необходимы для облегчения создания программ, которые, в конечном итоге, должны быть представлены на одном и том же языке – языке команд для процессора. Поскольку человек имеет дело с текстом, написанном на языке программирования, то этот текст также называют *программой*. В настоящее время существует много алгоритмических языков, ориентированных на описание различных особенностей решаемых задач.

В целом, процесс разработки программы включает в себя три этапа:

- представление алгоритма в виде, удобном для уяснения логики решения задачи, для чего используют графические, текстовые, формульные средства;
- описание алгоритма средствами определенного алгоритмического языка;

– преобразование текста в готовую программу, которая будет исполняться процессором.

Первый этап выполняется вручную, а для других двух этапов используют специальные программные средства, которые называются **системами программирования**.

Основу Visual Basic for Application (VBA) составляет язык программирования BASIC, разработанный в начале 1960-х годов. Название *BASIC* образовано из начальных букв английской фразы *Beginner's All-purpose Symbolic Instruction Code* (универсальный язык символических инструкций для начинающих). Это один из наиболее простых языков программирования, предназначенный для решения вычислительных задач. Несмотря на свою простоту, он позволяет решать довольно большой круг задач. На его примере можно достаточно быстро понять особенности и других языков программирования.

В настоящее время существует много разных вариантов, сохраняющих ядро BASIC и включающих различные дополнительные возможности. Одним из наиболее развитых расширений этого языка является Visual Basic for Application, который появился в 1991 году как средство автоматизации расчетов в Microsoft Excel.

1 ОСНОВЫ АЛГОРИТМИЗАЦИИ

1.1 Основные понятия

Задача, как правило, формулируется на обычном языке с применением различных конструкций, принятых в той области знаний, к которой относится данная задача. Например, мы можем использовать запись суммы n чисел в виде

$$S = \sum_{i=1}^n r_i$$

или написать $0 < x < 10$.

Эти конструкции есть в алгебраическом языке и используются для краткости записи. Если мы знаем этот язык, то понимаем, что первая конструкция означает «для вычисления S следует сложить n чисел r_1, r_2, \dots, r_n », а вторая конструкция – это краткая запись двух условий $0 < x$ и $x < 10$, которые должны выполняться одновременно.

Однако в языке программирования таких языковых конструкций может и не быть. Любая программа представляет собой описание последовательности *элементарных* действий (операций), которые надо выполнить, чтобы решить задачу. Поэтому, чтобы изложить эту последовательность на языке программирования, необходимо сначала ясно понимать, как представить языковые конструкции обычного языка в виде таких элементарных операций. В связи с этим, прежде чем писать текст программы, следует написать алгоритм.

Алгоритм – это *конечный* набор правил, позволяющий решать *любую* конкретную задачу из некоторого класса однотипных, при условии, что исходные данные для решения могут изменяться в заданных пределах. В алгоритме подробно описывается последовательность действий, необходимая для решения задачи. Составление такого пошагового описания процесса решения задачи называется **алгоритмизацией**. Существуют различные способы описания алгоритма, основными из которых являются словесно-формульный (пошаговый) и структурный (в виде блок-схемы). **Словесно-формульным** называется способ записи алгоритма на естественном языке, алгоритмическом (языке программирования) или псевдоязыке (сочетание элементов естественного и алгоритмического языка). **Структурный** способ – это компактная и наглядная форма записи в виде специальных графических знаков с указанием направленных

связей между ними. Полученное таким способом описание называют блок-схемой (или структурной схемой).

Блок-схема (структурная схема) – это удобное для человека графическое изображение алгоритма в виде плоских геометрических фигур (их называют **блоками** или **вершинами**), соединенных направленными линиями (их называют **дугами**). Внутри каждого блока (вершины) записывается действие, которое следует выполнить, или условие, которое необходимо проверить. Направление дуг показывает последовательность выполнения действий.

В блок-схеме присутствуют вершины разного типа, основными из которых являются:

– **вершины начала и окончания** (изображаются овалами). У вершины-начала нет входящих дуг, у нее есть лишь *одна* исходящая, направленная к вершине, с которой начинается алгоритм (рис. 1).

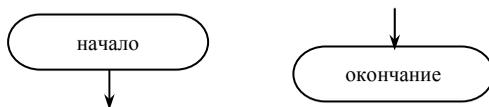


Рис. 1. Вершины начала и окончания алгоритма.

У вершины-окончания нет исходящих дуг и может быть несколько входящих;

– **вершины-действия** (изображаются прямоугольниками) соответствуют шагам, в которых выполняются действия по изменению значений, форм представления или расположения данных (рис 2).



Рис. 2. Вершина, изображающая действие.

Каждая такая вершина может иметь несколько входящих дуг и только одну исходящую;

– **вершины-условия** (изображаются ромбами) соответствуют шагам, в которых проверяются условия. Такая вершина может иметь несколько входящих дуг и не менее двух исходящих, каждая из которых соответствует одному из результатов проверяемого условия и отмечается соответствующей меткой (рис. 3).

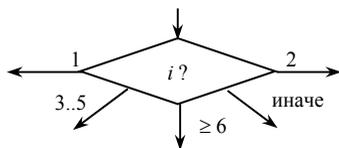


Рис. 3. Вершина, изображающая проверку условия.

Если условием является выражение, предполагающее альтернативный выбор (*логическое выражение*), то одна из исходящих дуг соответствует отношению следования при условии, что проверяемое логическое выражение истинно. Такая дуга отмечается меткой «да». Другая исходящая дуга соответствует отношению следования при условии, что проверяемое логическое выражение ложно, и отмечается меткой «нет» (рис.4);

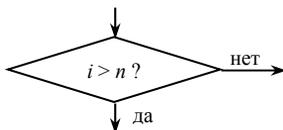


Рис. 4. Вершина альтернативного выбора.

– *вершины-узлы*, указывающие на объединение (*слияние*) нескольких входящих дуг. Они обозначаются точками или кругами небольшого размера (рис. 5).

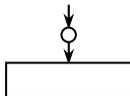


Рис. 5. Объединение нескольких дуг.

Иногда вершины-действия делят на *операционные (вычислительные)* и *вершины ввода/вывода*. Тогда первые изображают прямоугольниками, а последние – параллелограммами (рис. 6).

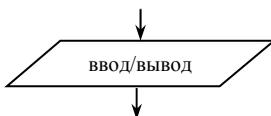


Рис. 6. Вершина ввода или вывода.

1.2 Основные алгоритмические структуры

Любой алгоритм представляет собой комбинацию трех элементарных алгоритмических структур: линейная, ветвящаяся, циклическая.

Линейная структура описывает процесс, в котором операции выполняются последовательно в порядке их описания. Вершины, отображающие эти действия, располагаются в линейной последовательности. Такие процессы имеют место, например, при вычислении арифметических выражений, когда имеются конкретные числовые данные и с ними выполняются соответствующие условию задачи действия. Например, вычисление

$$y = \frac{b^2 - ac}{a + c}$$

можно представить линейной структурой, изображенной на рис.7.

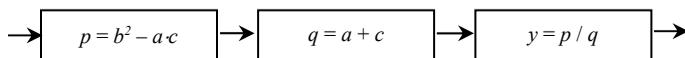


Рис.7. Линейная алгоритмическая структура.

Обратите внимание на действия, указанные в блоках. Здесь использована очень важная операция **присваивания**. Запись

переменная = выражение

означает, что необходимо провести вычисления, указанные справа от знака равенства, и полученное значение **присвоить** переменной, стоящей слева от этого знака. Здесь знак = означает не проверку условия, а выполнение действия “обозначить именем значение” (“поместить значение в переменную”). Это действие называется **присваиванием** значения переменной. Например, операция увеличения i на 1 записывается $i = i + 1$ (читается “переменной i присвоить значение $i + 1$ ”).

Ветвящаяся структура представляет процесс, для реализации которого предусмотрено несколько направлений (**ветвей**). Каждое отдельное направление является отдельной ветвью. Направление ветвления выбирается логической проверкой. Например, вычисление

$$q = \begin{cases} a + b, & \text{если } x \leq 0 \\ a - b, & \text{если } x > 0 \end{cases}$$

можем представить ветвящейся структурой, изображенной на рис.8

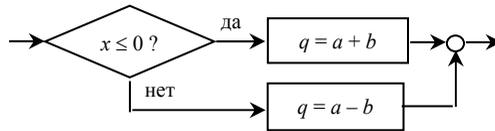


Рис. 8. Альтернативный ветвящийся процесс.

Ветвящийся процесс, включающий в себя две ветви, называется **простым** (или **альтернативным**). Если процесс предполагает более двух ветвей, то он называется **сложным**. Он реализует **множественный выбор** направлений алгоритма. Например, структура, изображенная на рис. 9,

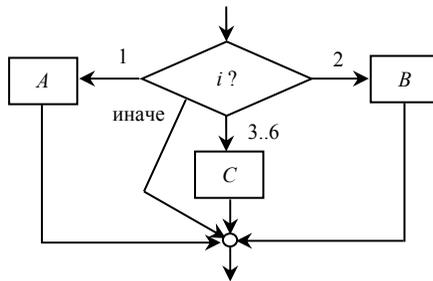


Рис. 9. Сложный ветвящийся процесс.

включает четыре ветви. Передвижение по ним осуществляется в зависимости от значения переменной i : если $i = 1$, производится переход к блоку A , при $i = 2$ – к блоку B , при значениях i от 3 до 6 – к блоку C , при других значениях – выход из процесса.

Заметим, что сложный ветвящийся процесс можно представить с помощью простых.

Циклической называется структура, описывающая процесс, содержащий цикл. **Цикл** – это последовательность многократно повторяющейся группы действий.

В описании цикла можно выделить следующие этапы:

- *подготовка (инициализация)* цикла включает действия по подготовке значений параметров, участвующих в действиях цикла;
- *выполнение (тело цикла)* включает действия, составляющие цикл;
- *модификация параметров* включает действия, изменяющие значения тех параметров, от которых зависит условие окончания цикла;

– проверка условия окончания цикла.

Цикл называется **детерминированным**, если число повторений тела цикла заранее (до начала цикла) известно или определено. Цикл называется **итерационным**, если число повторений тела цикла заранее не известно, а зависит от переменных, участвующих в вычислениях.

Проверка условия окончания может производиться как в конце цикла, так и в начале. В зависимости от его расположения различают *цикл с нижним окончанием* или *с постусловием* (условие проверяется после тела цикла) и *цикл с верхним окончанием* или *с предусловием* (условие проверяется перед телом цикла). Принципиальное отличие заключается в том, что в первом случае тело цикла обязательно выполняется по крайней мере один раз, а во втором – может не выполняться ни разу.

Для того, чтобы понять смысл циклического процесса, рассмотрим следующий пример. Пусть необходимо написать алгоритм вычисления суммы чисел, причем как количество чисел, так и сами числа заранее неизвестны и вводятся пользователем. Обозначим результат суммирования S , количество чисел – n , а каждое из них – r_i ($i = 1, \dots, n$), тогда

$$S = \sum_{i=1}^n r_i .$$

Если бы числа r_1, r_2, \dots, r_n и их количество n были известны заранее, то для вычисления S достаточно было записать арифметическое выражение $S = r_1 + r_2 + \dots + r_n$. В данном случае нам эти числа и их количество *заранее неизвестны*, поэтому поступим следующим образом. Сначала следует узнать у пользователя количество чисел и поместить в переменную n . После этого в S поместим нуль и n раз повторим одну и ту же последовательность действий: “ввести очередное число”, “прибавить число к значению S ”, “поместить результат сложения в переменную S ”. В результате этих действий в S будет накоплена сумма всех введенных пользователем чисел. Группа указанных действий составляет цикл, который должен выполняться n раз.

Соответствующий алгоритм можно представить в виде блок-схемы, изображенной на рис.10.

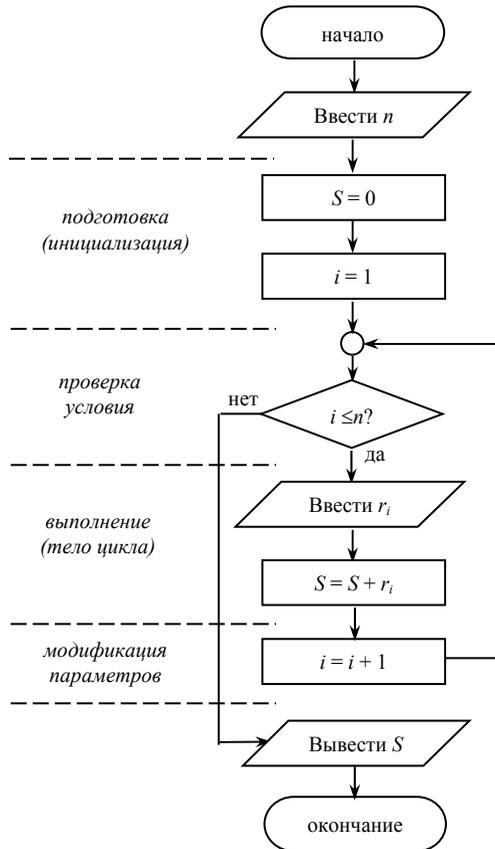


Рис. 10. Циклический процесс.

Здесь действия $S = 0$ и $i = 1$ составляют этап подготовки цикла. Тело цикла состоит из двух действий: ввод числа r_i и $S = S + r_i$. Модификация параметра выполняется действием $i = i + 1$. Такую переменную еще называют **переменной цикла**, т.к. ее значение показывает, сколько раз выполнено тело цикла. Кроме того, в данном случае это еще и номер очередного числа r_i . Цикл выполняется до тех пор, пока $i \leq n$. Заметим, что здесь имеем детерминированный цикл с верхним окончанием (с пред-условием).

2 ОСНОВНЫЕ ПОНЯТИЯ ЯЗЫКА VBA

Общаясь между собой, мы формулируем мысли, пользуясь средствами того языка, который известен собеседникам. Знание языка означает, как минимум, знание алфавита (т.е. множества допустимых символов) и правил построения слов и предложений из элементов этого алфавита (т.е. синтаксиса). Но знание этих средств не гарантирует правильность понимания: нас могут неправильно понять, если мы недостаточно точно выразим свою мысль. Другими словами, от нас требуется еще и знание семантики, т.е. смысла элементов языка (элементов алфавита, слов и предложений). Таким образом, если мы хотим, чтобы собеседник нас правильно понял, мы должны синтаксически и семантически правильно изложить свою мысль на том языке, на котором происходит общение.

Язык программирования играет аналогичную роль, но предназначен не для общения между людьми, а для общения человека с ЭВМ. Как и другие языки, VBA имеет алфавит, синтаксис и семантику. При этом, требования к написанию символов, слов и предложений в языке программирования значительно жестче, чем в любом разговорном языке. Это объясняется тем, что в отличие от человека, ЭВМ не обладает возможностями ассоциативного понимания смысла фразы. Другими словами, ЭВМ понимает только то, что написано в предложении, и не может догадаться о чем-либо, если это не выражено явно и однозначно. Например, если мы сначала укажем, что переменная h обозначает целое число, а затем этой переменной попытаемся присвоить текст “имя”, то эта попытка будет воспринята как ошибка (ведь перед этим мы объявили h целой, а не текстовой переменной). Как и любой язык программирования, VBA построен таким образом, чтобы исключить многозначность понимания написанных на этом языке слов и предложений.

Предложение, написанное на языке программирования, называется *оператором*. Операторы играют роль инструкций по выполнению определенных действий. Выполнение программы сводится, по существу, к выполнению последовательности содержащихся в ней операторов. В них указываются действия присваивания значений переменным, изменения последовательности выполнения других операторов, организации многократного выполнения какой-либо группы операторов и т.д.

Каждый оператор состоит из слов, операций (арифметических, логических и других) и специальных знаков (запятая, скобки и т.д.). Какие-то

из этих слов указывают, какое действие должно быть выполнено, эти слова называются *ключевыми*. Другие указывают, с чем именно надо произвести действие, эти слова составляют так называемые *операнды*. Ключевые слова, операции и операнды разделяются пробелами.

Например, в операторе

$$d = a + 5 * b$$

записано, что необходимо сложить значение переменной a с произведением числа 5 и значения переменной b , после чего приравнять d этому результату (говорят: *присвоить* переменной d значение, равное полученному результату). Здесь использованы знаки арифметических операций, ключевое слово = (*присваивание*) и операнды – переменные a , b , d и число 5.

