

**Программирование, практикум по Visual Basic for Applications.**



## Глава 1. Алгоритмы и программы

### 1.1. Понятие алгоритма

**Алгоритм.** Фундаментальной концепцией информатики является понятие алгоритма. *Алгоритм* – это конечная последовательность инструкций (действий, предписаний), предназначенных для решения поставленной задачи. Инструкции должны быть точными: два исполнителя одного и того же алгоритма должны прийти к одному и тому же результату.

Совокупность всех исходных данных, к которым алгоритм применим, называется *областью применимости алгоритма*. Каждый алгоритм задает функцию, относящую каждому элементу области применимости соответствующий результат, т.е. область применимости совпадает с областью определения этой функции. Говорят тогда, что *алгоритм вычисляет эту функцию*. Функция, которая вычисляется некоторым алгоритмом, называется *вычислимой*. Множество значений вычислимой функции, определенной на  $\mathcal{N}$  (натуральный ряд), образует *перечислимое множество* (значения функции перечисляются алгоритмом). Область применимости и область результатов любого алгоритма – перечислимые множества.

Не всякая математическая задача может быть решена с помощью алгоритма. Это связано с невычислимостью (неперечислимостью) некоторых областей, на которых определены решения этих задач. Например, существуют перечислимые подмножества натурального ряда с неперечислимым дополнением. Задачи, не решаемые с помощью алгоритма, называют *алгоритмически неразрешимыми*. К числу таких задач относятся, например, *проблема распознавания эквивалентности (равенства) слов* (существуют конечный алфавит, конечный набор правил составления и преобразования слов в этом алфавите, но нельзя построить алгоритм, который по двум произвольным словам этого алфавита всегда определит, преобразуются они или нет к одному и тому же слову). Другая известная алгоритмически неразрешимая задача – *проблема остановки* – не существует алгоритма, отвечающего на вопрос: остановится или нет некоторая программа, запущенная на некотором наборе начальных данных (не распознается наличие в программах бесконечных циклов). *Проблема самоприменимости* также неразрешима (нельзя с помощью алгоритма распознать для машины, преобразующей слова, распознает она или нет шифр самой себя). Задачи, решаемые алгоритмами, относятся к области исследований конструктивной математики.

**Средства записи алгоритмов.** Записать алгоритм можно на естественном языке, в виде схемы, на каком-либо (специальном) языке программирования, к числу которых относятся и языки машинных команд, ассемблеры, автокоды. Одним из популярных схемных средств представления алгоритмов являются блок-схемы.

*Блок-схема* – это диаграмма специального вида, на которой фигуры обозначают операторы (действия алгоритма), а стрелки – последовательность исполнения операторов.

Основные алгоритмические структуры (операторы) – конструкции, с помощью которых можно записать алгоритм, - это: *присваивание, условный оператор, оператор цикла и последовательность операторов*. В различных языках программирования эти конструкции могут изображаться по-разному (иметь разный синтаксис), но отличия, как правило, несущественны.

## 1.2. Основные алгоритмические конструкции

### 1.2.1. Последовательность

Последовательность операторов означает последовательное их исполнение друг за другом. На блок-схемах эта конструкция изображается стрелкой ↓. В языках программирования последовательно выполняемые операторы отделяются друг от друга символом «;» или (как в VBA) символом конца строки (каждый оператор начинается с новой строки) или двоеточием.

### 1.2.2. Присваивание

Обычный синтаксис оператора присваивания:

<переменная> <знак присваивания> <выражение> ,

где <знак присваивания> может иметь вид «:=» или «=» (как, например, в VBA). Выполняется присваивание так: вычисляется выражение в правой части этого оператора и полученное значение присваивается переменной левой части (переменная получает это значение, «стирая» предыдущее). Например, последовательность операторов присваивания (в языке VBA):

$a = 4 + 7$

$a = a + 2$

$v = 2$

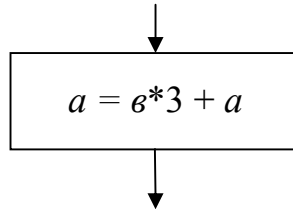
$a = v * 3 + a$

исполняется так: вычисляется  $4+7$ , результат 11 присваивается переменной  $a$ . Прежнее значение  $a$  стирается, новое значение  $a$  есть 11. Во второй строке к этому значению прибавляется 2, результат выражения есть 13 ( $11+2=13$ ). Это значение присваивается переменной  $a$ , теперь уже значение  $a$  есть 13. В третьей строке  $v$  получает значение 2. Наконец, в четвертой строке снова пересчитывается значение переменной  $a$ : вычисляется выражение  $v * 3 + a$  (вместо  $v$  и  $a$  подставляются их значения 2 и 13), результат  $2 * 3 + 13 = 19$  присваивается переменной  $a$  (говорят, 19 «записывается» в переменную  $a$  или «запоминается» в  $a$ ). Результат выполнения всей последовательности:  $a=19$ ,  $v=2$ .

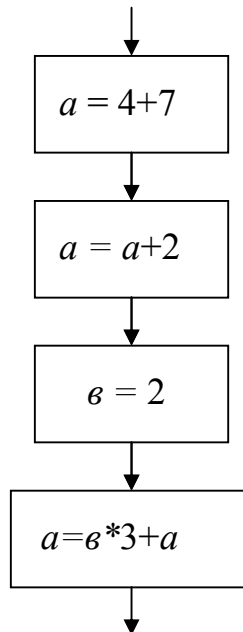
Присваивание можно трактовать как запоминание (сохранение) вычисляемых в ходе исполнения алгоритма значений для последующего их использования. Сохраняясь в переменной, значение приобретает имя, например, в пре-

в предыдущем примере далее в алгоритме вместо числа 19 можно использовать его (временное) имя  $a$ , вместо числа 2 – имя  $v$ . Обратную операцию – извлечение значения из переменной называют *разыменованием*.

На блок-схеме присваивание представляется прямоугольником со входом и выходом. Внутри прямоугольника записывается сам оператор. Например,  $a = v * 3 + a$  выглядит так:



Вся последовательность предыдущего примера изображается так:



### 1.2.3. Условный оператор (ветвление)

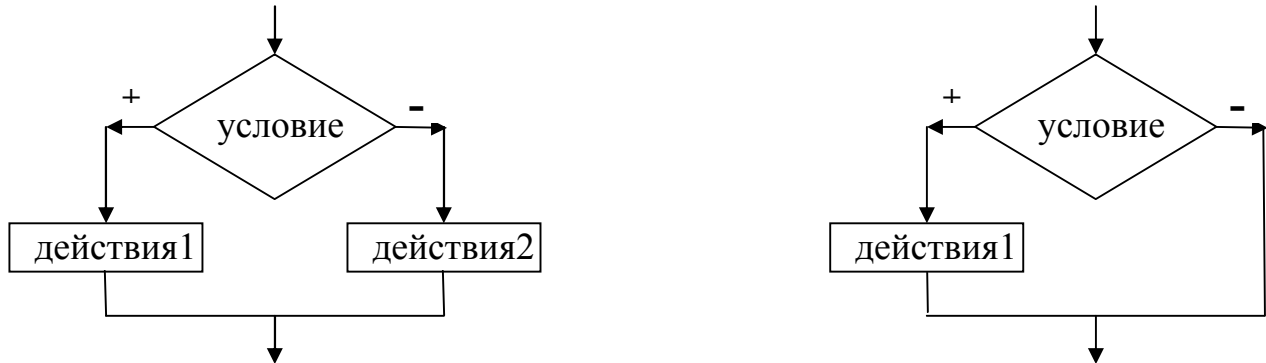
Эта алгоритмическая структура представляет разветвление алгоритма в зависимости от значения (истинности или ложности) некоторого условия. В общем виде конструкция выглядит так:

**<если>** <условие> **<то>** <действия1> **<иначе>** <действия2> ,

и читается как «если условие истинно, то выполнить действия1, иначе (если условие ложно), выполнить действия2». Слова «если», «то», «иначе» в разных языках могут иметь разный синтаксис, но в большинстве языков это «if», «then», «else». В VBA синтаксис условного оператора:

**If** <условие> **Then** <действия1> **Else** <действия2> **End If**

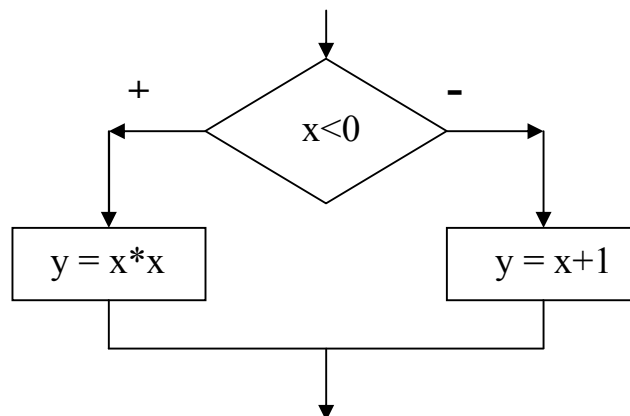
Условный оператор может быть неполным, без ветки <иначе> <действия2>. Тогда, если условие ложно, управление передается следующему в общей последовательности оператору. На блок-схемах эти два случая изображаются так:



Здесь знаки «+» и «-» обозначают «да» (условие выполняется) и «нет» (условие не выполняется). Например, вычисление  $y$ , заданного формулой:

$$y = \begin{cases} x^2 & \text{при } x < 0, \\ x + 1, & \text{при } x \geq 0 \end{cases}$$

можно представить следующей блок-схемой:



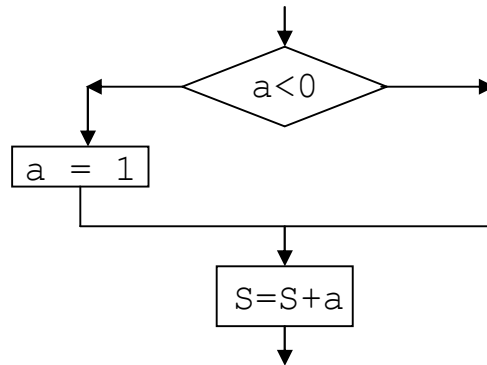
На языке программирования VBA эта конструкция выглядит так:

```
If  x<0  Then
y=x*x
Else
y=x+1
End If
```

Обратите внимание, что операторы после **Then** (ветка «+») и **Else** (ветка «-») начинаются с новой строки, а сам условный оператор заканчивается фразой **End If** – признаком конца конструкции ветвления.

Пример использования неполного условного оператора: суммируются числа, вводимые с клавиатуры; если число отрицательное, оно прежде заменяется единицей. Пусть переменная  $a$  хранит значение введенного числа, а пере-

менная  $S$  хранит сумму введенных чисел. Фрагмент блок-схемы решения такой задачи:



На языке программирования VBA эта конструкция выглядит так:

```

If a < 0 Then
a = 1
End If
S = S + a

```

Каждая из ветвей условного оператора может содержать произвольное количество операторов, среди которых могут быть снова ветвления (вложенные «если»). Например, следующая программа:

```

x = 10
If x > 5 Then
  x = x - 5
  If x > 7 Then
    x = x + 4
    y = x
  Else
    x = x * x - 1
    y = x + 8
  End If
Else
  x = x - 4
  y = 1
End If

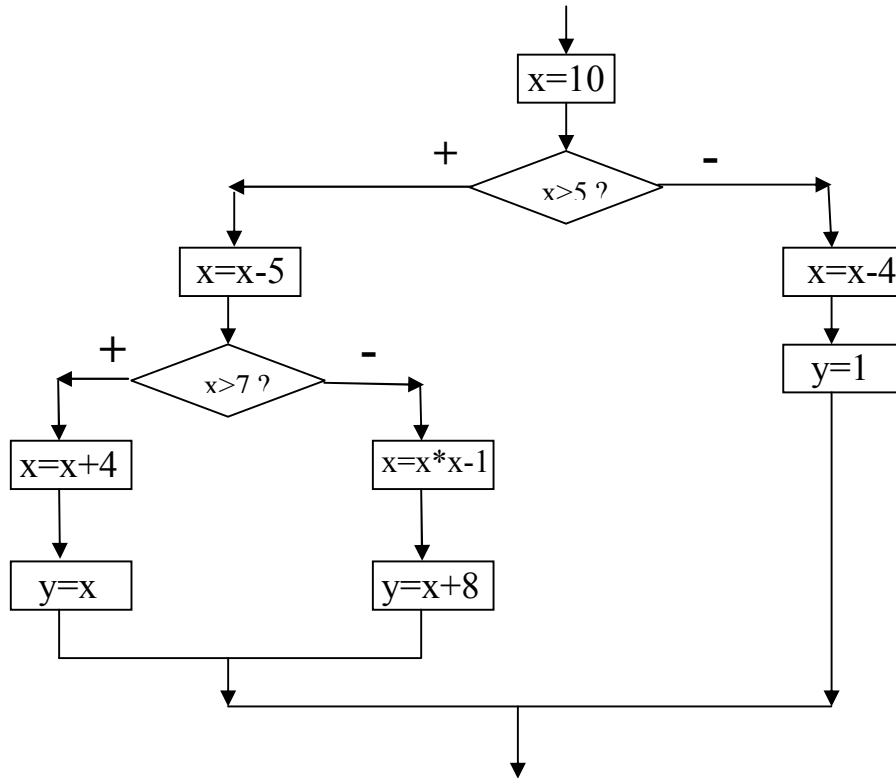
```

выполняется так: переменная  $x$  получает в результате присваивания значение 10. Затем проверяется условие  $x > 5$ ? В данном случае оно выполняется, поэтому исполняются операторы после **Then**: вначале  $x$  получает новое значение 5, затем проверяется (вложенное) условие  $x > 7$ ? Оно ложно, тогда последовательно исполняются операторы, идущие после **Else**:  $x = x * x - 1$  и  $y = x + 8$ , результат  $x = 24$ ,  $y = 32$ . На этом внутренний «**If**» заканчивается, ветка «**Else**» внешне-

го «**If**» не исполняется, общий результат остается тем же:  $x=24$ ,  $y=32$ . Если же изменить начальное присваивание, например, вместо  $x=10$  записать  $x=2$ , то исполнение программы будет идти по другой ветке, соответствующей внешнему «**Else**»:  $x=x-4$ ,  $y=1$ , и результат будет другой:

$x = -2$ ,  $y = 1$ .

Блок-схема для этой программы имеет вид:



Какие операции можно включать в условие? Условие – это логическая формула, значение которой может быть вычислено. Простейшее условие – это отношение: больше ( $>$ ), меньше ( $<$ ), больше или равно ( $>=$ ), меньше или равно ( $<=$ ), не равно ( $<>$ ). Более сложные условия составляются из простых с помощью операций конъюнкции (в VBA это **And**), дизъюнкции (**Or**), отрицания (**Not**), импликации (**Imp**). Например, условие  $x$  меньше пяти, но больше или равно двум, записывается как  $(x < 5)$  **And**  $(x >= 2)$ .

#### 1.2.4. Оператор цикла

Цикл означает повторяющийся набор действий. Например, суммируется набор из 20-ти чисел, вводимых с клавиатуры. Действия: «ввести число» и «добавить его к сумме» будут повторяться 20 раз. В задаче: складывать числа, вводимые с клавиатуры, до тех пор, пока не встретится 0, эти действия будут повторяться столько раз, сколько выполняется (истинно) условие: «введенное число не равно нулю». Для обозначения повторений в записи алгоритмов используют конструкции, называемые циклами: *цикл с параметром* (счетчиком), *цикл с предусловием* и *цикл с постусловием*. Цикл с предусловием является

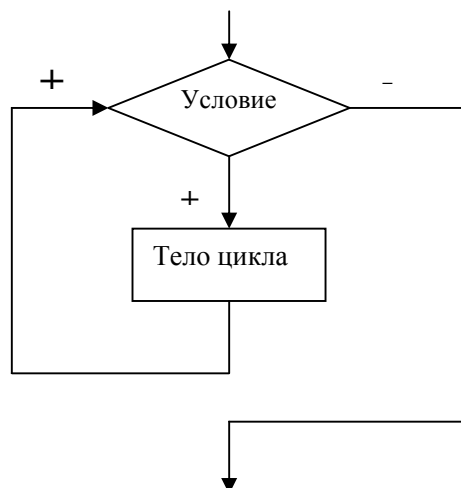


наиболее общей конструкцией: этого оператора достаточно, чтобы записать любые циклические действия алгоритмов.

Повторяющаяся группа действий называется *телом цикла*, однократное выполнение этой группы – *шагом цикла*. Часть конструкции, в которой определяется, продолжать выполнение цикла или нет, называют *заголовком цикла*. Заголовок, как правило, содержит условие, истинность или ложность которого требует повторения операторов тела цикла. Если условие задано так, что оно выполняется всегда, например,  $4 < 7$ , то повторения будут длиться «вечно», тогда говорят, что программа *зациклилась*. Если, наоборот, условие никогда не выполнится, например,  $2 < 0$ , то операторы цикла не исполняются, вся конструкция игнорируется исполнителем алгоритма и управление передается следующим за оператором цикла действиям.

**Цикл с предусловием.** Заголовок этого цикла содержит условие, которое проверяется всякий раз *перед* очередным исполнением тела цикла. Если условие истинно, тело исполняется, если ложно, управление передается следующему за циклом оператору в алгоритме. Таким образом, тело цикла исполняется столько раз, сколько раз истинно условие цикла. Блок-схема этого оператора:

Циклы с предусловием называют обычно циклами типа While или цикла-



ми ПОКА (работает, пока условие выполняется). Синтаксис этих циклов в VBA:

**While** <Условие> <Тело цикла> **Wend**

или

**Do While** <Условие> <Тело цикла> **Loop**

Например, исполнение фрагмента программы:

```

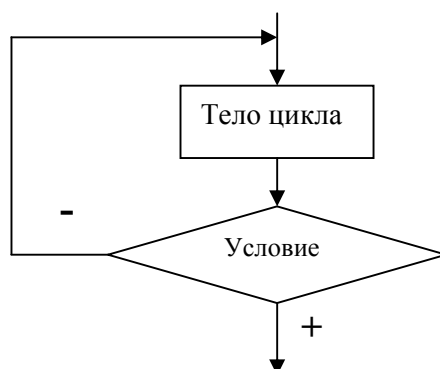
a = 7
While a > 0
a = a - 1
Wend
  
```

даст в результате значение  $a=0$ , т.к. тело цикла  $a = a - 1$  выполнялось столько раз, сколько выполнялось условие  $a > 0$ . Последний раз оно выполнилось, когда

$a=1$ , в результате  $a=1-1=0$ . Проверка условия ( $0>0$ ) показала «ложь» и управление передалось операторам, следующим за **Wend** (граница цикла **While**). Таким образом, значение  $a$  осталось равным 0. Если заменить первый оператор фрагмента  $a=7$  на  $a=0$ , то цикл не выполнится ни разу, т.к. условие  $a>0$  ложно.

Тело цикла не исполнится ни разу, если условие не выполняется уже вначале. В некоторых задачах необходимо, чтобы тело цикла хотя бы раз отработало. Можно, конечно, операторы тела цикла «продублировать», написав их еще раз до оператора цикла, тогда они исполнятся независимо от истинности условия. Но, если их много, программа выглядит громоздко. Для таких случаев удобно использовать цикл с постусловием.

**Цикл с постусловием.** В таких циклах условие проверяется *после* того, как операторы тела цикла хотя бы раз отработают. Блок-схема этого цикла такова:



В отличие от цикла с предусловием, данный цикл работает *до* выполнения условия, пока оно *не* верно, т.е. здесь истинность условия означает выход из цикла.

Циклы с постусловием называют **Until**-циклами, циклами **ПОКА НЕ** или циклами **ДО** (работают до того, как условие выполнится). Синтаксис этих циклов в VBA:

**Do** <Тело цикла> **Loop Until** <Условие>.

Программа предыдущего примера, но с **Until**-циклом:

```
a = 7
Do
a = a - 1
Loop Until a < 0
```

даст другой результат:  $a=-1$ , т.к. после того, как значение  $a$  стало равным 0, тело цикла отработало еще раз (условие  $a<0$  еще не выполнилось). Если заменить первый оператор фрагмента  $a=7$  на  $a=0$ , то этот оператор цикла сработает, т.к. до проверки условия выполнится  $a=a-1$ , и уже после этого произойдет

выход из цикла (значение  $a$  станет  $-1$ , поэтому условие завершения  $a < 0$  станет верным).

При использовании рассмотренных видов циклов необходимо помнить, что в теле цикла должны быть операторы, которые влияют на значение условия и, в конце концов, приведут к изменению его значения. Неизменные, «окаменевшие» условия, если они истинны, приведут к зацикливанию программы.

**Цикл с параметром.** Эти циклы используются тогда, когда число повторений известно заранее – количество шагов задано, например, 20, 100,  $N$ , или может быть вычислено как результат какого-либо выражения до исполнения цикла. *Параметром* в цикле является счетчик шагов.

Счетчик (это значение специально выделенной переменной) может изменяться на единицу с каждым шагом или получать некоторое заданное приращение, например, 0.15. Цикл тогда исполняется до тех пор, пока значение счетчика не достигнет указанного в заголовке цикла значения. Циклы с параметром называют часто *циклами типа For*, т.к. в большинстве языков программирования их заголовки начинаются со слова For. Для VBA синтаксис этого цикла:

```
For <переменная-счетчик> = <начальное значение> To <конечное значение>
<Тело цикла>
Next
```

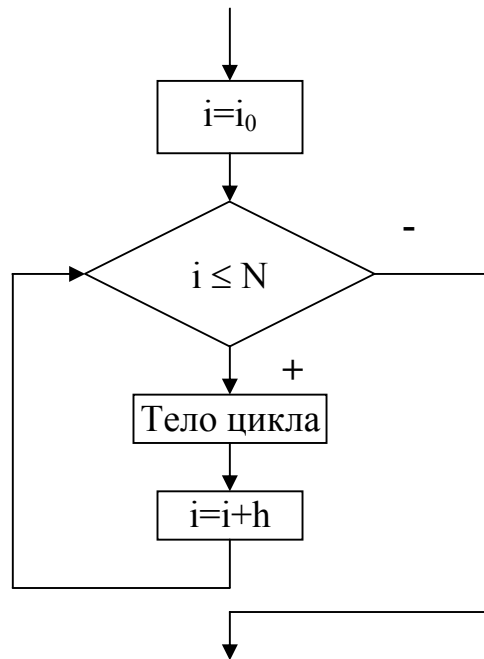
или, если используется счетчик с приращением значения, отличным от 1:

```
For <переменная-счетчик> = <начальное значение> To <конечное значение>
Step <приращение>
<Тело цикла>
Next
```

Здесь переменная-счетчик – это имя переменной, взятой для счета шагов; начальное значение, конечное значение – выражения, в частности, целые числа или переменные, значения которых берутся в качестве начала и, соответственно, конца отсчета повторений тела цикла, приращение – выражение, значение которого на каждом шаге добавляется к счетчику цикла (приращение может быть и отрицательной величиной, тогда оно вычитается из счетчика и начальное значение в этом случае должно быть больше конечного).

Оператор цикла с параметром работает следующим образом. Тело цикла исполняется столько раз, сколько значение счетчика меньше или равно конечному значению. После каждого шага цикла к счетчику добавляется 1, если приращение не указано, или значение приращения, если оно указано. Условие на счетчик после этого проверяется и, если конечное значение не превышено, тело цикла вновь исполняется. После присваивания счетчику начального значения условие также проверяется.

Обозначим начальное значение счетчика  $i$  как  $i_0$ , конечное значение –  $N$ , приращение –  $h$ . Тогда блок-схема этого оператора имеет вид:



Как видно, цикл с параметром является частным случаем цикла с предусловием типа While. Пример использования цикла-For в программе VBA:

```

k = 2
m = 4
For i = k + 1 To m * 2 + 1 Step 0.5
k = k + i
Next
  
```

Этот цикл выполняется, начиная со значения счетчика  $i$ , равного 3-м, и продолжается, пока счетчик не превзойдет величину 9 ( $4*2+1=9$ ); на каждом шаге значение счетчика увеличивается на 0.5, поэтому число шагов будет не 7, а 13. В результате значение  $k = 80$ . Несмотря на то, что значение  $k$  на каждом шаге меняется, начальное значение  $k+1=3$  не пересчитывается и к нему исполнение цикла не возвращается.

Обратите внимание, что заголовок цикла For: начальное, конечное значения счетчика и шаг - вычисляется один раз, до первого выполнения тела цикла, и не перевычисляется, несмотря на возможные изменения переменных, входящих в выражения для этих значений.

