Информатика и информационные технологии

Онацкий Александр Николаевич

Скоробогатова Марина Викторовна

ЗФО

Лабораторная работа 4
Тема 7. Основы программирования на языках высокого уровня
Управляющие структуры VBa. Операции с массивами и функциями VBa

Управление объектами и коллекциями. Контроль за выполнением кода. Циклическая обработка инструкций. Синтаксис функции. Вызов функции из процедуры и использование функции на рабочем листе. Встроенные функции. Объявление одномерных, многомерных и динамических массивов. Пример программы для сортировки элементов массива.

Управление объектами и коллекциями

VBa располагает двумя конструкциями, которые упрощают управление объектами и коллекциями:

- конструкция With — end With;
- конструкция For each — Next.

Конструкция With — end With позволяет выполнять несколько операций над одним объектом.

Чтобы понять, как она работает, проанализируем следующую процедуру, которая изменяет шесть свойств выделенного объекта (подразумевается, что выделен объект Range).

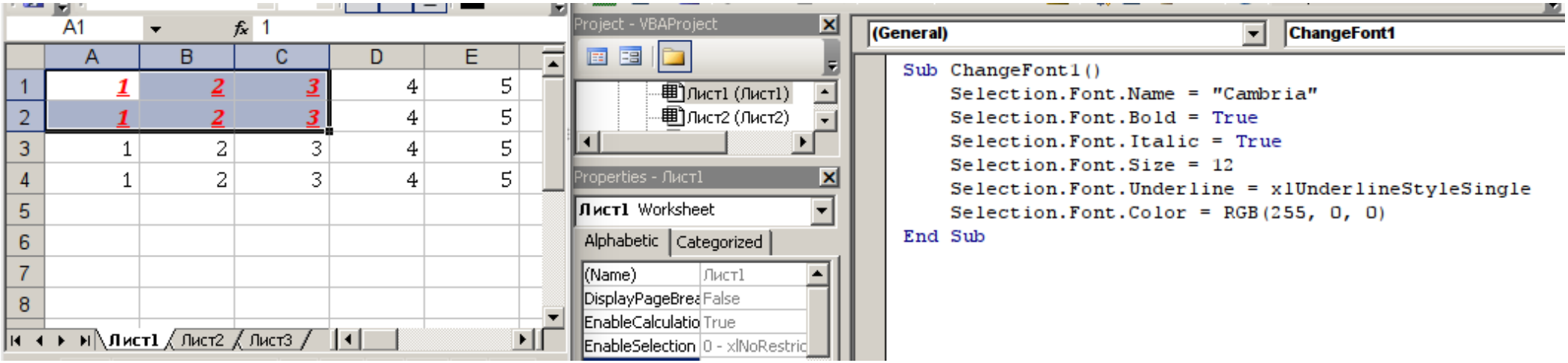
Здесь **Font** — это свойство объекта **Selection**.

Свойство Font (характеристики шрифта) — составное и представляет собой **структуру данных**, в которую, среди прочих, входят элементарные свойства:

- Name — название шрифта,
- Font.Size — кегль символов (число),
- Font.Italic — признак курсивного начертания (TRUe/FaLSe),
- Bold — признак жирного начертания (TRUe/FaLSe),
- Underline — подчеркивание,
- color — цвет шрифта.

Задание 1. Ввести программный код, запустить на выполнение и проанализировать.

```
Sub   changeFont1()
      Selection.Font.Name = "cambria"
      Selection.Font.Bold = True
      Selection.Font.Italic = True
      Selection.Font.Size = 12
      Selection.Font.Underline = xlUnderlineStyleSingle 'подчеркивание
      Selection.Font.color = RGB(255, 0, 0) ' red
end   Sub
```



Эту процедуру можно переписать с помощью конструкции **With — end With**.

Задание 2. Ввести программный код, запустить на выполнение и проанализировать.

```
Sub   changeFont2()
    With   Selection.Font
        .Name = "cambria "
        .Bold = True
        .Italic = True
        .Size = 12
        .Underline = xlUnderlineStyleSingle
        .color = RGB(255, 0, 0)
    end With
end Sub
```

Процедура, использующая для изменения нескольких свойств одного объекта конструкцию With — end With, помогает повысить эффективность выполнения кода.

Контроль за выполнением кода

Некоторые процедуры VBa начинают выполняться с первых строк кода. Этот процесс происходит построчно до самого конца процедуры (например, так всегда работают макросы, записанные при выполнении действий).

Однако иногда необходимо контролировать последовательность операций, пропуская отдельные операторы, повторно выполняя некоторые команды и проверяя условия для определения следующего действия, выполняемого процедурой.

Контроль за выполнением процедур, написанных на VBa, можно выполнять с помощью управляющих конструкций ветвления и цикла.