Оператор

$$\text{If } a > b \text{ Then } d = a * b \text{ Else } d = a / b$$

указывает, что переменной d следует присвоить результат умножения a на b , если значение переменной a больше значения переменной b , в противном случае переменной d следует присвоить результат деления a на b . Здесь используются операнды a , b и d , арифметические операции умножения (*) и деления (/), операция сравнения (>) и ключевые слова *If*, *Then*, *Else* и = (присваивание).

Операторы могут быть указаны как в разных строках, так и в одной строке. В последнем случае между ними должен быть указан знак двоеточия. Оператор можно продолжать в следующей строке. Для этого в конце продолжаемой части через пробел должен быть указан знак подчеркивания (_). Для облегчения понимания текста рекомендуется продолжаемую часть записывать с отступом от начала строки. Например,

$$\begin{array}{l} \text{If } a > b \text{ Then } d = a * b \text{ _} \\ \quad \text{Else } d = a / b \end{array}$$

Строки в программе можно нумеровать (как все, так и некоторые из них). При этом, после номера должно быть не менее одного пробела. Номера строк могут быть любыми, но номер любой строки должен превышать номера предыдущих строк.

Алфавит языка VBA включает в себя буквы латинского и русского алфавитов, цифры от 0 до 9, знаки математических операций и специальные знаки (круглые скобки, знаки #, !, \$, %, @ и т.д.).

В программе можно поместить фрагменты, поясняющие какие-либо ее особенности, но не являющиеся частью ее. Такие фрагменты называются комментариями. Комментарии используются для удобства просмотра текста программы и не выполняются в качестве действий. Они могут быть указаны как в отдельной строке, так и в одной строке с операторами *после них*. Перед комментариями должен быть указан знак апострофа (‘). Если комментарии указаны в отдельной строке, то вместо апострофа можно указать ключевое слово **Rem**.

Программа, написанная на языке VBA, называется макросом и выполняется под управлением программной компоненты, называемой интерпретатором. **Интерпретатор** последовательно читает операторы и обеспечивает их выполнение. Для этого он интерпретирует каждый оператор (определяет смысл данного оператора), переводит его на машинный язык и передает соответствующие инструкции (команды) процессору ЭВМ. Если интерпретатор не может выполнить какой-либо оператор (например, в связи с синтаксической ошибкой), он сообщает пользователю об ошибке и прекращает исполнение программы.

Ввод и редактирование текстов программ осуществляется под контролем специального *текстового редактора VBE – Visual Basic Editor* (см. приложение А).

В данном пособии при описании операторов будем использовать следующие обозначения:

– в квадратных скобках ([...]) указываем ту часть оператора, которая является необязательной, т.е. при определенных условиях может не указываться в программе;

– в угловых скобках (<...>) указываем часть оператора, которая требует специального пояснения.

Эти скобки, разумеется, не указываются в программе, они нам нужны только для удобства записи в данном пособии. Например,

If <условие> **Then** <оператор 1> [**Else** <оператор 2>]

3 ТИПЫ ДАННЫХ

В VBA используются разнообразные типы данных, основными из которых являются:

- Integer** – целое обычное (целое число от -32768 до $+32767$);
- Long** – длинное целое (целое число от -2147483648 до $+2147483647$);
- Single** – вещественное с одинарной точностью;
- Double** – вещественное с двойной точностью;
- String** – строковый;
- Boolean** – логический.

Целые и вещественные (т.е. действительные) – это числовые данные, с ними можно производить математические операции. **Строковые** (текстовые) данные – это последовательности текстовых знаков (символов), для которых допустимы только текстовые преобразования (ввод, вывод, сравнение, объединение т.п.). **Логические** величины принимают только логические значения **True** (истина) и **False** (ложь). Они используются для проверки выполнения условий, при которых требуется выполнить различные действия. К ним применяются операции сравнения и специальные (логические) операции.

Данные делятся на две группы: константы и переменные. **Константы** не изменяются на протяжении работы программы, а значения **переменных** могут изменяться (с сохранением типа). Константы используются в программе в виде чисел (например, 35 и 65.9 – целые константы), текста (например, “*Фамилия*” – строковая константа), а переменные – в виде имен (**идентификаторов**).

Строковая константа в выражениях и операторах должна быть ограничена слева и справа кавычками.

Вещественные константы могут быть указаны как в формате **с фиксированной точкой**, так и в формате **с плавающей точкой**. В первом случае целая часть отделяется от дробной точкой (например, 7.4 и 3290.75 – вещественные константы). Во втором случае число состоит из двух частей, мантиссы и порядка, разделенных латинской буквой *e* (для чисел с одинарной точностью) или *d* (для чисел с двойной точностью). **Мантисса** – это число в формате с фиксированной точкой, **порядок** – целое число, которое указывает степень, в которую надо возвести число 10, чтобы

при умножении полученного числа на мантиссу получить вещественное число, которое имеется в виду. Например, число 234.47 в формате с плавающей точкой может быть записано как 2.3447e+2 (означает $2.3447 \cdot 10^2$). Здесь 2.3445 – мантисса, а 2 – порядок. Изменяя порядок (и, соответственно, мантиссу), можно указать то же самое число и другим образом: 23.447e+1, 0.23445e+3, 2344.7e-1. Формат с плавающей точкой используется для удобства ввода и вывода вещественных чисел большой длины, т.е. чисел, для записи которых требуется большое количество цифр.

Термины “*одинарная точность*” и “*двойная точность*” связаны с объемом памяти, который отводится для хранения вещественных чисел. Для вещественных чисел с двойной точностью этот объем больше, поэтому они обеспечивают большую точность вычислений.

Исключительно важным понятием языков программирования является понятие *переменной*. Оно позволяет использовать конкретные данные, обращаясь к ним с помощью *имен (идентификаторов)*. Такое имя представляет собой последовательность букв и цифр, первый символ – обязательно буква (например, grand5). Можно еще использовать знак подчеркивания (например, My_Value). Величина букв (строчные или прописные) в VBA не имеет значения, поэтому например имена MyValue, myvalue и MYVALUE считаются одинаковыми по смыслу, т.е. именами одной и той же переменной.

Переменная – это именованное место хранения данных в памяти компьютера. Имя переменной *указывает на значение*, которое содержится в этом месте (в этой переменной). В процессе работы программы эти значения можно менять, обращаясь к ним по имени. Таким образом, переменная – это величина, которая может принимать различные значения (аналогично тому, как это понимается в алгебре). Например, в выражении

$$f1 * 5.08 - f4 / (gray + 4)$$

f1, *f4* и *gray* – имена переменных, а 5.08 и 4 – числовые константы.

Имена устанавливаются пользователем, но они не должны совпадать со служебными («*зарезервированными*») словами (ключевыми словами, именами встроенных функций VBA и т.д.).

Использование переменных в программе позволяет записать алгоритм решения задачи в общем виде и сделать программу пригодной для многократного использования с различными исходными данными.

Различают два вида переменных: простые и с индексами. *Простая переменная* указывает на одно значение, т.е. один элемент данных. В приведенном выше примере все переменные – простые.

Часто в программе необходимо описать выполнение действий с последовательностью однотипных данных (например, с целыми числами a_1, a_2, \dots, a_{100}). В этом случае удобнее использовать так называемый массив. **Массив** – это именованное упорядоченное множество однотипных данных (например, a – массив ста целых чисел). Элементы этого упорядоченного множества называются **элементами массива**. Элементы массива пронумерованы и обратиться к каждому из них можно по номеру (или по нескольким номерам – например, для элемента таблицы задается номер строки и столбца). Эти номера называются **индексами**. В языке VBA индексы указываются в круглых скобках после имени массива (например, $a(4)$, $b(3, 10)$). Подобное указание на элемент массива называется **переменной с индексами** или **индексированной переменной**.

Если элемент указывается одним индексом, то массив называется одномерным, если двумя индексами, то – двумерным, тремя – трехмерным и т.д. Например, $g(4,7)$ указывает на элемент двумерного массива g .

В **одномерном** массиве индекс указывает на порядковый номер элемента. Например, переменная с индексом $d(5)$ указывает на пятый элемент среди всех значений, составляющих одномерный массив d .

Двумерный массив, чаще всего, используется для представления таблиц (матриц). Например, массивом w можно представить следующую таблицу

	1	2	3
1	25	12	53
2	42	87	592
3	267	429	84
4	5	32	410

Смысл индексов определяет для себя сам программист, например, первый индекс может указывать на номер строки, а второй индекс – на номер столбца, на пересечении которых находится данный элемент. Тогда переменная с индексом $w(2,3)$ указывает на элемент двумерного массива w , соответствующий элементу таблицы, находящемуся на пересечении второй строки и третьего столбца, т.е. 592. Аналогично, число 429 находится на пересечении 3-ей строки и 2-го столбца и, следовательно, будет представлено элементом $w(3,2)$. Эту же таблицу можно представить и одномерным массивом, например, составляя его из строк последовательно друг за другом. Тогда это же число 429 будет восьмым элемен-

том такого массива (например, $s(8)$, если s – имя массива). Очевидно, в подобных случаях удобнее пользоваться все-таки двумерным массивом.

Соответственно значениям, на которые указывают переменные, различают числовые (целые, вещественные), строковые, логические переменные.

4 ВЫРАЖЕНИЯ

Самыми простыми являются операторы, которые определяют какие-либо вычисления. Выражения могут указывать на проведение арифметических действий (*арифметические выражения*), действий, направленных на проверку каких-либо условий (*логические выражения*), а также действий с текстовыми данными (*строковые выражения*). Для их записи используются знаки соответствующих операций: для арифметических – арифметические операции, для логических – операции сравнения и логические операции.

Например,

– *арифметическое* выражение $f + w$ указывает, что необходимо произвести действие сложения значений переменных f и w ;

– *логическое* выражение $h < 3 * p$ используется для проверки, является ли значение переменной h меньшим, чем произведение целого числа 3 и значения переменной p .

Арифметические операции:

- $+$, $-$ – знаки положительности и отрицательности числа;
- $+$ – сложение,
- $-$ – вычитание,
- $*$ – умножение,
- $/$ – деление,
- $^$ – возведение в степень.

Эти операции производятся над числами и результатом также является число.

Операции сравнения:

- $=$ – равно,
- $<>$ – не равно,
- $>$ – больше,

- $>=$ – больше либо равно,
- $<$ – меньше,
- $<=$ – меньше либо равно.

Эти операции предназначены для сравнения двух величин (например, в выражении $h >= k$ сравниваются значения двух переменных h и k). Результат принимает логическое значение **True** (истина), если условие выполняется, и значение **False** (ложь), если данное условие не выполняется. Подобные выражения называются *логическими*.

Например, результат логического выражения

$$15 < r * g ^ 2$$

равен **True** в том случае, если результат перемножения значения переменной r и квадрата значения переменной g больше 15. Если в процессе выполнения программы значения переменных r и g будут таковы, что результат этого произведения окажется меньшим или равным 15, то указанное условие не выполнится и результат выражения будет равен **False**.

Конкатенация.

Конкатенация – это операция, применяемая к строковым данным (константам, переменным) для их объединения. Для ее обозначения используется знак **&**. Например, результатом выражения

"Иван" & "Петров"

будет строковая константа "Иван Петров".

Эту операцию можно применять и к числовому данному. В этом случае происходит автоматическое преобразование в строковый тип. Например, результатом выражения

"Величина затрат: " & 30000.86 & " руб."

будет строковая константа "Величина затрат: 30000.86 руб."

Если объединяются только текстовые данные, можно вместо знака **&** указывать знак **+**.

Логические операции.

Эти операции применяются к логическим величинам, т.е. величинам, принимающим значения **True** или **False**. Результатом является также логическая величина. Основными из них являются:

- Not** – *инверсия* (отрицание),
- And** – *конъюнкция* (логическое И),
- Or** – *дизъюнкция* (логическое ИЛИ).

Операция **Not** выполняется над одной величиной и результат противоположен ее значению. В качестве такой величины может быть указано выражение с операциями сравнения и логическими операциями. Например,

$$\text{NOT } f > g(3)$$

принимает значение **False**, если значение переменной f больше третьего элемента массива g , и **True** в противном случае.

Операция **And** применяется к двум величинам. Результат равен **True** только в том случае, если оба значения этих величин равны **True**. Например, логическое выражение

$$i < 5 \text{ AND } j * k > m$$

принимает значение **True** только в том случае, если значение переменной i меньше целой числовой константы 5 и произведение значений переменных j и k больше значения переменной m .

Операция **Or** также, как и **And**, применяется к двум логическим величинам, но результат равен **False** только в том случае, если обе величины равны **False**. Другими словами, результат равен **True**, если хотя бы одна величина равна **True**. Например,

$$i < 5 \text{ OR } j * k > m$$

принимает значение **True**, если значение переменной i меньше 5 или результат произведения значений переменных j и k больше значения переменной m .

Приоритетность выполнения операций.

Операции в выражениях выполняются в соответствии с приоритетами. Среди арифметических операций самый высокий приоритет у операции возведения в степень (^), следующий приоритет – у операций умножения (*) и деления (/) и самый низкий приоритет у операций сложения (+) и вычитания (-). В связи с этим, например, результатом выражения – 5^2 будет -25.

У операций сравнения равный приоритет. Логические операции выполняются в следующем порядке: сначала **Not**, затем **And** и последним **Or**.

В смешанных выражениях наиболее высокий приоритет у арифметических операций, затем выполняется операция & (конкатенация), после нее – операции сравнения и самый низкий приоритет у логических операций. Операции с одинаковым приоритетом выполняются слева направо.

во. Порядок выполнения операций может быть изменен использованием круглых скобок, как это принято в математических выражениях.

Например, в выражении

$$\text{NOT } (b \geq 5.7 \text{ AND } (b > h - 1.4 \text{ OR } g / 4.5 < 7.3))$$

указан следующий порядок вычислений: сначала вычисляются значения выражений $h - 1.4$ и $g / 4.5$, затем выполняются операции сравнения в скобках, после этого – операция **Or**, затем – операция сравнения $b \geq 5.7$, операция **And** и последней – **Not**.

5 ПРОЦЕДУРЫ VBA

5.1 Виды процедур

Любая последовательность операторов в VBA называется **кодом**. Действия осуществляются в результате его выполнения. В частности, им может быть фрагмент, оформленный независимо от других фрагментов. Такие коды называются **процедурами**. Их ввод и редактирование осуществляется под контролем специального текстового редактора *Visual Basic Editor (VBE)* в так называемых **модулях** (см. приложение А).

Первый оператор такого кода должен начинаться с ключевого слова объявления процедуры и заканчиваться оператором ее окончания. Процедуры могут располагаться как в одном модуле VBA, так и в разных.

В VBA поддерживаются процедуры трех типов: программа (**макрос**), подпрограмма и функция. В любой процедуре можно оформить запуск на выполнение любой другой процедуры.

Макрос в отличие от других процедур может быть запущен на выполнение вручную пользователем. Функция и подпрограмма могут быть запущены на выполнение только из других процедур. **Функция** в отличие от других процедур описывает только вычисления и возвращает единственное полученное значение через свое имя. **В подпрограмме** могут быть описаны не только вычисления, но и действия с объектами Excel (например, рабочая книга, рабочий лист, область листа и т.д.). Кроме этого, в отличие от функции подпрограмма может вернуть несколько результатов.

Как программа, так и подпрограмма, оформляются одинаково и отличаются лишь отсутствием списка передаваемых аргументов у макроса. Эти процедуры объявляются с помощью ключевого слова **Sub** и заканчиваются оператором **End Sub**:

```
Sub <имя> ([<список>])  
    <оператор 1>  
    <оператор 2>  
    .....  
    <оператор N>  
End Sub
```

Здесь <список> – список аргументов, для макроса он отсутствует. В нем указываются переменные в качестве входных данных и переменные для результатов, полученных во время выполнения подпрограммы. Подпрограмма принимает входные данные через переменные, указанные в списке, производит различные вычисления с использованием этих данных, получает результаты и возвращает их через другие переменные, указанные в этом же списке.

Макрос можно запустить на выполнение двумя способами:

1) на рабочем листе кнопкой **Выполнить** в диалоговом окне команды **Разработчик** ⇒ **Макросы** (рис.11).