**Досрочный выход из цикла.** При использовании циклических конструкций может возникнуть необходимость досрочного выхода из цикла. Например, получен искомый результат, а условие цикла еще истинно и позволяет продолжить исполнение этого оператора. В языке VBA оператором досрочного выхода является **Exit**, причем в циклах **For** он имеет вид **Exit For**, а в циклах, начинающихся с **Do**, он имеет вид **Exit Do**. Например, программа

```

a = 7
Do
  a = a - 1
  If a = 5 Then
    Exit Do
  End If
End Do
  
```

**Loop Until**  $a < 0$

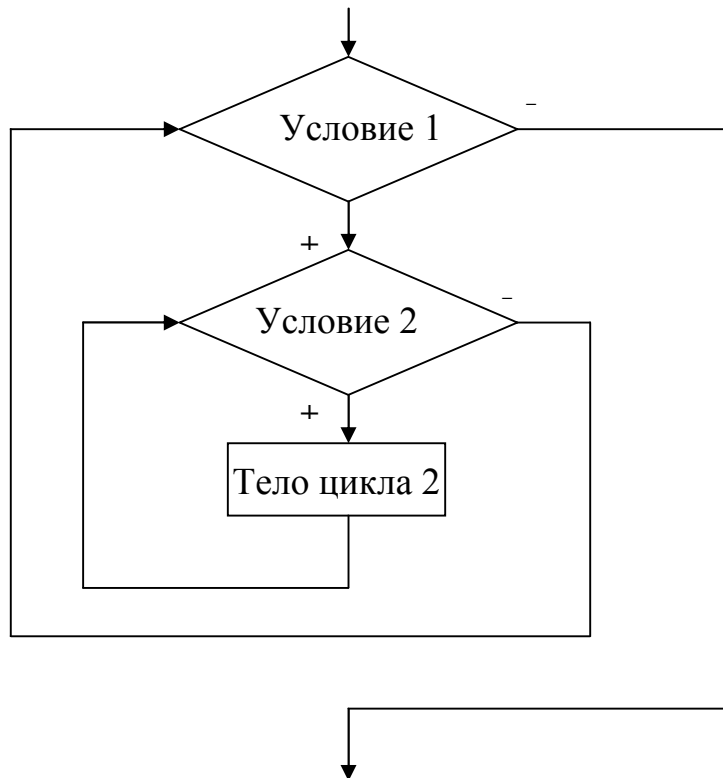
даст на выходе значение 5, т. к. цикл принудительно прервался, когда значение переменной  $a$  стало равным пяти. Того же результата можно достичь и в программе с циклом **For**:

```

k = 2
m = 4
For i = k + 1 To m * 2 + 1 Step 0.5
    k = k + i
    If k = 5 Then
        Exit For
    End If
Next

```

**Вложенные циклы.** Тело любого оператора цикла может содержать другие циклы. Такие конструкции называют *вложенными циклами*. Вложенные циклы (цикл в цикле) применяют обычно в задачах, когда требуется связать или сравнить *каждый* элемент одного множества с *каждым* элементом другого множества. Блок схема вложенных циклов для, например, циклов с условием выглядит так:



Здесь тело внешнего цикла (цикла 1) состоит из оператора внутреннего цикла (цикла 2). Условие выхода из всей конструкции – это условие внешнего цикла. На каждом шаге внешнего цикла полностью «отрабатывает» внутренний цикл. Таким образом, если внешний цикл перечисляет элементы некоторого множества  $X$ , а внутренний цикл – элементы множества  $Y$ , то за  $i$ -й шаг работы первого цикла для  $i$ -го элемента  $x_i$  из  $X$  могут быть проверены *все* элементы  $y_j$

из Y, т.к. второй цикл не завершит работу, пока истинно его условие (Условие 2 на схеме).

Пример программы с вложенными циклами в VBA:

```

For i = 1 To 7
  For j = 1 To 5
    If i <= j Then
      Cells(i, j) = 1
    End If
  Next
Next

```

Здесь Cells(i, j) – имя ячейки листа Excel, расположенной в строке i и столбце j. Эта программа «рисует» единицами треугольник на листе Excel: перебираются все строки листа от первой до седьмой и для *каждой* строки проверяются *все* пять столбцов: они заполняются единицами, если текущий номер строки меньше или равен текущему номеру столбца, и не заполняются ничем в противном случае.

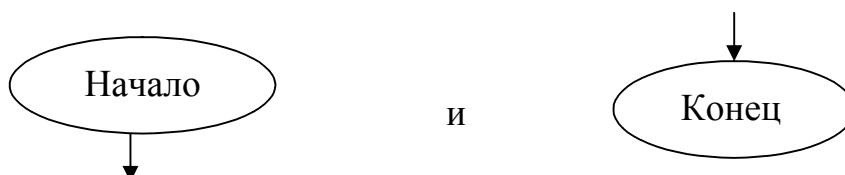
Циклы в теле другого цикла имеют точно такой же статус при исполнении, как и любой другой оператор, поэтому «вкладывать» один в другой можно циклы любого типа и глубина вложения (количество вложенных друг в друга циклов) может ограничиваться лишь реализацией языка программирования.

### 1.2.5. Операторы ввода-вывода

Эти действия выполняют ввод и вывод информации (значений переменных) в ходе работы программ. Операторы ввода-вывода с точки зрения выразимости алгоритма не имеют самостоятельного значения, т.к. могут быть заменены набором соответствующих присваиваний, но для написания программ они существенны, т.к. исполняются особым образом, с привлечением дополнительных устройств. Для более точного представления алгоритмов в блок-схемах эти операторы имеют специальное изображение, например,



Кроме того, для обозначения начала и конца алгоритма используют фигуры



В языке VBA ввод значения переменной  $x$  можно осуществить с помощью функции **InputBox**, например:

```
x = InputBox("Введи значение x")
```

Вывод в специальное окно экрана можно реализовать оператором **MsgBox**, например:

```
MsgBox (x)
```

в итоге на экран выведется только значение переменной  $x$ , например, 5. Оператор

```
MsgBox "x =" & x
```

выведет на экран фразу:  $x = 5$ .

Можно осуществить ввод и вывод, используя лист Excel. Ячейки листа имеют имена `Cells(i, j)`, где  $i$  – номер строки,  $j$  – номер столбца. Например, ввод значения переменной  $x$  из ячейки A1 выглядит в программе так:

```
x = Cells(1, 1),
```

а вывод результата – значения  $x$  в ячейку, например, C2 (третий столбец) реализуется так:

```
Cells(2, 3) = x
```

### 1.3. Структуры и типы данных

Понятие «данные» обычно связывают с информацией, перерабатываемой алгоритмом. Эта информация может быть неоднородной, данные могут иметь различную структуру, которая зависит от решаемой задачи. Соответственно, размещение таких данных в памяти компьютера и доступ к ним будет различным. Языки программирования должны иметь средства для представления и обработки данных различной природы.

Например, программы анализа анкет служащих работают с таблицами, содержащими сведения о сотрудниках, поэтому языки представления алгоритмов обработки таких данных должны иметь операции, позволяющие извлекать из ячеек таблицы их содержимое, компоновать части таблиц, связывать таблицы друг с другом. Программы, вычисляющие физические параметры производственного цикла, работают с последовательностями действительных чисел. Над числовыми значениями должны быть определены, по крайней мере, арифметические операции сложения, умножения, вычитания, деления. Алгоритмы переработки текстов имеют дело с наборами символов, над которыми возможны другие операции: конкатенация («склеивание») символов в слова, вырезка части слова или предложения, замена части слов и т.п. Сами программы также

можно рассматривать как данные, имеющие свою структуру и свойства и допускающие определенные операции, например, вызов одной программы из другой.

Из этих соображений возникло понимание *типа данных* как *множества значений, характеризуемое множеством операций* над ними. Например, тип целых чисел (**Integer** в VBA) включает все целые числа и арифметические операции (хотя деление может вывести из этого типа данных). Символьный тип (**String**) включает символьные значения и операции склейки символов и специальные операции обработки строк. Логический тип (**Boolean**) состоит из двух значений **True** (истина) и **False** (ложь) и допускает все логические операции – отрицание (**Not**), конъюнкцию (**And**), дизъюнкцию (**Or**), импликацию (**Imp**).

Дальнейшее развитие концепции типа данных привело к понятию объектно-ориентированного программирования, в котором решение задач (как и весь «окружающий мир») представлено совокупностью взаимодействующих друг с другом объектов разных типов. Здесь *тип* – это *множество объектов, имеющих общее поведение и общую структуру (свойства)*.

Например, людей можно считать объектами типа «Человек». Этот тип обладает определенной структурой, воплощенной в его свойствах: вес, рост, количество рук, объем мозга и т.п. К человеку можно применять операции (*методы*), определяющие его поведение – ходить, спать, говорить, учиться и т.п. Какие именно свойства и методы рассматривать, зависит от задачи. Например, в задаче «определить, кто придет первым в стометровке, если известно, что толстый бежит медленнее худого», важны такие свойства, как вес, и метод бегать, но не важны устройство скелета и операция хлопнуть в ладоши, которые в другой задаче могли играть главную роль. Таким образом, разработчик программы (приложения) решает, какие типы данных ему нужны, и *может самостоятельно их определить*, задав соответствующие свойства и операции, применимые к объектам этих типов. Современные языки программирования предоставляют средства для задания собственных, *пользовательских*, типов данных.

**Объекты** некоторого типа Т – это конкретные представители этого типа данных. Например, Сидоров А.А., Иванов Б.Б – объекты типа Человек, ВАЗ 2109 с номером У 3040 – объект типа Автомобиль, соседский кот Барсик – объект типа Коты. Именно к объектам можно применять *методы* (операции) соответствующего типа данных. Также и свойства типа: только применительно к конкретному объекту свойство может получить определенное значение. Если использовать принятый в языках программирования синтаксис обращений к методам и свойствам: *объект.метод* и *объект.свойство* = значение, то допустимы, например, такие конструкции в программах: Сидоров.Бегать, Барсик.Кусаться, Барсик.Вес=3. Применение операции копирования к объекту – диапазону А1:А3 листа Excel в VBA будет выглядеть так: Range("A1:A3").Copy. Метод **Copy** применим к любому объекту типа данных Диапазон. Этот тип задан в программе приложения Excel (описан разра-



ботчиками этого приложения), для пользователей же Excel он является *встроенным*, стандартным, как, например, тип **Integer**.

Типы данных в языках программирования можно условно разбить на следующие группы:

- примитивные типы (целые, вещественные числа, логические и символьные значения);
- конечные последовательности однотипных элементов (линейные массивы, матрицы, многомерные массивы);
- конечные последовательности разнотипных элементов (записи, структуры);
- динамические последовательности со специальными операциями доступа (очереди, стеки, списки);
- сложные типы, ориентированные на специальные предметные области (электронные таблицы, листы Excel, диапазоны ячеек листов, базы данных и т.п.);
- произвольные типы, определяемые пользователем (например, самолет, автомобиль, животное, собаки, студенты).

Программы обращаются к данным, хранящимся в памяти компьютера с помощью имен, реализуемых в понятии *переменной*. Переменная имеет тип, указывающий на то, какого сорта (типа) значения она может обозначать (говорят, *хранить*), и значение – конкретный объект. В компьютере каждой объявленной в программе переменной отводится область памяти, которую эта переменная будет именовать. Размер этой памяти зависит от типа переменной. Например, для переменной типа **Integer** это два байта, для переменной типа **Double** (число с плавающей точкой удвоенной точности) – восемь байтов, для массива из десяти целых чисел – блок памяти объемом  $2 \times 10 = 20$  байтов.

Таким образом, каждое хранимое значение имеет в компьютере физический адрес памяти, которая в данный момент его содержит и (в программе) имя переменной, которая им обладает (ссылается на этот адрес). Например, при присваивании  $a=b$  значение переменной  $b$  пересылается по адресу, отведенному для переменной  $a$ . Чтобы компьютер правильно выполнил эту пересылку, необходимо, чтобы значение, «сидящее» в  $b$  «поместилось» в память, отведенную для значений переменной  $a$ . Это означает, что типы этих переменных должны совпадать или быть *совместимыми*. Совместимость означает, что можно разместить данное без потери его значения или смысла. Например, присваивание  $a=b$  для  $a$  типа **Integer** и  $b$  типа **Double** невозможно реализовать без существенной потери значения  $b$ .

Ошибки, возникающие при таких присваиваниях, называют *типовыми ошибками*. Если типы используемых переменных объявлены в программе заранее, то эти ошибки можно выловить на этапе компиляции (перевода программы в машинный код). Такой контроль типов называют *статическим*, в отличие от динамического, когда типовые «разборки» начинаются при исполнении программ (исполнении готового машинного кода). Статический контроль типов повышает эффективность программ, поэтому большинство языков программи-

рования требует описания типов переменных в программах до того, как эти переменные будут использованы в операторах. В VBA описание переменных сопровождается ключевым словом **Dim** и выглядит так:

**Dim** <имя переменной> **As** <тип переменной>.

Например, **Dim a As Double, b As Boolean, c As Integer**. Эти описания помещаются в начале программы, до первого оператора.

Для реализации возможности совмещения типов используются так называемые операции *приведения типов*. Приведение, если оно возможно, означает преобразование значения одного типа в значение другого типа. Преобразуется именно значение, типы же переменных остаются прежними. Как правило, могут «приводиться» типы меньшего или равного физического размера к большему. В противном случае, если приведение допускается, то заранее оговаривается, каким образом большее значение будет урезано. В языках программирования существуют специальные операции (функции) приведения типов. Например, в VBA к таким функциям относятся **CStr** (преобразование к символьному типу `String`), **CInt** (преобразование к целочисленному типу `Integer`). Применять функцию **CInt**(*x*) можно к числовому аргументу *x* (значение *x* тогда округляется до целого) или к числовой строке *x* (значение строки, например, *x*="123" преобразуется в число 123). Нечисловые строки, например, *x*="abcd" не могут быть приведены к целому числу, и выражение **CInt**(*x*) даст типовую ошибку. Функция **CStr**(*x*), примененная к числовому аргументу *x*, например, *x*=35.4 преобразует это число в строку "35.4", и к ней тогда можно применять любые строковые операции, например, склейку – конкатенацию (&) строк: результат выражения **CStr**(*x*) &"abcd" будет 35.4abcd.

**Доступ к данным.** Для каждого типа данных должны быть определены специальные *операции* (говорят, *функции*) *доступа* к значениям этого типа. Создавая программу, разработчик должен знать, каким образом можно извлечь нужное значение. Для встроенных, стандартных типов языка функции доступа (программы, реализующие извлечение значения) также встроены, их синтаксис дается в описании языка программирования. Для пользовательских типов реализация (программа) функций доступа является неотъемлемой частью описания типа.

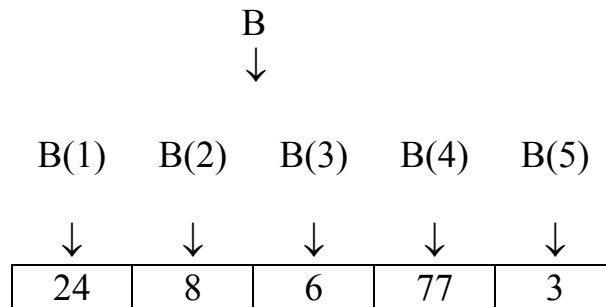
*Доступ к значениям примитивных типов* осуществляется просто указанием имени переменной, содержащей значение, или указанием самого значения непосредственно. Например, в выражении  $b+7*(d-c)$  для *b*, *d*, *c*, имеющих числовой тип, значения переменных извлекаются из ячеек памяти, именуемой этими переменными.

Для более сложных, структурированных, типов часто требуется иметь доступ к отдельным компонентам (элементам) этих структур. В зависимости от

«устройства» типа доступ может быть осуществлен по имени элемента, по его порядковому номеру в последовательности.

*Доступ к значениям элементов массивов.* Массивы – это упорядоченные конечные последовательности элементов одного и того же типа. Например, массив целых чисел, массив символов, массив котов. Массив может быть одномерным (линейным) или многомерным. Двумерные массивы называют обычно матрицами. В памяти компьютера массиву отводят блок памяти, в котором элементы массива располагаются друг за другом. Переменная, содержащая массив, ссылается на адрес начала этого блока. Доступ к элементу массива осуществляется по его порядковому номеру в последовательности. Таким образом, чтобы получить значение, например, третьего элемента некоторого массива нужно указать имя переменной, содержащей массив, и номер (говорят, *индекс*) три. Например, в VBA:  $A(3)$  – третий элемент массива  $A$ .

Пусть переменная  $B$  содержит массив пяти целых чисел: 24, 8, 6, 77, 3. Тогда  $B(3)$  – имя, назвав которое можно получить значение 6, т.е.  $B(3) = 6$ . Этот массив можно изобразить так:



Если выполнить присваивание, например,  $B(3) = B(5)$ , то значением элемента  $B(3)$ , как и  $B(5)$ , будет число 3, прежнее значение 6 «сотрется». Обозначение  $B(i)$  можно считать *именем  $i$ -го элемента* массива  $B$ .

Для двумерного массива – матрицы - доступ к элементу можно получить, указав номер строки и номер столбца, на пересечении которых он находится, например,  $M(4,3)$  – имя элемента четвертой строки и третьего столбца матрицы  $M$ .

Чтобы можно было правильно реализовать присваивания элементам массивов, нужно знать тип этих элементов. Так, если  $B$  – массив целых чисел, то недопустимо присваивание  $B(2) = \text{“Маша ела кашу”}$ . Для того, чтобы обеспечить правильную работу программы и распределение памяти, компьютер должен знать «размер» каждого элемента массива и их количество. Поэтому минимум того, что необходимо сообщить при описании массива, это – имя переменной массива, количество и тип элементов. Например, описание в VBA линейного массива  $A$ , состоящего из десяти элементов – целых чисел выглядит так:

**Dim A(10) As Integer,**

а описание матрицы  $M$  из пяти строк и семи столбцов символьных значений – так:

**Dim M(5,7) As String**

Таким образом, слово «массив» не имеет статуса типа данных, это скорее способ конструирования «массивоподобных» типов. Тип массива определяется количеством и типом его элементов (иногда, если в языке программирования разрешается нумеровать элементы не целыми числами, а, например, константами или буквами, то и типом индексов). Так, переменные *A* и *B*, описанные как массивы из десяти целых чисел: **Dim A(10) As Integer, Dim B(10) As Integer** имеют один и тот же тип, поэтому для любого  $i = 1, 2, \dots, 10$  допустимо присваивание  $A(i) = B(i)$ . В некоторых языках программирования допустимо и присваивание содержимого массива «целиком»:  $A = B$ . Если же переменная *B* описана как массив действительных чисел: **Dim B(10) As Single**, то теперь переменные *A* и *B* имеют разные типы, и присваивание  $A(i) = B(i)$  недопустимо из-за типовой ошибки. Если описать переменную *B* как массив целых чисел, но другой длины, например, **Dim B(20) As Integer**, то *A* и *B* также будут иметь разные типы, но присваивания компонент  $A(i) = B(j)$ ,  $B(j) = A(i)$  для  $i \leq 10$ ,  $j \leq 20$  допустимы, т.к. элементы этих массивов имеют один и тот же тип (*Integer*). Присваивание же значения элементу  $A(i)$  для  $i > 10$  приведет к ошибке, т.к. такого элемента у массива *A* нет.

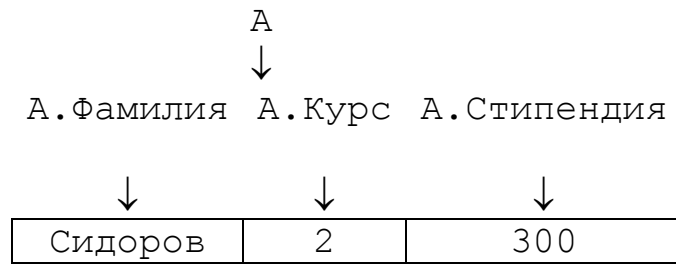
*Доступ к значениям элементов записей.* Записи (структуры) – это конечные наборы элементов разного типа. Каждый элемент имеет свое собственное имя, по которому можно извлечь его значение. Подобно массиву слово «запись» само по себе не является типом данных, тип записи определяется именами и типами ее элементов (говорят, *полей*). Например, набор полей для записей: *Фамилия* (символьная строка), *Курс* (целое число), *Стипендия* (вещественное число) определяет один тип данных (назовем его *S*), а набор: *Деталь* (символьная строка), *Материал* (символьная строка), *Количество* (целое число), *Цена* (вещественное число) определяет другой тип данных (назовем его *D*). На языке VBA такое описание типов выглядит так:

```
Type S
  Фамилия As String
  Курс As Integer
  Стипендия As Single
End Type
```

```
Type D
  Деталь As String
  Материал As String
  Количество As Integer
  Цена As Single
End Type
```

Переменные *A* и *B*, описанные как **Dim A As S, Dim B As D**, являются переменными *разных* типов.

Доступ к полям переменных-записей осуществляется с указанием имени переменной и имени поля (через точку), например, *A.Фамилия*. Пусть, например, поле *Фамилия* переменной *A* имеет значение «Сидоров», поле *Курс* – значение 2, поле *Стипендия* – значение 300. Запись *A* тогда можно изобразить так:



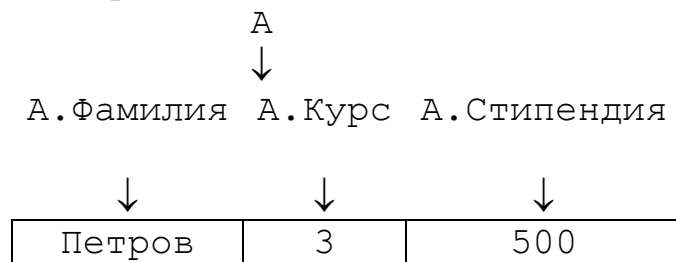
Допустимые присваивания, например:

A.Фамилия = «Петров»

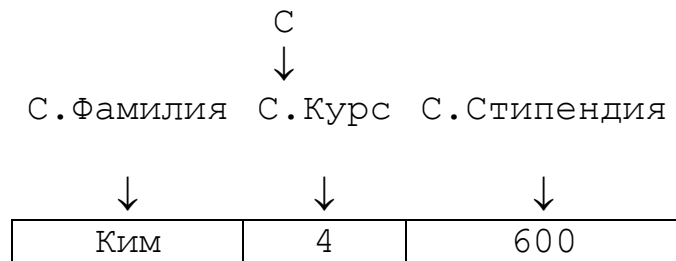
A.Курс = 3

A.Стипендия = A.Стипендия + 200

изменяют значение этой переменной:

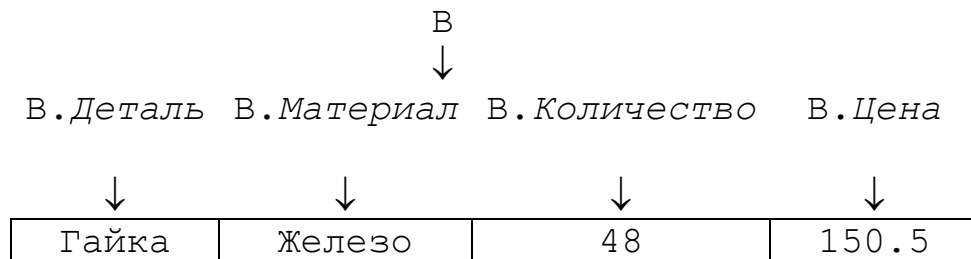


Переменная C, описанная как запись типа S, может иметь другое значение, но его структура будет той же:



Поэтому допустимы присваивания: A.Курс = C.Курс, A.Фамилия = C.Фамилия или даже A=C.

Переменную B рассмотренного типа D можно представить как:



Несмотря на то, что переменная B имеет другой тип, присваивания *однотипным элементам*, например, A.Фамилия = B.Деталь допустимы в большинстве языков программирования, но недопустимо присваивание «целиком»: A = B.

Из однотипных записей можно образовать массив, например: **Dim** M(10) **As** S. Тогда доступ к полю Фамилия i-го элемента массива будет осу-

ществлен по имени:  $M(i)$ . Фамилия. Фактически массив записей – это таблица, заголовки столбцов которой – имена полей, а строки – значения записей. В свою очередь поле записи может быть составным: иметь тип массива или тип записи, таким образом можно создавать достаточно сложные структуры данных.

*Доступ к значениям очередей, стеков, списков.* Очереди, стеки, списки – это чаще всего динамически создаваемые наборы не известного заранее количества элементов не обязательно одного типа. Динамическое создание элемента означает появление значения во время исполнения программы, в зависимости от реализации какого-либо условия или события, которого могло и не случиться. Новый элемент с таким значением может быть включен в набор. Обычно программисту, использующему подобные типы данных, доступны две операции: *взять элемент* и *добавить элемент*. Различие (для пользователя) между указанными типами данных состоит в том, какое именно значение из набора будет выдано, и куда будет помещен добавляемый элемент.

*Очередь* – последовательность, в которой выделены начальный элемент («голова») и конечный элемент («хвост»). Пополнение очереди происходит добавлением элемента к «хвосту» (пришедший становится «хвостом»), а извлечение значения возможно только для первого элемента очереди (элемент «уходит», и следующий в очереди становится «головой»). Очереди характеризуются в соответствии с этой процедурой обработки как «последним пришел – последним ушел».

*Стек* – последовательность, в которой выделен первый элемент (говорят, «верхушка»). Пополнение стека происходит добавлением элемента сверху (пришедший становится «верхушкой»), извлечение значения возможно тоже только для верхнего элемента (элемент «уходит», «верхушкой» становится следующий за ним). Стеки характеризуются способом обработки «последним пришел – первым ушел».