Ранее была описана конструкция For each - Next, которая является циклической структурой.

Управляющие конструкции VBa для контроля кода:

- конструкция For each — Next;
- оператор GoTo;
- конструкция If — Then;
- конструкция Select case;
- цикл For — Next;
- цикл Do While;
- цикл Do Until.

Оператор Goto.

Самый простой способ изменить последовательность операций в коде — использовать оператор GoTo.

Он перенаправляет ход выполнения программы на новую инструкцию, которая **помечена специальным образом**:

- текстовая строка, заканчивающаяся двоеточием,
- число, заканчивающееся пробелом.

Процедуры VBa могут содержать любое количество меток, а оператор GoTo не определяет переход за пределы процедуры.

В приведенной ниже процедуре применена функция VBa InputBox для получения имени пользователя.

Если имя пользователя отличается от «Иван», то процедура переходит к метке WrongName, на чем заканчивает свою работу.

В противном случае процедура выполняет дополнительные операции.

Оператор exit Sub заканчивает выполнение процедуры.

Задание 3. Ввести программный код, запустить программу на выполнение и проанализировать.

```
Sub   GoToDemo()
    UserName = InputBox("Введите свое имя:")
    If   UserName <> "Иван"   Then   GoTo   WrongName
    MsgBox("Привет, Иван!")
    exit   Sub
WrongName:
    MsgBox   "Извините, эту процедуру может запускать только Иван."
end   Sub
```

Представленная процедура работает, но оператор GoTo, как правило, используется, если другого способа выполнить действие просто не существует. Единственной ситуацией, когда оператор GoTo в VBa действительно необходим, является **перехват ошибок**.

Конструкция If — Then

Конструкция If — Then чаще остальных используется для группирования инструкций VBa.

Эта популярная конструкция наделяет приложения способностью **принимать решения**. Такая способность является ключевой при создании эффективных программ.

Стандартный синтаксис конструкции If — Then таков:

If условие Then инструкции_истина [else инструкции_ложь]

Конструкция If — Then используется для выполнения одного или более операторов при справедливости заданного условия. Оператор else необязателен. Он позволяет выполнять одну или более инструкций в случае невыполнения условия.

В описанной ниже процедуре применена структура If — Then без оператора else.

Время дня выражается дробным числом, например полдень представлен как 0.5.

Функция VBa **Time()** возвращает значение, представляющее время в формате системных часов.

В следующем примере сообщение отображается, если текущее время меньше полудня.

Задание 4. Ввести программный код, запустить программу на выполнение и проанализировать.

```
Sub   GreetMe1()
    If   Time() < 0.5   Then   MsgBox "Доброе утро"
end   Sub
```

Задание 5. Ввести программный код, запустить программу на выполнение и проанализировать.

```
Sub   GreetMe1a()
    If   Time() < 0.5 Then
         MsgBox "Доброе утро"
    end   If
end   Sub
```

Оператору If в этом случае должен соответствовать оператор end If.

В этом примере вызывается на выполнение только один оператор, если условие равно True.

Между операторами If и end If ***можно поместить любое количество операторов***.

В более сложных приложениях рекомендуется использовать следующий синтаксис.

If условие Then

[инструкции_истина]

[elseIf условие — n Then

[альтернативные_инструкции]]

[else

[операторы_по_умолчанию]]

end If

Ниже продемонстрирован пример процедуры, которая запрашивает у пользователя значение переменной Quantity и отображает скидку на основе полученного значения.

Если в окне InputBox пользователь щелкнет на кнопке «Отмена», то переменная Quantity приравняется к пустой строке и процедура будет завершена.

Эта процедура не выполняет проверку введенного числовое значение на отрицательность.

Задание 6. Ввести программный код и запустить программу на выполнение.

```
Sub   Discount()
    Dim   Quantity   as   Variant
    Dim   Discount   as  Double
    Quantity = InputBox("Укажите   количество: ")
    If   Quantity = " "   Then   exit   Sub
```

```
    If  Quantity >= 0   and   Quantity < 25 Then
        Discount = 0.1
    elself   Quantity < 50 Then
        Discount = 0.15
    elself   Quantity < 75 Then
        Discount = 0.2
    else
        Discount = 0.25
    endIf
    MsgBox   "Скидка: "   & Discount
end   Sub
```

Вложенные структуры If — Then достаточно *громоздкие*. Поэтому рекомендуется использовать их только для принятия простых бинарных решений. Если же необходимо выбрать между тремя и более вариантами, то целесообразно обратиться к конструкции Select case, которая рассматривается далее.

Конструкция Select case

Конструкция Select case применяется при выборе между тремя и более вариантами. Она справедлива также для двух вариантов и является хорошей альтернативой структуре If — Then — else.

Конструкция Select case имеет следующий синтаксис.