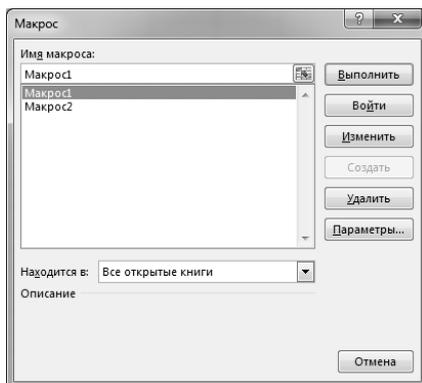


Рис. 11. Диалоговое окно выбора программ для запуска.

2) в окне текстового редактора VBE командами вкладки **Run** (см. приложение А).

Функция объявляется с помощью ключевого слова **Function** и заканчивается оператором **End Function**:

```
Function <имя> ([<список>])  
    <оператор 1>  
    <оператор 2>  
    .....  
    <оператор N>  
End Function
```

Здесь <список> – это список, в котором указываются только переменные для входных данных. Функция принимает входные данные через переменные, указанные в списке, производит вычисления с их использованием и возвращает полученный результат через свое имя.

Как подпрограмма, так и функция может не иметь входных данных.

5.2 Вызов подпрограмм и функций

Подпрограммы и функции используются для исключения дублирования в тексте программы одних и тех же кодов. Кроме этого, они используются для разбиения больших текстов на части с целью облегчения написания (небольшие коды значительно легче воспринимаются) и отладки.

Такой код оформляется в виде отдельного текста (см. п. 5.1), а в процедуре там, где требуется его выполнение, указывается специальный оператор, который называется оператором **вызова подпрограммы** (или **функции**).

Для вызова подпрограммы можно использовать ключевое слово **Call**:

```
Call <имя>[(<список>)]
```

или просто написать имя подпрограммы и список (но тогда через пробел после имени и без скобок)

```
<имя> [<список>]
```

В обоих случаях <список> – это список, в котором указываются входные данные, передаваемые в подпрограмму, и переменные для результа-

тов. Входные данные могут быть переданы в подпрограмму константами, переменными или выражениями.

Функция в отличие от подпрограммы возвращает полученный результат через свое имя, поэтому в тексте функции должен быть соответствующий оператор присваивания результата:

$$\langle \text{имя} \rangle = \langle \text{результат} \rangle$$

Например, если функция называется *FuncSum* и возвращаемый результат находится в переменной *S*, то в тексте этой функции должен быть оператор

$$\text{FuncSum} = S$$

Форма вызова функции такая же, что и для подпрограммы, но без использования ключевого слова **Call**, т.е.

$$\langle \text{имя} \rangle ([\langle \text{список} \rangle])$$

или
$$\langle \text{имя} \rangle [\langle \text{список} \rangle]$$

Здесь *<список>* – это список передаваемых в функцию данных (констант, переменных, выражений). В первом случае результат, возвращаемый функцией, обязательно должен быть использован в каком либо операторе. Например, оператор

$$\text{MsgBox}(\text{"Продолжать вычисления?"}, \text{vbYesNo})$$

во время выполнения кода будет воспринят интерпретатором как ошибочный, поскольку эта функция возвращает результат, означающий, какую кнопку «нажал» пользователь в окне. Но не ясно, что же делать с этим результатом, ведь в операторе ничего об этом не сказано. Чтобы этого не произошло, можно, например, написать так

$$b = \text{MsgBox}(\text{"Продолжать вычисления?"}, \text{vbYesNo})$$

Однако, если возвращаемый результат нам не нужен, лучше использовать вторую форму вызова функции

$$\text{MsgBox} \text{"Продолжать вычисления?"}, \text{vbYesNo}$$

В операторах объявления процедур и операторах их вызова в списках указываются только имена *локальных* переменных (подробнее см. п. 7). Это означает, что область действия этих переменных ограничивается текстом той процедуры, в которой используются. Поэтому имена переменных в списке при вызове процедуры могут не совпадать с именами переменных в списке при ее объявлении. Главное, чтобы элементы этих списков строго соответствовали друг другу.

Например, пусть в подпрограмме

```
Sub Ex6_1(d, e, f)
```

```
...
```

```
End Sub
```

переменная *d* предусмотрена для входного данного, а переменные *e* и *f* – для возвращения результатов.

Чтобы вызвать эту подпрограмму, ей надо передать входное значение для переменной *d* и указать, в какие переменные вернуть результаты.

Например,

```
Sub Ex_6()
```

```
...
```

```
Call Ex6_1(a, b, c)
```

```
...
```

```
Ex6_1 g, w, q
```

```
...
```

```
End Sub
```

Здесь в процедуре *Ex_6* дважды вызывается подпрограмма *Ex6_1*. В первый раз вызов оформлен с помощью ключевого слова **Call**, входное данное передается с помощью переменной *a* и указаны переменные *b* и *c*, в которые должны вернуться результаты. Во втором вызове входное данное передается через переменную *g*, а для получения результатов указаны переменные *w* и *q*.

5.3 Встроенные функции VBA

Как и в других языках программирования, в VBA есть так называемые встроенные функции. Эти функции упрощают вычисления и выполнение различных действий. Например,

Abs(<arg>) – модуль числа,

Int(<arg>) – целая часть числа,

Round(<arg>) – округленное число,

Sqr (<arg>) – квадратный корень числа,

Str(<arg>) – строковое представление числа.

В этих функциях <arg> – числовое данное (константа, переменная, выражение).

В VBA имеются функции, которые преобразуют тип аргумента:

CInt(<arg>) – Integer,

CLng(<apε>) – Long,
CSng(<apε>) – Single,
CDbl(<apε>) – Double,
CStr(<apε>) – String,
CBool(<apε>) – Boolean,

Разумеется, данные, передаваемые в эти функции, должны подходить (быть пригодными) для того типа, в который преобразуются, иначе будет ошибка. Например, при попытке преобразовать длинное целое (**Long**) в обычное целое (**Integer**) длинное целое должно находиться границах, допустимых для обычного целого (от -32768 до +32767).

6 ОПЕРАТОР ПРИСВАИВАНИЯ

Действие присваивания заключается в вычислении результата выражения (арифметического, логического и т.д.) и его размещении в области памяти, обозначенной именем переменной (говорят «присвоить переменной результат вычисления выражения» или «в переменную поместить результат вычисления выражения»). Соответствующий оператор записывается следующим образом

$\langle \text{имя} \rangle = \langle \text{выражение} \rangle$

Здесь $\langle \text{имя} \rangle$ – это идентификатор переменной (простой или с индексами), $\langle \text{выражение} \rangle$ – арифметическое, текстовое или логическое выражение, которое, в частности, может быть переменной или константой.

Например,

$$d = 34.5$$

означает, что переменной d присвоить значение 34.5 (в переменную d “поместить” константу 34.5);

$$f(2) = d + r(5) * y ^ 3$$

означает, что необходимо возвести значение переменной y в третью степень, умножить результат на значение пятого элемента массива r , сложить полученный результат со значением переменной d и результат этого действия поместить в массив f в качестве второго элемента. Другими словами, второму элементу массива f присвоить значение, полученное в результате выполнения арифметических действий, указанных справа от знака равенства.

При записи выражений необходимо следить за согласованием типов данных слева и справа от знака присваивания. Например, следует иметь в

виду, что запрещено присваивать числовым или текстовым переменным логические значения и наоборот. Кроме этого, если числовой переменной присваивается числовое значение другого типа, то это значение преобразовывается в тип, указанный для переменной, стоящей слева от знака присваивания. Например, если f – переменная целого типа, то после выполнения оператора

$$f = 34.2$$

произойдет округление числа 34.2 и переменная f будет указывать на целое число 34, а не на вещественное 34.2.

7 ОБЪЯВЛЕНИЕ ПЕРЕМЕННЫХ

Как мы уже знаем, переменные могут простыми или с индексами, а также разного типа (строковые, целые, вещественные с одинарной или двойной точностью). Для того, чтобы интерпретатор правильно понимал имя используемой переменной (простая, с индексами, тип), необходимо с помощью специального оператора сообщить ему соответствующую информацию (говорят “*описать переменную*”). Например, можно указать, что под именем r следует понимать простую переменную целого типа, а под именем g – одномерный массив не более 26 вещественных значений с одинарной точностью.

Если таких явных указаний не сделать, интерпретатор берет на себя ответственность за понимание используемых имен переменных (говорят “*по умолчанию*”). При этом, тип каждой переменной устанавливается автоматически в момент использования в программе в зависимости от того, какого типа значение присваивается переменной.

Например, если переменная i использована в операторе присваивания $i = 1$, то с этого момента под i будет пониматься целая простая переменная, а если – в операторе $i = 1.0$, то – простая переменная вещественного типа. То есть тип переменной здесь устанавливается автоматически в зависимости от типа присваиваемой константы. При этом, вещественный тип по умолчанию устанавливается с двойной точностью.

Иногда это бывает удобно, но рекомендуется все-таки все переменные описывать явно, причем в самом начале программы (сразу после оператора **Sub**), чтобы уменьшить вероятность ошибок, связанных с несогласованностью типов данных в операторах. Для этого используется оператор

Dim <перем.1> [**As** <тип>], <перем.2> [**As** <тип>], ...

Здесь <перем.1>, <перем.2> и т.д. – имена переменных, **Dim** и **As** – ключевые слова.

Если указывается массив, то после его имени необходимо указать в круглых скобках наименьший и наибольший индекс, которые будут допустимы для нумерации элементов этого массива в операторах. Между ними указывается ключевое слово **To**. Например,

Dim d(1 To 5, 3 To 24) As Single

означает, что *d* – это имя двумерного массива вещественных чисел, причем при указании любого элемента этого массива первый индекс должен быть от 1 до 5, а второй индекс – от 3 до 24.

Наименьшее значение индекса вместе с **To** можно не указывать. В этом случае он автоматически будет установлен равным 0. Поэтому, например, операторы

Dim d(0 To 10) As Single

и

Dim d(10) As Single

приведут к одному и тому же результату – объявлен одномерный массив вещественного типа, состоящий из 11 элементов, начиная с нулевого *d*(0) и заканчивая десятым *d*(10).

Оператор **Dim** может быть указан в любом месте программы перед первым использованием описанных переменных, но, как уже говорилось вначале, рекомендуется объявление всех переменных производить в начале программы (сразу после оператора **Sub**).

Следует иметь в виду важную особенность использования **Dim**: типы переменных и размеры массивов, объявленные этим оператором, остаются неизменными во всем последующем тексте программы. Если заранее не известно, сколько элементов потребуется в массиве, можно объявить его *динамическим*, т.е. имеющим только имя и тип, но не имеющим размерности. Для этого достаточно при его объявлении (с помощью **Dim**) в скобках не указывать размерность, а указать ее позже (перед использованием массива) с помощью оператора **ReDim**.

Например,

Dim Прибыль () As Single, Затраты() As Single, _

n As Integer, m As Integer

.....
n = 5

m = 20

.....
ReDim Прибыль (1 To 16), Затраты(1 To n, 1 To m)

.....
Здесь оператором **Dim** объявлены два массива данных вещественного типа *Прибыль* и *Затраты*, а затем с помощью оператора **ReDim** указано, что массив *Прибыль* – одномерный, а массив *Затраты* – двумерный, причем размер массива *Прибыль* задан константой 16, а размер массива *Затраты* задан переменными *n* и *m*.

Оператор **ReDim** можно использовать повторно, он позволяет *перепределять* имена массивов и использовать их по мере необходимости.

Например,

Dim Прибыль () As Single

.....
n = 5

ReDim Прибыль (1 To n)

.....

m = 20

ReDim Прибыль (1 To n, 1 To m)

.....

В данном случае мы сначала с помощью **ReDim** указали, что массив *Прибыль* – одномерный и задали его размер переменной *n*, а затем перепределили его, указав, что он теперь двумерный, и задав размер переменными *n* и *m*.

Обычно используют так называемые *локальные имена* переменных. Область действия такой переменной ограничивается текстом той процедуры, в которой объявлена (с помощью **Dim**). В разных процедурах могут быть использованы одни и те же локальные имена, при этом все они считаются уникальными в своих процедурах.

Если необходимо, чтобы переменная была доступна во всех процедурах модуля, ее следует объявить (оператором **Dim**) *перед* первой процедурой. В этом случае во всех процедурах этого модуля имя такой переменной будет означать одну и ту же область памяти.

Чтобы сделать переменную доступной во всех процедурах *всех* модулей, ее следует объявить с помощью оператора **Public** (а не **Dim**).

8 ДИАЛОГОВЫЙ ВВОД И ВЫВОД ДАННЫХ С ПОМОЩЬЮ ВСТРОЕННЫХ ФУНКЦИЙ

Ввод и вывод данных может осуществляться по-разному. Диалоговый ввод (или вывод) означает, что для этих целей используются диалоговые окна, в которых пользователь читает сообщения или указывает данные, которые хочет ввести. В VBA имеются соответствующие разнообразные возможности. Проще всего, для этого использовать встроенные функции **InputBox** (диалоговый ввод данных) и **MsgBox** (диалоговый вывод).

8.1 Диалоговый ввод данных

Функция **InputBox** автоматически создает диалоговое окно, в котором имеется поле для ввода текста и две командные кнопки: **OK** (подтверждение ввода) и **Cancel** (отмена). От пользователя требуется указать константу в поле ввода и щелкнуть по кнопке **OK**. В результате, эта функция возвращает введенную константу. Общий формат этой функции имеет следующий вид:

```
InputBox(<текст> [, <заголовок>] [, <default>]  
        [, <слева>] [, <сверху>] [, <справка, раздел>])
```

Здесь

- <текст> – единственный обязательный аргумент с текстом комментария для вывода в окне;
- <заголовок> – текст заголовка окна (по умолчанию – Microsoft Excel);
- <default> – константа, которая выводится в поле ввода при открытии окна (для “умолчания”);
- <слева> и <сверху> – координаты верхнего левого угла окна;
- <справка, раздел> – файл и раздел справочной системы, прилагаемой к программе. Если указать этот аргумент, в диалоговом окне будет отображена кнопка **Справка**.

Чаще всего, эту функцию используют с первыми тремя аргументами. Например, оператор

```
Затраты = InputBox("Введите размер затрат", "Ввод данных", 30000)
```

приведет к выводу на экран диалогового окна (рис. 12)

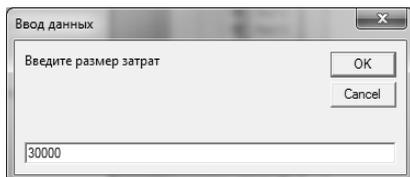


Рис. 12. Диалоговое окно ввода данных.

Пользователь может изменить значение, указанное в поле ввода. После щелчка по кнопке **OK** переменной *Затраты* будет присвоено введенное пользователем значение, автоматически преобразованное к типу, заданному ранее в операторе **Dim** для этой переменной. Если тип этой переменной не был задан, ей будет присвоено строковое (**String**) значение.

Значения аргументов *<текст>* и *<заголовок>* могут быть заданы текстовыми константами, переменными или выражениями с операцией конкатенации (&). Например,

```
number = 5
```

```
Q = InputBox("Введите количество " & number & "-го товара", _  
            "Ввод данных", 0)
```

приведет к выводу на экран диалогового окна (рис. 13)

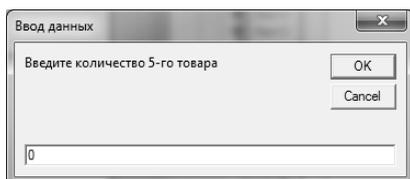


Рис. 13. Конкатенация текста комментария.

Текст комментария (<текст >) может быть многострочным. Для этого используется встроенная функция **Chr** с аргументом 13. Например, оператор

```
Затраты = InputBox("Введите размер затрат" + Chr(13) + _  
"в июне", "Ввод данных", 0)
```

приведет к выводу на экран диалогового окна (рис. 14)

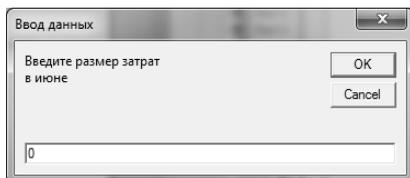


Рис. 14. Многострочный текст комментария.