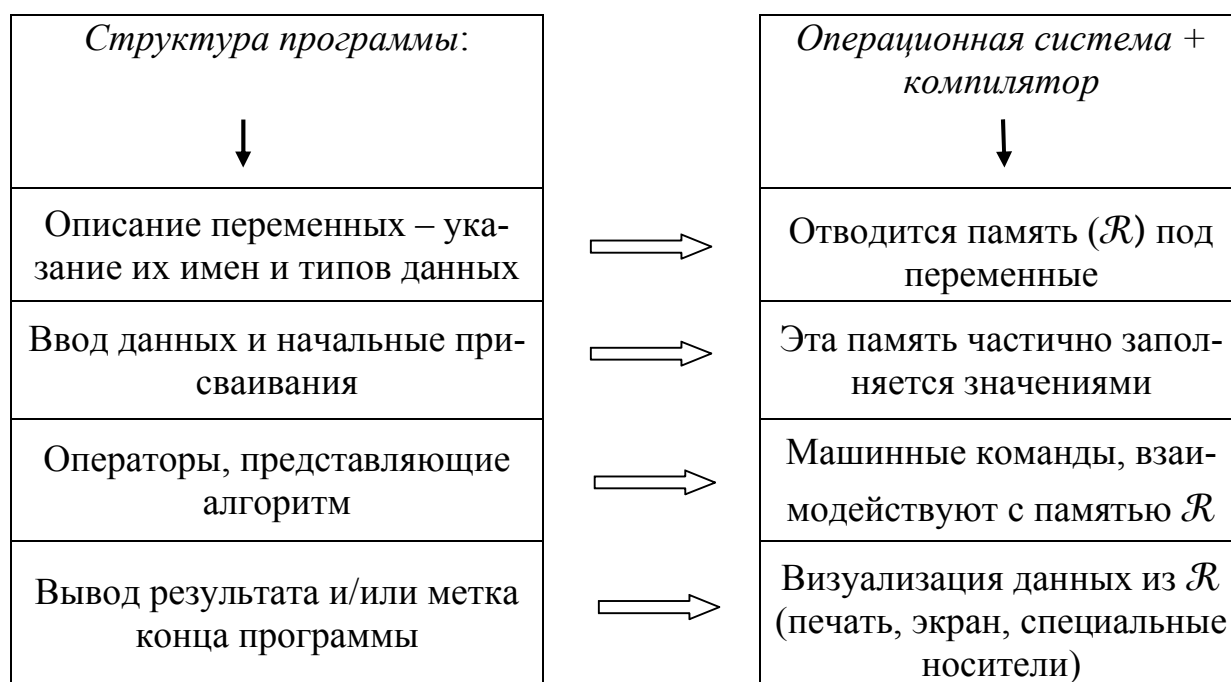
Для работы со значениями переменных-очередей или переменных-стеков задаются две операции: добавить элемент: `put(elem)` или `set(elem)` и взять элемент `get()`. Функция `get` «выдает» очередной или верхний элемент. Так, если переменная `Q` имеет тип очереди, то результатом обращения `Q.get` будет значение первого элемента очереди, при этом очередь укоротится. Аналогично для переменной `S` типа стека результатом обращения `S.get` будет значение верхушки стека, при этом стек также сократится. Доступа к другим элементам нет.

*Список* – последовательность, в которой известно местоположение первого элемента, возможно, последнего и каждый элемент хранит (помимо какой-либо информации) указатель (адрес) следующего за ним элемента списка. Таким образом, элементы списка могут быть «разбросаны» по памяти компьютера. Доступ возможен к любому элементу, удовлетворяющему некоторому условию, например: «первый в списке», «последний», «следующий», «пятый», «такой, у которого значение поля `d` равно 20» и т.п. Применение операции `get` к переменной-списку `G` даст то значение, которое определено программой этой операции.

Рассмотренные типы – очереди, стеки, списки, как правило, не входят в состав встроенных типов языков программирования, а моделируются разработчиками приложений. Использование этих типов данных удобно при создании таких структур, как деревья, графы, потоки, рекурсивные вызовы программ, вызовы процедур с параметрами-процедурами – всего «джентльменского набора» разработчиков операционных систем, компиляторов, on-line систем.

#### 1.4. Программы и программные единицы

Структуру программы и системные действия по ее обработке можно изобразить схемой:



При написании программы стоит помнить некоторые **технологические правила**:

1. При составлении алгоритма сразу определите его вход и выход: исходный набор данных и тот, который требуется получить.

2. Если задача сложная, попытайтесь разбить ее на несколько более простых задач. Для каждой части также определите ее вход и выход.

3. При обнаружении в алгоритме циклических конструкций четко определите условие выхода из цикла и набор повторяющихся операторов – тело цикла. Проверьте, не приводит ли условие к заикливанью. Проверьте также, не приводит ли приращение счетчика к выходу за пределы массивов, если они используются в цикле.

4. Все переменные, используемые в программе, следует описать в начале программы, до их использования. Тогда компилятор будет следить за соответствием типов.

5. Имена для переменных лучше выбирать так, чтобы они «говорили» о назначении этих переменных или, хотя бы, о типе их значений. Удобнее читать программу (и исправлять в ней ошибки), в которой вместо «всеядных»  $x$ ,  $y$ ,  $z$

используются информативные имена «цена», «среднее», «температура» или их понятные сокращения.

6. Переменные, используемые в *правых* частях операторов присваивания, в частности, переменные для накопления значений - счетчики, суммы, произведения - должны предварительно получить какое-то начальное значение. В противном случае результат такого присваивания может быть непредсказуем – память, отведенная под переменную, заполнена неизвестно чем. Не всякий компилятор предупредит о не определенном в программе значении переменной.

7. Типы левой и правой частей любого оператора присваивания должны быть совместимы. Следя за этим самостоятельно, вы застрахуетесь от «изуродованных» значений, полученных при автоматическом, «по умолчанию», приведении типов.

8. Проверьте «выходы за пределы»: если в задаче предусматриваются какие-либо предельные, граничные значения счетчиков, индексов, интервалов, отдельных величин, проанализируйте (самостоятельно, с помощью оператора If или дополнительной печати промежуточного результата), не превзойдены ли эти границы. Ошибки «пограничных ситуаций» наиболее типичны и трудно уловимы при просмотре текста программы.

9. Проверяйте ситуации «дурака»: всегда может найтись пользователь вашей программы, не выполнивший «очевидных» действий. Например, вы просите ввести ненулевое число и надеетесь поэтому, что ввели не ноль. Все надежные программы имеют «защиту от дурака» - операции проверки правильности ввода, предупреждения о возможном неверном ходе выполнения и т.п.

### 1.4.1. Сборка программ

Большие, сложные программы трудно писать без ошибок. Даже автор программы часто не может оценить, что же именно происходит в его творении, и малейшее исправление уже работающей программы может привести к полному краху всей системы. История программирования показывает все большее стремление к модульности в технологиях создания программ. Языки программирования и поддерживающие их системы включают средства разбиения программ на части (модули), отдельной компиляции этих частей и последующей сборки. При сборке программы могут сообщить друг другу свои результаты и эти результаты могут стать входом других программ. Такую *связь по входу и выходу* обеспечивает механизм входных и выходных параметров, которые можно указать в заголовке программы. Так, если программа P работает для входных данных X и выдает результат – данные Y, а программа G использует эти данные, вычисляя результат Z, который в свою очередь принимает на вход программа R, то образуется цепочка  $P(X) \rightarrow Y \Rightarrow G(Y) \rightarrow Z \Rightarrow R(Z) \rightarrow \dots$ . Очевидно, чтобы такие последовательности вызовов программ работали правильно, необходимо, чтобы типы входных и выходных данных соответствовали друг другу, т.е. были совместимы. Так, если, например, Y на выходе P имеет целочисленный тип, а данные, воспринимаемые программой G должны быть сим-



вольными, то  $G(Y)$  работать не будет, либо будет работать с непредсказуемым результатом. Поэтому сборщику программ для каждой включаемой программы необходимо знать имя программы, тип ее входных данных (говорят, *входных*, *input-параметров*) и тип результата (говорят, *выходных*, *output-параметров*).

Такого рода рассуждения приводят к пониманию программы как данного особого типа, который характеризуется типами входных параметров и типом результата программы. Имя программы в таком случае может рассматриваться как имя переменной соответствующего типа. Например, программа с именем  $P$ , имеющая два входных параметра, один из которых типа `Integer`, а другой – типа `String`, и выдающая результат типа `Single`, может рассматриваться как переменная  $P$  типа  $(\text{Integer}, \text{String}) \rightarrow \text{Single}$ . Другая программа  $R$  с таким же типом входа и выхода будет другой переменной *того же типа*. Структура программы в таком случае не учитывается, программа рассматривается (с точки зрения сборщика) как «черный ящик». Значениями таких переменных можно считать состояния памяти, изменяемые при исполнении программы. Поэтому переменные-программы имеют особый статус: *им нельзя присваивать значений в программах, которые их используют*.

Однако многие языки программирования допускают присваивания этим переменным внутри их тел, т.е., например, в программе  $P$  может (иногда, должен) встретиться оператор  $P = \langle \text{значение} \rangle$ . Таким значением обычно является *результат* ее работы, т.е. значение переменной-программы отождествляется с результатом ее выполнения. Это бывает удобно при определении программы как функции, т.е.  $P(x,y)$  рассматривается как функция от аргументов  $x$  и  $y$ . Тогда вызов этой программы для заданных данных  $a$  и  $b$  может участвовать в выражениях другой программы, например, допустимо выражение  $0.15 + P(a,b) - 2$ . При исполнении выражения вместо  $P(a,b)$  будет подставлен результат работы программы  $P$  на входных данных  $a$  и  $b$ .

Для использования в программе других программ, нужно отличать описание программы от ее вызова.

### 1.4.2. Описание программ

**Описание программы**  $P(x,y,z)$  – это текст программы  $P$ , в котором  $x$ ,  $y$ ,  $z$  являются *обозначениями* входных (и, возможно, выходных) данных. Эти обозначения называют *формальными параметрами* программы  $P$ . Описание обычно делят на заголовок и тело.

В *заголовке* указываются типы входных данных и результата (выходных данных). Если программа не вычисляет конкретное значение, например, просто печатает текст или производит различные действия со своими собственными переменными, не сообщая результаты в качестве выходных данных, то считается, что результат имеет неопределенный тип (например, тип `void` в языке C++). Такие программы называют часто *процедурами*. В отличие от них программы, вычисляющие определенное значение (некоторое число, текст, массив данных и т.п.), называют *функциями*. Говорят тогда, что *функция возвращает значение*. Синтаксис заголовка программ в VBA:

**Function** <имя функции> (<список формальных параметров>)  
<тип результата>

или

**Procedure** <имя процедуры> (<список формальных параметров>).

Список формальных параметров – это последовательность пар: <имя формального параметра> <тип формального параметра>, разделенных запятой.

Например, функция, вычисляющая  $n!$  – факториал числа  $n$ , будет иметь в языке VBA заголовок: **Function fact(n As Integer) As Integer**. Заголовок процедуры, вычисляющей сумму любых двух чисел и печатающей результат в диалоговом окне, может быть таким: **Procedure Sum(x As Single, y As Single)**.

**Тело программы** – это сама программа, описание ее собственных переменных и операторы, реализующие алгоритм. Так, полное описание функции fact:

```
Function fact(n As Integer) As Integer
fact = 1
If n > 1 Then
    For i = 1 To n
        fact = fact * i
    Next
End If
End Function
```

Полное описание процедуры Sum:

```
Sub Sum(x As Integer, y As Integer)
Dim S As Integer
S = x + y
MsgBox S
End Sub
```

Программа, используемая другой программой, может иметь статус *подпрограммы*, и тогда ее описание может быть помещено внутри использующей программы, среди других ее описаний. Такая возможность предусмотрена, например, в языках Pascal, Modula, Modula 2, Algol. Эти языки разрешают вложенные описания подпрограмм (описание одной программы содержит описание другой, оно, в свою очередь, содержит описание третьей и т. д.). В языках C++, VBA программы имеют независимый статус, т.е. описываются отдельно от использующей программы, вложения описаний недопустимы. По описанию программы компилятор выделяет память, содержащую команды программы, память для переменных программы и память для ее формальных параметров (пока пустую).

Для того, чтобы программа, содержащая параметры, заработала, нужно ее *вызвать*, т.е. использующая программа должна содержать оператор вызова.

**Вызов программы** должен включать имя программы и набор конкретных значений, подставляемых вместо ее формальных параметров. Синтаксис вызова процедуры в VBA такой:

<имя процедуры> <список фактических параметров>

или

**Call** <имя процедуры> (<список фактических параметров>).

Вызов функции имеет вид:

<имя функции> <список фактических параметров>.

Список фактических параметров – это последовательность фактических параметров, разделенных запятой. В качестве фактического параметра может выступать значение, переменная, выражение, вызов функции.

*Вызов процедуры* – это независимый оператор, он не может стоять в правой части присваиваний, поскольку *не возвращает* значения.

*Вызов функции*, напротив, *должен* стоять в правых частях присваиваний, он может быть операндом выражения, т.к. возвращает значение, которое и используется при вычислении выражения.

Механизм реализации вызова таков. В точке вызова управление передается указанной программе; переменные, указанные в качестве имен формальных параметров получают конкретные значения, заданные через фактические параметры (говорят, фактические параметры *подставляются* вместо формальных). Программа исполняется с этими фактическими значениями параметров. Если программа является функцией, то ее значение (результат работы) возвращается в точку вызова и может быть использовано, например, в выражении. Если программа – процедура, то после ее исполнения просто происходит возврат управления в точку вызова, следующему за вызовом оператору.

Например, программа, содержащая вызовы процедуры Sum и функции fact, может быть такой (после знака ‘ идет однострочный комментарий):

**Sub** vysov ()

**Dim** k **As Integer**

k = fact (5)            ‘функция fact вызвана с фактическим параметром  
5, результатом 120

k = fact (4)+10+k    ‘функция fact вызвана с фактическим  
параметром 4, ее результатом 64

**MsgBox** k

Sum 5, 10            ‘процедура Sum вызвана с фактическими параметрами  
5 и 10

**Call** Sum (10, 20)    ‘другой способ вызова Sum с фактическими  
параметрами 10 и 20

Sum k, 2            ‘процедура Sum вызвана с фактическими параметрами  
k и 2

**End Sub**

Поскольку при вызове фактические значения замещают формальные, используя отведенную для формальных параметров память, *типы формальных и фактических параметров должны совпадать или быть совместимыми*. Так, вызов `Call Sum(2.5, 4)` не осуществится из-за типовой ошибки – число 2.5 не является целым.

Программы – процедуры и функции могут быть собраны в автономно существующие, например, в рамках проекта, более «крупные» единицы – модули. Такое понимание модуля характерно для VBA. В других языках программирования модулем часто называют программу, имеющую автономный статус: она может быть откомпилирована независимо от других программ. Для обеспечения этой возможности программе сообщаются только типы входных данных, в том числе – типы других программ, используемых в теле. Эти типы оформляются синтаксически специальным образом и называют «интерфейсом». Таким образом, для компиляции модуля необходимо описание его программы и интерфейсов используемых в нем других программ.

*Видимость переменных.* При всякой сборке возникает вопрос: могут ли программы использовать имена и значения переменных из других программ? Этот вопрос решается для каждого языка программирования в разделе, называемом *правилами видимости* переменных. Если «чужую» (не описанную в данной программе) переменную можно использовать, то, говорят, что эта переменная *видна* в программе.

Считается хорошим стилем, если программа достаточно автономна: все, что ей требуется извне, должно быть передано через ее фактические параметры. Но соблюдение этого принципа может привести к необозримому разрастанию списка параметров. Поэтому большинство языков программирования допускает использование так называемых *глобальных* данных, переданных через глобальные, внешние (extern-), общие (common-) переменные. Такие переменные видны всем программам модуля или приложения. В VBA переменные, описанные в модуле отдельно от всех его программ, делают их доступными в теле любой программы модуля. Например, описание:

```

Dim a As Integer, b As Integer
Sub P1 (x As Single, y As String) As Single
.....
    a = 7
.....
End Sub
Sub P2 ( x As Integer) As Boolean
.....
    b = a*3 + x
.....
End Sub

```

позволяет использовать глобальные переменные a и b в любом операторе программ P1 и P2 и изменять их первоначальные значения. Описание этих же пе-

ременных с ключевым словом **Public**, т.е. **Public a As Integer**, **b As Integer** делает их видимыми (доступными для использования и изменения) в любом модуле проекта.

Слова **Public** и **Private**, называемые иногда *спецификаторами доступа* регламентируют возможность использования переменных и объектов, к которым они приписаны, в других программных единицах: модулях, классах, проектах. Как правило, **Public** означает общедоступность объекта, **Private** – доступность только в конкретном модуле или классе. Если спецификатор не указан, то тип доступа считается **Private**.

Для VBA действуют следующие правила видимости:

- Переменные, описанные в процедуре или функции, видны только в ней. Они объявляются с помощью ключевого слова **Dim** или **Static** и называются *локальными*.
- Переменные, описанные в модуле вне описаний программ модуля с ключевыми словами **Dim** или **Private**, видны во всех программах этого модуля. В других модулях они не видны.
- Переменные, описанные в модуле вне описаний программ модуля с ключевым словом **Public**, видны во всех модулях (процедурах, функциях) проекта.

*Декомпозиция программ.* Разбиение программы на части – (под)программы имеет смысл, когда программа слишком объемна или содержит группы операторов, которые реализуют одну и ту же цель, но с разными данными или в разных местах исходной программы. Например, требуется найти максимальный элемент в массиве, значения которого в программе периодически меняются или необходимо определить истинность формулы для разных сочетаний значений ее переменных. В этом случае удобно группу операторов, выполняющую эту задачу, оформить в отдельную программу – процедуру или функцию. Данные, для которых эти операторы выполняли свою работу, могут передаваться в эту программу в качестве параметров. Тогда в исходной программе вместо группы операторов будет стоять вызов соответствующей процедуры. Например, «некрасивая» исходная программа вычисляет значение истинности формулы  $x \& y \vee x \vee \neg (x \rightarrow y)$  для значений переменных  $x=1, y=0$ ;  $x=0, y=1$ ;  $x=1, y=1$ :

```

Sub P ( )
Dim x As Boolean, y As Boolean, res As Boolean
x=1
y=0
res = x And y Or x Or Not ( x Imp y)
MsgBox res
x=0
y=1
res = x And y Or x Or Not ( x Imp y)
MsgBox res

```

```

x=1
y=1
res = x And y Or x Or Not( x Imp y)
MsgBox res
End Sub

```

Эта программа может быть преобразована в две программы более простого вида:

```

Sub P ( )
    MsgBox F(1, 0)
    MsgBox F(0, 1)
    MsgBox F(1, 1)
End Sub

Function F (x As Boolean, y As Boolean) As Boolean
    F = x And y Or x Or Not( x Imp y)
End Function

```

Помимо более простого (а, значит, и более понимаемого и легче тестируемого) вида программ такая декомпозиция дает возможность использовать функцию F в других программах и модулях. Все это позволяет делать приложения более технологичными, менее «кустарными», хорошо «вписываемыми» в другие программные комплексы.

## Глава 2. Введение в разработку приложений

Приложение – это программа или комплекс программ специального назначения, решающие класс задач какого-либо направления. Например, приложение Microsoft Word обрабатывает документы (иллюстрированные тексты), приложение Excel решает задачи, связанные с табличными вычислениями, приложение Paint позволяет создавать рисунки, обрабатывать графические изображения. Можно написать программу-приложение, вычисляющее, например, прибыль предприятия по заданным исходным экономическим показателям и т.п.

Важной частью разработки приложения является создание сервисных средств, обслуживающих диалог с пользователем приложения. Чаще всего такими средствами являются разнообразные формы – специальные окна, снабженные специальными элементами управления – кнопками, меню, полями ввода, переключателями, вкладками для удобного ввода исходных данных и вывода – визуализации результатов. Все языки программирования с приставкой «Visual» предоставляют для разработчиков приложений инструменты для создания таких форм, используя готовые «строительные блоки».

## 2.1. Объекты, свойства, методы, события в VBA

Одним из основных понятий VBA является объект. В VBA имеется более 100 встроенных объектов (рабочие книги (**Workbook**), рабочие листы (**Worksheet**), рабочие ячейки (**Cell**), формы (**UserForm**), элементы управления (**TextBox**, **CommandButton**, **Label** и др.), диалоги и т. д.). Объектом можно управлять с помощью программы на языке VBA. Каждый объект обладает некоторыми характеристиками, или свойствами. Например, диалог может быть видимым или невидимым в данный момент на экране. Можно узнать текущее состояние диалога с помощью свойства **Visible**. Шрифт и его тип, размер, цвет и т. д. также определяют различные свойства объекта, например содержимого ячейки. Изменяя свойства, можно менять характеристики объекта.

Таким образом, *свойство* представляет собой атрибут объекта, определяющий его характеристики, такие, как размер, цвет, положение на экране и состояние объекта, например доступность или видимость.

*Синтаксис применения свойства:*

Объект.Свойство

Объект содержит также список методов, которые к нему могут быть применены. Например, показать диалог (форму) на экране или убрать его можно с помощью методов **Show** и **Hide** соответственно.

Таким образом, *метод* представляет собой действие, выполняемое над объектом.

*Синтаксис применения метода:*

Объект.Метод

Из вышесказанного можно сделать вывод, что *объект* – это программный элемент, который имеет свое отображение на экране, содержит некоторые переменные, определяющие его свойства, и некоторые методы для управления объектом. Наиболее часто в VBA используются следующие встроенные объекты:

<b>Range</b>	Диапазон ячеек (может включать только одну ячейку)
<b>Cells</b>	Ячейка
<b>Sheet</b>	Лист
<b>Worksheet</b>	Рабочий лист
<b>DialogSheet</b>	Диалоговое окно

Большинство объектов принадлежит к группе подобных объектов. Эти группы называются *семействами*. Например, все рабочие листы рабочей книги образуют семейство, называемое **Worksheets**. Семейства можно использовать одним из двух способов: либо какое-либо действие совершается над всеми объ-

ектами семейства, (например, удалить – **Delete**), либо со ссылкой на семейство выбирается конкретный объект для работы с ним. Например, инструкция

**Worksheets** ("Первый")

выбирает рабочий лист «Первый» из активной рабочей книги. Другими примерами семейств являются:

<b>Sheets</b>	Листы
<b>DialogSheets</b>	Диалоговые окна
<b>DrawingObjects</b>	Графические объекты

Изменяя свойства, можно изменять характеристики объекта или семейства объектов. Установка значений свойств - это один из способов управления объектами. Для установки свойства необходимо ввести имя объекта, затем поставить точку и за ней – имя свойства. Далее должен следовать знак равенства и значение свойства. Синтаксис установки значения свойства объекта выглядит следующим образом:

Объект.Свойство = Выражение

В приведенном ниже примере для свойства **Value** диапазона ячеек Исходные\_данные устанавливается значение 0,1 (т.е. в ячейках этого диапазона будет записано число 0.1):

**Range** ("Исходные данные").**Value** = 0.1

Обратите внимание, что в MS Excel в представлении числа «0,1» используется запятая («,»), а в VBA - точка («.»).

В следующем примере в ячейку A2 вставляется формула путем изменения свойства **Formula** (Формула):

**Range** ("A2").**Formula**= "СУММ(A1:C1)"

Некоторые свойства являются неизменяемыми, т. е. допустимыми только для чтения. Иными словами, значение свойства можно узнать, но нельзя изменить. Например, для диапазона, состоящего из одной ячейки, свойства **Row** (Строка) и **Column** (Столбец) являются неизменяемыми. Другими словами, можно узнать, к какой строке и в каком столбце находится ячейка, но изменить ее положение путем изменения этих свойств нельзя. Для извлечения значения свойств объекта используется следующая конструкция:

Переменная = Объект.Свойство

В следующем примере переменной Процентная\_ставка присваивается значение из ячейки A1 текущего рабочего листа:

Процентная\_ставка = **Range** ("A1").**Value**



или

```
Процентная_ставка = Cells(1, 1).Value
```

Кроме свойств, как уже отмечалось выше, у объектов есть ряд методов, т. е. команд, применяемых к объекту. Например, у объекта – диапазон ячеек – имеется метод **Clear**, позволяющий очистить содержимое диапазона. Приводимый ниже пример показывает, как можно очистить диапазон Исходные\_данные:

```
Range ("Исходные данные") .Clear
```

В примере

```
Range ("A10:B12") .Select
```

выбирается диапазон ячеек A10:B12.

В Excel имеется много объектов, причем некоторые из них содержат другие объекты. Например, рабочая книга содержит рабочие листы, рабочий лист содержит диапазон ячеек и т. д. Объектом самого высокого уровня является **Application** (Приложение). Если вы изменяете его свойства или вызываете его методы, то результат применяется к текущей работе MS Excel. Например, можно завершить работу с Excel, применив метод **Quit** (Выход) к объекту **Application**:

```
Application.Quit
```

Как было отмечено, точка после имени объекта указывает на то, что далее следует имя свойства или метода. Но после точки можно указать и имя объекта для перехода от одного объекта к другому. Например, следующее выражение очищает 5-ю строку рабочего листа май в рабочей книге Отчет:

```
Application.Workbooks ("Отчет") .Worksheets ("Май") .Rows (5) .Delete
```

Таким образом, ссылки на объекты могут быть очень длинными. Приводимые выше примеры можно записать значительно короче:

- можно не писать имя объекта **Application**, так как это подразумевается по умолчанию;
- при работе с подобъектом уже активизированного объекта нет необходимости указывать содержащий его объект;
- VBA использует некоторые свойства и методы, которые возвращают объект, к которому они относятся.

Использование последнего правила позволяет быстро указывать нужный объект. Так, в следующем примере устанавливается значение активной ячейки.

```
ActiveCell.Value = "Да"
```

**ActiveCell** (Активная ячейка), **ActiveSheet** (Активный лист), **ActiveWorkbook** (Активная рабочая книга) и **Selection** (Выбор - указывает на выбранный объект) являются примерами свойств, возвращающих объект.

*Событие* представляет собой действие, распознаваемое объектом (например, щелчок мышью или нажатие клавиши), для которого можно запрограммировать отклик. События возникают в результате действий пользователя программы, или же они могут быть вызваны системой.