Select case тестируемое_выражение

[case список условий -n

[операторы — n]]

[case else

[операторы_по_умолчанию]]

end Select

Приведем пример конструкции Select case, который показывает еще один способ запрограммировать процедуру GreetMe, рассмотренную ранее.

Задание 7. Ввести программный код и запустить программу на выполнение.

```
Sub   GreetMe()
    Dim   Msg   as   String
    Select   case   Time()
        case   Is < 0.5
            Msg = "Доброе утро"
        case   0.5   To   0.75
            Msg = "Добрый день"
        case   else
            Msg = "Добрый вечер"
    end   Select
    MsgBox   Msg
end   Sub
```

В операторе case можно использовать запятую, разделяющую несколько значений для одного варианта:

case 1, 7

Задание 8. Ввести программный код и запустить программу на выполнение.

```
Sub   GreetUser1()
    Select   case   Weekday(Now)
        case   1, 7
            MsgBox "Это выходные"
        case   else
            MsgBox "Это рабочие дни"
    end   Select
end Sub
```

Под каждым оператором case может указываться любое количество инструкций, и они выполняются при условии case, имеющем значение True.

Интерпретатор VBa осуществляет выход из конструкции Select case, как только найдено условие True. Следовательно, для максимальной эффективности, в первую очередь, следует выполнить проверку наиболее вероятного случая.

Структуры Select case можно вкладывать друг в друга. Можно создавать конструкции Select case любой степени вложенности, но важно убедиться, что каждому оператору Select case соответствует свой оператор end Select.

Циклическая обработка инструкций

Цикл — это процесс повторения набора инструкций. Возможно, вы заранее знаете, сколько раз должен повториться цикл, или это значение определяется переменными в программе.

Поскольку в VBa встроено несколько структурированных команд циклов, в процессе принятия решений практически никогда не требуется прибегать к операторам GoTo.

VBa — структурированный язык. Он предлагает:

- разветвляющиеся конструкции:
  1. If — Then — else
  2. Select case,
- циклы:
  1. For — Next,
  2. Do Until
  3. Do While.

Циклы For — Next

Оператор цикла — For — Next имеет следующий синтаксис.

For счетчик = начало To конец [Step шаг]

[инструкции]

[exit For]

[инструкции]

Next [счетчик]

Ниже приведен пример цикла For — Next, в котором не используется значение шаг переменной Step и необязательный оператор exit For.

Эта процедура выполняет выражение Sum=Sum+Sqr(count) 100 раз и отображает результат — сумму квадратных корней первых 100 целых чисел.

Задание 9. Ввести программный код и запустить программу на выполнение.

```
Sub   SumSquareRoots()
    Dim   Sum   as   Double
    Dim   count   as   Integer
    Sum = 0
    For   count = 1   To   100    Step   1
        Sum = Sum + Sqr(count)
    Next   count
    MsgBox   Sum
end   Sub
```

В данном примере count (переменная — счетчик цикла) имеет начальное значение 1 и увеличивается на 1 при каждом повторении цикла.

Переменная Sum суммирует квадратные корни каждого значения count.

Используя циклы For — Next, важно понимать, что в качестве счетчика цикла используется **обычная переменная**, и ничего больше.

В результате значение счетчика цикла можно изменять в блоке программы, который выполняется между операторами For и Next. Однако это **очень неудачный прием**, поскольку он может привести к непредсказуемым результатам.

Необходимо принять специальные меры предосторожности, чтобы удостовериться в том, что программа не изменяет счетчик цикла.

Можно также указать значение ***шаг*** для переменной Step, чтобы пропустить отдельные итерации цикла.

Ниже рассмотрена исходная процедура, переписанная так, что суммируются квадратные корни всех нечетных чисел в диапазоне от 1 до 100.

В цикл можно помещать любое количество операторов и вкладывать циклы For — Next в другие циклы For — Next.

```
For   i = 1   To   10
    For   j = 1   To   10
        ' оператор 1
        ' оператор 2
        ' оператор 3
    Next   j
Next   i
```

Циклы Do While

Оператор Do While — еще один тип циклической структуры, представленной в VBa.

В отличие от цикла For — Next, цикл Do While выполняется до тех пор, пока удовлетворяется заданное условие.

Цикл Do While может иметь один из двух представленных ниже синтаксисов.

Do [While условие]

[инструкции]

[exit Do]

[инструкции]

Loop

или

Do

[инструкции]

[exit Do]

[инструкции]

Loop [While условие]

Как мы видим, VBa позволяет проверять условие While в начале или в конце цикла. Разница между этими двумя синтаксисами связана с моментом, когда оценивается условие. В первом синтаксисе содержимое цикла может вообще не выполняться. Во втором содержимое цикла всегда выполняется (как минимум один раз).