8.2 Диалоговый вывод данных

Функция **MsgBox** выводит на экран диалоговое окно с текстовым сообщением и набором командных кнопок (**OK**, **Отмена**, **Да**, **Нет** и т.д.). После выбора пользователем одной из этих кнопок **MsgBox** возвращает результат этого выбора.

Общий формат этой функции имеет следующий вид:

```
MsgBox(<текст > [, <кнопки>] [, <заголовок>]  
[, <справка, раздел>])
```

Здесь аргументы <текст>, <заголовок> и <справка, раздел> имеют такой же смысл, что и в функции **InputBox**, и оформляются аналогично. Значение аргумента <кнопки> определяет набор кнопок в окне:

- vbOKOnly – **OK**;
- vbOKCancel – **OK** и **Отмена**;
- vbYesNo – **Да** и **Нет**;
- vbYesNoCancel – **Да**, **Нет** и **Отмена**;
- vbAbortRetryIgnore – **Прервать**, **Повтор** и **Пропустить**;
- vbRetryCancel – **Повтор** и **Отмена**.

Если этот аргумент не указан (по умолчанию), имеется в виду vbOKOnly.

Данная функция возвращает результат “нажатия” кнопки пользователем:

vbYes – “Да”; **vbNo** – “Нет”; **vbCancel** – “Отмена” и т.д.

Этот результат можно присвоить переменной или использовать для выполнения различных действий в зависимости от того, на какую кнопку нажал пользователь. Например,

```
...
v = 10
If vbNo = MsgBox("Использовать " + CStr(v) + "?", vbYesNo, _
    "Подтверждение") Then End
...
```

В результате (благодаря **MsgBox**), на экране появится окно (рис. 15):

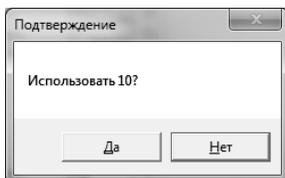


Рис. 15. Диалоговое окно вывода данных.

Как только пользователь «нажмет» на одну из кнопок (**Да** или **Нет**), в программе будет произведена проверка (**If**) значения, которое возвратила функция **MsgBox**. Если это значение окажется равным **vbNo** (“Нет”), то (**Then**) произойдет прекращение выполнения программы (**End**).

Функцию **MsgBox** можно оформить, не предполагая использование возвращаемого значения (например, если она используется исключительно для вывода сообщения). Для этого достаточно указать список аргументов без скобок. Например,

```
v = 10
MsgBox "Будет использовано " + CStr(v), vbOKOnly, "Сообщение"
```

В окно, которое формируется функцией **MsgBox**, можно добавить знак, комментирующий вид диалога. Для этого аргумент **<кнопки>**

оформляется с добавлением (с помощью знака +) соответствующего значения:

- vbInformation – информация, не требующая ответа ⓘ;
- vbQuestion – вопрос ⓘ;
- vbExclamation – важная информация ⓘ;
- vbCritical – предупреждение ⓧ.

Например, в результате выполнения операторов

```
v = 10
```

```
MsgBox "Будет использовано " + CStr(v), _  
vbOKOnly + vbInformation, "Сообщение"
```

увидим следующее окно вывода (рис. 16):

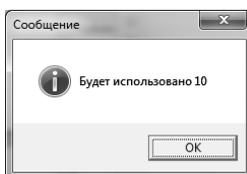


Рис. 16. Добавление знака, комментирующего вид диалога.

Здесь оформлен вывод диалогового окна с названием *Сообщение*, текстом *Будет использовано* и числовым значением переменной *v*, командной кнопкой (*OK*) и сопроводительным знаком ⓘ.

9 ОРГАНИЗАЦИЯ ЦИКЛА

Напомним (см. п. 1.2), что циклом называется процесс многократного повторения одной и той же последовательности действий. Указание группы операторов для выполнения цикла производится с помощью разных операторов.

9.1 Цикл с постусловием *Do – Loop Until*

Во время выполнения программы могут возникать ошибочные ситуации, связанные с неверными данными, которые вводит пользователь. Например, в следующей части программы

```
Dim product() As String, n As Integer
n = InputBox("Введите количество товаров", "Ввод данных", 1)
ReDim product(1 To n)
```

...

определен динамический массив *product* (с помощью *Dim*), затем вводится число (с помощью *InputBox*) и присваивается переменной *n*, после чего она используется для задания размера массива в операторе *ReDim*.

Разумеется, если пользователь введет число меньше 1, возникнет ошибка в операторе *ReDim*, поскольку в данном случае наибольший индекс массива *product* не может быть нулевым или, тем более, отрицательным.

Человеку свойственно ошибаться и желательно в программах предусматривать проверку корректности вводимых данных. Один из простых способов реализации такого контроля основан на использовании операторов *Do* и *Loop Until*.

```
Do
  <оператор 1>
  <оператор 2>
  .....
  <оператор N>
Loop Until <условие>
```

Эти операторы выполняют указанную между ними последовательность операторов до тех пор, пока условие, записанное после ключевых слов *Loop Until*, не выполняется, т.е. принимает значение *False*. Как только это условие станет равным *True*, цикл закончится.

В данном случае условие проверяется *после* выполнения тела цикла, поэтому указанная последовательность обязательно выполнится по крайней мере один раз. Другими словами, эти операторы реализуют цикл с постусловием.

Например, если необходимо ввести число в пределах от единицы до 31, то добиться от пользователя правильного ввода можно, если действовать в соответствии с блок-схемой, представленной на рис. 17.

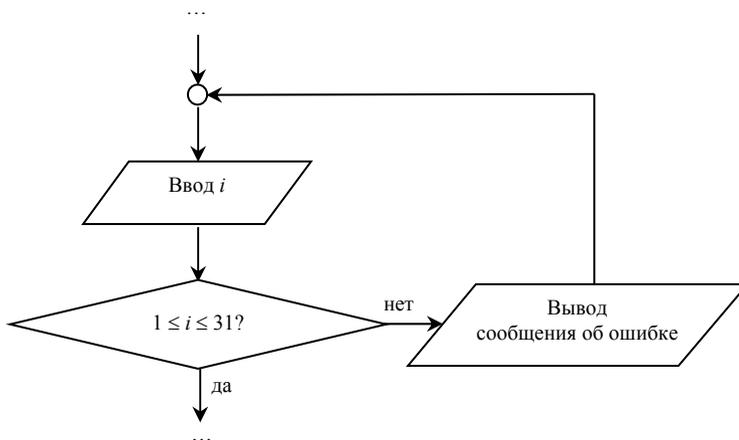


Рис. 17. Проверка корректности ввода.

Как видно из блок-схемы, алгоритм не продолжится до тех пор, пока не будет введено число, удовлетворяющее проверяемому условию. Для реализации этого достаточно написать

```

...
Do
  i = InputBox("Введите номер элемента (от 1 до 31)", _
    "Ввод данных", 1)
  If (i < 1) Or (i > 31) Then _
    MsgBox "Недопустимый номер элемента: " + Str(i) + _
      Chr(13) + "Повторите ввод.", vbOKOnly + vbCritical, _
      "Ошибка"
Loop Until (i >= 1) And (i <= 31)
...

```

В результате, если пользователь, например введет 32, на экран будет выведено окно с сообщением об ошибке (рис. 18):

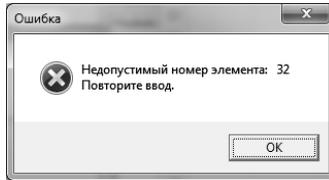


Рис. 18. Предупреждение об ошибке ввода.

и требование ввода номера элемента повторится. Так будет продолжаться до тех пор пока условие $(i \geq 1)$ And $(i \leq 31)$, указанное после **Loop Until**, не примет значение **True**.

9.2 Цикл с постусловием *Do – Loop While*

Do

<оператор 1>
<оператор 2>
.....
<оператор N>

Loop While <условие>

Операторы **Do** и **Loop While** выполняют указанную между ними последовательность операторов до тех пор, пока условие, записанное после ключевых слов **Loop While**, выполняется, т.е. принимает значение **True**. Как только это условие станет равным **False**, цикл закончится.

9.3 Цикл с предусловием *While – Wend*

Операторы **While** и **Wend** выполняют указанную между ними последовательность операторов, пока условие, записанное после ключевого слова **While**, имеет значение **True**.

While <условие>

<оператор 1>
<оператор 2>
.....
<оператор N>

Wend

Заметим, что эти операторы реализуют цикл с предусловием (см. п. 1.2), т.е. тело цикла может не выполниться ни разу. Например, рассмот-

Алгоритм решения этой задачи должен включать ввод пользователем величины предельной суммы (обозначим ее S_{\max}), циклический ввод чисел и их суммирование, а также вывод результата. Кроме этого, необходимо проводить контроль правильности ввода чисел, они должны быть неотрицательными.

Сначала делаем сумму равной нулю ($S = 0$), а затем в цикле наращиваем, прибавляя к ней каждый раз очередное введенное пользователем число ($S = S + r$). Вычисления должны остановиться, как только сумма введенных чисел превысит заданный предел, т.е. как только *не* выполнится условие $S \leq S_{\max}$.

Обратите внимание, что внутри цикла вычисления суммы мы имеем цикл ввода очередного числа с проверкой на правильность ввода. Для описания внешнего цикла мы используем *While* и *Wend*, а для описания внутреннего – *Do* и *Loop Until*.

```
Sub Ex9_1()
    Dim n As Integer, r As Single, S As Single, Smax As Single
    Smax = InputBox("Введите размер предельной суммы", _
        "Ввод данных", 0)
    If Smax <= 0 Then End
    n = 0
    S = 0
    While S <= Smax
        n = n + 1
        Do
            r = InputBox("Введите " + CStr(n) + _
                "-е неотрицательное число", "Ввод данных", 0)
            If r < 0 Then _
                MsgBox "Недопустимое число: " + CStr(r) + _
                    Chr(13) + "Повторите ввод.", _
                    vbOKOnly + vbCritical, "Ошибка"
        Loop Until r >= 0
        S = S + r
    Wend
    MsgBox "Сумма введенных " + CStr(n) + " чисел равна " + _
        CStr(S), vbOKOnly + vbInformation, "Вывод результата"
End Sub
```

Возвращаясь к задаче вычисления суммы заранее неизвестных чисел (см. п. 1.2), соответствующую программу с помощью операторов **While** – **Wend** можно написать в следующем виде.

```
Sub Ex9_2()
    Dim n As Integer, r() As Single, S As Single
    n = InputBox("Введите количество элементов", "Ввод данных", 0)
    If n <= 0 Then End
    ReDim r(1 To n)
    S = 0
    i = 1
    While i <= n
        r(i) = InputBox("Введите " + CStr(i) + "-е число", _
            "Ввод данных", 0)
        S = S + r(i)
        i = i + 1
    Wend
    MsgBox "Сумма введенных " + CStr(n) + " чисел равна " + _
        CStr(S), vbOKOnly + vbInformation, "Вывод результата"
End Sub
```

9.4 Цикл со счетчиком *For* – *Next*

Операторы **For** и **Next** повторяют выполнение указанной между ними последовательности операторов заданное число раз.

```
For <имя> = <начало> To <конец> [Step <шаг>]
```

```
    <оператор 1>
```

```
    <оператор 2>
```

```
    .....
```

```
    <оператор N>
```

```
Next [<список>]
```

Здесь **For**, **To**, **Step** – ключевые слова. **For** требуется для определения имени так называемой **переменной цикла** (<имя>) и ее начального значения (<начало>), т.е. значения, при котором цикл выполняется в первый раз. **To** определяет значение (<конец>) этой переменной, при котором цикл выполняется в последний раз, а ключевое слово **Step** – шаг изменения (<шаг>) значений переменной цикла, т.е. то число, на которое должно изменяться это значение после очередного выполнения группы операторов, составляющих цикл.

Величины *<начало>*, *<конец>* и *<шаг>* могут быть заданы арифметическими выражениями (чаще всего, это константы или переменные).

Оператор **For** указывает, что за ним следует группа операторов, которая должна повторяться до тех пор, пока значение переменной цикла не превысит значение *<конец>*. Начинается цикл со значения переменной цикла, равного значению *<начало>*. После каждого повторения цикла оператор **Next** автоматически наращивает значение этой переменной на величину, равную значению *<шаг>*. Если **Step** (вместе с *<шаг>*) не указан, то интерпретатор считает его равным единице.

<список> в операторе **Next** используется, если применяется несколько циклов **For – Next**, вложенных друг в друга. Каждый из них, разумеется, начинается с оператора **For**, и в каждом из них своя переменная цикла. Тогда, если **Next** один для всех, то для него необходим список, в котором перечисляются переменные всех этих циклов в порядке от последнего к первому (по вложенности циклов). Если вложенных циклов нет, т.е. переменная цикла одна, то список не нужен.

Например,

For i = -10 To 8 Step 2	и	For i = -10 To 8 Step 2
For j = 1 To 10		For j = 1 To 10
MsgBox Str(i) + ", " + Str(j)		MsgBox Str(i) + ", " + Str(j)
Next j, i		Next
		Next

реализуют один и тот же алгоритм (рис. 20).

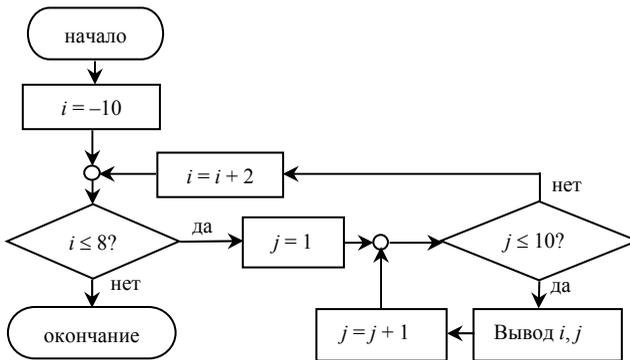


Рис. 20. Вложенные циклы.

Следует иметь в виду, что оператор **For**:

- проверяет достижение переменной цикла ее предельного значения и
- производит выход из цикла, как только значение этой переменной превысит значение, указанное после ключевого слова **To**.

Это означает, что цикл может ни разу не выполниться, если начальное значение переменной цикла (<начало>) превышает предельное значение (<конец>). Другими словами, операторы **For** и **Next** так же, как и **While** и **Wend**, реализуют цикл с предусловием.

Основное отличие между **While – Wend** и **For – Next** заключается в простоте описания действий, связанных с переменной цикла. Если применение **While – Wend** предполагает явную запись операторов присваивания начального значения этой переменной (например, $i = 1$), изменение ее значения (например, $i = i + 1$) и условия выхода из цикла (например, $i \leq n$), то применение **For – Next** этого не требует. Достаточно лишь указать <начало>, <конец> и <шаг> в операторе **For**, что упрощает текст программы.

Например, программу вычисления суммы заранее неизвестных чисел (см. п. 1.2 и п. 9.3) с помощью операторов **For – Next** можно написать достаточно просто в следующем виде.

```
Sub Ex9_3()
    Dim n As Integer, r() As Single, S As Single
    n = InputBox("Введите количество элементов", "Ввод данных", 0)
    If n <= 0 Then End
    ReDim r(1 To n)
    S = 0
    For i = 1 To n
        r(i) = InputBox("Введите " + CStr(i) + "-е число", _
            "Ввод данных", 0)
        S = S + r(i)
    Next
    MsgBox "Сумма введенных " + CStr(n) + " чисел равна " + _
        CStr(S), vbOKOnly + vbInformation, "Вывод результата"
End Sub
```

10 ОПЕРАТОРЫ ВЕТВЛЕНИЯ

Операторы перехода служат для описания ветвлений, т.е. принудительного изменения последовательности выполнения операторов в программе.

10.1 Оператор безусловного перехода

GoTo <метка>

Этот оператор обеспечивает переход в то место программы, где указана <метка>. Такая метка может быть задана номером строки или именем. Номер указывается в начале строки перед оператором. Между ними должно быть не менее одного пробела. Номер любой строки должен превышать номера предыдущих строк.