Процедуры обработки событий формы условно можно разделить на две группы: процедуры без параметров и с параметрами. К первой группе относятся, например, процедуры `Initialize`, `Load`, `Click` и `DoubleClick`. Эти процедуры имеют следующий синтаксис:

*Синтаксис:*

```
Sub UserForm_Событие ()
    Последовательность инструкций
End Sub
```

Например, процедура загрузки формы с отображением ее на экране:

```
Sub UserForm_Load()
    Show
End Sub
```

Ко второй группе процедур относятся `Unload`, `MouseDown`, `MouseUp` и `KeyDown`. Эти процедуры имеют следующий синтаксис:

*Синтаксис:*

```
Private Sub Form_Событие (СписокПараметров)
    Последовательность инструкций
End Sub
```

Процедура обработки события может быть связана с действием над любым элементом управления (кнопкой, полем, списком, флажком и т.п.). Имя такой процедуры может быть выбрано пользователем самостоятельно или формироваться по умолчанию следующим образом:

ИмяЭлементаУправления\_Событие (СписокПараметров)

Например,

```
Private Sub МояКнопка_Click()
    Последовательность инструкций
End Sub
```

## 2.2. Панель инструментов «Элементы управления»

Панель инструментов Элементы управления (рис.1), обычно содержит следующие кнопки:

- Выбор объектов.
- Надпись.
- Поле.
- Поле со списком.
- Список.
- Флажок.
- Переключатель.
- Выключатель.
- Рамка.
- Кнопка
- Набор вкладок.
- Набор страниц.
- Полоса прокрутки.
- Счетчик.
- Рисунок.
- RefEdit.

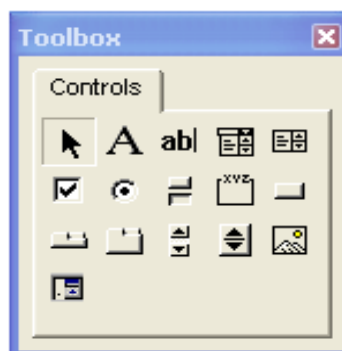


Рис. 1. Панель элементов

### 2.2.1. Поле (*TextBox*)

Элемент управления **TextBox** позволяет ввести в форму информацию, которая затем может быть использована в программе. Также элемент управления **TextBox** может служить и для вывода информации.

Для добавления любого элемента управления в форму необходимо нажать соответствующую кнопку на панели элементов управления, а затем щелкнуть по форме в требуемой позиции.

После создания любого элемента управления желательно сразу же присвоить ему новое имя, иначе будет использоваться имя, заданное по умолчанию, и при последующем изменении имени придется редактировать все процедуры, в которых имеется старое имя объекта. По умолчанию поля имеют имена `TextBox1`, `TextBox2` и т. д.

Новое значение имени любого элемента управления можно установить либо в окне свойств, вызвав его с помощью кнопки, либо непосредственно в коде процедуры, связанной с этим элементом управления. При присвоении имен полей используется следующее правило:

`TxtИмяОбъекта`

Например, `txtAge` – название поля, в которое вводится возраст, а `txtFirstName` – название поля, в которое вводится фамилия.

Для установки и получения содержимого поля используется свойство **Value**. Это свойство имеет тип **Variant**.

Например, установка значения свойства поля `txtFirstName` имеет вид:

`TxtMyFirstName.Value` = "Петров" 'в поле заносится значение Петров

Для получения значения элемента управления **TextBox** можно использовать следующие инструкции:

```
Dim X As Variant
X=txtFirstName.Value
```

Если нужно запретить изменение содержимого поля (например, объект **TextBox** применяется для отображения доступной только для чтения информации, такой, как имена файлов), следует «отключить» поле с помощью свойства **Enabled**, установив его равным значению **False**. Если значение свойства равно **True**, то изменение содержимого поля разрешено.

```
TextBox1.Enabled=False
```

### 2.2.2. Надпись (Label)

Элемент управления **Label** предназначен для вывода текста в форме, например для вывода заголовка для тех элементов управления, у которых отсутствует собственное свойство **Caption**. В качестве примера таких элементов можно назвать поле или рисунок в форме. В этом случае надпись находится около этого элемента управления, указывая его назначение.

Для задания текста надписи можно использовать свойство **Caption**. Например:

```
Label1.Caption="Адрес"
```

По умолчанию надписи имеют имена Label1, Label2 и т. д.

### 2.2.3. Кнопка (CommandButton)

Элемент управления **CommandButton** задает выполнение некоторого действия, например запуск, прерывание или останов некоторого процесса.

По умолчанию кнопкам присваиваются имена CommandButton1, CommandButton2 и т. д. Для изменения имени кнопки откройте окно свойств и введите новое имя в поле Имя (**Name**).

Можно задать текст, который будет выводиться на кнопке вместо установленного по умолчанию значения CommandButtonN (где N – порядковый номер данной кнопки в общем списке кнопок в соответствии с очередностью их создания). Для этого установите новое значение свойства **Caption**. Например:

```
CommandButton1.Caption="Моя новая программа"
```

Можно задать автоматическое изменение размеров элемента управления **CommandButton** с помощью свойства **AutoSize**. Если установлено значение этого свойства, равное **True**, то весь текст надписи, заданный свойством **Caption**, будет умещаться на кнопке. Например:

```
CommandButton1.AutoSize=True
```

Если в форме имеется несколько кнопок, то одну из них можно назначить применяемой по умолчанию. Например, при выводе окна сообщений, в котором содержится запрос на подтверждение удаления данных, кнопка **Да** обычно задана по умолчанию. Если по ошибке нажать клавишу «Пробел» или **Enter**, то вся информация будет уничтожена. Поэтому нужно назначить применяемой по умолчанию кнопку **Нет**. Для того чтобы назначить кнопку по умолчанию, нужно присвоить значение **True** ее свойству **Default**. Тогда свойству **Default** остальных кнопок формы автоматически будет присвоено значение **False**. Например:

```
CommandButton1.Default=True
```

С нажатием кнопки можно связать выполнение некоторого действия, если назначить эту кнопку некоторому событию, например, *Нажатие кнопки* (**Click**). Процедура обработки события **Click** не имеет параметров. Например, следующая процедура обработки события выводит в окне отладки **Debug** сообщение «Моя новая программа», после того как нажата кнопка **CommandButton1**.

```
Private Sub CommandButton1_Click()  
Debug.Print "Моя новая программа"  
End Sub
```

Можно изменить состояние кнопки: запретить пользователю нажатие кнопки, если оно приведет к опасным или нежелательным последствиям. Например, можно отключить кнопку печати, пока не выбран принтер. При запрете доступа кнопка выглядит серой. Для отключения объекта используется значение **False** свойства **Enabled**.

#### 2.2.4. Список (*ListBox*)

Элемент управления **ListBox** предназначен для хранения списка значений, из которого можно выбрать один или несколько элементов. По умолчанию списки имеют имена **ListBox1**, **ListBox2** и т. д.

Существуют следующие варианты выбора элементов в списке:

- 1 – один элемент,
- 2 – несколько последовательно расположенных элементов,
- 3 – несколько произвольно расположенных элементов.

Способ выбора элементов в списке определяется свойством **MultiSelect**, значение которого (числовое или заданное константой) можно указать в окне свойств или в программе.

Вариант	Значение	Константа
1	0	<b>fmMultiSelect</b>

2	1	<b>fmMultiSelectExtended</b>
3	2	<b>fmMultiSelectMulti</b>

Например, следующая инструкция позволяет выделить в списке несколько последовательно расположенных элементов:

```
ListBox1.MultiSelect=fmMultiSelectExtended
```

Для добавления новых элементов в список используется метод **AddItem**. При этом нужно задать параметр, который определяет строку с названием добавляемого в список элемента:

```
ListBox1.AddItem элемент
```

В качестве элемента может выступать, в частности, число, имя переменной, элемент массива ( $a(i)$ ), в этом случае в список добавляется их значение.

В следующей процедуре метод **AddItem** добавляет в список названия месяцев года:

```
Public Sub Months ()
ListBox1.AddItem "January"
ListBox1.AddItem "February"
ListBox1.AddItem "December"
End Sub
```

Для заполнения списка последовательными числами можно использовать процедуру

```
Public Sub NumberList ()
For i=1 To 20
ListBox1.AddItem i
Next i
End Sub
```

Пусть в программе требуется определить выбранные элементы списка, например, узнать их значения. Если в списке задан выбор только одного элемента, то свойство **Text** элемента управления **ListBox** содержит выделенный элемент, в противном случае свойство **Text** равно пустой строке. Свойство **ListIndex** содержит номер выделенного пункта в списке. Выбранный в списке элемент можно вывести, например, в окне отладки **Debug** с помощью инструкции

```
Debug.Print ListBox1.Text
```

Если известно, что в списке выделено несколько элементов, то необходимо

проверить каждый пункт списка, чтобы определить, выделен он или нет. Для этого используется свойство **Selected**, которое по индексу пункта возвращает значение **True**, если пункт выбран, и значение **False** – в противном случае.

Например, если необходимо найти сумму выделенных элементов в списке, то можно воспользоваться следующей инструкцией:

```

For i = 0 To listBox1.ListCount-1
If listBox1.Selected(i) = True Then
  S = S + listBox1.List(i)
End If
Next

```

Свойство **ListCount** содержит общее количество элементов (пунктов) в списке. При этом первый элемент имеет номер «0», а последний **ListCount**-1. Свойство **List** возвращает по номеру пункта его текст. Для удаления элемента из списка используется метод **RemoveItem**, при этом в качестве параметра метода указывается номер удаляемого пункта.

Например, для очистки списка может использоваться следующая процедура:

```

Public Sub NumberList ()
  For i=0 To listBox1.ListCount-1
    listBox1.RemoveItem i
  Next i
End Sub

```

### 2.2.5. Поле со списком (ComboBox)

Если используется поле со списком, то необходимый элемент можно выбрать сразу из списка или ввести его имя вручную для автоматического поиска. Текущее значение в элементе управления **ComboBox** отображается в поле, а список всех возможных значений выводится при нажатии кнопки со стрелкой. Элемент управления **ComboBox** отличается от элемента управления **ListBox** тем, что в нем можно явно выделить требуемое значение.

Существует два типа полей со списком. С помощью объектов первого типа можно ввести в поле данные, которые затем можно использовать как:

- критерий выбора элементов в списке; например, если список содержит названия месяцев года и вводится слово «May», то осуществляется перемещение на этот пункт списка;
- новое значение; например, для задания новой величины масштаба изображения; таким образом, в программе должен быть предусмотрен случай, когда введенного значения нет в списке.

Если элемент управления **ComboBox** относится ко второму типу, то для выбора элемента необходимо открыть список, нажав кнопку со стрелкой, а

затем указать в списке требуемый элемент списка. Этот элемент появится в поле элемента управления **ComboBox**.

Тип объекта **ComboBox** можно указать с помощью свойства **Style**, значение которого в программе указывается либо числом (0, 2), или константой:

Тип	Значение	Константа
Ввод данных	0	<b>fmStyleDropDownCombo</b>
Выбор значения	2	<b>fmStyleDropDownList</b>

Для заполнения поля со списком применяется метод **AddItem**. Для получения значения, содержащегося в поле элемента управления **ComboBox**, можно использовать свойства **Value** и **Text**.

Например, два следующих оператора выполняют одно и то же действие - выводят в окне отладки **Debug** текст, содержащийся в поле элемента управления **ComboBox**:

```
Debug.Print ComboBox1.Value
Debug.Print ComboBox1.Text
```

При присвоении значения свойству **Text** автоматически выполняются следующие действия:

- заданный текст выводится в поле элемента управления **ComboBox** (если заданный текст не является элементом списка, то выдается сообщение об ошибке);
- свойству **ListIndex** элемента **ComboBox** присваивается индекс элемента списка, соответствующего заданному значению.

### 2.2.6. Флажок (*Checkbox*)

Элемент управления **CheckBox** имеет вид маленького квадрата. С флажком можно связать некоторый заголовок (надпись). Если этот квадрат пуст, то при щелчке по нему в нем появляется галочка, и, наоборот, если квадрат был помечен галочкой, то при щелчке по нему галочка исчезает. Если флажок установлен (есть галочка), то свойство **Value** элемента управления **CheckBox** имеет значение **True**, в противном случае его значение **False**.

Состояние флажка используется в процедурах обработки события **Click** флажка или в других процедурах, где требуется анализ установки этого флажка.

Если флажок недоступен (блокирован свойством **Enabled**), то его значение (свойство **Value**) равно константе **Null**.

По умолчанию флажки имеют имена: **CheckBox1**, **CheckBox2** и т. д. С помощью свойства **Name** можно присвоить флажку новое имя. Свойство **Caption** позволяет установить текст (надпись), который будет появляться рядом с



элементом управления **CheckBox**. Если заголовок флажка очень длинный, то можно разместить его в нескольких строках, присвоив свойству **WordWrap** значение **True**.

Описать работу элемента управления **CheckBox** можно, например, с помощью следующих инструкций:

```
Public Sub CheckBox1_Click()
    If CheckBox1.Value=True Then
        ' инструкции 1
    Else
        ' инструкции 2
    End If
End Sub
```

### 2.2.7. Переключатель (*OptionButton*)

Элемент управления **OptionButton** предназначен для выбора одного варианта из нескольких. Переключатель может менять значения независимо от других или входить в состав группы переключателей. В любое время в группе может быть выбран только один переключатель. Отмена выбора одного элемента управления **OptionButton** при выделении другого в группе осуществляется автоматически.

По умолчанию переключатели имеют имена `OptionButton1`, `OptionButton2` и т. д.

Группировка переключателей может быть выполнена двумя способами:

- С помощью элемента управления Рамка (**Frame**). Все объекты управления **OptionButton**, расположенные в одной рамке, рассматриваются как члены одной группы. Для каждого набора переключателей должна использоваться своя рамка.

- С помощью свойства для группировки объектов – **GroupName**. При выборе элемента управления **OptionButton** отменяется выбор всех переключателей, значение свойства **GroupName** которых совпадает со значением того же свойства выделенного элемента управления **OptionButton**. При использовании свойства **GroupName** отпадает необходимость в создании элемента управления **Frame**. Свойство **GroupName** может быть установлено как в окне свойств, так и в программе.

Свойство **Value** активизированного переключателя имеет значение **True**.

Процедура, описывающая работу трех переключателей, может иметь вид:

```
Public Sub CheckOptionButton()
    If OptionButton1.Value Then
        ' инструкции 1
    ElseIf OptionButton2.Value Then
        ' инструкции 2
```

```

Elseif OptionButton3.Value Then
' инструкции 3
End If
End Sub

```

### 2.2.8. Рамка (*Frame*)

Элемент управления **Frame** предназначен для группирования элементов в форме. По умолчанию рамки имеют имена `Frame1`, `Frame2` и т. д. Установить новое значение имени рамки можно с помощью свойства **Name**. Свойство **Caption** определяет текст (надпись), который появляется вверху рамки. Например:

```
Frame1.Caption = "Варианты заданий"
```

## Глава 3. Лабораторный практикум

Для того чтобы открыть редактор VBA, выберите команду **Сервис, Макрос, Редактор Visual Basic** или нажмите комбинацию клавиш `<Alt>+<F11>`. Далее можно воспользоваться меню редактора: выбрать команды **Insert, Module**, еще раз войти в меню **Insert** и выбрать **Procedure**. Затем заполнить поле **Name** диалогового окна (ввести имя своей процедуры), и если программируется функция, то выбрать переключатель **Function** (рис. 2), нажать **Ok**.

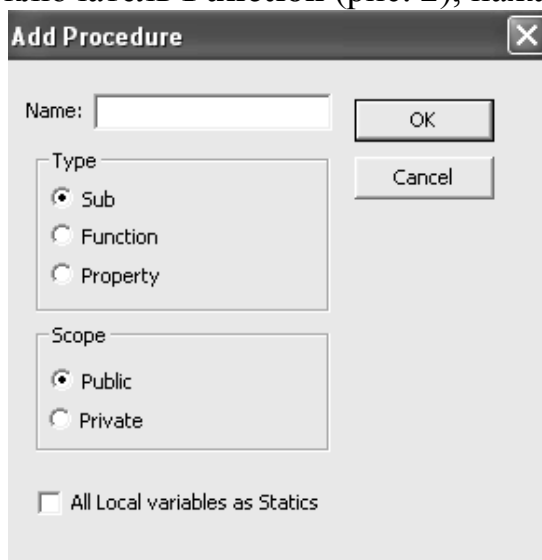


Рис. 2. Вид окна *Добавление процедуры*

На листе созданного модуля появятся стандартные строки начала и конца процедуры:


```


Public Sub ИмяПроцедуры ()

End Sub

```

Между ними можно набрать код (инструкции) программы.

Для запуска программы можно выбрать в меню **Run** команду **Run Sub/UserForm** или нажать кнопку  на панели инструментов.

Если в программе обнаружены ошибки, то компилятор выдает сообщение и отладчик отмечает строку с ошибкой желтым цветом. Чтобы снять это выделение, можно нажать кнопку  на панели инструментов. После этого необходимо исправить ошибку и снова запустить программу на выполнение.

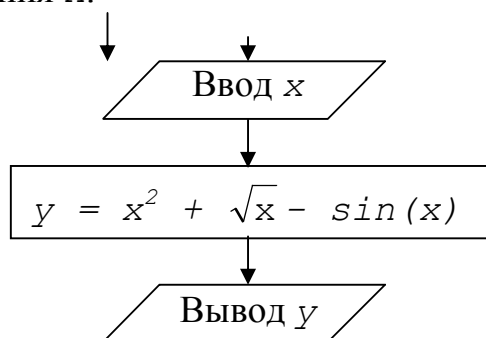
Более подробное изложение возможности отладчика программ **Debug** можно найти в справочнике VBA, приведенного в конце методического пособия.

## Лабораторная работа № 1. Линейные алгоритмы

**Задача 1.** Найти значения переменной  $y$ , заданные формулой

$$y = x^2 + \sqrt{x} - \sin(x) \text{ для трех различных значений } x.$$

**Решение.** Вход алгоритма: значение переменной  $x$ , выход: значение переменной  $y$ . Алгоритм состоит из трех шагов: 1) ввод значения  $x$ , 2) вычисление  $y$  для этого значения, 3) вывод полученного результата. Эти три шага следует также выполнить трижды – для каждого нового значения  $x$ . Блок-схема алгоритма для одного значения  $x$ :



Для иллюстрации возможных методов ввода будем задавать  $x$  разными способами: присваиванием, вводом с клавиатуры, вводом из ячейки листа Excel.

Определяем переменные, участвующие в программе: их имена и типы. Пусть это будут  $x$  и  $y$  вещественного типа `Single` (целый тип `Integer` для  $y$  брать нельзя, т.к. операция извлечения корня и функция  $\sin(x)$  могут дать в качестве результата вещественные числа). В начале программы следует поставить описания этих переменных.

Вывод реализуем оператором `MsgBox`. Его используем в двух режимах – просто вывод числа и вывод числа с текстом. Текст в этом случае нужно взять в двойные кавычки и соединить его с числом операцией `&` склейки строк. Аргумент оператора `MsgBox` имеет строковый тип (`String`), при выводе числового значения происходит автоматическое преобразование типов (здесь – из `Single` в `String`).

В программе участвуют встроенные функции – извлечение корня и вычисление синуса. В VBA это  $\text{Sqr}(x)$  и  $\text{Sin}(x)$ . Отметим, что извлечение корня из числа эквивалентно операции возведения этого числа в дробную степень (здесь – в степень  $1/2$ ).

```

Public Sub Task1()
Dim x As Single, y As Single           ‘описываем переменные
x = 2.4                                   ‘вводим x присваиванием значения
y = x^2 + Sqr(x) - Sin(x)                ‘вычисляем y, используя функцию sqr
MsgBox y                                 ‘выводим значение y без текста

x = InputBox ("Vvedi x")                  ‘вводим x с клавиатуры
y = x^2 + x^(1/2) - Sin(x)                ‘используем возведение в дробную степень
MsgBox "y=" & y                           ‘выводим значение y с текстом "y="
x = Cells(1, 1)                            ‘вводим x из ячейки A1 листа Excel
y = x ^ 2 + Sqr(x) - Sin(x)
MsgBox "y=" & y
End Sub

```

Перед тем, как запустить эту программу на счет, необходимо ввести какое-нибудь значение в ячейку A1 активного листа, иначе оператор  $x=\text{Cells}(1,1)$  присвоит  $x$  значение 0.

В этой программе есть недостаток: она не проверяет, что введенное значение положительно ( $x > 0$ ). Если пользователь программы введет отрицательное значение, то операции  $x^{(1/2)}$  и  $\text{Sqr}(x)$  приведут к ошибке, система выдаст сообщение о неверном аргументе этих операций. Программа будет гораздо технологичней, если после каждого ввода поставить защиту, например: проверку значения, выдачу сообщения и выход из процедуры:

```

if x < 0 Then
MsgBox "введено отрицательное значение переменной x"
Exit Sub
End if

```

### Задача 2. Обменять значения двух переменных.

**Решение.** Пусть имена переменных  $x$  и  $y$ . Задача состоит в том, чтобы переменная  $x$  получила значение переменной  $y$  и наоборот:  $y$  получил значение переменной  $x$ . Типичное *неправильное* решение задач с обменом значений:

```

x = y ‘неверно!
y = x

```

В этом случае переменная  $x$  получит значение  $y$ , но переменная  $y$  в присваивании  $y=x$  получит уже новое значение  $x$ , приобретенное в результате первого оператора  $x=y$ . Таким образом,  $y$  приобретет свое прежнее значение. Необходимо помнить, что в любых операциях обмена значений переменных

*требуется посредник* – дополнительная переменная для хранения промежуточного результата (старого значения одной из исходных переменных). Пусть такой переменной-посредником будет rab. Тогда правильное решение задачи:

rab = x

x = y

y = rab

Программа целиком:

**Sub** Обмен ()

**Dim** x **As Integer**, y **As Integer**, rab **As Integer**

x = 10

y = 7

rab = x

x = y

y = rab

**MsgBox** (x)

**MsgBox** (y)

**End Sub**

### *Задачи для самостоятельной работы:*

1. Найти значение переменной y при различных значениях A, B, C, вводимых с листа Excel, и значениях x, z, вводимых с клавиатуры:

$$\text{a) } y = \frac{A - 3B}{A + B} (A + C) \cos^2 x - z; \quad \text{b) } y = \frac{x A}{B} \sqrt{x^C + 1} \frac{1}{A^x + 1};$$

2. Задать длину ребра куба. Найти объем куба и площадь его поверхности.

3. Задать три действительных положительных числа. Найти среднее арифметическое и среднее геометрическое этих чисел.

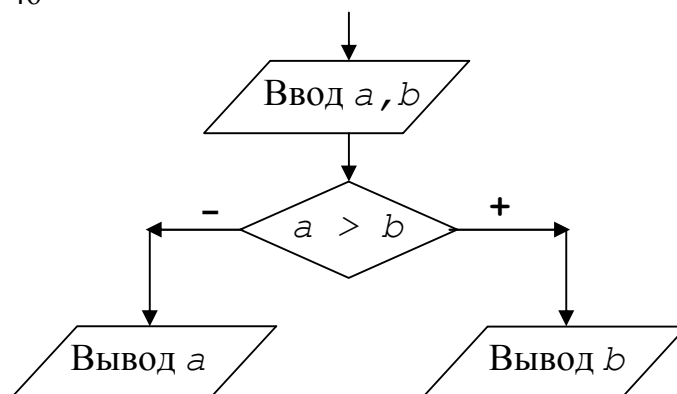
4. Определить высоту треугольника, если его площадь равна S, а основание больше высоты на величину A.

## **Лабораторная работа № 2. Разветвляющиеся алгоритмы**

### ***Задача 3. Найти минимальное из двух чисел, вводимых с клавиатуры.***

**Решение.** Вход алгоритма: два неравных числа, назовем их a и b. Выход – число, меньшее другого числа. Алгоритм: сравнить значения a и b: если a > b, то вывести в качестве результата значение a, иначе - значение b. Блок-схема алгоритма:

Определим тип переменных  $a$  и  $b$ . Для простоты используем целые числа, т.е. тип – Integer. По условию задачи значения этих переменных вводятся с клавиатуры, поэтому используем функцию `InputBox` с текстовым аргументом. Для вывода используем функцию `MsgBox`.