В следующем примере в активный рабочий лист вставляется набор дат. Эти даты соответствуют дням текущего месяца, а их ввод в столбец осуществляется, начиная с активной ячейки.

В этом примере используются ***некоторые функции VBa, обрабатывающие даты***:

- Date возвращает текущую дату;
- Month возвращает номер месяца для даты, являющейся аргументом;

- DateSerial конструирует дату на основе значений дня, месяца и года, указанных в качестве аргументов.

Рассмотрим примере цикла Do While, в котором реализована проверка условия в начале цикла:

процедура enterDates1 вводит даты текущего месяца в столбец рабочего листа, начиная с активной ячейки.

Задание 10. Ввести программный код, запустить программу на выполнение и проанализировать.

```
Sub   enterDates1()
    ' цикл   Do   While, условие проверяется в начале
    Dim   TheDate   as  Date
    TheDate = DateSerial(Year(Date), Month(Date), 1)
    Do   While   Month(TheDate) = Month(Date)
```

```
        activecell = TheDate
        TheDate = TheDate + 1
        activecell.offset(1, 0).activate
    Loop
end   Sub
```

В этой процедуре используется переменная TheDate, которая хранит даты, записанные в рабочем листе. Для инициализации переменной используется первый день текущего месяца.

В процессе выполнения цикла значение переменной TheDate было введено в активную ячейку, затем это значение было увеличено на единицу, после чего активизируется следующая ячейка.

Цикл выполняется до тех пор, пока значение месяца, присвоенное переменной TheDate, совпадет со значением месяца текущей даты.

Результат выполнения следующей процедуры будет таким же, что и результат выполнения процедуры enterDatesl, но в данном случае используется вторая форма синтаксиса цикла Do While, предусматривающая проверку условия в конце цикла.

Задание 11. Ввести программный код, запустить программу на выполнение и проанализировать.

```
Sub   enterDates2()
    ' цикл Do While, условие проверяется в конце
    Dim   TheDate   as   Date
    TheDate = DateSerial(Year(Date), Month(Date), 1)
    Do
        activecell = TheDate
        TheDate = TheDate + 1
        activecell.offset(1, 0).activate
    Loop   While   Month(TheDate) = Month(Date)
end   Sub
```

Циклы Do Until

Структура цикла Do Until имеет много общего с конструкцией Do While. Разница заключается лишь в том, как проверяется условие цикла.

В варианте Do While цикл выполняется до тех пор, пока выполняется условие.

В цикле Do Until цикл выполняется, пока условие не станет выполняться.

Структура Do Until может быть представлена двумя вариантами синтаксиса.

```
Do [Until условие]
    [инструкции]
    [exit Do]
    [инструкции]
Loop
```

или

Do

[инструкции]

[exit Do]

[инструкции]

Loop [Until условие]

Ниже показаны два примера кода, которые выполняют те же действия, что и рассмотренные ранее примеры циклов Do While.

Отличие между этими процедурами заключается в месте проверки условия (в начале либо в конце цикла).

Задание 12. Ввести программный код, запустить программу на выполнение и проанализировать.

```
Sub   enterDates3()
    ' Цикл Do Until, проверка условия в начале
    Dim   TheDate   as   Date
    TheDate = DateSerial(Year(Date) ,Month(Date), 1)
    Do   Until   Month(TheDate) <> Month(Date)
        activecell = TheDate
        TheDate = TheDate + 1
        activecell.offset(1, 0).activate
    Loop
end   Sub
```

Задание 13. Ввести программный код, запустить программу на выполнение и проанализировать.

```
Sub   enterDates4()
    ' Цикл Do Until, проверка условия в конце
    Dim   TheDate   as   Date
    TheDate = DateSerial(Year(Date), Month(Date), 1)
    Do
        activecell = TheDate
        TheDate = TheDate + 1
        activecell.offset(1, 0).activate
    Loop   Until   Month(TheDate) <> Month(Date)
end   Sub
```

Создание функций

Задание 14. Изучить.

Функция — это процедура VBa, которая выполняет вычисления и возвращает значение. Функции можно использовать в коде Visual Basic for applications (VBa) или в формулах.

Процедуру можно рассматривать как команду, которая выполняется пользователем или другой процедурой. С другой стороны, процедуры типа *Function* обычно возвращают отдельное значение (или массив) подобно функциям рабочих листов excel и встроенным функциям VBa.

Как и встроенные функции, процедуры типа *Function* имеют аргументы.

Процедуры типа *Function* универсальны и **используются в двух ситуациях**:

- как часть выражения в процедуре VBa;
- в формулах, которые создаются на рабочем листе.

Процедуру типа Function можно использовать в работе с функциями excel и встроенными функциями VBa. Единственное исключение — ***невозможно использовать функцию VBa в формуле проверки данных***.