Если метка задана именем, она также должна начинаться с первой позиции в строке, но заканчиваться двоеточием. Например,

```
Sub Ex10_1()
  If InputBox("Введите свое имя") <> "Мария" Then _
    GoTo InvalidName
  MsgBox ("Привет, Мария!")
  ...
  ...
End
InvalidName:
  MsgBox "Неверное имя."
  ...
  ...
End Sub
```

10.2 Альтернативное ветвление If – Then – Else

Конструкция *If – Then – Else* обеспечивает *простое (альтернативное)* ветвление, когда в качестве проверяемого условия записывается *логическое* выражение и в зависимости от истинности или ложности результата выбирается одна из указанных групп операторов.

```
If <условие> Then  
    <группа 1>  
    [Else  
    <группа 2>]  
End If
```

Здесь

<условие> – логическое выражение;
<группа 1> – группа операторов, которые выполняются, если <условие> выполняется, т.е. результат выражения равен **True**;
<группа 2> – группа операторов, которые выполняются, если <условие> не выполняется, т.е. результат выражения равен **False**;
End If – окончание ветвления.

Например, рассмотрим задачу вычисления суммы от 1 до числа, указанного пользователем, если это число больше 10, и факториала этого числа в противном случае. Алгоритм решения можно представить блок-схемой, изображенной на рис. 21.

Здесь сначала вводится число (n) с проверкой правильности ввода (число должно быть больше 0). Затем проверяется его значение и выполняется суммирование, если оно больше 10, или вычисляется его факториал в противном случае.

В обоих случаях вычисление результата (S) выполняется циклически, но есть отличия в действиях, совершаемых с этой переменной. В первом случае (суммирование) начальное значение переменной S должно быть равным 0 и наращивание этого значения должно производиться с помощью операции сложения с числами от 1 до введенного пользователем числа. Во втором случае (вычисление факториала) начальное значение переменной S должно быть равным 1 и наращивание этого значения должно производиться с помощью операции умножения.

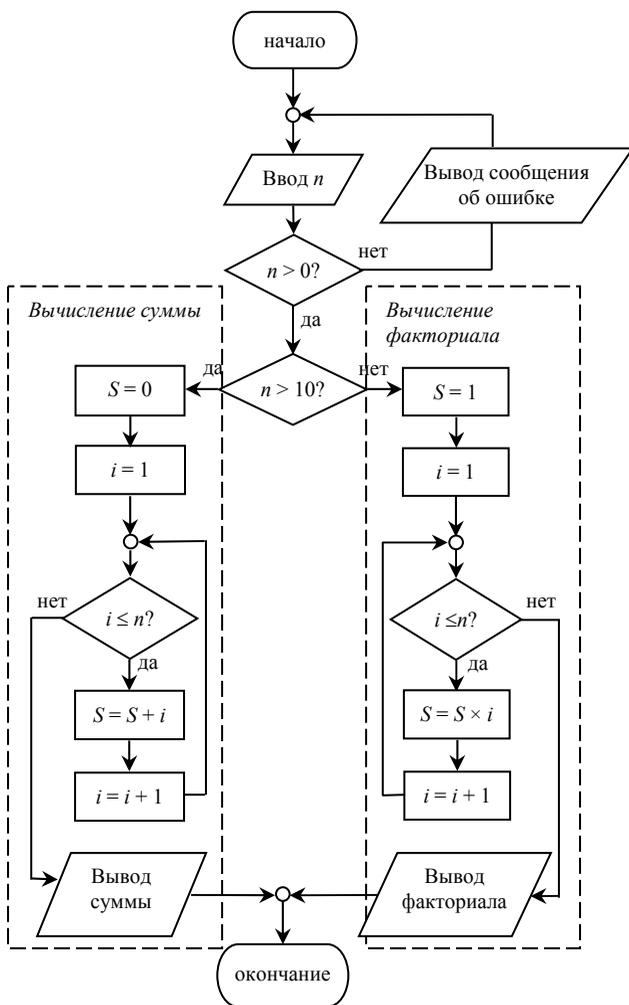


Рис. 21. Пример альтернативного ветвления.

Для реализации этого алгоритма можно использовать следующий программный текст:

```
Sub Ex10_2()
```

```
Dim n As Integer, S As Integer
```

```

Do
    n = InputBox("Введите целое положительное число", _
                "Ввод данных", 1)
    If n < 1 Then _
        MsgBox "Недопустимое число: " + CStr(n) + _
            Chr(13) + "Повторите ввод.", _
            vbOKOnly + vbCritical, "Ошибка"
Loop Until n > 0
If n > 10 _
    Then
        S = 0
        For i = 1 To n
            S = S + i
        Next
        MsgBox "Сумма чисел от 1 до " + CStr(n) + " равна " + _
            CStr(S), vbOKOnly + vbInformation, "Вывод результата"
    Else
        S = 1
        For i = 1 To n
            S = S * i
        Next
        MsgBox "Факториал числа " + CStr(n) + " равен " + _
            CStr(S), vbOKOnly + vbInformation, "Вывод результата"
    End If
End Sub

```

Если <группа 1> состоит из одного оператора и <группа 2> также состоит из одного оператора, то оператор **End If** можно не указывать, а оператор записывается в одной строке:

If <условие> **Then** <оператор 1> [**Else** <оператор 2>]

Вместо этой конструкции можно использовать встроенную функцию **If**:

If (<условие>, <оператор 1>, <оператор 2>)

Эта функция возвращает результат, полученный с помощью <оператор 1>, если <условие> равно **True**, или результат, полученный с помощью <оператор 2>, если <условие> равно **False**.

10.3 Оператор множественного выбора

Оператор множественного выбора обеспечивает описание *сложного* ветвящегося процесса, когда в качестве проверяемого условия записывается *арифметическое или строковое* выражение и в зависимости от результата выбирается одна из нескольких групп операторов.

Этот оператор оформляется с помощью ключевых слов *Select Case* (описание проверяемого условия), *Case* (описание ветви) и *End Select* (окончание оператора выбора).

```
Select Case <выражение>  
  Case <результат 1>  
    <группа 1>  
  Case <результат 2>  
    <группа 2>  
  .....  
  Case <результат N>  
    <группа N>  
  [Case Else  
    <группа Else>]
```

End Select

Здесь

<выражение> – арифметическое или строковое выражение (в частности, это может быть число, текстовая константа, переменная числового или строкового типа);

<результат i > – i -й результат ($i=1, \dots, N$) вычисления выражения;

<группа i > – группа операторов, которые выполняются, если получен i -й результат ($i=1, \dots, N$);

<группа Else> – группа операторов, которые выполняются, если полученный результат не совпадает ни с одним из <результат i >.

В качестве результата после *Case* можно указывать

- константу, например, *Case 1*;
- список (через запятую), например, *Case 2, 3*;
- диапазон (с помощью ключевого слова *To*), например, *Case 4 To 6*;
- логическое выражение (с помощью ключевого слова *Is* и оператора сравнения), например, *Case Is >= 7*.

Например, вычисление

$$S = \begin{cases} a + b, & \text{если } x = 1 \\ a - b, & \text{если } x = 2 \text{ или } x = 3 \\ a \times b, & \text{если } 4 \leq x \leq 6 \\ a / b, & \text{если } x \geq 7 \\ \text{иначе } 0 \end{cases}$$

представляется сложной ветвящейся структурой, изображенной на рис. 22.

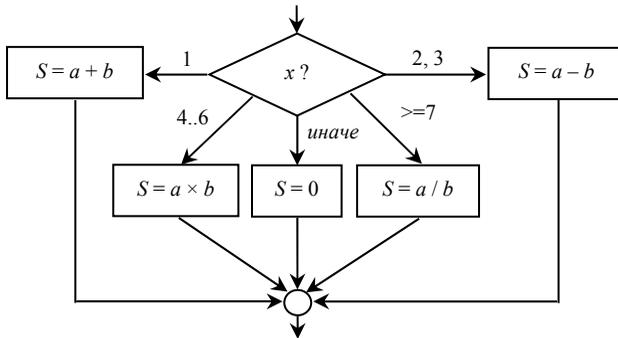


Рис. 22. Пример множественного выбора.

Для ее реализации можно использовать следующий программный текст:

```

...
Select Case x
  Case 1
    S = a + b
  Case 2, 3
    S = a - b
  Case 4 To 6
    S = a * b
  Case Is >= 7
    S = a / b
  Case Else
    S = 0
End Select
...

```

11 ОПЕРАТОРЫ ПЕРЕРЫВАНИЯ

Можно прервать выполнение программы, подпрограммы, а также циклов *Do – Loop Until* и *For – Next*. Это применяется, если необходимо их завершить немедленно, не продолжая дальнейших действий. Для этого используется ключевое слово *Exit* с добавлением ключевого слова, с которого начинается соответствующая программная структура:

Exit Sub – прерывание программы или подпрограммы с возвращением в вызывающую процедуру,

Exit Do – прерывание циклов *Do–Loop Until* и *Do–Loop While*,

Exit For – прерывание цикла *For – Next*.

Например, необходимо вычислить сумму (обозначим ее S) n чисел, которые вводит пользователь, но производить вычисления только до тех пор, пока эта сумма не превышает фиксированной величины S_{max} . Тогда часть программы

```
...
S = 0
For i = 1 To n
    v = InputBox("Введите " + Str(i) + "-е число")
    If S + v > Smax Then Exit For
    S = S + v
Next
...
```

реализует алгоритм, представленный на рис. 23.

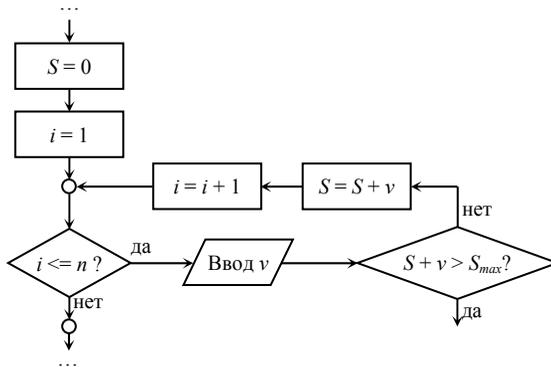


Рис. 23. Пример с прерыванием выполнения цикла.

Если необходимо вообще прекратить вычисления, используется оператор **End**. Например, можно это сделать, если пользователь ввел неверное число:

```
n = InputBox("Введите целое положительное число")
If n <= 0 Then End
```

12 ОСНОВЫ АВТОМАТИЗАЦИИ ПРОВЕДЕНИЯ РАСЧЕТОВ В EXCEL

12.1 Объектно-ориентированный принцип VBA

До сих пор мы рассматривали возможности VBA работы с данными. Однако наиболее интересно применять VBA для создания программ, автоматизирующих решение различных задач с использованием возможностей Excel. С помощью программирования можно создавать рабочие книги, вводить данные и осуществлять расчеты в табличной форме, использовать функции Excel и многое другое. Все это возможно благодаря тому, что VBA относится к группе так называемых объектно-ориентированных языков программирования. В таком языке главным является не понятия «данные», которые надо обработать, и «процедура», определяющая способ обработки, а понятия «класс» и «объект».

Класс – это готовая программная структура, предназначенная для работы с какой-либо совокупностью взаимосвязанных элементов, с которой мы хотим работать как единым целым, т.е. **объектом**. Например, в VBA есть классы, обеспечивающие работу с такими объектами, как приложение Microsoft Excel (*Application*) рабочая книга (*Workbook*), рабочий лист (*Worksheet*), диапазон ячеек (*Range*). В VBA каждый элемент рабочего пространства MS Excel рассматривается как объект. Объектами могут быть области таблицы, листы, меню команд, графические объекты и т.д.

Объект может включать в себя множество других объектов, т.е. является **контейнером** для них. Самый крупный объект – приложение MS Excel. Это контейнер для всех остальных объектов. Он включает в себя рабочие книги, каждая из которых является контейнером для листов (рабочих, модульных и других), рабочие листы содержат диапазоны и т.д.

Каждый класс объединяет в себе так называемые свойства, методы и события.

Свойство – это характеристика объекта (параметр, характеризующий этот объект) или указатель на внутренний объект (например, указатель на диапазон рабочего листа). Например, свойством рабочего листа может быть его имя (*Name*), определенный диапазон ячеек, строк или столбцов (*Range-объекты*), свойством диапазона ячеек – шрифт (объект *Font*), заливка (объект *Interior*).

Методы – это процедуры, с помощью которых можно выполнять различные операции с объектом (например, удалить рабочий лист).

Событие – это воздействие на объект, для которого можно написать процедуру отклика (*процедуру обработки события*). Такие события могут возникать в результате действий пользователя (например, если пользователь выполнил щелчок кнопкой мыши или изменил что-либо на рабочем листе) или генерироваться программной системой (например, в результате ошибки). Как только возникает подобное событие, автоматически запускается на выполнение процедура его обработки. Если такая процедура не написана, система не будет реагировать на событие.

В текстовом редакторе VBE с помощью команды **View ⇒ Object Browser** (или клавишей **F2**) можно вывести на экран список всех классов и предусмотренных для них свойств, методов и событий, которые возможны для соответствующих объектов.

Любой класс можно создать на основе одного из имеющихся с помощью изменения его свойств и методов или добавления новых. Новый класс-потомок *наследует* все свойства и методы класса-предшественника (родительского класса). В любом объектно-ориентированном языке задан так называемый *базовый класс*, который не имеет предшественника. С него и начинается создание всех остальных классов. Наследование свойств и методов, в частности, означает, что одинаково называемые методы, применяемые к разным объектам, могут приводить к разным результатам.

12.2 Объекты

В связи особенностями классов (методы и свойства являются его элементами) операторы применения действий к объектам имеют специальную структуру: в таком операторе сначала указывается объект и только после него – действие, которое необходимо с ним совершить. Действие

может быть задано применением метода или изменением значения свойства. В качестве разделителя между ними указывается точка. Например,

```
Sheets(1).Name = "Пример"  
Sheets(1).Delete
```

Здесь в обоих операторах объектом является первый рабочий лист. В первом операторе этому листу дается имя *Пример* (действие заключается в присваивании текстовой константы *Пример* свойству *Name*). Во втором операторе к этому объекту применяется метод удаления *Delete*.

Если необходимо выполнить действие с объектом, который находится в другом объекте, а тот – в третьем и т.д. (например, ячейка находится на рабочем листе определенной рабочей книги), то в операторе первым указывается самый внешний объект, а затем через точку перечисляются остальные по мере их вложенности (рабочая книга, затем – лист, последний объект – ячейка). Если в операторе объект не указан, имеется в виду активный, т.е. тот, к которому было применено последнее действие. Например,

```
Sheets(1).Range("B2:B6").Select
```

Здесь к объекту *Range("B2:B6")* (диапазон ячеек от *B2* до *B6*), который находится в объекте *Sheets(1)* (первый лист) активной рабочей книги (она не указана) применяется метод *Select* (выделить).

Ссылки на объекты в операторах можно указывать с использованием переменных, которые называются *объектными*. Для этого предусмотрен специальный тип *Object*, с помощью которого можно определить такую переменную в операторе *Dim*. Если необходима переменная для объектов определенного вида, то в операторе *Dim* она может быть определена с помощью соответствующего типа, например,

Workbook – рабочая книга,
Worksheet – рабочий лист,
Range – область рабочего листа.

Поскольку объекты – это не просто данные (числа, тексты и т.д.), к ним нельзя применять обычные операторы сравнения и присваивания. Для них в VBA имеются специальные операторы: *Set*, с помощью которого можно присвоить объектной переменной какое-либо значение, имеющее смысл ссылки на объект, и *Is* – сравнение на эквивалентность. Например,

```
Dim MyObject As Object  
Set MyObject = Workbooks("Книга1").Sheets(1).Range("B2:B6")
```

MyObject. Select

Здесь в операторе **Dim** определена переменная *MyObject* типа **Object**, во втором операторе этой переменной присвоено значение – ссылка на объект в виде диапазона ячеек *B2:D6* первого листа рабочей книги *Книга1*, а в третьем операторе эта область выделена.

12.3 Коллекции

Есть объекты, которые существуют только вместе с другими аналогичными. Совместно эти объекты образуют так называемую **коллекцию (набор)**. Все коллекции имеют системные имена. Например,

Workbooks	– <i>открытые</i> рабочие книги;
Sheets	– <i>все листы</i> открытой рабочей книги;
Worksheets	– <i>рабочие листы</i> рабочей книги;
Charts	– <i>листы диаграмм</i> рабочей книги;
DialogSheets	– <i>диалоговые листы</i> рабочей книги.