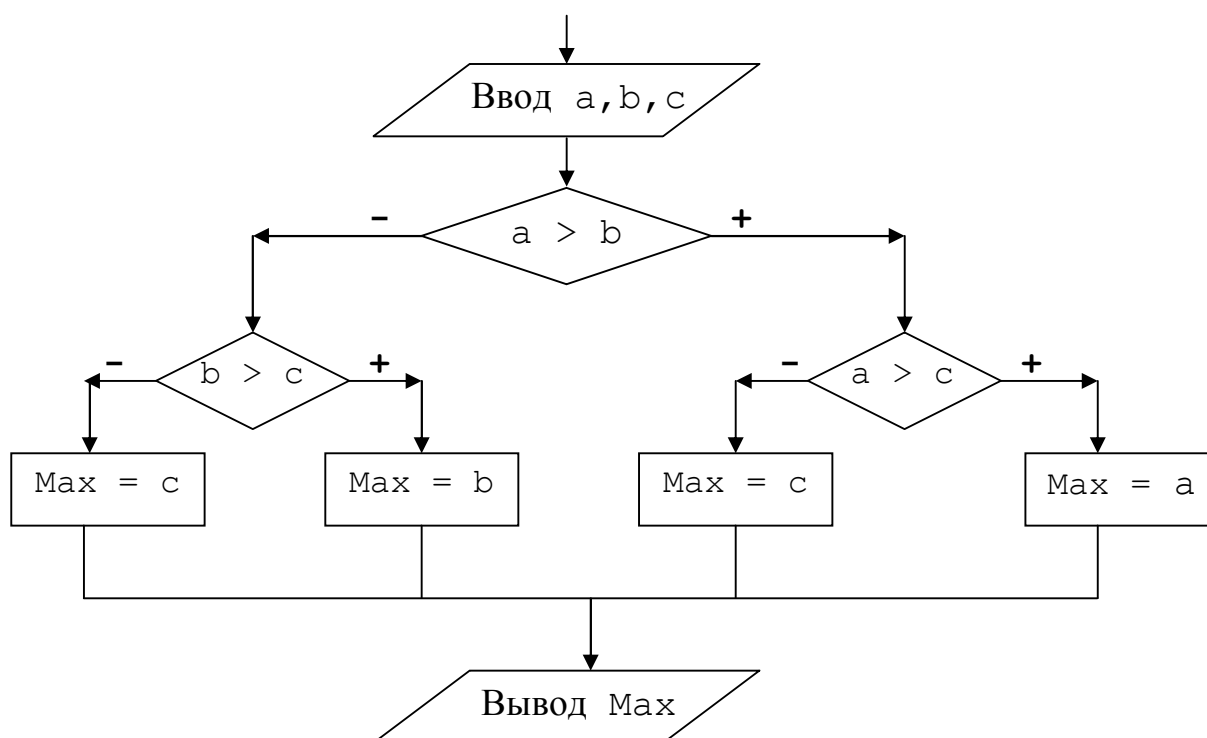


```

Public Sub min2()
Dim a As Integer, b As Integer
a = InputBox("Введи a")
b = InputBox("Введи b")
If a > b Then
MsgBox "min = " & b
Else
MsgBox "min = " & a
End If
End Sub
  
```

**Задача 4. Найти максимальное из трех чисел, вводимых с клавиатуры.**

**Решение.** Вход алгоритма: три произвольных числа, назовем их  $a$ ,  $b$ ,  $c$ . Выход: наибольшее из них. Алгоритм состоит в попарном сравнении этих чисел и выборе максимального в паре. Программа иллюстрирует применение вложенных операторов `If... Then... Else...`, причем новые `If... Then... Else...` появляются как в части `Then`, так и в части `Else`. Блок-схема алгоритма:



В программе тип всех переменных для простоты выбран Integer. Для сокращения двустрочной записи:

```
Else  
Max = c
```

использована однострочная **Else**: Max = c, допустимая в VBA.

```
Public Sub max3()  
Dim a As Integer, b As Integer, c As Integer, Max As Integer  
a = InputBox ("Vvedi a")  
b = InputBox ("Vvedi b")  
c = InputBox ("Vvedi c")  
If a > b Then  
    If a > c Then                ‘вложенный условный оператор If...  
        Max = a  
    Else: Max = c  
    End If                        ‘конец вложенного If...  
Else  
    If b > c Then                ‘вложенный условный оператор If...  
        Max = b  
    Else: Max = c  
    End If                        ‘конец вложенного If...  
End If  
MsgBox Max  
End Sub
```

### ***Задачи для самостоятельной работы:***

1. Вывести на печать переменные A, B, C в порядке их возрастания. Значения переменных: а) ввести с клавиатуры; б) взять из диапазона B2:B4 листа Excel; в) задать оператором присваивания.

2. Даны значения x, y, z. Определить, могут ли они быть сторонами: а) обычного треугольника; б) равнобедренного треугольника.

3. Список доходов клиентов расположен в диапазоне A2:A11 листа Excel. Определить налог конкретного клиента (его номер в списке ввести с клавиатуры), если налоговое начисление составляет 13% от дохода при доходе меньшем 5000 руб., 20% от дохода, если он находится в промежутке от 5000 до 40000 руб. и 30%, если доход превышает 40000 руб.

4. Даны действительные числа x, y, z. Получить минимальное из них по модулю.

## **Лабораторная работа № 3. Циклические алгоритмы**

***Задача 5. Найти среднее арифметическое четных чисел из N чисел, вводимых с клавиатуры.***

**Решение.** Среднее арифметическое – это значение суммы чисел, деленной на их количество. Поэтому для решения задачи потребуются переменные

для накопления суммы и количества четных чисел. Пусть это будут  $S$  и  $k$ , для простоты – целого типа. Кроме того, в программе нужно ввести  $N$  или задать его с помощью присваивания. Для проверки на четность можно использовать функцию `Mod`, дающую целочисленный остаток от деления ее левого операнда на правый: результат операции  $x \text{ Mod } y$  равен целому числу - остатку от деления  $x$  на  $y$ ; так он будет 0, если  $x$  делится на  $y$ . Если  $x$  или  $y$  не целые числа, то они предварительно будут округлены до целого. Основу алгоритма составит циклическая процедура: ввести число, проверить: если оно четное, то добавить его к сумме  $S$  и увеличить счетчик четных чисел  $k$  на 1. Эти операторы нужно повторить  $N$  раз. На выходе из цикла сумма четных чисел и их количество будут посчитаны, останется только разделить  $S$  на  $k$  и запомнить результат. Для хранения результата возьмем переменную  $r$ , ее тип – `Single` – действительное число, т.к. искомое среднее – результат деления – не всегда будет целым числом:

```

Sub Среднее ()
Dim S As Integer, r As Single, k As Integer, a As Integer
    N = InputBox ("Введите N - количество чисел")
    S = 0
    k = 0
    For i = 1 To N
        a = InputBox ("Введите число")
        If a Mod 2 = 0 Then
            S = S + a
            k = k + 1
        End If
    Next
    r = S / k
    MsgBox (r)
End Sub

```

Обратите внимание, что переменные  $S$  и  $k$  предварительно обнуляются («чистятся»), причем *до* начала цикла. Чистку необходимо делать, т.к. при переводе программы на машинный язык под переменные отводится память, в которой может что-нибудь уже находиться, какая-либо старая информация. Если убрать оператор  $S = 0$ , то в последующем присваивании  $S = S + a$  при вычислении правой части к значению  $a$  будет добавлено неизвестное значение  $S$  и результат будет неверный. Типичная ошибка, когда операции обнуления помещают в тело цикла, после заголовка:

```

For i = 1 To N
    S = 0
    k = 0
    a = InputBox ("Введите число")
    If a Mod 2 = 0 Then
        S = S + a

```



```

k = k + 1
End If
Next

```

Тогда чистка происходит на каждом шаге исполнения цикла, поэтому после завершения цикла в  $S$  окажется лишь последнее введенное четное  $a$ , и  $k$  будет равен 1.

**Задача 6.** *Посчитать произведение чисел, вводимых с клавиатуры до тех пор, пока не встретится 0.*

**Решение.** Здесь заранее не известно, сколько чисел будет введено, поэтому лучше воспользоваться циклом `While`:

```

Sub Произведение ()
Dim a As Integer, P As Integer
a = InputBox("Введите ненулевое число")
P = 1
While a <> 0
P = P * a
a = InputBox("Введите число")
Wend
MsgBox (P)
End Sub

```

Обратите внимание, что «чистка» переменной  $P$  заключается в присвоении ей значения 1, т.к.  $P$  участвует в произведении  $P = P * a$  и обнуление  $P$  привело бы к нулевому результату всей программы. Исполнение цикла продолжается до тех пор, пока не введен 0 в переменную  $a$ . В первой строке мы потребовали, чтобы вначале было введено ненулевое число. А что, если все-таки пользователь программы ввел 0? Тогда цикл не проработает ни разу и результат будет  $P=1$ . Такой же результат будет, если ввели 1, а затем 0. Как различить эти случаи? «Защититься» от первого нуля можно, поставив, например, «оберегающий» оператор `Until` с проверкой на ноль:

```

Sub Произведение ()
Dim a As Integer, P As Integer
Do
a = InputBox("Введите ненулевое число")
Loop Until a <> 0
P = 1
While a <> 0
P = P * a
a = InputBox("Введите число")
Wend
MsgBox (P)
End Sub

```

Здесь первый оператор цикла не позволит продолжить программу, если вводятся нули: условие выхода из цикла  $a < 0$ .

**Задача 7. Найти максимальное из 10-ти чисел, вводимых с клавиатуры.**

**Решение.** При нахождении максимума в последовательности значений, нужно определить начальное значение переменной (*max*), в которой будет храниться этот максимум. Затем каждое число в последовательности (здесь - каждое введенное число) сравнивается со значением *max* и, если это число превышает *max*, то оно теперь считается максимальным и поэтому заносится (присваивается) в *max*, стирая предыдущее значение. Таким образом, основной оператор алгоритма решения – это цикл, в котором тело составляют два действия: ввод нового значения и проверка, не является ли это значение максимальным (из тех, что уже были введены). По окончании цикла (когда все числа уже введены и проверены) в *max* будет находиться наибольшее из них.

Что взять в качестве начального значения *max*? Верное решение – взять любое из анализируемой последовательности, например, первое. Неверное решение – взять «с потолка», например, 0. Ноль сгодится, если вводятся только положительные числа (тогда любое из них «закроет» первоначальный максимум). Но, если могут быть введены только отрицательные числа, то этот 0 и окажется максимальным, хотя и не был введен. Ответ будет неверным.

Итак, возьмем в качестве начального значения первое из вводимых чисел и откроем цикл с проверкой оставшихся 9-ти чисел на максимум. Поскольку число шагов известно, проще воспользоваться циклом For:

```

Sub МаксЧисло ()
Dim max As Single, a As Single    ‘min переменных – Single
max = InputBox ("Введите число")
For i = 1 To 9
    a = InputBox ("Введите число")
    If a > max Then
        max = a
    End If
Next
MsgBox (max)
End Sub

```

Аналогично решаются задачи на поиск минимума, нужно только заменить знак неравенства на < и переобозначить переменную (для ясности): вместо *max* взять, например, *min*.

***Задачи для самостоятельной работы:***

1. Найти максимальный из отрицательных элементов среди произвольных 20 чисел, вводимых с клавиатуры.
2. Найти первый отрицательный член последовательности  $\sin(\text{tg}(n))$  для  $n$ , изменяющегося так:  $n=1, 2, 3, \dots$

3. Вычислить положительные значения функции  $y = \sin(x) + 4\cos(x-2)$  для  $x$ , изменяющемся на отрезке  $[-15, 10]$  с шагом 1.

4. Найти количество чисел, кратных трем, из последовательности, вводимой с клавиатуры до тех пор, пока не встретится ноль.

#### **Лабораторная работа № 4. Итерации (рекуррентные соотношения) и вложенные циклы**

Задачи такого типа можно представить как определение некоторой величины  $Y$ , которая является результатом последовательности вычислений  $Y_i = F(Y_{i-1})$ , т.е. каждое последующее значение  $Y$  вычисляется на основе его предыдущих значений. Процесс заканчивается при достижении какого-либо условия или конкретного значения.

Для решения подобных задач нужно понять закономерность образования каждого последующего значения, т.е. задать функцию (или выражение)  $F$ . Очевидно, что процесс вычисления является циклом и для начала проще всего определить его тело: какие действия повторяются, какие величины и на сколько изменяются на каждом шаге. После этого будет яснее условие продолжения (окончания) цикла.

**Задача 8. Вычислить значение  $Y$ , определяемое формулой:**

$$Y = \sqrt{1024 + \sqrt{512 + \dots + \sqrt{4 + \sqrt{2}}}}$$

**Решение.** Алгоритм будет понятнее, если представить числа в этом выражении как степени двойки:

$$Y = \sqrt{2^{10} + \sqrt{2^9 + \dots + \sqrt{2^2 + \sqrt{2^1}}}}$$

Вычисление  $Y$  можно организовать тогда, используя цикл из десяти шагов. На каждом шаге  $i$  выполняется операция: к  $2^i$  прибавляется значение, вычисленное на предыдущем шаге, и из результата извлекается квадратный корень. Таким образом, результат  $i$ -го шага:  $Y = \sqrt{2^i + Y}$ . Осталось только задать начальное значение  $Y=0$ . Программа тогда:

```

Sub Корень ( )
Dim Y As Single
Y=0
For i = 1 To 10
    Y = (2^i + Y) ^ (1/2)
Next
MsgBox Y
End Sub

```

Здесь символ  $^$  означает операцию возведения в степень:  $a^b$  в языке VBA можно представить как  $a^b$ , тогда и  $\sqrt{a}$  ( $a$  в степени  $1/2$ ) есть  $a^{(1/2)}$ .

**Задача 9.** Вычислить значение  $Y$ , определяемое формулой:

$$Y = \sum_{n=0}^{10} (-1)^n \frac{x^2}{2(n+1)!}.$$

**Решение.** Вычисление, очевидно, реализуется циклическим алгоритмом, состоящим из 11 шагов. На каждом шаге к сумме  $Y$  прибавляется очередной,  $Y_i$ , элемент ряда:  $Y = Y + Y_i$ . Сложность этой задачи можно понизить, если выявить зависимость каждого последующего  $Y_i$  от предыдущего  $Y_{i-1}$  значения. Величина  $x$  от шага к шагу не изменяется, в такой постановке задачи ее просто нужно один раз ввести, например, с клавиатуры.

Итак, для всякого  $i$  от 0 до 10,  $Y_i = (-1)^i \frac{x^2}{2(i+1)!}$ . Соответственно,

$Y_{i-1} = (-1)^{i-1} \frac{x^2}{2i!}$ . Разделим  $Y_i$  на  $Y_{i-1}$ , учитывая, что, по определению факториала,  $(i+1)! = i!(i+1)$ . После сокращений получим:  $\frac{Y_i}{Y_{i-1}} = \frac{-1}{i+1}$ , отсюда следует,

что  $Y_i = \frac{-1}{i+1} Y_{i-1}$ . В суммируемой последовательности последний  $Y_{i+1}$  — это  $Y_{10}$ , а первый  $Y_i$  — это  $Y_0$ . Вычислим это начальное значение, подставив в формулу

для  $Y_i$  вместо  $i$  — ноль:  $Y_0 = \frac{x^2}{2}$ . Окончательно, программа:

```

Sub MyY ( )
Dim Y As Single, yi As Single, x As Single
x = InputBox("Введите значение x")
yi = (x^2)/2           'значение начального -Y0 элемента ряда
Y = yi                 'начальное значение суммы
For i = 1 To 10
    yi = (-1/(i+1))*yi 'считаем очередной Yi -й элемент
    Y = Y + yi         'считаем сумму
Next
MsgBox Y
End Sub

```

Отметим, что (частая ошибка!) замена двух операторов тела цикла на один:  $Y = Y + (-1/(i+1)) * yi$  приведет к тому, что на каждом шаге цикла к сумме  $Y$  будет прибавляться одно и то же значение  $yi$ , равное  $Y_0$ , ведь  $yi$  тогда никак не изменяется. В программах, использующих зависимость  $i$ -го элемента в итерации от  $i-1$ -го, необходимо запоминать это предыдущее, вычисленное на  $(i-1)$ -м шаге значение, чтобы потом использовать именно его.

**Задача 10.** Для каждого числа  $b$  из 20-ти чисел, вводимых с клавиатуры определить наименьшее неотрицательное целое  $k$  такое, что  $b < 2^k$ .

**Решение.** Алгоритм составляет цикл из 20-ти шагов, на каждом из которых: вводится очередное число; проверяется, какую (наименьшую) степень

двойки оно не превышает; выводится показатель этой степени. Как получить нужную степень двойки? Пусть число  $b$  введено. Начинаем процесс проверки с  $2^0$ , т.е.  $b < 2^0$ ? Если да, процесс окончен, ответ  $k = 0$ . Если нет, проверяем  $2^1$ , т.е.  $b < 2^1$ ? Если да, процесс окончен, ответ  $k = 1$ . Если нет, проверяем  $2^2$ ,  $b < 2^2$ ? И так до тех пор, пока не найдем нужное  $k$ . Таким образом, на каждом шаге внешнего цикла (ввод очередного числа) работает внутренний цикл, перечисляющий степени  $k$  двойки. Условие выхода из этого внутреннего цикла – число  $b$  больше или равно  $2^k$ . Программа:

```

Sub MinK ( )
Dim b As Integer, k As Integer
For i = 1 To 20          ‘внешний цикл вводит очередное b и
                           печатает результат
    b = InputBox("Введите целое число")
    k = 0                  ‘для каждого b начинаем поиск k с нуля!
    While b >= 2 ^ k      ‘внутренний цикл; условие цикла на каждом
                           его шаге пересчитывается
        k = k + 1         ‘увеличиваем показатель степени
    Wend
    MsgBox "Для b =" & CStr(b) & " k=" & CStr(k)
Next
End Sub

```

Здесь операция & склейки строк при выводе текста потребовала преобразования типов CStr из целого в строковый, т.к. она определена для аргументов-строк.

### *Задачи для самостоятельной работы:*

1. Вычислить выражение:  $y = \sqrt{99 + \sqrt{96 + \sqrt{9 + \dots + \sqrt{6 + \sqrt{3}}}}$
2. Найти сумму бесконечного ряда с точностью до  $\varepsilon > 0$ :  $\sum_{n=1}^{\infty} \frac{1}{n(n+1)(n+2)}$ .
3. Найти сумму  $k$  первых членов ряда:  $\sum_{n=1}^k \frac{5^n x^{n-1}}{(2+n)!}$ .
4. Для каждого числа  $a$  из 20-ти чисел, вводимых с клавиатуры определить наибольшее целое  $k$  такое, что  $a > 3^k$ .

## **Лабораторная работа № 5. Задачи с данными – одномерными (линейными) массивами**

### **Задача 11. Ввод и вывод линейного массива.**

Ввод массива фиксированной длины можно осуществить с клавиатуры или, в VBA, также с листа Excel. Для ввода проще всего использовать цикл For

с числом шагов, равным длине (количеству элементов) массива. Пусть, например, требуется ввести массив М из 10 целых чисел.

**Ввод массива с клавиатуры:**

```
For i = 1 To 10
    M(i) = InputBox("Введите элемент массива")
Next
```

**Ввод массива с листа Excel** (массив расположен на листе в диапазоне, например, A1:A10):

```
For i = 1 To 10
    M(i) = Cells(i, 1) 'номер строки меняется в цикле, столбец первый (A)
Next
```

**Вывод массива** в VBA нагляднее всего реализовать непосредственно на лист Excel, например, в диапазон B1:B10:

```
For i = 1 To 10
    Cells(i, 2) = M(i) 'номер строки меняется в цикле, столбец второй (B)
Next
```

**Вывод массива** с помощью оператора **MsgBox**:

```
For i = 1 To 10
    MsgBox M(i)
Next
```

**Задача 12. В массиве из 10 чисел найти максимальное значение.**

**Решение.** Задача отличается от разобранных в [Задаче 7](#) только тем, что вначале последовательность вводится в массив (одномерный), а затем анализируется: каждый элемент A(i) массива сравнивается с уже полученным на предыдущих шагах максимумом. В качестве начального значения можно взять любой из A(i), но удобнее A(1) – циклическую проверку можно тогда начать со второго элемента, сократив число шагов:

```
Sub maxmas()
    Dim i, max, A(10) As Integer
    For i = 1 To 10
        A(i) = InputBox("Введите число - элемент массива")
    Next
    max = A(1)
    For i = 2 To 10
        If A(i) > max Then
            max = A(i)
        End If
    Next
    MsgBox (max)
End Sub
```

Здесь первый цикл выполняет ввод элементов массива, а второй – поиск максимума. В данной задаче оба этих действия можно совместить, воспользовавшись одним циклом, например:

```

Sub maxm()
Dim i, max, A(10) As Integer
max = InputBox ("Введите число – первый элемент массива")
For i = 2 To 10
    A(i) = InputBox ("Введите число – элемент массива")
    If A(i) > max Then
        max = A(i)
    End If
Next
MsgBox (max)
End Sub

```

**Задача 13.** Поменять местами первый положительный и первый отрицательный элементы массива  $A$  из 10 целых чисел.

**Решение.** Алгоритм состоит из четырех частей: 1) ввод массива, 2) поиск в нем первого положительного элемента, 3) поиск первого отрицательного элемента, 4) обмен их значений. Для демонстрации результата необходима еще одна часть – 5) вывод полученного массива. Для наглядности будем вводить массив с листа Excel (например, из диапазона A1:A10) и выводить результат на тот же лист (например, в диапазон B1:B10).

Поиск первого положительного элемента проведем, перебирая в цикле значения  $A$ . Как только встретится  $A(k) > 0$ , нужно запомнить его индекс (номер  $k$ ) и сразу выйти из этого цикла. Обратите внимание, что если продолжить цикл, то мы в результате получим номер не первого, а последнего положительного элемента массива.

Аналогично проводим поиск первого отрицательного элемента массива. Пусть это будет  $A(j)$ . Индекс  $j$  запоминаем.

Теперь осталось обменять значения  $A(k)$  и  $A(j)$ , используя переменную-посредника. Вся программа выглядит так:

```

Sub obmen()
Dim i, j, k, rab, A(10) As Integer
For i = 1 To 10      'вводим массив из листа Excel:
    A(i) = Cells(i, 1)
Next
For i = 1 To 10      'находим первый положительный элемент:
    If A(i) > 0 Then
        k = i
    Exit For
    End If
Next
For i = 1 To 10      'находим первый отрицательный элемент:

```

```

If A(i) < 0 Then
    j = i
    Exit For
End If
Next

```

```

rab = A(k)           'меняем значения
A(k) = A(j)
A(j) = rab

```

```

For i = 1 To 10    'выводим массив на лист Excel:
    Cells(i, 2) = A(i)
Next
End Sub

```

#### ***Задача 14. Отсортировать массив чисел по возрастанию.***

**Решение.** *Сортировка* – распределение элементов множества по группам в соответствии с определенными правилами. Например, сортировка «по невозрастанию» – это сортировка элементов массива, в результате которой получается массив, каждый элемент которого, начиная со второго, не больше стоящего от него слева. Существует достаточно большое число методов сортировки. Приведем лишь простейшие из них.

*Линейная сортировка (отбором).*

Пусть необходимо упорядочить массив по возрастанию (убыванию) элементов. Алгоритм:

1. Просматриваем элементы, начиная с 1-го. Ищем минимальный (максимальный). Меняем его местами с 1-м элементом.
2. Просматриваем элементы, начиная со 2-го. Ищем минимальный (максимальный). Меняем его местами со 2-м элементом.
3. И т. д. до предпоследнего элемента.

Программа такой сортировки:

```

Sub сортировка_отбором()
Dim c(1 To 100) As Single
Dim k As Integer, i As Integer, j As Integer
Dim vr As Single
k = InputBox("Введите количество элементов <= 100")
If k > 100 Then Exit Sub
Cells(1, 1) = k
For i = 1 To k
    c(i) = InputBox("введите" & i & " элемент")
    Cells(i, 2) = c(i)           'Вывод исходного массива на лист
Next
For i=1 To k-1 'Двигаемся по массиву, сокращая неотсортированную часть
    For j=i+1 To k 'Сравниваем по очереди i-й элемент неотсортированной

```



*части массива со всеми, от  $i+1$ -го до конца*

**If**  $c(j) < c(i)$  **Then** *'если в неотсортированной части массива нашли элемент, больший  $i$ -го, то меняем их местами:*

```
vr = c(i)
c(i) = c(j)
c(j) = vr
```

**End If**

**Next**

**Next**

**For**  $i = 1$  **To**  $k$

Cells( $i$ , 3) =  $c(i)$  *'Вывод результата на лист Excel*

**Next**

**End Sub**

*Сортировка методом «пузырька».*

Данный метод получил такое название по аналогии с пузырьками воздуха в стакане воды. Более «легкие» (максимальные или минимальные) элементы постепенно «всплывают». В отличие от линейной сортировки, сравниваются только пары соседних элементов, а не каждый элемент со всеми (поэтому такая сортировка выполняется за меньшее число шагов и, следовательно, быстрее).

Алгоритм:

1. Последовательно просматриваем пары соседних элементов массива ( $c$ ).
2. Если для соседних элементов выполняется условие  $c[i-1] < c[i]$ , то значения меняются местами.

Фрагмент программы сортировки методом «пузырька»:

**For**  $i = 2$  **To**  $k$

**For**  $j = k$  **To**  $i$  **Step**  $-1$

**If**  $c(j-1) < c(j)$  **Then** *'вытеснить элемент справа, тогда пузырек всплывает влево*

```
vr = c(j - 1)
c(j - 1) = c(j)
c(j) = vr
```

**End If**

**Next**

**Next**

**Задача 15.** *Дан массив целых чисел  $A(10)$ . Посчитать количество разных элементов в нем.*

**Решение.** Ниже приведена программа, разберите по ней алгоритм решения этой задачи самостоятельно:

**Sub** РазныеЭлементы()

**Dim**  $A(10)$  **As Integer**,  $k$  **As Integer**,  $k0$  **As Integer**

**For**  $i = 1$  **To** 10

```

A(i) = InputBox("Введите значение " & i & "-го элемента массива")
Next
k = 1
For i = 2 To 10
    k0 = 1
    For j = 1 To i - 1
        If A(j) = A(i) Then
            k0 = 0
            Exit For
        End If
    Next
    k = k + k0
Next
MsgBox "Разных элементов в массиве " & k
End Sub

```

### *Задачи для самостоятельной работы:*

1. Дано целое число N и набор из N целых чисел. Найти номера первого и последнего минимального элемента из данного набора и вывести их в указанном порядке.
2. Найти минимальный из элементов массива A(10), принадлежащий интервалу (2;14).
3. Составить массив B из неположительных элементов массива A(15).
4. Дано положительное число B и набор из десяти чисел. Вывести максимальный из тех элементов набора, которые больше B, а также его номер. Если чисел, больших B, в наборе нет, то дважды вывести 0.