Создавая пользовательские функции, которые используются в формуле рабочего листа, необходимо **убедиться, что код вводится в обычном модуле** VBa.

Если пользовательские функции поместить в модуле листа «Лист» (Sheet), в пользовательской форме (UserForm) или в модуле «ЭтаКнига» (ThisWorkbook), они **не будут выполняться в формулах**.

Синтаксис функции

Пользовательские функции имеют много общего с процедурами типа Sub.

Для объявления функции применяется следующий синтаксис.

[Public | Private] [Static] Function *имя* ([список_аргументов] [as тип])

[инструкции]

[имя = выражение]

[exit Function]

[инструкции]

[имя = выражение]

end Function

Функция состоит из следующих ключевых элементов.

- ***Public*** (необязательное ключевое слово). Указывает, что функция доступна для других процедур во всех остальных модулях активных проектов VBa.

- ***Private*** (необязательное ключевое слово). Указывает, что функция доступна только для других процедур в текущем модуле.
- ***Static*** (необязательное ключевое слово). Указывает, что значения переменных, которые объявлены в функции, сохраняются между вызовами функции.
- ***Function*** (обязательное ключевое слово). Обозначает начало функции, возвращающей значение или другие данные.
- ***Имя*** (обязательное ключевое слово). Представляет произвольное название функции. При именовании функций используются те же правила, что и при задании имен переменным. По окончании выполнения функции результат присваивается ее названию.
- ***Список_аргументов*** (необязательный). Список одной или нескольких переменных, представляющих аргументы, которые передаются в функцию. Аргументы заключены в скобки. Для разделения пар аргументов используется запятая.

- ***Тип*** (необязательный). Тип данных, который возвращает функция.
- ***Инструкции*** (необязательные). Корректные инструкции VBa.
- ***exit Function*** (необязательный). Оператор, вызывающий немедленный выход из функции до ее завершения.
- ***end Function*** (обязательное). Ключевое слово, обозначающее конец функции.

О пользовательских функциях, написанных на VBa, необходимо знать следующее: значение всегда присваивается названию функции минимум один раз и, как правило, тогда, когда функция завершила выполнение.

Создание пользовательской функции начинается с создания модуля VBa (можно также использовать существующий модуль).

Вводится ключевое слово *Function*, после которого указывается название функции и список ее аргументов (если они есть) в скобках. Также можно объявить тип данных значения, которое возвращает функция, используя ключевое слово *as* (это делать необязательно, но рекомендуется).

Затем пишется код VBa, выполняющий требуемые действия.

Необходимое значение должно быть присвоено переменной процедуры, соответствующей названию функции, минимум один раз в теле функции. Функция заканчивается оператором *end Function*.

Имена функций подчиняются тем же правилам, что и имена переменных.

Выполнение функций

В отличие от процедур, функции не отображаются в диалоговом окне ***Макрос*** (Macro) при выполнении команды *Разработчика* → *Код* → *Макросы* (*Developer* → *code* → *Macros*).

Кроме того, функцию нельзя выбрать при использовании команды VBe *Run → Sub/UserForm* (*Выполнить → Процедуру/пользовательскую форму*) (или нажатии <F5>), если курсор установлен в тексте функции (так как при этом отображается диалоговое окно «Макрос», в котором можно выбрать макрос для выполнения).

Именно поэтому необходимо выполнить дополнительные действия по тестированию функций еще в процессе разработки.

Один из возможных подходов — создать простую процедуру, вызывающую функцию.

Если функция должна использоваться в формулах рабочего листа, то для ее проверки следует ввести простую формулу.

<u>Функцию (Function) можно вызвать одним из следующих способов</u>:

- вызвать ее из другой процедуры;
- включить ее в формулу рабочего листа;
- включить в формулу условного форматирования;
- вызвать ее в окне отладки VBe (Immediate).

Вызов из процедуры

Пользовательские функции можно вызывать из процедуры точно так же, как и встроенные функции. Например, после определения функции с названием Sumarray в код вводится оператор, показанный ниже.

Total = Sumarray(MyArray)

Этот оператор выполняет функцию Sumarray с аргументом MyArray, возвращает результат функции и присваивает его переменной Total.

Кроме того, можно использовать метод Run объекта application, например:

Total = application.Run ("Sumarray", "MyArray")

Первый аргумент метода Run — имя функции. Дополнительные аргументы представляют аргумент (аргументы) функции.

Аргументами метода Run могут быть строки, числа или переменные.

Использование функции на рабочем листе

При вводе формулы, в которой используется пользовательская функция *MyFunction*, excel выполняет программу для получения конечного значения.