Все элементы коллекций имеют имена, которые можно изменять. Каждый элемент коллекции идентифицируется индексом либо своим именем (в кавычках), указываемом в круглых скобках после имени коллекции (например, *Sheets(3)* – третий среди всех листов, *Sheets("Лист3")* – лист с именем *Лист3*).

Есть коллекции, которые являются подмножествами других коллекций (например, **Worksheets**, **Charts** и **DialogSheets** являются непересекающимися подмножествами коллекции **Sheets**). В этом случае один и тот же объект может быть одновременно элементом нескольких коллекций. Например, если в книге четыре листа *Лист1*, *Лист2*, *Лист3*, *Лист4*, причем *Лист1* – лист для диаграммы, а остальные – рабочие, то *Лист3* в операторе можно указать по-разному:

Sheets("Лист3"), *Sheets(3)*, *Worksheets("Лист3")*
или *Worksheets(2)*

Напоминаю, что если объект содержится в другом объекте, то задать его нужно в виде цепочки вложенных объектов от внешнего к внутреннему, разделяя их точкой. Например, если нужно указать *Лист1* конкретной рабочей книги *Книга2*, то необходимо написать

Workbooks("Книга2").Sheets("Лист1")

12.4 Управление объектами и коллекциями

12.4.1 Операторные скобки With – End With

В случае, если необходимо произвести несколько действий с одним и тем же объектом, этот объект и применяемые к нему действия можно поместить между так называемыми *операторными скобками* **With** и **End With**

```
With <объект>  
    <действие 1>  
    <действие 2>  
    .....  
    <действие N>
```

End With

Эта конструкция позволяет сократить текст и сделать его более ясным. Например,

```
With Sheets(1).Range("B2:B6")  
    .Clear  
    .Select  
    .Name = "МоиДанные"
```

End With

Здесь к области *B2:B6* первого рабочего листа применено три действия: удаление содержимого в ячейках (метод **Clear**), выделение области (метод **Select**) и присваивание ей имени *МоиДанные* (свойство **Name**).

12.4.2 Цикл For Each – Next

Конструкция **For Each – Next** используется, когда надо применить последовательность операторов ко всем элементам коллекции.

```
For Each <элемент> In <коллекция>  
    <оператор 1>  
    <оператор 2>  
    .....  
    <оператор N>  
Next [<элемент>]
```

Здесь <коллекция> – коллекция, к элементам которой надо применить операторы; <элемент> – имя объектной переменной, которая будет использоваться в операторах для ссылки на очередной элемент коллекции.

Например,

```
Dim ItemSheet As Worksheet
For Each ItemSheet In Worksheets
    ItemSheet.Range ("A1") = ItemSheet.Name
Next
```

Здесь сначала с помощью **Dim** определяется объектная переменная типа **Worksheet** (рабочий лист), а затем в ячейку **A1** каждого рабочего листа введено его имя.

12.5 Свойства объектов

Объекты обладают свойствами, значения которых можно прочитать или изменить. Изменение значения свойства описывается с помощью присваивания. Для некоторых свойств значения задаются пользователем по своему усмотрению, например,

```
Sheets(1).Name = "Главный".
```

Для других свойств возможные значения уже заданы и пользователю остается лишь выбрать одно из них. Например, для горизонтального центрирования значений в ячейках диапазона **B2:D6** следует присвоить значение **xlCenter** свойству **HorizontalAlignment**, т.е.

```
Range("B2:B6").HorizontalAlignment = xlCenter
```

Есть свойства, которые указывают на объект в другом объекте. Например,

ActiveWorkbook	– активная рабочая книга;
ActiveSheet	– активный лист в коллекции листов;
ActiveCell	– активная ячейка в области рабочего листа;
Selection	– выделенная область рабочего листа;
Font	– шрифт в диапазоне ячеек;
Interior	– заливка в диапазоне ячеек.

Такие свойства можно указывать в операторах в качестве объектов. Например, в операторе

```
Worksheets(1).ActiveCell.Value = 500
```

свойству **Value** активной ячейки первого рабочего листа присваивается значение 500, в операторе

Worksheets(1).ActiveCell.Font.Size = 14,
для активной ячейки первого рабочего листа устанавливается размер шрифта 14 пт (свойство **Size** объекта **Font**).

С помощью свойств можно изменить и другие особенности шрифта: **Name** (тип, например, Arial Суг), **FontStyle** (стиль, например, полужирный), **Underline** (тип подчеркивания), **Color** (цвет) и т.д.

Свойства могут иметь аргументы. К таким свойствам относятся те, которые указывают на элементы объектов в виде внутренних объектов. Например, **Range** – свойство рабочего листа, которое задает диапазон ячеек для совершения с ним каких-либо действий. Например,

Range("B2:D6")

задает объект в виде диапазона ячеек от B2 до D6. Теперь этот объект является элементом рабочего листа, к которому можно применить какое-либо действие. Разумеется, как и любой объект, он обладает своими свойствами, которые можно изменить или прочитать. Например, с помощью его свойства **Value** во все ячейки этого диапазона можно поместить значение 12.4

Range("B2:D6").Value = 12.4

Многие подобные объекты (объекты, которые указываются с помощью свойств) имеют свойство, которое можно не указывать. Например, для объектов, указанных с помощью **Range**, таким свойством является **Value**. Поэтому последний оператор можно записать проще:

Range("B2:D6") = 12.4

12.6 Применение методов к объектам

Методы определяют те действия, которые можно выполнить с объектом. Например,

Select – выделяет объекты в виде области рабочего листа.

Методы связаны с определенными объектами, поэтому выполняемые ими действия могут отличаться в зависимости от объекта, к которому они применяются. Например,

Delete – удаляет объект, если он может быть удален, например для удаления листа *Лист1* в рабочей книге *Книга2* можно написать оператор

Workbooks("Книга2").Sheets("Лист1").Delete

или, если рабочая книга *Книга2* активная,

Worksheets("Лист1").Delete

Если объект не может быть удален, этот метод выполняет действия в зависимости от объекта, к которому применяется. Например, применительно к столбцу рабочего листа этот метод приводит к перемещению влево содержимого ячеек всех столбцов, расположенных справа от него.

Все аргументы методов имеют имена. Какие-то из аргументов обязательны для указания, другие можно не указывать. Аргументы указываются списком через пробел после имени метода двумя способами.

Первый способ называется **обращением по имени** (*by-name syntax*). В этом случае аргументам присваиваются определенные значения. При этом, знаком присваивания является равенство с двоеточием (:=). Такой знак (с двоеточием) используется только для задания аргументов метода:

`<метод> <arg_1> := <знач_1>, ..., <arg_N> := <знач_N>`

При использовании этого способа аргументы можно указывать в любом порядке. Необязательные аргументы можно не указывать.

Второй способ называется **обращением по порядку** (*by-position syntax*). В этом случае в списке указываются только значения аргументов, причем в строго определенном порядке, предусмотренном для данного метода:

`<метод> <знач_1>, ..., <знач_N>`

Значения необязательных аргументов могут отсутствовать, но позиции *пропущенных* значений должны быть обязательно обозначены разделителем списка. Последние неуказанные аргументы никак не отмечаются. Например, если для метода предусмотрено пять аргументов, причем второй из них обязательный и мы хотим еще указать четвертый, то спецификация будет иметь следующий вид:

`<метод> , <arg_2> , , <arg_4>`

Рассмотрим метод **Add**.

Этот метод добавляет новый элемент в коллекцию. Он обеспечивает добавление элемента в коллекции разных типов. Имеет разные аргументы в зависимости от того, к какому объекту (коллекции) применяется. Если **Add** применяется к коллекции листов **Sheets**, то спецификация имеет следующий вид:

`Sheets.Add [before], [after], [count], [type]`

Этот метод добавляет в *Sheets* новый лист типа *type* в количестве *count* перед листом *before* или после листа *after*. Все аргументы этого метода являются необязательными. По умолчанию новый лист добавляется перед активным, а *count* равен единице. Для аргумента *type* предусмотрены следующие значения:

- xlWorkSheet – рабочий лист,
- xlChart – лист для диаграммы,
- xlDialogSheet – диалоговый лист,
- xlModule – модульный лист.

По умолчанию этот аргумент имеет значение *xlWorksheet*.

Например, оператор добавления нового рабочего листа перед листом *Лист2* в активную рабочую книгу может иметь следующий вид

```
Sheets.Add Before := Sheets("Лист2")
```

или

```
Sheets.Add Sheets("Лист2")
```

Методы можно использовать не только для выполнения действий, но и для указания на объект, с которым необходимо произвести следующее действие. В этом случае список аргументов, если он необходим, надо поместить в круглые скобки (без пробела после имени метода).

Например,

```
Workbooks.Add.Sheets.Add(Sheets(2)).Name = "НовыйЛист"
```

Здесь в одном операторе описана цепочка, включающая три действия: создание новой рабочей книги (добавление нового элемента в набор *Workbooks*), добавление в нее нового рабочего листа (добавление нового элемента в набор *Sheets*) перед вторым листом и присваивание ему имени *НовыйЛист*.

Другой пример:

```
Workbooks.Add.Sheets.Add.Range("B2:D8"). _  
Interior.Color = 49407
```

Здесь создается новая рабочая книга, в нее добавляется новый рабочий лист, в области *B2:D8* которого устанавливается цвет заливки (свойство *Color* объекта *Interior*).

12.7 Объекты Range

Объекты Range (*Range-объекты*) – это области рабочего листа. Такой областью может быть ячейка, диапазон ячеек, строк или столбцов и даже вся область рабочего листа. Это один из самых важных типов объектов MS Excel.

12.7.1 Указание Range-объекта в операторе

Указать Range-объект в операторе можно по-разному. Обычно это делается с помощью свойств объекта *WorkSheet* (рабочий лист): *ActiveCell* (активная ячейка), *Selection* (выделенная область), *Range* (диапазон ячеек, строк или столбцов), *Columns* (столбец), *Rows* (строка) и *Cells* (ячейка).

С *ActiveCell* и *Selection* мы уже знакомы (см. п. 12.5), обращение к остальным свойствам оформляется следующим образом:

Range[(*<ссылка>*)] – указывает на область, заданную ссылкой. Это самый универсальный способ. Областью может быть ячейка (например, *Range*("B2")), диапазон ячеек (например, *Range*("B2:D8")), строк (например, *Range*("2:4")) или столбцов (например, *Range*("B:D")). Допустимо указывать и несколько областей, например, *Range*("A4:C8, D6:F10");

Rows[(*<ссылка>*)] – указывает на строку (например, *Rows*(4)) или диапазон строк (например, *Rows*("2:4"));

Columns[(*<ссылка>*)] – указывает на столбец или диапазон столбцов. В ссылке могут быть номера столбцов или их имена. Например, *Columns*("C") или *Columns*(3) указывают на один и тот же столбец;

Cells[(*<arg_1>*,*<arg_2>*)] – указывает на ячейку, находящуюся на пересечении строки и столбца, номера которых заданы соответственно *<arg_1>* и *<arg_2>*. Эти аргументы могут представлять собой арифметические выражения. В качестве *<arg_2>* для задания столбца можно указывать не только номер, но и его имя. Например, для выделения ячейки B3 можно написать

```
Cells(3, 2).Select  
или Cells(3, "B").Select.
```

Ссылки на области могут быть заданы переменными. Для свойств *Rows*, *Columns* и *Cells* можно не указывать аргументы (вместе со скобками).

ми). В этом случае имеются в виду все строки, столбцы или ячейки соответственно. Например,

`Cells.Select` – выделяет всю область активного рабочего листа.

Для указания аргумента свойства **Cells** можно использовать сквозную нумерацию ячеек на рабочем листе. Поскольку на рабочем листе 256 столбцов, то ячейке *A2*, например, будет соответствовать номер 257, т.е.

`Cells(257)`

указывает на ячейку *A2*. Такая возможность бывает удобна, например, если надо организовать циклическое применение действий ко всем ячейкам какого-либо диапазона.

Совместно свойства **Range** и **Cells** позволяют задать область с помощью вычисляемых ссылок. Для этого применяется другой вид записи **Range**:

```
Range(Cells(<arg_1>,<arg_2>), Cells(<arg_3>,<arg_4>))
```

Здесь `Cells(<arg_1>,<arg_2>)` определяет начало области, а `Cells(<arg_3>,<arg_4>)` – ее окончание. Например,

```
Range(Cells(3, 2), Cells(5, 4))
```

указывает на область *B3:D5*.

Следующая подпрограмма обеспечивает выделение области, заданной формальными параметрами, и выравнивание содержимого ячеек в ней по центру:

```
Sub SelectRange(R_1, C_1, R_2, C_2)
  With Range(Cells(R_1, C_1), Cells(R_2, C_2))
    .Select
    .HorizontalAlignment = xlCenter
  End With
End Sub
```

Здесь с помощью `Cells(R_1, C_1)` задана ячейка с координатами *R_1* (номер строки) и *C_1* (номер или имя столбца), с которой начинается область, а с помощью `Cells(R_2, C_2)` – ячейка с координатами *R_2* и *C_2*, которой эта область заканчивается.

12.7.2 Свойства Range-объекта

Для Range-объекта предусматривается множество свойств. Среди них есть свойства, значения которых можно только прочитать (например, **Address** – адрес ячейки, **Count** – количество элементов в диапазоне), а

есть и такие, для которых значения может задать сам пользователь. К ним, в частности, относятся

Name – имя, например,

Selection.Name = "ИсхДанные"

Value – значение, например,

ActiveCell.Value = 43

Formula – содержимое в A1-формате, например,

Range("B10").Formula = "= B3^2"

FormulaR1C1 – содержимое в R1C1-формате, например,

Range("B10").FormulaR1C1 = "= R3C2^2"

ColumnWidth – размер столбцов, например,

Columns("B:D").ColumnWidth = 10

RowHeight – размер строк, например,

Rows("2:4").RowHeight = 14

Если ссылка на Range-объект получена с помощью свойства **Range**, то **Value**, **Formula** и **FormulaR1C1** можно не указывать, например,

Range("B2") = 43

Range("A1") = "Пример"

Range("B10") = "= B3^2"

Range("B10") = "= R3C2^2"

Range("D10") = "= СУММ(D1:D9)"

Range-объект, как и рабочий лист, обладает свойствами **Range**, **Rows**, **Columns** и **Cells**, которые позволяют указать диапазоны (Range-объекты) внутри него. При этом следует учитывать, что это самостоятельный объект, состоящий из ячеек, строк и столбцов, поэтому в нем своя (внутренняя) адресация этих элементов. В связи с этим внутренние Range-объекты будут определяться относительно левой верхней ячейки внешнего Range-объекта. Например, в результате выполнения оператора

Range("B2:D6").Range("B2").Select

будет выделена ячейка рабочего листа C3. Оператор

Range("B3:F9").Rows(4).Select

выделяет 4-ую строку в области B3:F9, то есть диапазон рабочего листа B6:F6. Оператор

Range("B3:F9").Cells(8).Select

выделяет ячейку D4. Оператор

Columns("C").Cells(4).Select

выделяет ячейку C4. Оператор

```
Range("B3:F9").Columns(3).Select
```

выделяет 3-й столбец в области B3:F9, то есть диапазон рабочего листа D3:D9.

Рассмотрим еще одно полезное свойство Range-объекта – **Offset**, которое позволяет указать ячейку относительно левой верхней ячейки Range-объекта:

```
<объект>. Offset(<смещ_стр>, <смещ_столбца>)
```

Первый аргумент (<смещ_стр>) задает смещение строки, а второй аргумент (<смещ_столбца>) – смещение столбца. Например,

```
ActiveCell. Offset(1, 0).Value = 4
```

```
Range("B3:F9").Offset(-1, 0).Value = 12
```

Первый оператор означает, что в ячейку, находящуюся под активной ячейкой, следует ввести число 4. В результате выполнения второго оператора число 12 будет введено в ячейку B2.

Это свойство особенно эффективно при использовании в цикле, когда действия необходимо выполнить с множеством ячеек, расположение которых определяется относительно начальной позиции на рабочем листе.

Есть свойства, для которых допустимые значения уже заданы, и пользователь может лишь выбрать какое-либо из них. К таким свойствам, например, относятся

HorizontalAlignment – тип выравнивания по горизонтали,

VerticalAlignment – тип выравнивания по вертикали,

WrapText – перенос слов.