### **Лабораторная работа № 6. Вычисление значений формулы, зависящей от нескольких переменных, каждая из которых изменяется на своем интервале и со своим шагом**

Речь идет о задачах типа: вычислить  $v = f(x,y,z)$ , где  $x \in [a,b]$ ,  $\Delta x = h_1$ ;  $y \in [c,d]$ ,  $\Delta y = h_2$ ;  $z \in [e,k]$ ,  $\Delta z = h_3$ . Если переменные, как здесь, не зависят друг от друга, то нужно использовать столько вложенных циклов, сколько переменных участвует в формуле, здесь - три. Вложенные циклы позволяют связать каждое значение с каждым, т.е. перебрать все возможные варианты значений  $x$ ,  $y$ ,  $z$ . Каким типом цикла пользоваться? Если границы интервала включены, то проще всего – циклом For, например:

```

For x = a To b Step h1
    For y = c To d Step h2
        For z = e To k Step h3
            v = f (x, y, z)
        Next
    Next
Next

```

При этом безразлично, в каком порядке идут эти циклы (можно начинать с  $y$  или  $z$ ) – все равно будут перечислены все наборы значений  $x, y, z$ . Если же границы интервалов не включаются, например,  $y \in [c, d)$ , то для  $y$  можно использовать цикл While с условием  $y < d$ :

```

For x = a To b Step h1
    y = c
    While y < d
        For z = e To k Step h3
            v = f (x, y, z)
        Next
        y = y + h2
    Wend
Next

```

Если в исходной задаче переменные зависят друг от друга, например,  $y = 2x + 4$ , то количество циклов уменьшается на число зависимых переменных, здесь – на 1. Тогда первый вариант задачи (с поправкой на новый  $y$ ) будет иметь вид:

```

For x = a To b Step h1
    y = 2*x + 4
    For z = e To k Step h3
        v = f (x, y, z)
    Next
Next

```

Таким образом, для каждого  $x$  будет вычислен соответствующий  $y$ . Здесь порядок вычисления уже играет роль: значение  $y$  не может быть посчитано, если значение  $x$  не известно, поэтому операции с переменной  $y$  должны идти *после* операций, устанавливающих значение переменной  $x$ .

**Задача 16. Вычислить значения  $v$ , определяемые формулой:**

$$v = \begin{cases} xy + z^2 + w & \text{при } x < 3 \\ x^2 + xw - z & \text{при } x \geq 3 \end{cases}; \quad x \in [2; 4], \quad \Delta x = 1;$$

$$y \in [2; 6], \quad \Delta y = \begin{cases} 0,5 & \text{при } y < 3 \\ 1 & \text{при } y \geq 3 \end{cases}; \quad z = 1, 2; 1, 5; 3; \quad w_{i+1} = 0,5w_i, \quad w_1 = 2; \quad 1 \leq i \leq 4$$

**Решение.** Проанализируем условие. Переменные-аргументы в формуле  $v=f(x, y, z, w)$  не зависят друг от друга, поэтому для перебора их значений достаточно четырех циклов. Границы интервалов включены, значит, конечные значения переменных-аргументов известны. Однако переменная  $y$  изменяется с переменным шагом, поэтому цикл типа For для перечисления ее значений не подойдет, лучше воспользоваться циклом While. Переменная  $w$  вычисляется 4 раза, первое значение известно, значит, для вычисления остальных можно использовать цикл For с начальным значением 2 и конечным значением 4. Переменная  $z$  изменяется, не подчиняясь какой-либо закономерности, поэтому ее

значения лучше записать в массив и затем перебирать элементы  $z(j)$  этого массива в цикле For со счетчиком  $j$ . Значения  $v$  и  $\Delta y$ , кроме того, меняются в зависимости от условия, поэтому для их вычисления нужно использовать условный оператор If. Получаем в итоге программу:

```

Sub KrutoyV( )
  Dim x As Single, y As Single, z(3) As Single, w As
Single, v As Single
  z(1) = 1.2
  z(2) = 1.5
  z(3) = 3
  For x = 2 To 4           'цикл для x
    For j = 1 To 3       'цикл для z
      y = 2
      While y <=6        'цикл для y
        w =2
        For i =2 To 4   'цикл для w
          If x <3 Then
            v = x*y + z(j)^2 +w
          Else
            v = x^2 + x*w - z(j)
          End If
          MsgBox v      'печатаем результат - полученное v
          w = w*0.5      'меняем значение w
        Next           'повторяем тело цикла for i для нового w
        If y < 3 Then 'меняем значение y
          y = y + 0.5
        Else
          y = y + 1
        End If
      Wend           'повторяем цикл While y для нового значения y
    Next           'повторяем цикл For j для следующего z(j)
  Next           'повторяем цикл For для следующего x
End Sub

```

### **Задачи для самостоятельной работы:**

1. Для функции  $v$  определить количество ее значений:

$$v = \begin{cases} x^3 y^2 (2 - x - y) + z^2 & \text{при } x < 4 \\ x^2 + xw + \frac{1}{2w^2} & \text{при } x \geq 4 \end{cases} ; \quad x \in [2;4], \quad \Delta x = 1;$$

$$y \in [2;6], \quad \Delta y = \begin{cases} 0,5 & \text{при } y < 3 \\ 0,75 & \text{при } y \geq 3 \end{cases} ; \quad z = 1,2; 1,5; 2,3; 3;$$

$$w_{i+1} = 0,3w_i, \quad w_1 = 2,4; \quad 1 \leq i \leq 4$$

2. Определить сумму значений функции  $v$ , больших числа 5:

$$v = \begin{cases} (y + \sqrt{y})z & \text{при } y < 5 \\ \frac{3w - 5y}{z} & \text{при } y \geq 5 \end{cases}; \quad z = 1,1; \quad 1,3; \quad 1,5; \quad 2,1;$$

$$w_{i+1} = 1,7w_i, \quad w_1 = 0,2; \quad 1 \leq i \leq 4;$$

$$y = \begin{cases} 29, & \text{при } j < 3 \\ 7, & \text{при } j \geq 3 \end{cases}; \quad 1 \leq j \leq 5.$$

### Лабораторная работа № 7. Задачи с данными – двумерными массивами

Матрицы – двумерные массивы вводятся и выводятся с помощью вложенных циклов: один перебирает строки, другой – столбцы матрицы. Какой из циклов внешний – неважно, главное – вложенные циклы позволяют перечислить все сочетания номеров строк и столбцов, т.е. учесть *все* элементы матрицы. Рассмотрим, например, матрицу  $M$  размера  $5 \times 7$ .

#### Задача 17. Ввод и вывод двумерного массива (матрицы).

*Ввод матрицы с клавиатуры:*

```

For i = 1 To 5           'перебираем строки
  For j = 1 To 7         'перебираем столбцы
    M(i,j) = InputBox("Введите элемент матрицы")
  Next
Next

```

*Ввод матрицы с листа Excel* (матрица расположена на листе в диапазоне, например, A1:G5):

```

For i = 1 To 5           'перебираем строки
  For j = 1 To 7         'перебираем столбцы от A до G
    M(i,j) = Cells(i,j)
  Next
Next

```

Работает вложенный цикл следующим образом. Счетчик строк  $i$  получает очередное значение (здесь вначале это 1), затем полностью выполняется цикл со счетчиком  $j$ , т.е. перебираются *все* элементы  $i$ -той строки. После этого значение счетчика строк увеличивается на единицу ( $i = i + 1$ ) и вновь полностью выполняется цикл по  $j$ , перебирающий столбцы матрицы. Процедура завершается после перебора последней строки.

Если  $i$  и  $j$  поменять местами в заголовках циклов, т.е.:

```

For j = 1 To 7
  For i = 1 To 5
    A(i, j) = Cells(i, j)
  Next
Next

```

то вначале будут перебираться все элементы первого столбца, потом второго и т. д.

**Вывод матрицы** на лист Excel, например, в диапазон A8:G12:

```

For i = 8 To 12      'меняем строки, начиная с восьмой
  For j = 1 To 7      'меняем столбцы от А до G
    M(i, j) = Cells(i, j)
  Next
Next

```

**Задача 18.** Занести отрицательные элементы массива  $A(N \times M)$  в массив  $B$  и напечатать его.

**Решение.** Алгоритм задачи состоит из трех частей: ввести матрицу  $A$ ; проверить каждый ее элемент: если он отрицательный, поместить его в массив  $B$ ; вывести полученный массив  $B$  на печать. Первая и третья части – ввод и вывод данных решаются стандартным способом ввода и вывода массивов. Вторая часть – основа решения, рассмотрим ее подробнее. Анализ элементов  $A(i, j)$  матрицы  $A$  можно провести, как обычно для матриц, с помощью вложенных циклов **For**, перечисляющих все строки ( $i$ ) и столбцы ( $j$ ) матрицы. Если  $A(i, j)$  – отрицательный, нужно присвоить *очередному* элементу массива  $B$  это значение. Здесь проблема (и источник ошибок) в том, как определить очередной элемент массива  $B$ . Поскольку заранее количество отрицательных элементов в  $A$  неизвестно, то заранее неизвестна и точная длина массива  $B$ . Очевидно только, что она не превысит  $N \times M$  – количества элементов исходной матрицы  $A$ . Ясно также, что счетчик элементов массива  $B$  никак не зависит от очередных  $i$  и  $j$ . Отсюда следует главный вывод для решения задач такого типа: *для массива-приемника необходимо завести свой, независимый счетчик элементов*. Отведем для него отдельную переменную. Пусть это будет  $k$ . При записи отрицательного  $A(i, j)$  в  $B(k)$  счетчик  $k$  будет увеличиваться на единицу, и по окончании всей работы значение  $k$  покажет истинную длину массива  $B$ . Для примера в программе взята исходная матрица  $A$  размером  $N=4$ ,  $M=6$ . Вся программа имеет вид:

```

Sub Перенос ()
Dim A(4, 6) As Integer, B(24) As Integer, k As Integer
For i = 1 To 4
  For j = 1 To 6
    A(i, j) = InputBox ("Введите значение элемента матрицы ")
    Cells(i, j) = A(i, j) 'вывод вводимой матрицы на лист Excel для
                          наглядности
  Next
Next

```

```

k = 0      'обнуление счетчика элементов массива B перед началом его
           заполнения
For i = 1 To 4
  For j = 1 To 6
    If A(i, j) < 0 Then
      k = k + 1
      B(k) = A(i, j)
    End If
  Next
Next
For i = 1 To k
  Cells(7, i) = B(i)      'вывод массива B на лист Excel в строку 7
Next
End Sub

```

Эту программу можно написать короче, совместив анализ с вводом и выводом и сократив, соответственно, количество внешних циклов с трех до одного:

```

Sub Перенос()
Dim A(4, 6) As Integer, B(24) As Integer, k As Integer
k = 0
For i = 1 To 4
  For j = 1 To 6
    A(i, j) = InputBox("Введите значение элемента матрицы")
    Cells(i, j) = A(i, j)
    If A(i, j) < 0 Then
      k = k + 1
      B(k) = A(i, j)
      Cells(7, k) = B(k)
    End If
  Next
Next
End Sub

```

**Задача 19. Посчитать число четных чисел в матрице  $A(5 \times 5)$ .**

**Решение.** Алгоритм включает ввод матрицы, перебор ее значений с проверкой на четность, подсчет суммы четных значений и вывод результата.

Для программирования задачи возьмем переменную (например, k) в качестве счетчика четных чисел. Проверку на четность реализуем с помощью функции Mod. Вся программа:

```

Sub matrixNumber()
Dim i, j, k, A(5, 5) As Integer
For i = 1 To 5      'вводим матрицу из листа Excel:
  For j = 1 To 5

```

```

        A(i, j) = Cells(i, j)
    Next j
Next
k = 0                'обнуляем счетчик четных чисел
For i = 1 To 5      'начинаем перебор элементов матрицы:
    For j = 1 To 5
        If A(i, j) Mod 2 = 0 Then 'проверяем на четность
            k = k + 1            'считаем четные значения
        End If
    Next
Next
MsgBox (k)          'печатаем результат
End Sub

```

**Задача 20.** Посчитать сумму элементов, стоящих на четных позициях строк и столбцов матрицы  $A(7 \times 7)$ .

**Решение.** Здесь уже не важна четность элементов, важна четность позиций: элемент  $A(2,4)$ , например, нам подходит, а элемент  $A(3,4)$  уже нет. Чтобы попасть на четную позицию, излишне использовать функцию Mod для счетчиков строк  $i$  и столбцов  $j$ . Достаточно изменять шаг каждого цикла с приращением 2. Программа:

```

Sub matrixIndex()
Dim i, j, s, A(7, 7) As Integer

For i = 1 To 7
    For j = 1 To 7
        A(i, j) = Cells(i, j)
    Next
Next

s = 0                'обнуляем хранилище суммы
For i = 2 To 7 Step 2 'перебираем только четные строки,
                    'начиная с 2
    For j = 2 To 7 Step 2 'перебираем только четные столбцы,
                        'начиная с 2
        s = s + A(i, j) 'элементы A(i, j) лежат на пересечении
                        'четных строк и столбцов
    Next
Next
MsgBox (s)
End Sub

```



**Задача 21.** *Обнулить элементы главной диагонали матрицы  $A(5 \times 5)$ .*

**Решение.** В задачах, связанных с диагоналями матриц, следует обращать внимание на зависимость значений индекса  $j$  (счетчика столбцов) от  $i$  (счетчика строк). Если обнаружена закономерность, то можно избавиться от множества лишних операторов и сделать программу более внятной и компактной. Например, все элементы главной диагонали имеют номера столбцов, совпадающие с номерами строк, т.е. «типичный» представитель этой диагонали – элемент  $A(i,i)$ . На побочной диагонали – это элемент  $A(n-i+1)$ , если число столбцов матрицы равно  $n$  и счет строк начинается с 1. Поэтому для работы с этими диагоналями достаточно простого цикла, без вложенных. Программа задачи:

```

Sub topDiagonal()
Dim i, j, A(5, 5) As Integer
For i = 1 To 5           'вводим матрицу из листа Excel:
  For j = 1 To 5
    A(i, j) = Cells(i, j)
  Next
Next
For i = 1 To 5         'обнуляем диагональ
  A(i, i) = 0
Next
For i = 1 To 5       'выводим матрицу на лист Excel:
  For j = 1 To 5
    Cells(i, j) = A(i, j)
  Next
Next
End Sub

```

Таким образом, весь алгоритм обнуления диагонали – один простой цикл-For и одно присваивание.

**Задача 22.** *Дана матрица размером  $5 \times 5$ . Посчитать среднее значение элементов матрицы, расположенных строго ниже главной диагонали.*

**Решение.** Для каждой строки этого куска матрицы номера столбцов нужных элементов изменяются от первого до пересекающегося с диагональю. На диагонали номер столбца совпадает с номером строки, т.е.  $j = i$ . Поскольку диагональ по условию не включена, то  $j$  изменяется не до  $i$ , а до  $i - 1$ .

```

Sub Матрица()
Dim M(5, 5) As Integer, S As Integer, k As Integer, Avg
As Single
For i = 1 To 5
  For j = 1 To 5
    M(i, j) = InputBox("Введите элемент матрицы M(" & i & "," & j & ")")
  Next
Next

```

```

S = 0
k = 0
For i = 2 To 5           'номера строк берутся от 2 до конца матрицы
  For j = 1 To i - 1     'номера столбцов берутся от 1 до диагонали
    S = S + M(i, j)
    k = k + 1
  Next
Next
Avg = S / k
MsgBox "Среднее значение = " & Avg
End Sub

```

**Задача 23.** Обнулить элементы матрицы  $A(5 \times 5)$ , лежащие правее ее главной диагонали и левее побочной, включая диагональные элементы.

**Решение.** Диагонали делят матрицу на 4 части, в задаче требуется обнулить верхнюю четверть. Обратите внимание, что для каждой строки этого куса обнуляются элементы, лежащие между диагоналями, т.е. из столбцов с номерами: для 1-ой строки – от 1 до 5, для 2-ой строки – от 2 до 4, для 3-ей - от 3 до 3, т.е., в общем случае, для  $i$ -той строки – от  $i$  до  $5-i+1$ . Программа:

```

Sub matrixKvota()
Dim i, j, A(5, 5) As Integer
For i = 1 To 5
  For j = 1 To 5
    A(i, j) = Cells(i, j)   'ввод матрицы
  Next
Next
For i = 1 To 3           'номера строк берутся от 1 до середины матрицы
  For j = i To 5 - i + 1 'номера столбцов берутся от
                               'главной до побочной диагонали
    A(i, j) = 0
  Next
Next
For i = 1 To 5
  For j = 1 To 5
    Cells(i, j) = A(i, j)   'вывод матрицы
  Next
Next
End Sub

```

**Задачи для самостоятельной работы:**

1. Посчитать количество нулевых элементов в матрице  $A(5 \times 5)$ .
2. Перенести элементы, кратные трем, из массива  $A(5 \times 5)$  в массив В.
3. Обнулить элементы матрицы  $M(5 \times 5)$ , лежащие ниже ее побочной диагонали, включая диагональные.

4. Поменять местами симметрично элементы верхней четверти и нижней четверти матрицы  $A(5 \times 5)$ , разделенной диагоналями на четыре части.

### Лабораторная работа № 8. Задачи с данными строкового типа

**Задача 24.** Подсчитать количество символов, не являющихся цифрами, в произвольной строке.

**Решение** поставленной задачи сводится к проверке каждого символа строки. Известна функция Asc(Строка), которая возвращает ASCII-код начальной буквы строки. Если код лежит в интервале [49,57], то это цифра.

```

Sub количество_цифр ()
Dim s As String
Dim i As Byte, nd As Byte
s = InputBox("введите произвольную строку")
nd = 0

For i = 1 To Len(s)
  If Asc(Mid(s, i, 1)) > 57 Or Asc(Mid(s, i, 1)) < 49 Then
    nd = nd + 1
  End If
Next

MsgBox (nd)
End Sub

```

**Задача 25.** Создать программный код зашифровывающий, а затем расшифровывающий предложение, записанное на русском языке. Использовать шифр простой замены, в котором каждая буква русского алфавита заменяется другой буквой этого же алфавита. При этом замена осуществляется по правилу: первая буква заменяется последней, вторая – предпоследней и т.д. Так, А заменяется на Я, Б – на Ю, В - на Э и т. д.

**Решение** поставленной задачи сводится к замене букв исходного текста (алфавит по порядку) буквами алфавита записанного в обратном порядке. *Исходные данные:* Буквы русского алфавита, записанные в алфавитном порядке за исключением буквы Ё, буквы русского алфавита, выписанные в обратном порядке. Шифруемое сообщение: «Простая замена один из самых древних шифров». *Выходные данные:* сообщение после шифровки.

Для решения поставленной задачи определяем количество символов преобразуемой строки. Образует новую строку по длине равную исходной строке. Далее организуем цикл, в котором просматриваем все символы преобразуемой строки, определяем позицию номер k этого символа в исходном алфавите. Если в исходном алфавите символ не найден, то в данную позицию новой строки заносим этот символ без изменений, в противном случае в данную позицию но-

вой строки заносим символ из нового алфавита, позиция которого совпадает с позицией номер k исходного алфавита.

При решении задачи используем следующие функции:

1. **Len** (Строка) – возвращает число символов строки, например, Len(“мама”)=4.
2. **LCase** (Строка) – все прописные символы строки преобразует в строчные, например, b=Lcase(“Мир”). b = “мир”.
3. **Space** (количество\_символов) – возвращает строку пробелов длины количество\_символов.
4. **Mid**(string, start[, length]) – возвращает подстроку строки, содержащую указанное число символов исходной строки, например, Mid(“программирование”, 4, 4)=“грам”.
5. **InStr**([start, ] string1, string2[, compare]) – возвращает позицию первого вхождения одной строки внутри другой строки, например, k = **InStr**(1, “XXpXXpXXPXXP”, “W”) возвращает 0, т.к. символа “W” нет в строке “XXpXXpXXPXXP”.

```

Private Sub Шифрование ()
Const АБВ As String = "абвгдежзийклмнопрстуфхцщгъыьэюя"
Const НовАБВ As String = "яюэьыгъщчцхфутсрпномлкийизжед-
гвба"
Dim STR As String, STRS As String
Dim n As Long
STR = "Простая замена один из самых древних шифров "
STR = LCase (STR)           'преобразуем все символы в строчные
n = Len (STR)              'находим длину строки
STRS = Space (n)          'организуем строку пробелов длины n
For i = 1 To n
    tmp = Mid (STR, i, 1) 'по одному символу «отрываем» от
    строки STR
    k = InStr (1, АБВ, tmp) 'находим позицию вхождения символа
    If k = 0 Then
        Mid (STRS, i, 1) = tmp
    Else
        Mid (STRS, i, 1) = Mid (НовАБВ, k, 1)
    End If
Next i
Msgbox STRS
End Sub

```

**Задача 26.** Гласные буквы русского алфавита не изменяются. Первый десяток согласных заменяется на второй десяток согласных (второй - на первый) по следующей таблице:

<b>Б</b>	<b>В</b>	<b>Г</b>	<b>Д</b>	<b>Ж</b>	<b>З</b>	<b>К</b>	<b>Л</b>	<b>М</b>	<b>Н</b>
<b>Щ</b>	<b>Ш</b>	<b>Ч</b>	<b>Ц</b>	<b>Х</b>	<b>Ф</b>	<b>Т</b>	<b>С</b>	<b>Р</b>	<b>П</b>

**Решение.** Исходными данными задачи является строка произвольной длины. Кроме этого, можно определить две строки-алфавита, состоящие из согласных букв “бвгджзклмнщшчцхфтсрп” и “щшчцхфтсрпбвгджзклмн”. Если в тексте встречается буква из первой строки, то она заменяется на букву с таким же номером из второй строки-алфавита.

```

Private Sub Гласные ()
Dim tmp As String, Alf1 As String, Alf2 As String
Dim a As String, b As String
    a = InputBox ("Введите исходную строку символов")
    Alf1 = "бвгджзклмнщшчцхфтсрп"
    Alf2 = "щшчцхфтсрпбвгджзклмн"
    n = Len (a)
    b = Space (n)
    For i = 1 To n
        tmp = Mid (a, i, 1)
        k = InStr (1, Alf1, tmp)
        If k = 0 Then
            Mid (b, i, 1) = tmp
        Else
            Mid (b, i, 1) = Mid (Alf2, k, 1)
        End If
    Next i
    MsgBox b
End Sub

```

### *Задачи для самостоятельной работы:*

1. Для любого введенного предложения заменить слоги «ма» на слоги «ле».
2. Посчитать количество слов «кот» в предложении, вводимом с клавиатуры. Разделителем слов считать пробелы и запятые.
3. Поменять местами первое и последнее слово в предложении, разделителем слов считать пробел.
4. Образовать по заданному слову слово-перевертыш, в котором все буквы идут в обратном порядке.

## **Лабораторная работа № 9. Автоматическая запись макроса и его редактирование**

В настоящее время почти каждое приложение, предназначенное для ведения деловой документации, имеет макроязык и средство записи макросов.

Макрос – это программа, автоматически записанная редактором VBA по действиям пользователя, которые он выполняет на листе Excel. В результате эти действия можно потом в любое время воспроизвести, просто вызвав этот макрос как обычную программу.

Код этой программы можно также просмотреть и отредактировать или написать самостоятельно от начала до конца. Однако самый простой способ разработать макрос – это записать его, а затем изменить, если требуется, созданный код. Этот метод позволяет быстро освоить язык VBA, тщательно изучая код программы, созданный автоматически.

Для записи макроса и редактирования его в редакторе VBA:

1. Откройте лист рабочей книги Excel.
2. Запустите средство автоматической записи макросов с помощью команды **Сервис/Макрос/Начать запись**. При этом на экране появится диалоговое окно **Запись макроса** (рис. 3).

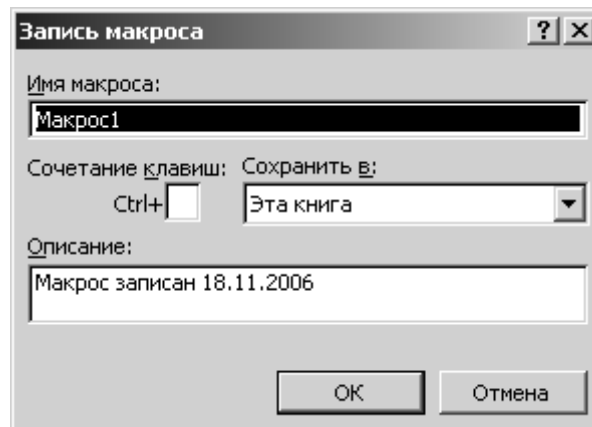


Рис. 3. Диалоговое окно «Запись макроса».

3. Задайте имя макроса (по умолчанию Макрос1, Макрос2 и т. д.) и нажмите кнопку **ОК**. При этом появится панель записи макроса **Останов** с кнопкой **Остановить запись**. Выполните нужную Вам последовательность действий и остановите запись, нажав кнопку **Остановить запись**.

4. Далее можно использовать этот записанный макрос или изменить его. В первом случае, например, можно открыть другой лист, выбрать записанный макрос в списке макросов, используя команды **Сервис/Макрос/Макросы**, а затем нажмите кнопку **Выполнить**. Во втором случае можно использовать команды **Сервис/Макрос/Макросы**, и кнопку **Изменить**. При этом запускается редактор VBA и в окне модуля выводится текст созданного макроса, который можно изменить или добавить комментарии.

**Задача 27.** *Создать макрос, очищающий содержимое ячеек A1, B1, C1 рабочего листа и изменить его, введя комментарии и новые инструкции.*

**Решение.**

1. Выполните команду **Сервис/Макрос/Начать запись**.
2. Присвойте макросу имя ClearCell и нажмите кнопку **ОК**.

3. Выделите диапазон ячеек A1:C1 с помощью мыши и нажмем кнопку **Delete** на клавиатуре.

4. Остановите запись, нажав кнопку **Остановить запись**.