Эту функцию можно использовать в следующей формуле: **=MyFunction(a1)**

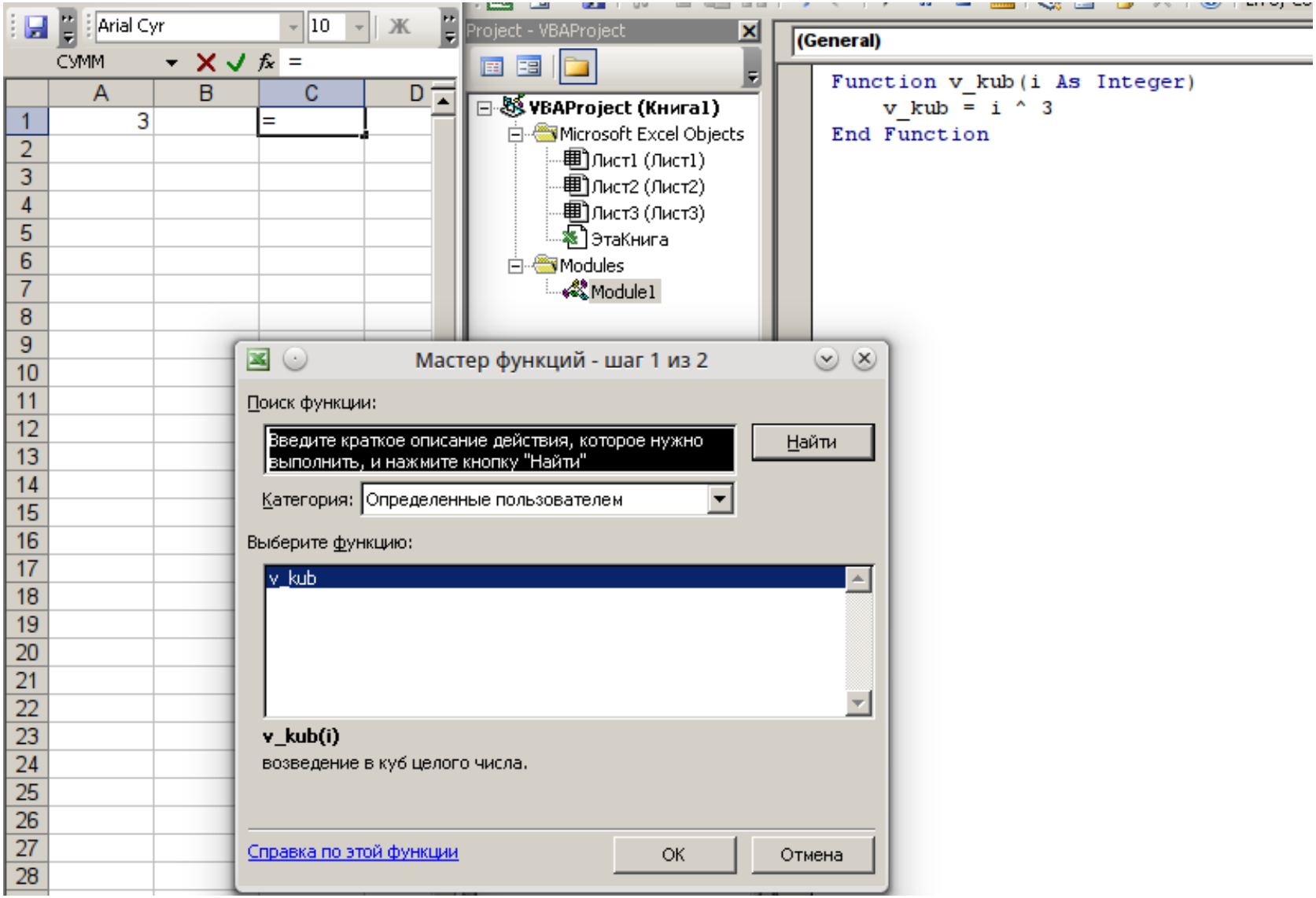
Функция в качестве аргумента используется текст в столбце А.

Фактически такая функция действует подобно любой встроенной функции рабочего листа.

Можно вставить функцию в формулу, используя команду *Формулы → Библиотека функций → Вставить функцию* (*Formulas → Function Library → Insert Function*) или кнопку «Мастера функций» в левой части строки формул.

Любое из этих действий приведет к появлению диалогового окна «Мастер функций» (Insert Function), в котором находятся пользовательские функции (обычно — в категории «Определенные пользователем» (User Defined)).