Для **HorizontalAlignment** предусмотрены следующие значения:

xlLeft – по левому краю,

xlRight – по правому краю,

xlCenter – по центру,

xlCenterAcrossSelection – по центру объединенной области нескольких ячеек.

Например,

```
Range("C2:E2").HorizontalAlignment = xlCenterAcrossSelection
```

```
Range("B2:B6;C3:E6").HorizontalAlignment = xlCenter
```

Для **VerticalAlignment**:

xlTop – по верхнему краю,

xlBottom – по нижнему краю,
xlCenter – по центру.

Свойство **WrapText** позволяет установить (или отменить) перенос слов. Для него предусмотрено два значения: **True** (установить) и **False** (отменить).

Кроме свойств, значения которых можно изменить, есть и такие, которые предназначены только для чтения. К ним относятся, например, **EntireColumn** и **EntireRow**, которые указывают соответственно на все столбцы и строки рабочего листа, ячейки которых принадлежат Range-объекту. Например, в результате выполнения операторов

```
Range("B2:D6").EntireColumn.ColumnWidth = 10
```

```
Range("B2:D6").EntireRow.RowHeight = 10
```

будет установлена ширина столбцов диапазона *B:D* в 10 символов и высота строк диапазона *2:6* в 10 пт.

К свойствам, значения которых можно лишь прочитать, относится и свойство **CurrentRegion**, указывающее на прямоугольный диапазон ячеек, окруженный пустыми строками и столбцами, внутри которого находится данный Range-объект. Например,

```
Range("B4").CurrentRegion.Select
```

выделяет область листа, внутри которой находится ячейка *B4*.

Для обрамления *ячеек внутри области* используется свойство **Borders**:

```
<область>.Borders(<аргумент>)
```

Тип линии определяется аргументом с помощью следующих значений:

xlLeft	– слева,
xlRight	– справа,
xlTop	– сверху,
xlBottom	– снизу,
xlInsideVertical	– между ячейками по вертикали,
xlInsideHorizontal	– между ячейками по горизонтали.

Линия обладает свойствами, значения которых определяют ее особенности. Например, тип линии устанавливается ее свойством **LineStyle** (xlContinuous – непрерывная, xlDouble – двойная, xlDash – прерывистая), толщина – свойством **Weight** (xlThin – тонкая, xlMedium – толстая), цвет – свойством **Color**.

```
Например, группа операторов
With Range("B2:D6").Borders(xlLeft)
    .LineStyle = xlDouble
    .Weight = xlThin
    .Color = -4165632
End With
```

проводит двойную тонкую линию синего цвета слева для каждой ячейки в области *B2:D6*.

Свойство **Borders** можно использовать и для проведения границ рядом с областью. В этом случае аргумент должен иметь значения: *xlEdgeLeft* (слева), *xlEdgeRight* (справа), *xlEdgeTop* (сверху) или *xlEdgeBottom* (снизу).

12.7.3 Методы Range-объекта

Range-объект обладает большим количеством методов, обеспечивающих выполнение с ним разнообразных действий, например, автозаполнение, копирование, перемещение, автоподгонку размеров строк и столбцов и многое другое. В приведенных выше примерах мы уже встречались с методами

- Select – выделение области,
- Clear – удаление содержимого в ячейках области,
- Delete – удаление области.

Рассмотрим еще некоторые полезные методы.

Для определения граничной непустой ячейки в столбце или строке относительно заданной ячейки предназначен метод **End**:

```
<ячейка>.End(<arg>)
```

Здесь

<ячейка> – ячейка, относительно которой должен осуществляться поиск непустой ячейки;

<arg> задает направление поиска. Для него предусмотрены значения: **xlDown** (вниз), **xlUp** (вверх), **xlToLeft** (влево), **xlToRight** (вправо).

Например,

```
Range(ActiveCell, ActiveCell.End(xlDown)).Select
```

выделяет диапазон от активной ячейки до последней непустой ячейки в том же столбце вниз.

Автозаполнение, копирование и перемещение содержимого области производится методом соответственно **AutoFill**, **Copy** и **Cut**.

<обл_1>.<метод> <обл_2>

Copy и **Cut** осуществляют соответственно копирование и перемещение из области <обл_1> в область <обл_2>. Например,

Range("B7").Copy Range("D7")

Для **AutoFill** продолжаемая область <обл_1> должна быть частью (началом) области <обл_2>, в которую продолжается. Например,

Range("B5").AutoFill Range("B5:B8")

Размер столбцов или строк можно устанавливать методом **AutoFit**, который выполняет автоподгонку. Например,

Columns("B:D").AutoFit

Range("B2:D6").EntireColumn.AutoFit

Оба эти оператора дают один и тот же результат: автоподгонку размера столбцов от *B* до *D*.

Объединение ячеек производится методом **Merge**. Например,

Range("B2:B3").Merge

Для проведения линий вокруг области удобно применять метод **BorderAround**:

<область>.BorderAround Weight := <значение>

Значение аргумента **Weight** определяет толщину линии (xlThin – тонкая или xlMedium – толстая). Например, оператор

Range("B2:D6").BorderAround Weight := xlMedium

проводит толстую линию вокруг области *B2:D6*.

12.7.4 Диалоговый ввод ссылки на Range-объект

Как уже упоминалось, MS Excel также является объектом (**Application**), к которому можно применять различные методы. Например, мы уже знаем функцию VBA **InputBox**, которая позволяет ввести какое-либо данное с помощью диалогового окна. Для объекта **Application** предусмотрен метод **InputBox**, используемый для аналогичной цели, но предоставляющий дополнительные возможности:

- возможность задать тип возвращаемого значения,
 - возможность указать диапазон листа путем выделения с помощью мыши или вводом в поле,
 - автоматическая проверка правильности введенных данных.
- Этот метод имеет следующую спецификацию:

InputBox(*Prompt* [, *Title*] [, *Default*] [, *Left*] [, *Top*] [, *HelpFile*],

HelpContextID] [, *Type*])

Здесь, в сравнении с функцией, добавлен аргумент *Type* – тип возвращаемого значения: 0 – формула, 1 – число, 2 – текстовая строка, 4 – логическое значение (True или False), 8 – ссылка на область рабочего листа, 16 – значение ячейки, 64 – массив значений области листа.

При использовании метода *InputBox* может возникнуть проблема, если пользователь закроет диалоговое окно кнопкой *Отмена*, тем самым отказавшись от ввода. Такое действие приведет к ошибке выполнения программы. Если Вы этого не хотите, следует перед оператором, в котором переменной присваивается значение, полученное с помощью *InputBox*, поместить оператор игнорирования ошибки

```
On Error Resume Next
```

В этом случае объектной переменной присваивается специальное значение *Nothing*, которое означает «отсутствие ссылки на объект», и программа выполняется дальше.

Например, получение от пользователя диапазона ячеек рабочего листа с помощью диалогового окна может быть оформлено следующим образом

```
Dim UsRange As Range
On Error Resume Next
Set UsRange = Application.InputBox( _
    Prompt := "Укажите диапазон для вычислений", _
    Title := "Ввод диапазона", Default := ActiveCell.Address, _
    Type := 8)
If UsRange Is Nothing Then End
```

Здесь оператором *Dim* определена *объектная* переменная (см. п. 12.2) *UsRange* типа *Range* (область рабочего листа). Второй оператор защищает от остановки выполнения программы, если пользователь откажется от ввода кнопкой *Отмена*. Затем с помощью оператора *Set* (присваивание значений объектным переменным, см. п. 12.2) этой переменной присваивается ссылка на область рабочего листа, полученная с помощью *метода InputBox*. В последнем операторе проверяется (с помощью *If*): не является ли переменная *UsRange* «пустой», т.е. ее значение равно (*Is*, см. п. 12.2) специальному значению *Nothing*. И, если это так (*Then*), выполнение программы прекращается (*End*).

Кстати, обратите внимание, что для *Default* (размещение значения «по умолчанию» в поле ввода) мы использовали свойство *Address* (адрес) объекта *ActiveCell* (активная ячейка).

12.8 Автоматическое создание программ

Тексты программ (макросов) можно создавать в автоматическом режиме. Процесс автоматического создания макроса состоит из трех этапов.

Прежде всего, необходимо активизировать режим автоматического создания макроса командой

Разработчик* ⇒ *Запись макроса

(кнопка ) и в появившейся диалоговой панели *Запись макроса* определить его имя (в поле *Имя макроса*) и другие параметры (комментарий в поле *Описание* и сочетание клавиш клавиатуры для быстрого вызова). Кроме этого, следует указать параметр *Сохранить в*, определив место расположения создаваемого макроса. Если указать *В этой книге*, то макрос будет записан в автоматически созданный модульный лист и созданный макрос будет доступен только в том случае, если данная книга открыта. Если указать *Личная книга макросов*, то он будет доступен при каждой загрузке MS Excel.

На втором этапе следует выполнить все действия, которые необходимо записать в макрос (команды меню, выделение ячеек, ввод данных и формул и т.д.). При этом надо иметь в виду, что в макрос включаются *все* выполняемые действия, т.е. следует внимательно следить за тем, чтобы не выполнять лишние операции. Лучше всего заранее подготовить сценарий и руководствоваться им.

Завершение процесса выполняется командой

Разработчик* ⇒ *Остановить запись

(кнопка ) , которая автоматически заменяет команду *Запись макроса* (кнопку ) после ее выбора.

Автоматическое создание текста макроса, конечно, удобный способ. Однако, необходимо иметь в виду, что в этом случае получается слишком большой, далекий от оптимального, текст. Это может быть связано с разными причинами, например,

1) в результате изменения параметров (например, шрифта) в диалоговом окне в текст макроса будут вставлены соответствующие операторы

для всех параметров, в том числе и тех, значения которых Вы не изменяли;

2) если один и тот же эффект может быть достигнут несколькими способами, в текст вставляется определенный оператор (или даже последовательность операторов), что может оказаться не эффективным;

3) в текст вставляются операторы, соответствующие всем выполняемым действиям, в том числе таким, которые можно и не описывать в программе (например, выделение ячеек перед вводом в них данных).

Поэтому после автоматического создания текста макроса, его необходимо оптимизировать вручную.

Например, активизируйте команду *Разработчик* ⇒ *Запись макроса* (кнопка ) и выполните следующие действия:

- задайте имя макроса (например, *AutoText*);
- выделите ячейку B2 и введите в нее текст *Фирма "Привет"*;
- введите в ячейку C4 число *12.5*;
- введите в ячейку D6 формулу $=C8^2$;
- завершите процесс командой *Разработчик* ⇒ *Остановить запись* (кнопка )

В результате этих действий в модульном листе появится текст

```
Sub AutoText()  
,  
,  
' AutoText Макрос  
,  
,  
    Range("B2").Select  
    ActiveCell.FormulaR1C1 = "Фирма ""Привет"""  
    Range("C4").Select  
    ActiveCell.FormulaR1C1 = "12.5"  
    Range("D6").Select  
    ActiveCell.FormulaR1C1 = "=R[-2]C[-1]^2"  
    Range("D7").Select  
End Sub
```

В этом тексте сначала указаны строки с комментариями. Удалим их.

Первая пара операторов описывает ввод текстовой константы. Поскольку вручную Вы выполнили два действия, то и в программе этому соответствует два оператора: выделение (с помощью метода *Select*) ячеек-

ки *B2* (на нее указывает *Range*) и присваивание свойству *FormulaR1C1* (содержимое ячейки в R1C1-формате) текстовой константы. Этот же результат можно получить, если оставить только присваивание ячейке *B2* текстовой константы (см. п. 2.7.2), т.е.

```
Range("B2") = "Фирма ""Привет"""
```

Кстати, обратите внимание, как обозначены внутренние кавычки в этой текстовой константе.

Вторая пара операторов появилась в результате ввода числовой константы. Заметим, что она оформлена как текстовая. Именно так происходит в автоматическом режиме: все вводимые данные оформляются в форме текстовых констант, которые присваиваются свойству *FormulaR1C1* (содержимое ячейки в R1C1-формате). В данном случае, поскольку константа все-таки числовая, во время выполнения программы будет производиться автоматическое преобразование текстовой константы в числовую. Это лишнее действие, поэтому кавычки лучше удалить:

```
Range("C4") = 12.5
```

В следующей паре операторов описан ввод формулы. Убрав лишнее, получаем

```
Range("D6") = "=R[-2]C[-1]^2"
```

Заметим, что в автоматическом режиме ссылки на ячейки в формуле оформляются в формате R1C1. Если Вам удобнее формат A1, это можно исправить:

```
Range("D6") = "= C4^2"
```

В итоге получаем следующий текст:

```
Sub AutoText()  
    Range("B2") = "Фирма ""Привет"""  
    Range("C4") = 12.5  
    Range("D6") = "= C4^2"  
End Sub
```

ЛИТЕРАТУРА

1. *Слепцова Л.Д.* Программирование на VBA в Microsoft Excel 2010. М. : ООО "Вильямс", 2010. – 432 с.
2. *Кузьменко В.Г.* VBA. Эффективное использование. М. : «Бином. Лаборатория знаний», 2015. – 624 с.
3. *Лебедев В.М.* Программирование на VBA в MS Excel. Учебное пособие для академического бакалавриата. М. : Юрайт, 2017. – 272 с.
4. *Уокенбах Джон.* Excel 2013: Профессиональное программирование на VBA. М. : Диалектика/Вильямс, 2014. – 960 с.
5. *Уокенбах Джон.* Microsoft Excel 2013: Библия пользователя. М. : Диалектика/Вильямс, 2015. – 928 с.

ТЕКСТОВЫЙ РЕДАКТОР VBE

Ввод и редактирование текстов процедур осуществляется под контролем специального *текстового редактора Visual Basic Editor (VBE)*.

Для краткости изложения в тексте используются следующие обозначения основных операций, выполняемых мышью:

CL – поместить указатель мыши на элемент и выполнить один щелчок *левой* кнопкой мыши (*Click Left*);

CR – поместить указатель мыши на элемент и выполнить один щелчок *правой* кнопкой мыши (*Click Right*);

DC – поместить указатель мыши на элемент и выполнить *двойной* щелчок *левой* кнопкой мыши (*Double Click*);

DD – операция “тащить и бросать” (*Drag and Drop*), что означает: установить указатель мыши на элемент, нажать на *левую* кнопку мыши, передвинуть мышь (элемент синхронно будет передвигаться на экране) и отпустить кнопку мыши (новое положение элемента зафиксировается).

Для сокращения записи команд используется следующая форма:

Лента ⇒ *Группа* ⇒ ... ⇒ *Команда*

Например, если требуется выбрать команду *Visual Basic* из группы *Код* ленты (вкладки) *Разработчик*, то соответствующая запись будет выглядеть так

Разработчик ⇒ *Код* ⇒ *Visual Basic*

А.1 Основные особенности

VBE можно запустить из рабочей книги. Для этого необходима лента (вкладка) *Разработчик* главного меню команд Excel. Для ее отображения следует: вывести на экран окно настройки главного меню команд Excel (выбрав команду *Настройка ленты* в окне *Файл* ⇒ *Параметры* или в контекстном меню для области главного меню) и в категории *Настройка ленты* установить флажок *Разработчик* в списке *Основные вкладки*.

Для установки уровня безопасности, разрешающего выполнение всех макросов, следует выбрать команду

Разработчик ⇒ *Код* ⇒ *Безопасность макросов*

и установить переключатель *Включить все макросы* в группе *Параметры макросов*.

Переход в окно VBE осуществляется командой
Разработчик ⇒ Код ⇒ Visual Basic
В результате, на экране появится программное окно (рис. 24)

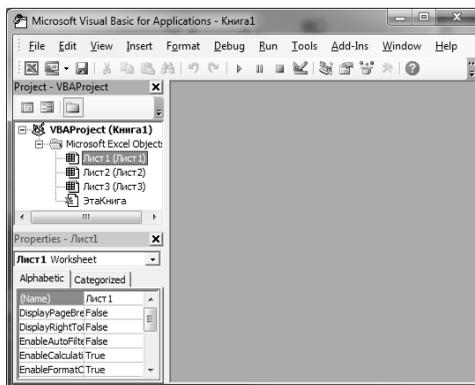


Рис. 24. Окно текстового редактора *Visual Basic Editor (VBE)*.