5. Просмотрите результат: выполните команду **Сервис/Макрос/Макросы**. Выберите в списке макросов макрос с именем ClearCell и нажмите кнопку **Изменить**. В результате на экране откроется окно редактора **VBA** с текстом только что созданного макроса:

```
Sub ClearCell()
' ClearCellData Макрос
' Макрос записан 23.05.00 ()
Range("A1:C1").Select
Selection.ClearContents
End Sub
```

6. Добавьте комментарии к строкам программы (комментарий должен начинаться с апострофа). Например:

```
Range("A1:C1").Select      'Выделение диапазона ячеек
Selection.ClearContents   'Очистка содержимого выделенного диа-
пазона
```

7. Добавьте новую строку в программу, например,

```
Range("A3:C3").Select     'Выделение нового диапазона ячеек
```

8. Проследите, какие действия выполнит макрос после редактирования. Для этого: перейдите на лист Excel, заполните ячейки A1:C3 любыми значениями, выполните команды меню **Сервис/Макрос/Макросы**, выберите макрос ClearCell и нажмите кнопку **Выполнить**.

Для упрощения работы с макросом можно запускать его кнопкой, без использования команд меню. Кнопку можно поместить на лист Excel следующим способом:

1. Войти в меню **Вид/Панели Инструментов/Формы** листа Excel.
2. На появившейся панели выбрать элемент управления **Кнопка**, щелкнув по нему мышью. Указатель мыши превратится в крестик.
3. Щелкните мышью на листе Excel, откроется диалоговое окно **Назначить макрос объекту**.
4. Выберите в списке программ нужный макрос.

Аналогичным образом можно создать кнопку для вызова любой разработанной Вами программы. Для этого выполняются те же действия, только на шаге 4 выбирается имя этой программы.

**Задачи для самостоятельной работы:**

1. Создайте макрос, меняющий формат ячеек диапазона A1:C4: цвет шрифта, заливку, центрирование, тип и размер шрифта.
2. Создайте макрос, заполняющий диапазон A1:A12 месяцами года. Предусмотрите его вызов с помощью кнопки.
3. Измените программу макроса, созданного в первой задаче, изменив диапазон на D5:J12 и цвет заливки ячеек.
4. Создайте макрос, позволяющий строить диаграмму по таблице, представленной в диапазоне A1:B11 с заголовками столбцов – Товар и Цена.

**Лабораторная работа № 10. Размещение элементов управления на рабочем листе Excel**

С помощью панели инструментов **Элементы управления** (рис. 4), вызванной командами меню **Вид/Панели Инструментов/Элементы управления**, можно разместить на листе Excel разные элементы управления, такие как CommandButton1, CommandButton2, CommandButton3, OptionButton1, OptionButton2, OptionButton3, TextBox1, Label1.



Рис. 4. Панель инструментов **Элементы управления** в Excel

**Задача 28. Создать:**

- кнопку «**Вывод сгенерированного массива на лист**» (при нажатии на эту кнопку генерируется массив вещественных чисел в диапазоне (-5, 5) и выводится в диапазон A1:A20 листа Excel),
- кнопку «**Отменить вывод значений**» (при нажатии на данную кнопку из диапазона A1:A20 удаляются значения элементов массива),
- кнопку «**Ok**» (при нажатии на нее решается задача поиска минимального или максимального элемента массива, либо находится среднее арифметическое значение, в зависимости от того, какой элемент **Option-Button** выбран) (см. рис. 5).

**Решение.** В начале работы необходимо подготовить рабочий лист, создав на рабочем листе «Лист1» элементы управления, используя команды меню **Вид/Панели Инструментов/Элементы управления**. Чтобы изменить название элемента управления, например, CommandButton1, можно воспользоваться кнопкой **Свойства** (вторая слева) на панели **Элементы управления**. В



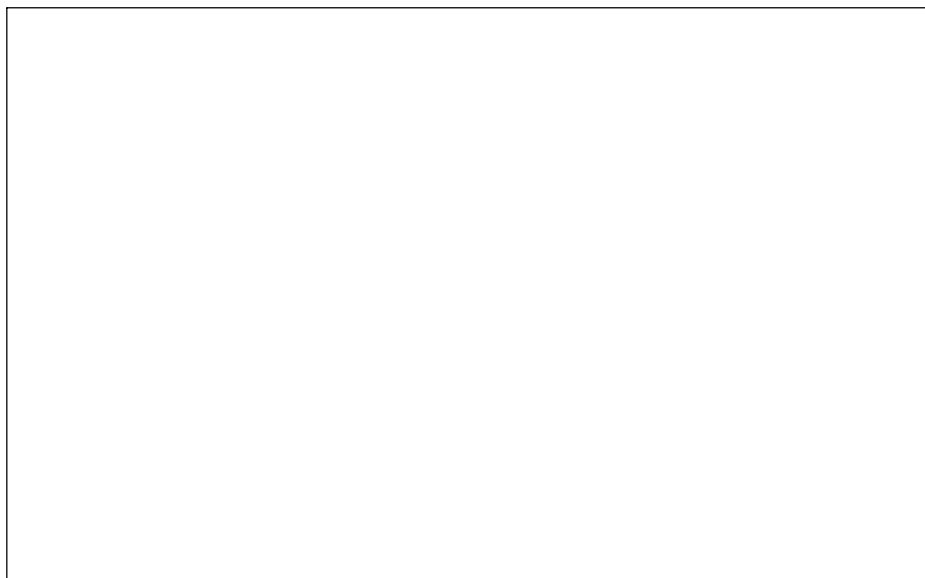


Рис. 5. Вид листа Excel с созданными на нем элементами управления.

открывшемся окне свойств нужно выбрать строку **Caption** и изменить текст в правом столбце.

При двойном щелчке на созданном элементе управления открывается редактор VBA, подготовленный для создания программы на листе «Лист1».

Для решения поставленной задачи необходимо набрать следующий код:

```
Private Sub CommandButton1_Click()'Кнопка генерации данных  
For i = 1 To 20  
Cells(i, 1) = -5 + 10 * Rnd()
```

```
Next
```

```
End Sub
```

```
Private Sub CommandButton2_Click()'Кнопка очистки диапазона
```

```
Range ("A1:A20") .Select
```

```
Selection.ClearContents
```

```
End Sub
```

```
Private Sub CommandButton3_Click()'Кнопка основной программы
```

```
Dim a(1 To 20) As Single, Max As Single
```

```
Dim Min As Single, S As Single
```

```
For i = 1 To 20
```

```
a(i) = Cells(i, 1)
```

```
Next
```

```
If OptionButton1.Value = True Then
```

```
Max = a(1)
```

```
For i = 2 To 20
```

```
    If Max < a(i) Then Max = a(i)
```

```
Next
```

```
TextBox1.Text = Max
```

```
End If
```

```
If OptionButton2.Value = True Then
```

```
    Min = a(1)
```

```

For i = 2 To 20
If Min > a(i) Then Min = a(i)
Next
TextBox1.Text = Min
End If
If OptionButton3.Value = True Then
    s = 0
    For i = 1 To 20
        s = s + a(i)
    Next
    TextBox1.Text = s / 20
End If
End Sub

```

### Лабораторная работа № 11. Применение пользовательских форм

Пользовательская форма (UserForm) изначально представляет собой заготовку для создания собственного диалогового окна. На ней, так же, как и на рабочем листе можно размещать разнообразные элементы управления, назначать им процедуры и устанавливать значение свойств.

С помощью одной или нескольких форм создается удобный интерфейс разрабатываемого приложения его пользователем. По умолчанию формы имеют имена UserForm1, UserForm2, UserForm3 и т. д. Для изменения имени формы необходимо изменить значение свойства Caption формы.

Для создания формы необходимо выполнить команду **Сервис/Макрос/Редактор Visual Basic**, затем команду **Insert/UserForm**. При этом будет создана пустая форма с именем UserForm1 (рис. 6).

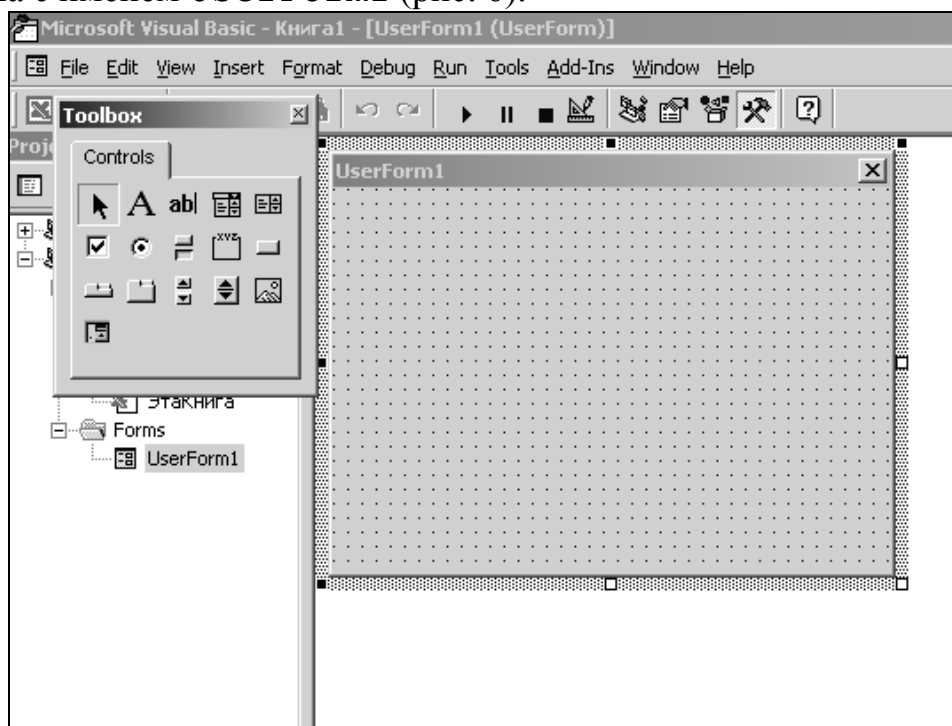


Рис 6. Форма UserForm1 и панель элементов.

Можно выделить форму, щелкнув по ней правой кнопкой мыши. При этом откроется контекстное меню, в котором пункт **Свойства (Properties)** позволяет открыть свойства формы. Например, чтобы изменить заголовок формы нужно заменить значение по умолчанию на новое имя.

Также на форме можно разместить любые из элементов управления, описанных в предыдущем параграфе. Для удобства работы при размещении элементов управления на форме имеется разметка в виде точек. Кроме того, используя контекстное меню, можно выравнять размеры и положение элементов управления на форме.

Для удаления формы с экрана можно использовать метод `Hide`. Для этого необходимо какой-либо кнопке на форме, например кнопке Отмена, назначить следующую процедуру:

```
Sub CommandButton1_Click ()
    UserForm1.Hide
End Sub
```

Для вывода формы на экран из Excel необходимо предварительно в модуле создать процедуру

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

После этого можно выполнить команду **Сервис/Макрос/Макросы**, выбрать из списка макросов макрос «Открытие\_формы», и нажать кнопку **Выполнить** (например, макрос Задача). При этом на экран будет выведена форма с соответствующими элементами управления.

Форма может быть активизирована не только из Excel, но и из среды VBA:

- установите курсор в области процедуры, выводящей форму на экран, или сделайте активным окно с необходимой формой;
- выполните команду **Run/ Run Sub /UserForm**.

При этом откроется лист Excel, с расположенной на нем формой.

Каждому элементу управления в пользовательской форме можно назначить всплывающую подсказку, установив значение свойства `ControlTipText` в окне свойств для каждого элемента управления. Подсказка появляется при наведении указателя мыши на соответствующий элемент управления.

В результате установки значений свойства `ControlTipText` всех элементов управления пользовательской формы можно будет получить всплывающую подсказку об интересующем элементе формы (поле, надписи и т. д.), указав на этот элемент управления мышью.

**Задача 29.** Создать приложение, которое позволит при исходных данных  $A, B, C$  вычислить значение  $z = \frac{A+B}{\sqrt{C^2-A}} + \sin\left(\frac{B}{A}\right)$ . В случае, если произвести вычисление невозможно, то должно появиться соответствующее сообщение.

**Решение.**

1. Подготовьте соответствующим образом UserForm2 (рис. 7):

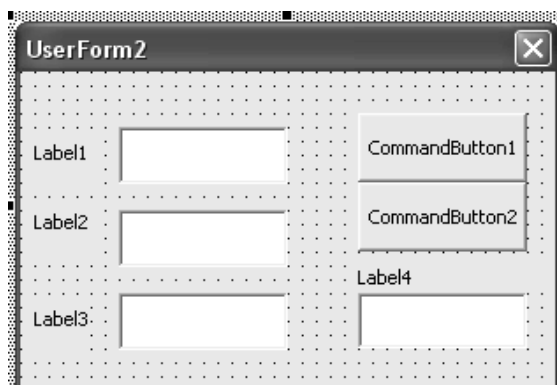


Рис. 7. Вид формы UserForm2.

С помощью свойств элементов управления переименуйте их следующим образом (рис. 8):

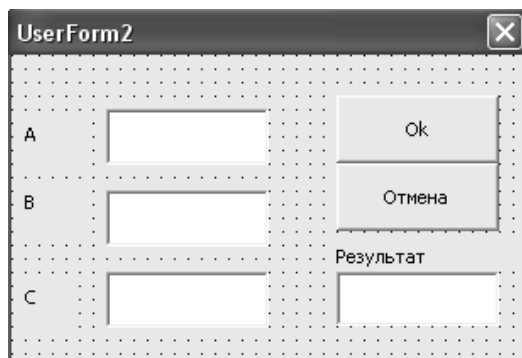


Рис. 8. Вид формы UserForm2 после переименования элементов управления.

2. Щелкнув дважды по кнопке **Ok**, войдите в область программного кода, где наберите следующий код:

```
Private Sub CommandButton1_Click()  
    A = CInt(TextBox1.Value) 'присваиваем переменным a,b,c значения, введенные пользователем в текстовые поля  
    B = CInt(TextBox2.Value)  
    C = CInt(TextBox3.Value)  
    If C ^ 2 - A < 0 Or A = 0 Then  
        TextBox4.Text = "Решений нет"  
    Exit Sub
```

**End If**

$z = (A + B) / (C^2 - A)^{(1/2)} + \sin(B / A)$

`TextBox4.Value = Format(z, "###.00")` *'форматирует действительное число*

**End Sub**

**Private Sub** CommandButton2\_Click()

`UserForm2.Hide`

**End Sub**

Обратите внимание, что, несмотря на то, что надписи на кнопках нами поменялись, внутренние имена этих кнопок, используемые в программе (`CommandButton1`, `CommandButton2`) остались прежними.

3. Запустите форму на выполнение **Run/ Run Sub /UserForm**. Введите в поля A, B, C необходимые данные и проведите тестирование программы.

**Задача 30.** Создать приложение, которое позволит считать данные с рабочего листа, обработать их и выдать результат в соответствующие поля пользовательской формы.

**Содержание приложения:** на активном рабочем листе представлены данные о студентах и их среднем балле успеваемости (рис. 9). Для удобства средний балл успеваемости генерируется произвольным образом в интервале от 2 до 5 при запуске формы на экран (т.е. при инициализации формы). Данные считываются с листа и обрабатываются процедурой, вызываемой нажатием кнопки **Запуск**. В результате выбирается студент с максимальным баллом, с минимальным баллом и вычисляется средний балл среди всех студентов.

**Решение.**

1. Подготовьте пользовательскую форму `UserForm3` следующим образом (рис. 10):

	A	B
1	Студенты	Средний балл
2	Иванов И.И.	2.85
3	Сидоров П.А.	2.14
4	Батулин П.А.	2.89
5	Соловьева О.Т.	3.15
6	Пушникова А.И.	2.9
7	Сидорня К.Ю.	4.85

Рис. 9. Вид активного рабочего листа

Рис.10. Вид формы `UserForm3`.

2. Так как данные генерируются при запуске формы, то необходимо создать процедуру `UserForm_initialize`. Код процедуры инициализации можно записать в той же области программного кода, где и остальной код про-

граммы (можно щелкнуть мышью по форме дважды).

```

Private Sub UserForm_Initialize()
For i = 1 To 6
    Cells(i+1,2) = Format(2+(5-2)*Rnd(), "###.##")
Next
End Sub

```

Таким образом, при запуске формы на экран во второй столбец активного рабочего листа будут выводиться сгенерированные данные из интервала (2; 5).

### 3. Запрограммируем кнопку CommandButton1 (Запуск).

```

Private Sub CommandButton1_Click()
Max = Cells(2, 2): k = 2
For i = 2 To 6           'перебираем строки второго столбца (B)
    If Max < Cells(i, 2) Then
        Max = Cells(i, 2) 'ищем максимум и запоминаем его номер строки
        k = i
    End If
Next
TextBox1.Text = Cells(k, 1) 'выводим результат на форму
Min = Cells(2, 2): k = 2
For i = 2 To 6           'аналогично ищем минимум
    If Min > Cells(i, 2) Then
        Min = Cells(i, 2)
        k = i
    End If
Next
TextBox2.Text = Cells(k, 1) 'выводим результат на форму
s = 0
For i = 1 To 6
    s = s + Cells(i + 1, 2) 'суммируем баллы
Next
sred = s / 6              'находим среднее
TextBox3.Text = Format(sred, "###.##") 'выводим его на форму в формате действительного числа
End Sub

```

4. Запустите форму на выполнение с помощью **Run/ Run Sub /UserForm**. Проверьте как генерируются данные на активном рабочем листе: нажав на кнопку **Запуск** вы должны получить данные о студенте с наивысшим балом, с наименьшим баллом и средний балл успеваемости среди всех студентов.

**Задача 31.** Создать программное приложение, которое позволит обрабатывать данные о товаре, количестве поставок и поставщиках. Данные представлены на рабочем листе. Количество поставок генерируется при инициализации формы. На пользовательской форме при вводе в поле названия поставщика должны появиться следующие данные: общее количество его поставок, рассчитанный на этой основе статус поставщика или процент от общего количества.

**Решение.**

1. Подготовьте рабочий лист в следующем виде (рис. 11):

	А	В	С
1	Товар	Количество поставок	Поставщик
2	Порошок Миф		ООО Солнышко
3	Порошок Ариель		ООО Цум
4	Порошок Лоск		ООО Окей
5	Шампунь Пантин		ОАО Рамстор
6	Шампунь ГлиссКур		ООО Цум
7	Шампунь Дав		ООО Солнышко
8	Дезодорант Фа		ИЧП Парадиз
9	Дезодорант Рексона		ООО Метро
10	Дезодорант ОлдСпайс		ООО Цум
11	Мыло Дуру		ООО Окей
12	Мыло Палмолив		ОАО Рамстор
13	Мыло Камэй		ИЧП Парадиз
14	Мыло Дав		ООО Солнышко
15	Мыло Хозяйственное		ООО Цум
16	Гель/д/Д Нивея		ОАО Рамстор
17	Гель/д/Д Фа		ООО Цум
18	Гель/д/Д ИвРоше		ООО Окей
19	Салфетки Зева		ОАО Рамстор
20	Салфетки Ясамая		ООО Солнышко
21	Салфетки Клинекс		ООО Солнышко

Рис. 11. Вид активного рабочего листа с данными.

2. Подготовьте пользовательскую форму, разместив на ней текстовое поле для ввода поставщика, текстовое поле для вывода общего количества поставок и переключатель, который позволяет либо вывести на экран статус поставщика, либо процентное отношение его поставок к общему количеству его поставок всех поставщиков (рис. 11). При этом считаем, что статус поставщика равен 3, если поставщик поставляет более 50% от общего количества, статус равен 2, если количество поставок от 30% до 50%; равен 1 – если количество поставок более 10% и меньше или равно 30% и, наконец, равен 0 в остальных случаях.

Рис. 11. Вид формы «Поставщики».

3. Первоначально необходимо сгенерировать данные с помощью процедуры инициализации формы:

```
Private Sub UserForm_Initialize()
  For i = 1 To 20
    Cells(i + 1, 2) = Format(100+900*Rnd(), "###.##")
  Next
End Sub
```

4. Кнопка «Ok» позволяет считать данные из поля «Поставщик», найти данного поставщика на рабочем листе в столбце «Поставщик» и выбрать данные о поставках для этого поставщика. Если поставщика нет в списке, то должно быть выдано соответствующее сообщение. Кроме того, должен быть произведен расчет общего количества поставок всех поставщиков и определение статуса или процента от общего количества в зависимости от выбранного переключателя OptionButton. В программе использованы переменные KolPost, statys, prozent, которые можно не описывать.

```
Private Sub CommandButton1_Click()
  Поставщик = TextBox1.Text
  s = 0: k = 0: KolPost = 0
  For i = 1 To 20
    If Cells(i + 1, 3) = Поставщик Then
      s = s + Cells(i + 1, 2)
      k = k + 1
    End If
    KolPost = KolPost + Cells(i + 1, 2)
  Next
  If k=0 Then
    MsgBox "Нет такого поставщика"
  Exit Sub
  End If
  TextBox2.Text = s
  statys = 0
  If OptionButton1.Value = True Then
    If s >= 0.5 * KolPost Then statys = 3
    If s<0.5*KolPost And s>=0.3*KolPost Then
      statys=2
    End If
    If s<0.3*KolPost And s>=0.1*KolPost Then
      statys=1
    End If
    TextBox3.Text = statys
  End If
```



```

If OptionButton2.Value = True Then
    prozent = s / KolPost
    TextBox3.Text = Format(prozent, "0.#0")
End If
End Sub

```

5. Кнопка «Очистить» позволяет удалить данные из диапазона B1:B21 и очищает данные в текстовых полях TextBox1, TextBox2, TextBox3.

```

Private Sub CommandButton2_Click()
    TextBox1.Text = ""
    TextBox2.Text = ""
    TextBox3.Text = ""
    Range("B2:B21").Select
    Selection.ClearContents
End Sub

```

### *Задачи для самостоятельной работы:*

1. В форму Студенты (см. задачу 31) добавить поля для вывода результата расчета поощрения и поле ввода с надписью Студент. Если его средний балл на 20% больше чем средний балл по группе, то поощрение – тетрадь, на 30% больше – пенал, на 40% – путевка в летний лагерь.

2. Создать программное приложение, которое позволит обработать данные о странах и ценах на туристические путевки в эти страны. Данные представлены на рабочем листе. Цена на путевки генерируется при инициализации формы. На пользовательской форме, при вводе в поле названия страны должны появиться следующие данные: средняя цена на путевки в эту страну, максимальная и минимальная цена на путевки в эту страну.

## **Глава 4. Справочник по языку программирования Visual Basic For Application**

В справочник включен минимум сведений для составления программ на VBA по предложенному курсу. При составлении справочника использованы следующие обозначения: выражения в [] являются необязательными, знак | означает альтернативный выбор конструкций, например, { k1 | k2 } – используем либо инструкцию k1, либо k2.

### *4.1. Описание переменных и основные типы данных*

Переменные в программах VBA должны быть описаны. Описание переменной включает объявление ее имени и типа. Имя переменной – набор букв и символов, начинающихся с буквы. В качестве символов могут быть использо-

ваны цифры, подчеркивания. Нельзя использовать имена, совпадающие с ключевыми словами VBA и именами встроенных функций. Регистр не имеет значения. Базовые типы переменных VBA приведены в табл.1.

Таблица 1

Тип данных	Занимаемый объем памяти (байт)	Допустимые значения	Описание
<b>Byte</b>	1	От 1 до 255	Байт
<b>Boolean</b>	2	True (Истина) или False (Ложь)	Логический
<b>Integer</b>	2	От -32768 до 32768	Целое
<b>Long</b>	4	От -2147483648 до -2147483648	Длинное целое
<b>Single</b>	4	По абсолютной величине от 1,4E-45 до 3,4E+38	Число с плавающей точкой
<b>Date</b>	8	От 1 января 100 г. до 31 декабря 9999г.	Дата
<b>Double</b>	8	По абсолютной величине от 4,9E-324 до 1,7E+308	Число с плавающей точкой двойной точности
<b>String ()</b>	10+длина строки	От 0 до 2×10 <sup>9</sup>	Строка переменной длины
<b>String</b>	Длина строки	От 0 до 65400	Строка постоянной длины
<b>Variant</b>	16	Любое значение вплоть до границ диапазона Double	Числовые подтипы
<b>Variant</b>	22+ длина строки	От 0 до 2×10 <sup>9</sup>	Строковые подтипы

Переменную в VBA можно описать с помощью следующей конструкции:

**Dim** Имя\_Переменной **As** Тип\_Переменной

Например, возможны следующие описания переменных:

**Dim** A **As Integer**

**Dim** C, D **As Integer**, E **As Single**

Часто при написании программ необходимо использовать одни и те же постоянные значения: числа, строки, даты и т.д. В этом случае вместо них лучше использовать имена, которые обозначают эти значения. В VBA можно задать константу с помощью одной из следующих конструкций:

**Const** Имя\_константы = Выражение

**Const** Имя\_константы **As** Тип\_Постоянной = Выражение

Примеры возможного определения констант:

**Const** FileName = "test.xls"

```
Const PI As Double = 3.14159
```

Константам нельзя присваивать новых значений, т.е. имя константы не должно встречаться в левых частях операторов присваивания.

Для описания переменных-массивов можно использовать несколько способов, например:

```
Dim B(3, 3) As Single
Dim A(12) As Integer
```

Первая строка объявляет двухмерный массив 3×3 (двумерный массив), состоящий из действительных чисел. Вторая строка объявляет одномерный массив (вектор), состоящий из 12 целых чисел, причем по умолчанию первый элемент массива будет A(0), а последний – A(11). В этом случае говорят, что 0 – базовый индекс. Можно изменить базовый индекс, написав в начале листа модуля оператор **Option Base 1**. После этого индексы массивов A и B будут начинаться с единицы. Другим способом изменения базового индекса является использование ключевого слова **To** при объявлении массива, например

```
Dim B(1 To 3, 1 To 3) As Single
Dim A(1 To 12) As Integer
```

Удобным способом определения одномерных массивов является функция **Array**, преобразующая список элементов, разделенных запятыми, в вектор из этих значений, и присваивающая их переменной типа **Variant**.

```
Dim A As Variant
A = Array (10, 20, 30)
```

*Комментарий* – фрагменты поясняющего текста, не являются операторами программы и игнорируются компилятором. Начало комментария – символ апострофа ('). Комментарием считается любой текст от этого символа до конца строки.