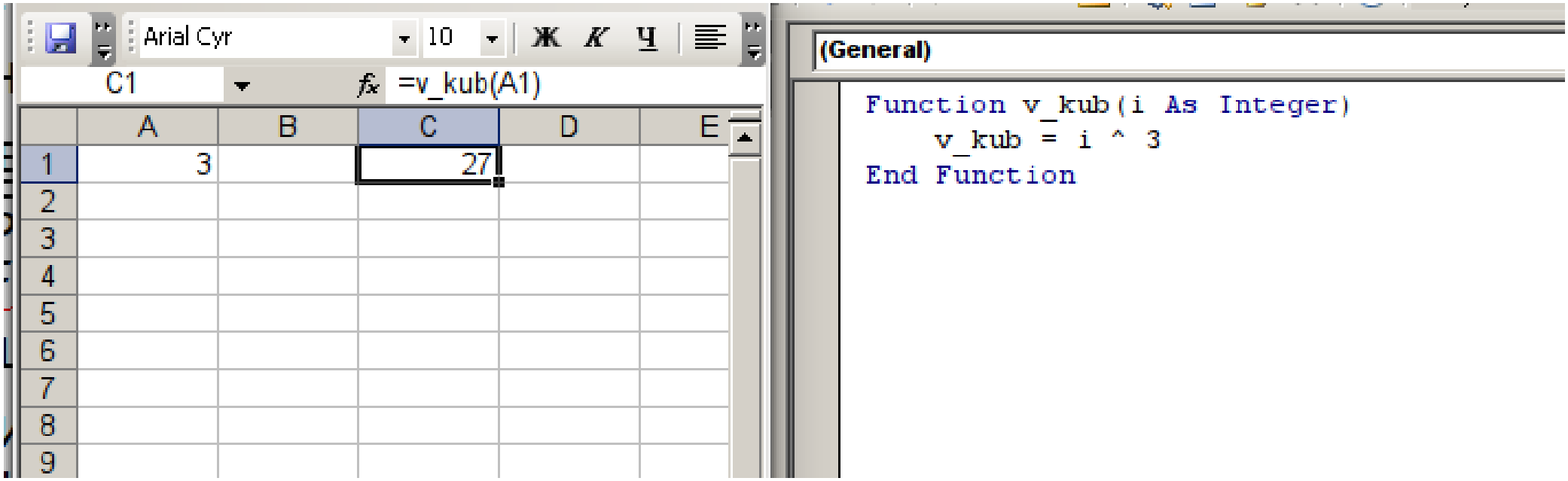
Задание 15. Ввести программный код (скрин ниже), запустить программу на выполнение и проанализировать.



Добавление описания к пользовательской функции:

Сервис → Макрос → Макросы. Попадаем в диалог запуска макросов. Но в нём будут отображаться только подпрограммы (SUB), а не функции (FUNCTION). Вводим в качестве имени макроса имя функции и нажимаем «Параметры». Заполняем поле описание и нажимаем «ОК».

Результат выполнения пользовательской функции:



Использование функции в процедуре VBa

Пользовательские функции можно применять не только в формулах рабочего листа, но и **в процедурах VBa**.

Следующая процедура VBa, определенная в том же модуле, что и пользовательская функция *MyFunction*, сначала отображает окно для ввода текста пользователем.

Затем процедура применяет встроенную функцию VBa *MsgBox* для отображения данных, введенных пользователем, но уже после их обработки функцией *MyFunction*. Первоначальные данные отображаются в заголовке окна сообщения.

Встроенные функции

Как и в большинстве других языков программирования, в VBa есть ряд встроенных функций, упрощающих вычисления и операции.

Часто функции позволяют выполнить операции, которые по-другому осуществить сложно или вообще невозможно. Многие функции VBa подобны (или идентичны) функциям excel.

Например, функция VBa *Ucase*, преобразующая строку в верхний регистр, эквивалентна функции excel *ПРОПИСН*.

Чтобы получить список функций VBa при написании кода, нужно ввести VBa и точку.

После этого VBe отобразит список всех вложенных в объект VBa объектов, включая функции).

Функции обозначаются зеленым значком. Если этот способ недоступен, необходимо проверить, включен ли параметр *auto List Members* (Автоматическая вставка объектов).

Для этого нужно выбрать команду *Tools → options* (*Сервис → Параметры*) и щелкните на вкладке *editor* (Редактор).

Функции VBa используются в выражениях почти так же, как и функции excel в формулах рабочего листа. Например, можно создавать вложенные функции VBa.

Задание 16. Ввести программный код, запустить и проанализировать.

```
Sub   ShowRoot()
    Dim   MyValue   as   Double
    Dim   SquareRoot   as   Double
    MyValue = 25
    SquareRoot = Sqr(MyValue)
    MsgBox   SquareRoot
end   Sub
```

Функция *Sqr* VBa эквивалентна функции рабочего листа КОРЕНЬ в excel. Имеется возможность использовать ряд (но не все) функций excel в коде VBa. Объект **WorksheetFunction**, который содержится в объекте application, располагает всеми функциями рабочего листа, которые можно вызвать в процедуре VBa.

Чтобы использовать функцию excel в операторе VBa, перед названием функции нужно ввести следующее выражение:

application.WorksheetFunction

Рассмотрим одну из наиболее часто используемых встроенных функций VBa — **Функция MsgBox**.