В окне VBE предусматривается несколько окон. В основном мы работаем с тремя из них: окно кода (**Code**), окно проектов (**Project**) и окно свойств (**Properties**). В окне кода осуществляется ввод и редактирование текста процедур, в окне проектов отображается древовидная структура всех открытых рабочих книг (каждая из них в VBA называется проектом), а в окне **Properties** – свойства, характеризующие элементы, отображенные в окне проектов (например, имя листа). Наличие или отсутствие этих окон определяется соответствующими командами меню **View (Code, Project Explorer и Properties Window)**.

Процедуры содержатся в модулях. Для добавления модуля в рабочую книгу следует в окне проектов **Project** (рис. 25)

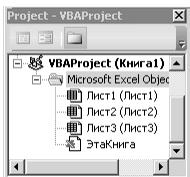


Рис. 25. Окно проектов **Project**.

выбрать команду **Insert** ⇒ **Module** в контекстном меню для рабочей книги (рис. 26)

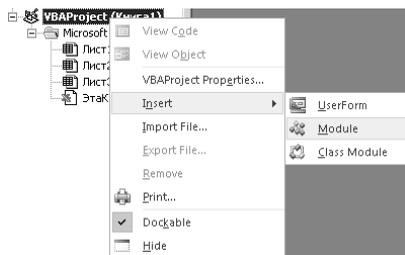


Рис. 26. Добавление модульного листа.

Процедуры могут располагаться как в одном модуле, так и в разных. Для удаления модуля следует в окне **Project** вывести для него контекстное меню и выбрать команду **Remove...** (рис. 27).

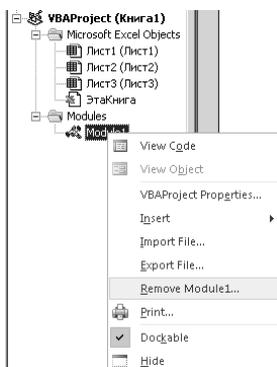


Рис. 27. Удаление модульного листа.

Для перехода в модуль достаточно применить **ДС** к его имени в окне **Project**.

А.2 Ввод и редактирование текста

Текстовый редактор VBE помогает синтаксически правильно оформить программный текст. В частности, при переходе к следующей строке автоматически производится

- преобразование некоторых букв, составляющих ключевые слова, в прописные;
- установка пробелов между частями оператора;
- установка отступа от начала строки, аналогичного предыдущей строке.

В некоторых случаях VBE производит автоматическое добавление фрагментов текста. Например, начиная текст процедуры, можно лишь указать *sub* и имя процедуры. После нажатия на клавишу <Enter>, мы увидим, что VBE автоматически добавил скобки после имени процедуры, пустую строку и последний оператор *End Sub* (оператор окончания процедуры).

В процессе редактирования возможна как вставка, так и замена знаков. В первом случае текстовый указатель изображается мигающей чертой между знаками, а во втором – мигающий текстовый указатель полностью накладывается на текстовый знак. Для изменения режима “вставка/замена” следует щелкнуть по клавише <Insert> (или <Ins>).

Для удаления текстового знака, находящегося слева от текстового указателя, используется клавиша <Backspace>, а справа от него – <Delete>. Если воспользоваться ими при нажатой клавише <Ctrl>, произойдет удаление *части слова* соответственно слева или справа от текстового указателя.

Перемещаться в тексте можно как мышью (щелчком левой кнопкой), так и клавишами управления указателем. Для быстрого перемещения (например, через слово, через процедуру) используются эти же клавиши, но при нажатой клавише <Ctrl>:

- <Ctrl>+<←> – перед словом;
- <Ctrl>+<→> – после слова;
- <Ctrl>+<↑> – начало текущей процедуры;
- <Ctrl>+<↓> – начало следующей процедуры;
- <Ctrl>+<Home> – начало текста модуля;
- <Ctrl>+<End> – конец текста модуля.

А.3 Выделение, удаление, перемещение и копирование

Перед выполнением действий с текстовым фрагментом необходимо его **выделить**. В качестве такого фрагмента может быть часть слова, слово, строка, текст процедуры и т.д. Для выделения текстового фрагмента, в основном, используется мышь. При этом применяются следующие две группы способов.

Первая группа (указатель мыши расположен слева от текста и изображен стрелкой с наклоном вправо ) предназначена для выделения фрагментов, состоящих из полных строк (таблица 1).

Таблица 1

Операции выделения фрагментов, состоящих из целых строк

Фрагмент	Операция
Строка	CL
Весь текст процедуры	DC
Весь документ	<Ctrl>+CL
От выделенной области до местоположения указателя мыши	<Shift>+CL
Группа подряд расположенных строк	DD

Вторая группа (указатель мыши расположен внутри фрагмента и изображен вертикальной чертой) предназначена для выделения фрагментов, состоящих из знаков или слов (таблица 2).

Таблица 2

Операции выделения фрагментов, состоящих из знаков или слов

Фрагмент	Операция
Слово	DC
От текстового указателя до местоположения указателя мыши	<Shift>+CL
От любой позиции до любой позиции	DD

Кроме мыши, для выделения фрагментов можно использовать и клавиатуру. При этом, началом фрагмента является положение текстового указателя или ранее выделенная часть текста, т. е. в процессе выделения клавиатурой размер фрагмента можно изменить. Для такого действия применяются клавиши управления указателем.

Если использовать клавиши управления указателем при нажатой клавише **<Shift>**, то выделение фрагмента изменяется соответственно перемещению текстового указателя. Клавиши со стрелками изменяют размер выделенного фрагмента на один знак (влево, вправо) или на одну строку (вверх, вниз), **<Home>** или **<End>** – соответственно до начала или окон-

чания строки, а <Page Up> или <Page Down> на одну страницу соответственно назад или вперед.

Можно изменять размер выделенного фрагмента **крупными блоками**. Для этого также используются клавиши управления указателем, но при одновременно нажатых *двух клавишах* <Ctrl> и <Shift>. Тогда клавиши <←> и <→> изменяют фрагмент на одно слово, клавиши <↑> и <↓> – соответственно до начала текущего или начала следующего текста процедуры, а <Home> и <End> – до начала или окончания всего модуля. Для выделения всего модуля достаточно выполнить <Ctrl>+<A> (латинская буква A).

Отмена выделения производится операцией **CL** в любом месте документа или любой клавишей управления указателем.

Для удаления выделенного фрагмента достаточно щелкнуть по клавише <Delete> () клавиатуры или воспользоваться командой *Edit ⇒ Clear*.

Перемещение выделенного фрагмента осуществляется применением к нему **DD**, копирование – этой же операцией при нажатой клавише <Ctrl>. Указатель мыши при этом должен быть изображен стрелкой с наклоном влево (). Во время копирования или перемещения при выполнении операции **DD** синхронно с указателем мыши передвигается изображение текстового указателя (вертикальная черта). Ее положение подсказывает место, в которое будет производиться вставка копируемого или перемещаемого фрагмента. Знак плюс справа от указателя мыши () подсказывает, что производится копирование. Отсутствие этого знака указывает на выполнение перемещения.

Можно переместить (“*вырезать*”) или скопировать выделенный фрагмент в *Буфер обмена* Windows, а затем содержимое этого буфера вставить в то место, где находится текстовый указатель. Такое перемещение и копирование производится командами соответственно *Cut* (кнопка ) и *Copy* (кнопка ) меню *Edit*, а вставка содержимого буфера – командой *Paste*  этого же меню.

А.4 Запуск программ на выполнение

В главном меню команд имеется вкладка **Run**, в которой находятся команды управления запуском программ на выполнение.

Запуск программы на выполнение осуществляется командой

Run ⇒ RunSub,

клавишей <F5> или инструментальной кнопкой . В результате, автоматически запускается интерпретатор VBA и под его контролем выполняется программа, в которой находится текстовый указатель.

Выполнение программы можно прервать, не ожидая ее окончания, командой

Run ⇒ Break,

клавишей клавиатуры <Ctrl>+<Break> или инструментальной кнопкой . Это, например, может понадобиться, если Вы считаете, что программа выполняется слишком долго, или Вы видите, что полученные промежуточные результаты не верны.

Если во время выполнения программы интерпретатор обнаружил ошибку, он *приостанавливает* процесс и выводит соответствующее сообщение (рис. 28).

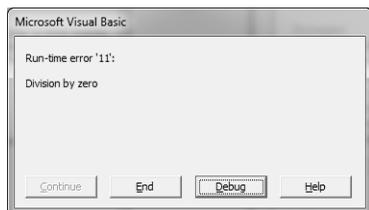


Рис. 28. Сообщение об ошибке выполнения программы.

Нажатие кнопки **End** приведет к прекращению выполнения программы, а если нажать кнопку **Debug**, интерпретатор возвращает пользователя в текстовый редактор и ожидает исправления этой ошибки. После этого команда **RunSub** приведет к продолжению выполнения программы, а не к запуску ее с самого начала. Чтобы заново *перезапустить* макрос с самого начала, следует перед командой **RunSub** выполнить команду

Run ⇒ Reset

или щелкнуть по инструментальной кнопке .

СПИСОК ОСНОВНЫХ ОПЕРАТОРОВ VBA

1. **Sub** <имя>([<список>]) – начало программы или подпрограммы. Здесь <список > – список переменных для приема входных данных и переменных для возвращения результатов выполнения подпрограммы. Для программы <список > не указывается.

2. **End Sub** – окончание программы или подпрограммы.

3. **Function** <имя>([<список>]) – начало функции. Здесь <список > – список переменных для приема входных данных.

4. **End Function** – окончание функции.

5. **Call** <имя>([<список>]) – вызов программы или подпрограммы.

6. **Оформление комментариев.** Комментарий указывается после оператора в той же строке или в отдельной строке. Перед ним должен быть указан знак апострофа ('). Если комментарий указан в отдельной строке, вместо апострофа можно использовать ключевое слово **Rem**.

7. <имя> = <выражение> – оператор присваивания. Здесь <имя> – идентификатор переменной, <выражение> – арифметическое, логическое или текстовое выражение, которое, в частности, может быть переменной или константой.

8. **Dim** <список> – оператор описания переменных. Здесь <список> – это список имен переменных и массивов. Для указания типа после имени в списке указывается ключевое слово **As** и обозначение типа: **Integer** – целое обычное; **Long** – длинное целое; **Single** – вещественное с одинарной точностью; **Double** – вещественное с двойной точностью; **String** – строковый тип; **Object** – объектный тип.

9. **ReDim** <список>– оператор переопределения динамического массива, т.е. массива, объявленного в операторе **Dim** без указания размерности.

10. Цикл с постусловием **Do – Loop Until**.

Do

...

Loop Until <условие>

Операторы **Do** и **Loop Until** выполняют указанную между ними последовательность операторов до тех пор, пока логическое выражение <условие> равно **False**.

11. Цикл с постусловием *Do – Loop While*.

Do

...

Loop While <условие>

Операторы *Do* и *Loop While* выполняют указанную между ними последовательность операторов до тех пор, пока логическое выражение <условие> равно *True*.

12. Цикл с предусловием *While – Wend*.

While <условие>

...

Wend

Операторы *While* и *Wend* выполняют указанную между ними последовательность операторов до тех пор, пока логическое выражение <условие> равно *True*.

13. Цикл со счетчиком *For – Next*.

For <имя> = <начало> **To** <конец> [**Step** <шаг>]

...

Next [<список>]

Операторы *For* и *Next* повторяют выполнение указанной между ними последовательности операторов заданное число раз.

Здесь <имя> – имя переменной цикла, <начало> – ее начальное значение, <конец> – ее последнее значение, <шаг> – шаг изменения значений переменной цикла. Переменная цикла должна быть целого типа. Величины <начало>, <конец> и <шаг> могут быть заданы константами, переменными и арифметическими выражениями. Для *Next* можно указать список имен переменных циклов, вложенных друг в друга. Эти имена указываются в порядке от последнего к первому по вложенности циклов.

14. **GoTo** <метка> – переход на строку, обозначенную меткой. Такая метка может быть задана номером или именем, указанным в начале строки. Если метка задана номером, то после него должен быть указан пробел, а если именем, то – знак двоеточия.

15. Альтернативное ветвление

If <условие> **Then**

<группа 1>

[**Else**

<группа 2>]

End If

Здесь

<условие> – логическое выражение; <группа 1> – группа операторов, которые выполняются, если <условие> выполняется, т.е. результат выражения равен **True**; <группа 2> – группа операторов, которые выполняются, если <условие> не выполняется, т.е. результат выражения равен **False**;
End If – окончание ветвления.

Если <группа 1> состоит из одного оператора и <группа 2> также состоит из одного оператора, то оператор можно записывать в одной строке без указания **End If**:

If <условие> **Then** <оператор 1> [**Else** <оператор 2>]

Вместо этой конструкции можно использовать функцию **If**:

If (<условие>, <оператор 1>, <оператор 2>)

Эта функция возвращает результат, полученный с помощью <оператор 1>, если <условие> равно **True**, или результат, полученный с помощью <оператор 2>, если <условие> равно **False**.

16. Множественный выбор.

Этот оператор оформляется с помощью ключевых слов **Select Case** (описание проверяемого условия), **Case** (описание ветви), **Case Else** (ветвь «иначе») и **End Select** (окончание оператора множественного выбора).

Select Case <выражение>

Case <результат 1>
 <группа 1>

.....
Case <результат N>
 <группа N>

[**Case Else**
 <группа Else>]

End Select

Здесь после ключевых слов **Case** указываются возможные результаты выражения и в следующих за ними строках – группы операторов, которые должны при этом выполняться. **Case Else** используется, если надо выполнить группу операторов, когда полученный результат не совпадает ни с одним из предусмотренных в **Case**.

17. **Exit Sub** – прерывание программы или подпрограммы с возвращением в вызывающую процедуру.

18. **Exit Do** – прерывание циклов **Do–Loop Until** и **Do–Loop While**.

19. **Exit For** – прерывание цикла **For – Next**.

20. **End** – оператор прекращения выполнения вычислений.

СОДЕРЖАНИЕ

Предисловие	3
1 Основы алгоритмизации	5
1.1 Основные понятия	5
1.2 Основные алгоритмические структуры	8
2 Основные понятия языка VBA	12
3 Типы данных	15
4 Выражения	18
5 Процедуры VBA	21
5.1 Виды процедур	21
5.2 Вызов подпрограмм и функций	23
5.3 Встроенные функции VBA	25
6 Оператор присваивания	26
7 Объявление переменных	27
8 Диалоговый ввод и вывод данных с помощью встроенных функций	30
8.1 Диалоговый ввод данных	30
8.2 Диалоговый вывод данных	32
9 Организация цикла	34
9.1 Цикл с постусловием <i>Do – Loop Until</i>	35
9.2 Цикл с постусловием <i>Do – Loop While</i>	37
9.3 Цикл с предусловием <i>While – Wend</i>	37
9.4 Цикл со счетчиком <i>For – Next</i>	40
10 Операторы ветвления	43
10.1 Оператор безусловного перехода	43
10.2 Альтернативное ветвление <i>If – Then – Else</i>	43
10.3 Оператор множественного выбора	47
11 Операторы прерывания	49
12 Основы автоматизации проведения расчетов в Excel	50
12.1 Объектно-ориентированный принцип VBA	50
12.2 Объекты	51
12.3 Коллекции	53
12.4 Управление объектами и коллекциями	54
12.4.1 Операторные скобки <i>With – End With</i>	54
12.4.2 Цикл <i>For Each – Next</i>	54
12.5 Свойства объектов	55
12.6 Применение методов к объектам	56

12.7 Объекты Range.....	59
12.7.1 Указание Range-объекта в операторе.....	59
12.7.2 Свойства Range-объекта.....	60
12.7.3 Методы Range-объекта.....	64
12.7.4 Диалоговый ввод ссылки на Range-объект.....	65
12.8 Автоматическое создание программ	67
Литература.....	70
Приложение А Текстовый редактор VBE.....	71
А.1 Основные особенности	71
А.2 Ввод и редактирование текста.....	74
А.3 Выделение, удаление, перемещение и копирование	75
А.4 Запуск программ на выполнение	77
Приложение Б.Список основных операторов VBA	78

Издание подготовлено в авторской редакции

Отпечатано на участке цифровой печати
Издательского Дома Томского государственного университета

Заказ № 2431 от «15» марта 2017 г. Тираж 120 экз.