#### 4.2. Отладка программ




После того, как написан код программы, необходимо провести компиляцию. Это можно сделать с меню **Debug, Compile VBAProject**. При этом могут возникнуть ошибки компиляции в случае, если VBA не может интерпретировать введенный код, например, при некорректном вводе числа скобок, неправильном имени, неполном вводе инструкции и т.д. Некоторые из этих ошибок обнаруживаются Visual Basic при завершении набора строки с инструкцией в редакторе кода и нажатием клавишу <Enter>. Строка, в которой содержится ошибка, выделяется красным цветом, и на экране отображается диалоговое окно о возможной причине, вызвавшей ошибку.

Другие ошибки компиляции обнаруживаются перед выполнением программы. Отметим, что VBA каждый раз автоматически компилирует программу при ее запуске на выполнение. В этом случае предполагаемое место-

положение ошибки выделяется синим цветом, и на экране отображается диалоговое окно **Microsoft Visual Basic** с сообщением о возможной причине, вызвавшей ошибку.

Ошибки выполнения возникают после успешной компиляции программы при ее выполнении. Причинами таких ошибок могут быть некорректные данные, введенные пользователем, например, требуется число, а пользователь вводит строковую информацию.

Если в диалоговом окне **Microsoft Visual Basic** нажать кнопку **Debug**, то в строке модуля желтым цветом будет выделена строка, вызвавшая ошибку, и на выполнение которой действие программы было прервано. Visual Basic перейдет в режим прерывания. Кроме режима прерывания, приложение может находиться в режиме разработки и режиме выполнения. Переключение между режимами работы удобно производить при помощи трех кнопок панели инструментов **Standart**.

Кнопка	Название	Описание
	Start	Доступна в режиме конструирования. Переключает в режим выполнения. В режиме прерывания она также доступна, но играет роль кнопки Continue.
	Break	Доступна в режиме выполнения. Переключает в режим прерывания.
	End	Доступна в режиме выполнения. Переключает в режим разработки.

Наиболее коварными являются логические ошибки (когда программа работает, но выдает неверные результаты). В этом случае очень полезными являются средства отладки, которые позволяют лучше понять, что действительно происходит в программе, проследить за ее работой шаг за шагом. Для проверки последовательности выполнения операторов в программе необходимо поместить курсор в процедуру, которую необходимо отладить, и осуществить одно из перечисленных ниже действий:

- Выбрать команду Выполнить, Выполнить пошагово (**Run, Step Into**)
- Нажать клавишу <F8>

### 4.3. Операции VBA

- Математические операции, выполняются над числами и их результатом являются числа:

Операции	Описание
[Операнд1] + [Операнд2]	Сложение
[Операнд1] – [Операнд2]	Вычитание
– [Операнд]	Перемена знака
[Операнд1] * [Операнд2]	Умножение
[Операнд1] / [Операнд2]	Деление

[Операнд1] <b>Mod</b> [Операнд2]	Остаток от деления по модулю
[Операнд1] <b>^</b> [Операнд2]	Возведение в степень

- Отношения, применяются не только к числам и их результатом являются логические значения, например  $x > y$ :

Операции	Описание
[Операнд1] <b>&lt;</b> [Операнд2]	Меньше
[Операнд1] <b>&gt;</b> [Операнд2]	Больше
[Операнд1] <b>&lt;=</b> [Операнд2]	Меньше или равно
[Операнд1] <b>&gt;=</b> [Операнд2]	Больше или равно
[Операнд1] <b>&lt;&gt;</b> [Операнд2]	Не равно
[Операнд1] <b>=</b> [Операнд2]	Равно

- Логические операции, применяются к логическим выражениям (значениям) и их результатом являются логические значения, например **Not**  $x$  **and**  $y$ .

[Операнд1] <b>And</b> [Операнд2]	Логическое умножение
[Операнд1] <b>Or</b> [Операнд2]	Логическое сложение
<b>Not</b> [Операнд]	Логическое отрицание
[Операнд1] <b>Imp</b> [Операнд2]	Логическая импликация

VBA выполняет операции в соответствии с их приоритетами. Приоритеты операций (по убыванию):

1. Вызов функции и скобки
2. **^**
3. **-** (унарный минус)
4. **\***, **/**
5. **Mod**
6. **+**, **-**
7. **>**, **<**, **>=**, **<=**, **<>**, **=**
8. **Not**
9. **And**
10. **Or**
11. **Imp**

#### 4.4. Встроенные функции VBA

Математические:

Функция	Возвращаемое значение
<b>Abs</b> (Число)	Модуль числа
<b>Atn</b> (Число)	Арктангенс
<b>Cos</b> (Число)	Косинус
<b>Exp</b> (Число)	Экспонента
<b>log</b> (Число)	Натуральный логарифм
<b>Sin</b> (Число)	Синус
<b>Sqr</b> (Число)	Квадратный корень из числа

<b>Tan</b> (Число)	Тангенс
<b>Sgn</b> (Число)	Знак числа
<b>Fix</b> (Число)	Отбрасывают дробную часть числа и возвращают целую, для положительных одинаковы; различие в отрицательном Числе
<b>Int</b> (Число)	

### Функции обработки строк:

Функция	Возвращаемое выражение
<b>Asc</b>	Возвращает ASCII-код начальной буквы строки. <i>Синтаксис:</i> Asc (Строка)
<b>Chr</b>	Преобразует ASCII-код в строку. <i>Синтаксис:</i> Chr (Код) Например: Chr (13) –переход на новую строку, Chr (97) = "a"
<b>Left</b>	Возвращает подстроку, состоящую из заданного числа первых символов исходной строки. <i>Синтаксис:</i> Left (string, length) <ul style="list-style-type: none"> <li>▪ Length – число символов,</li> <li>▪ String – исходная строка.</li> </ul>
<b>Right</b>	Возвращает строку, состоящую из заданного числа последних символов исходной строки. <i>Синтаксис:</i> Right (string, length) <ul style="list-style-type: none"> <li>▪ Length – число символов,</li> <li>▪ String – исходная строка.</li> </ul>
<b>Mid</b>	Возвращает подстроку строки, содержащую указанное число символов исходной строки.. <i>Синтаксис:</i> Mid (string, start[, length]) <ul style="list-style-type: none"> <li>▪ Length – число возвращаемых символов подстроки,</li> <li>▪ String – исходная строка,</li> <li>▪ Start – позиция символа в строке string, с которого начинается нужная подстрока.</li> </ul>
<b>Len</b>	Возвращает число символов строки. <i>Синтаксис:</i> len (Строка)
<b>String</b>	Возвращает строку, состоящую из указанного числа повторений одного и того же символа. <i>Синтаксис:</i> String (number, character)
<b>StrComp</b>	Возвращает результат сравнения двух строк. <i>Синтаксис:</i> StrComp (string1, string2 [, compare]) <ul style="list-style-type: none"> <li>▪ String1, string2 –два любых строковых выражения</li> <li>▪ Compare – указывает способ сравнения строк.( 0 – двоичное сравнение и 1-посимвольное без учета регистра)</li> </ul>
<b>InStr</b>	Возвращает позицию первого вхождения одной строки внутри другой строки. <i>Синтаксис:</i> InStr ([start, ] string1, string2 [, compare]) <ul style="list-style-type: none"> <li>▪ Start-числовое выражение, задающее позицию, с которой на-</li> </ul>

	<p>чинается каждый поиск. Если этот аргумент опущен, поиск начинается с первого символа строки</p> <ul style="list-style-type: none"> <li>▪ String1 – строковое выражение, в котором выполняется поиск</li> <li>▪ String2 – искомое строковое выражение</li> <li>▪ Compare – указывает способ сравнения строк.</li> </ul>
--	---

### Функции проверки типов:

Функция	Проверка
<b>IsArray</b> (переменная)	Является ли переменная массивом
<b>IsDate</b> (переменная)	Является ли переменная датой
<b>IsEmpty</b> (переменная)	Была ли переменная описана инструкцией Dim
<b>IsNull</b> (переменная)	Является ли переменная пустым значением
<b>IsNumeric</b> (переменная)	Является ли переменная числовым значением
<b>IsObject</b> (переменная)	Является ли переменная объектом

### Функции преобразования типов

Преобразование строки в число и обратно осуществляется следующими функциями:

<b>Val</b> (строка)	Возвращает числа, содержащиеся в строке, как числовое значение соответствующего типа
<b>Str</b> (число)	Возвращает значение типа Variant (String), являющееся строковым представлением числа

Другие функции преобразования типов из данного в указанный:

Функция	Тип, в который преобразуется значение аргумента
<b>CBool</b> (Выражение)	Boolean
<b>CByte</b> (Выражение)	Byte
<b>CCur</b> (Выражение)	Currency
<b>CDate</b> (Выражение)	Date
<b>CDouble</b> (Выражение)	Double
<b>CDec</b> (Выражение)	Decimal
<b>CInt</b> (Выражение)	Integer
<b>CLng</b> (Выражение)	Long
<b>CSng</b> (Выражение)	Single
<b>CVar</b> (Выражение)	Variant
<b>CStr</b> (Выражение)	String

## 4.5. Операторы VBA

**Последовательность операторов.** Операторы последовательности отделяются друг от друга либо двоеточием, либо признаком конца строки (т.е. каждый оператор начинается с новой строки). Использование знака двоеточия позволяет разместить несколько операторов на одной строке. Например,

a = b: c = d: **If** a > d **Then** d = a

Можно также осуществлять перенос строки, используя символы (Пробел)+(Знак подчеркивания) в конце строки.

**Оператор присваивания.** Оператор присваивает значение выражения переменной, константе или свойству объекта.

*Синтаксис:*

Переменная = Выражение           или  
**Const** Константа = Значение

**Условный оператор If...Then...Else..End If (выбор одной из двух ветвей).**

Оператор условного перехода задает выполнение одной из двух групп инструкций в зависимости от истинности условия.

*Синтаксис:*

```
If Условие Then
    Оператор_1
[Else
    Оператор_2]
End If
```

Для записи вложенных условных операторов может быть использована более простая конструкция:

```
If <Условие-1> Then
    Операторы 1
ElseIf <Условие-2> Then
    Операторы 2
    . . .
[ElseIf <Условие-n> Then
    [Операторы n]
    . . .
[Else
    [Операторы]
End If
```

**Оператор множественного ветвления Select Case (выбор одной из нескольких ветвей).** Оператор выбора выполняет одну из нескольких групп инструкций, в зависимости от значения выражения, и затем выходит из этого оператора.

*Синтаксис:*

```
Select Case Выражение
    Case Значение-1
        Оператор_1
    . . .
```



```

Case Значение-n
    Оператор_n
[Case Else
    Оператор_n+1]
End Select

```

Значение-*i* – это одно конкретное значение из множества значений или диапазон (интервал значений).

Сначала вычисляется значение выражения, стоящего после ключевых слов **Select Case**. Затем выполняется проверка того, попадает ли Выражение в одно из Case-значений.

Например:

```
Число=Cells(1,2)
```

```
Select Case Число
```

```
    Case 1
```

```
        Cells(2,2) = "Число равно 1"
```

```
    Case 2, 3
```

```
        Cells(2,2) = "Число равно 2 или 3"
```

```
    Case 4 To 6
```

```
        Cells(2,2) = "Число от 4 до 6"
```

```
    Case Is >=7
```

```
        Cells(2,2) = "Число не менее 7"
```

```
End Select
```

### Операторы циклов.

#### **For...To...Next**

Оператор **For-Next** повторяет выполнение группы инструкций, пока Счетчик изменяется от Начала до Конца с указанным Шагом. Если Шаг не указан, то он полагается равным 1. Альтернативный выход из цикла предоставляется инструкции **Exit For**.

*Синтаксис:*

```
For Счетчик = Начало To Конеч [ Step Шаг ]
```

```
    Операторы
```

```
    [ Exit For ]
```

```
    Операторы
```

```
Next [Счетчик]
```

#### **For...Each...Next**

Повторяет выполнение инструкций для каждого элемента семейства или массива. В качестве Семейства может выступать диапазон ячеек рабочего листа или семейство рабочих листов. В качестве элемента в последнем случае можно рассматривать рабочий лист.

*Синтаксис:*

```
For Each Элемент In Семейство
Операторы
[Exit For]
Операторы
Next [Элемент]
```

Например:

```
For Each Лист In WorkSheets
If Лист.Name= "Результат" Then
    Sheets("Результат").Delete
End If
Next Лист
```

### **While-Wend**

Оператор While-Wend выполняет последовательность инструкций, пока заданное условие имеет значение **True**.

*Синтаксис:*

```
While Условие
    Операторы
Wend
```

Оператор While-Wend в отличие от For-Next работает не заданное число раз, а пока выполняется условие.

### **Do – Loop**

Оператор Do-Loop повторяет выполнение набора инструкций, пока условие имеет значение True (случай While) или пока оно не примет значение True (случай Until).

*Синтаксис:*

```
Do {While | Until} Условие
    Операторы
    [Exit Do]
    Операторы
Loop
```

или

```
Do
    Операторы
    [Exit Do]
    Операторы
Loop {While | Until} Условие
```

В любом месте управляющей структуры Do-Loop может быть размещена любое число инструкций Exit Do, обеспечивающих досрочные возможности выхода из цикла Do-Loop.

### Оператор безусловного перехода GoTo.

*Синтаксис:*

**Goto** Метка

Аргумент Метка может быть любым именем или номером строки, на которую осуществляется переход. Нельзя осуществлять переход из тела функции в вызывающую процедуру, переход из охватывающей программы внутрь тела цикла, внутрь тела условного оператора. Если используется оператор GoTo, то в программе должен быть *единственный* оператор, перед которым стоит эта метка:

*Синтаксис:* Метка: Оператор

## 4.6. Модули, процедуры и функции

Программа на языке VBA является процедурой или функцией, и может состоять из одного или нескольких модулей.

Обычно текст *модуля* начинается с команд, которые управляют описанием переменных, способом сравнения строк и т.д. Затем идет объявление глобальных переменных и констант, которые можно использовать во всех процедурах либо модуля, либо проекта. Далее располагается код процедур Sub и функций Function, составляющих саму программу.

**Процедура** является самостоятельной частью кода, которая имеет имя и может содержать аргументы-параметры, выполнять последовательность инструкций и изменять значения своих аргументов.

*Синтаксис:*

[**Private** | **Public**] [**Static**] **Sub** Имя ( [СписокАргументов] )

Операторы

[**Exit Sub**]


Операторы

**End Sub**

Элементы описания:

<b>Public</b>	Указывает, что процедура Sub доступна для всех других процедур во всех модулях
<b>Private</b>	Указывает, что процедура sub доступна для других процедур только того модуля, в котором она описана
<b>Static</b>	Указывает, что локальные переменные процедуры sub сохраняются в промежутках времени между вызовами этой процедуры
Имя	Имя процедуры

СписокАргументов	Список переменных-(формальных параметров), представляющих аргументы, значения которых передаются в процедуру sub при ее вызове. Имена переменных разделяются запятой
------------------	--

Процедура без параметров (аргументов) может быть исполнена без вызова из другой программы, непосредственно в редакторе VBA. Для этого можно щелкнуть кнопку Run  на стандартной панели инструментов и нажать кнопку F5 на клавиатуре. Процедура с параметрами (аргументами) запускается только при ее вызове из другой программы, т.к. аргументы должны получить конкретные значения.

Инструкция `Exit Sub` приводит к немедленному выходу из процедуры `Sub`.

**Функция** – процедура, результат выполнения которой присваивается имени функции в качестве ее значения.

*Синтаксис:*

**[Private|Public] [Static] Function**

Имя ([СписокАргументов]) **As** Тип

Операторы

**[Exit Function]**

Операторы

**End Function**

Синтаксис инструкции `Function` содержит те же элементы, что и `Sub`. Инструкция `Exit Function` приводит к немедленному выходу из процедуры `Function`. Подобно процедуре `Sub`, процедура `Function` является самостоятельной процедурой, которая может получать фактические аргументы (параметры), выполнять последовательность инструкций и изменять значения этих аргументов. В теле функции должен быть оператор присваивания значения имени функции. Это значение может быть использовано в программе, вызывающей данную функцию.

Как и процедура `Sub`, функция *без параметров* (аргументов) может быть запущена автономно из редактора VBA.

**Вызов процедуры:** вызов процедуры `Sub` из другой процедуры можно произвести несколькими способами. Первый способ вызова процедуры `Sub`:

*Синтаксис:*

ИмяПроцедуры СписокФактическихПараметров

СписокФактическихПараметров – список аргументов, передаваемых процедуре. Он должен соответствовать списку, заданному в описании процедуры по количеству и типу.

Если требуется использовать несколько процедур с одинаковыми названиями, расположенных в разных модулях, при их вызове после имени процедуры через точку надо указывать имя модуля в котором они расположены:

*Синтаксис:*

ИмяМодуля.ИмяПроцедуры СписокФактическихПараметров

Второй способ вызова процедуры sub производится с помощью инструкции **Call**.

*Синтаксис:*

**Call** ИмяПроцедуры (СписокФактическихПараметров)

Обратите внимание, что в этом случае список фактических параметров заключается в скобки.

#### 4.7. Операторы ввода и вывода

С помощью встроенных диалоговых окон осуществляют ввод и вывод информации и результаты программ.

Окно сообщений (MsgBox) выводит простейшие сообщения для пользователя, а окно ввода (InputBox) обеспечивает ввод информации.

<p>Функция <b>InputBox</b></p>	<p>Выводит на экран диалоговое окно, содержащее сообщение и поле ввода, две кнопки <b>ОК</b> и <b>Cancel</b>. Устанавливает режим ожидания ввода текста пользователем и нажатия кнопки. Если нажата кнопка <b>ОК</b>, то функция возвращает текст, введенный в поле ввода. Если нажата кнопка <b>Cancel</b>, то возвращает пустую строку.</p> <p>Синтаксис:</p> <pre>InputBox(Prompt[, Title] [, Default]         [, xPos] [, yPos] [,Helpfile, Context])</pre> <ul style="list-style-type: none"> <li>• <b>Prompt</b> – строковое выражение, отображаемое как <i>сообщение</i> в диалоговом окне. Строковое выражение <b>Prompt</b> может содержать несколько строк. Для разделения строк допускается использование символа возврата каретки (Chr(13)), символа перевода строки (Chr(10)) или комбинацию этих символов (Chr(13) &amp; Chr(10));</li> <li>• <b>Title</b> – строковое выражение, отображаемое в <i>строке заголовка</i> диалогового окна. Если этот параметр опущен, то в строку заголовка помещается имя приложения;</li> <li>• <b>Default</b> – строковое выражение, отображаемое в <i>поле ввода</i> как используемое <i>по умолчанию</i>, если пользователь не введет другую строку. Если этот параметр опущен, то поле ввода изображается пустым;</li> <li>• <b>xPos</b> – числовое выражение, задающее расстояние по горизонтали между левой границей диалогового окна и левым краем экрана. Если этот параметр опущен, то диалоговое окно выравнивается по центру экрана по горизонтали;</li> <li>• <b>yPos</b> – числовое выражение, задающее расстояние по вертикали между верхней границей диалогового окна и верхним краем экра-</li> </ul>
------------------------------------	---


	<p>на. Если этот параметр опущен, то диалоговое окно помещается по вертикали примерно на одну треть высоты экрана;</p> <ul style="list-style-type: none"> <li>• <code>Helpfile</code> – строковое выражение, определяющее имя файла справки, содержащего справочные сведения о данном диалоговом окне. Если этот параметр указан, то необходимо указать также параметр <code>Context</code>;</li> </ul> <p><code>Context</code> – числовое выражение, определяющее номер соответствующего раздела справочной системы. Если этот параметр указан, то необходимо указать также параметр <code>Helpfile</code>.</p>
<b>Процедура MsgBox</b>	<p>Выводит на экран диалоговое окно, содержащее сообщение, устанавливает режим ожидания нажатия кнопки пользователем, а затем возвращает значение типа <code>Integer</code>, указывающее, какая кнопка была нажата.</p> <p>Синтаксис:  <code>MsgBox(Prompt[, Buttons] [, Title] [, Helpfile, Context])</code></p> <ul style="list-style-type: none"> <li>• <code>Prompt</code> – строковое выражение, отображаемое как <i>сообщение</i> в диалоговом окне;</li> <li>• <code>Buttons</code> – числовое выражение, представляющее сумму значений, которые указывают <i>число и тип отображаемых кнопок</i>, тип используемого значка, основную. Значение по умолчанию этого параметра равняется 0. Значения констант, определяющих число, тип кнопок и используемого значка, приведены в таблице 2;</li> <li>• <code>Title</code> – строковое выражение, отображаемое в строке заголовка диалогового окна. Если этот параметр опущен, то в строку заголовка помещается имя приложения;</li> </ul> <p><code>Helpfile</code> – строковое выражение, определяющее <i>имя файла справки</i>, содержащего справочные сведения о данном диалоговом окне. Если этот параметр указан, необходимо указать также параметр <code>Context</code>;</p>

Значения параметра `Buttons` процедуры `MsgBox`, определяющие отображаемые кнопки в диалоговом окне:

Константа	Значение	Отображаемые кнопки
<code>vbOKOnly</code>	0	
<code>vbOKCancel</code>	1	 
<code>vbAbortRetryIgnore</code>	2	  
<code>vbYesNoCancel</code>	3	  
<code>vbYesNo</code>	4	 
<code>vbRetryCancel</code>	5	 

При вызове процедуры можно использовать как приведенные константы, так и эквивалентные им числовые значения.

Значения параметра `Buttons` процедуры `MsgBox`, определяющие отображаемые значки в диалоговом окне:

Константа	Значение	Значок сообщения
<code>vbCritical</code>	16	
<code>vbQuestion</code>	32	
<code>vbExclamation</code>	48	
<code>vbInformation</code>	64	

При написании программ с откликом в зависимости от того, какая кнопка диалогового окна нажата, вместо возвращаемых значений удобнее использовать следующие константы VBA:

Константа	Значение	Нажатая кнопка
<code>vbOK</code>	1	ОК
<code>vbCancel</code>	2	Отмена (Cancel)
<code>vbAbort</code>	3	Прервать (Abort)
<code>vbRetry</code>	4	Повторить (Retry)
<code>vbIgnore</code>	5	Пропустить (Ignore)
<code>vbYes</code>	6	Да (Yes)
<code>vbNo</code>	7	Нет (No)

#### 4.8. Свойства, методы пользовательской формы *UserForm*

Свойство	Описание
<b>Name</b>	Возвращает или устанавливает имя пользовательской формы.
<b>Caption</b>	Возвращает или устанавливает текст, отображаемый в строке заголовка формы.
<b>BorderStyle</b>	Возвращает или устанавливает тип границы. Достижимо только на этапе конструирования. Допустимые значения: <ul style="list-style-type: none"> <li><code>fmBorderStyleNone</code> или 0 (нет видимой границы);</li> <li><code>fmBorderStyleSingle</code> или 1 (имеется граница, используется по умолчанию).</li> </ul>
<b>BorderColor, BackColor</b>	Возвращают и устанавливают цвет фона и текста формы. Цвета зашифровываются в шестнадцатеричной системе счисления. Для удобства работы с цветами часто вместо их шестнадцатеричного представления используются константы, встроенные в VBA.
<b>Left, Top</b>	Возвращают или устанавливают местоположение верхнего левого угла формы.
<b>Height, Width</b>	Возвращают или устанавливают высоту и ширину формы.

	Эти свойства включают в себя толщину границы формы и строку заголовка.
<b>InsideHeight, InsideWidth</b>	Возвращают высоту и ширину формы без учета толщины границы и строки заголовка.
<b>Метод</b>	<b>Описание</b>
<b>Show</b>	Отображает форму на экране.
<b>Hide</b>	Закрывает форму.
<b>Move</b>	Изменяет местоположение и размер формы.
<b>PrintForm</b>	Печатает изображение формы.
<b>Событие</b>	Описание
<b>Initialize</b>	Происходит во время конфигурирования формы, но до ее загрузки.
<b>Load</b>	Происходит после инициализации формы, но до ее отображения на экран.
<b>Unload</b>	Событие, противоположное Load. Обычно используется для того, чтобы уточнить, действительно ли пользователь желает закрыть форму.
<b>Click, DblClick</b>	Происходят при щелчке на форме. Происходят при двойном щелчке на форме.

Элементы управления – **объекты, которые разработчик приложения может поместить на форму:**

Элемент управления	Имя	а, его создающая
Поле	TextBox	
Надпись	Label	
Кнопка	CommandButton	
Список	ListBox	
Поле со списком	ComboBox	
Полоса прокрутки	ScrollBar	
Счетчик	SpinButton	
Переключатель	OptionButton	
Флажок	CheckBox	
Выключатель	ToggleButton	
Рамка	Frame	

**Общие свойства элементов управления:**

Свойство	Описание
<b>Name</b>	Имя элемента управления



<b>Caption</b>	Возвращает или устанавливает надпись, отображаемую при элементе управления.
<b>AutoSize</b>	Логическое свойство, которое устанавливает режим автоматического изменения размеров элемента управления так, чтобы на нем полностью помещался текст, являющийся значением свойства <code>Caption</code> .
<b>Visible</b>	Логическое свойство, которое определяет, надо ли отображать элемент управления во время выполнения программы.
<b>Enabled</b>	Логическое свойство, которое определяет, достижим ли для пользователя элемент управления во время работы приложения.