Функция MsgBox — одна из самых полезных в VBa. Она используется для отображения значений переменных.

Данная функция часто является достойной заменой простой форме. Кроме того, это превосходный *инструмент отладки*,

поскольку можно в любое время вставить функцию *MsgBox*, чтобы приостановить программу и отобразить результат вычисления или присваивания.

Как правило, функции возвращают одно значение, которое можно присвоить переменной.

Функция MsgBox не только возвращает значение, но и отображает диалоговое окно, в котором пользователь может выполнить определенные действия.

Значение, возвращаемое функцией MsgBox, является <u>ответом пользователя на отображенный запрос</u>.

Функция MsgBox может применяться даже в том случае, когда ответ пользователя не требуется, а нужно отобразить сообщение.

Формальный синтаксис функции MsgBox предполагает использование пяти аргументов (аргументы в квадратных скобках — *необязательные*).

MsgBox(сообщение[, кнопки] [, заголовок] [, файл_справки, контекст])

- *Сообщение* (обязательный аргумент) — сообщение, которое отображается в диалоговом окне.
- *Кнопки* (необязательный аргумент) — значение, определяющее, какие кнопки и пиктограммы (если нужно) отображаются в окне сообщения. Применяйте встроенные константы (например, vbYesNo).
- *Заголовок* (необязательный аргумент) — текст, который отображается в строке заголовка окна сообщения. По умолчанию отображается текст Microsoft excel.
- *Файл_справки* (необязательный аргумент) — название файла справки, соответствующего окну сообщения.

- *Контекст* (необязательный аргумент) — контекстный идентификатор раздела справки. Представляет конкретный раздел справки для отображения. Если используется аргумент *контекст*, следует также задействовать аргумент *файл_справки* .

Можно присвоить полученное значение переменной либо использовать функцию без оператора присваивания. В приведенном ниже примере результат присваивается переменной ans.

Задание 17. Ввести программный код, запустить и проанализировать.

```
Sub zadanie17()
    ans = MsgBox("Продолжить?",   vbYesNo + vbQuestion,    "Сообщи")
    If   ans = vbNo   Then   exit   Sub
end sub
```

Обратим внимание, что в качестве значения аргумента кнопки используется сумма двух встроенных констант (*vbYesNo* + *vbQuestion*).

Благодаря константе *vbYesNo* в окне сообщения отображаются две кнопки: одна с меткой *Yes*, а вторая — с меткой *No*. Добавление *vbQuestion* в состав аргумента также приведет к отображению значка вопроса.

Как только будет выполнен первый оператор, переменная *ans* получит одно из двух значений, представленных константами *vbYes* и *vbNo*.

В этом примере процедура завершает свою работу после щелчка на кнопке *No.*

Работа с массивами

Массив — это группа элементов одного типа, которые имеют общее имя. На конкретный элемент массива ссылаются, используя имя массива и индекс.

Например, можно определить массив из 12 строк так, чтобы каждая переменная соответствовала названию месяца.

Если назвать массив *MonthNames*, то можно обратиться к первому элементу массива как *MonthNames(0)*, ко второму — как *MonthNames(1)* и так до *MonthNames(11)*.

Объявление массивов

Массив, как и обычные переменные, объявляется с помощью операторов *Dim* и *Public*.

Кроме того, можно определить количество элементов в массиве: вводится нижний индекс, ключевое слово *To* и верхний индекс и вся конструкция должна быть заключена в скобки.

Например, объявить массив, содержащий 100 целых чисел, можно следующим образом:

Dim Myarray(1 To 100) as Integer

При объявлении массива в обязательном порядке указывается только верхний индекс; тогда VBa установит нижний индекс равным 0. Следовательно, два следующих оператора приведут к одинаковым результатам.

Dim Myarray(0 to 100) as Integer

Dim Myarray(100) as Integer

В обоих случаях массив состоит из 101 элемента.

По умолчанию в массивах VBa в качестве первого элемента используется ноль.

Если необходимо, чтобы в качестве первого индекса всех массивов использовалась единица, то перед первой процедурой модуля нужно сделать следующее объявление:

option Base 1

Объявление многомерных массивов

Массивы VBa могут иметь до 60 измерений, хотя на самом деле используется не более трех (трехмерные массивы). Показанный ниже оператор объявляет двухмерный 100-элементный массив целых чисел.

Dim Myarray(1 To 10, 1 To 10) as Integer

Этот массив можно рассматривать как матрицу значений размером 10x10. Чтобы обратиться к конкретному элементу двухмерного массива, используются два индекса.

Например, таким образом присваивается значение элементу предыдущего массива: **MyАггау(3, 4) = 125**

Трехмерный массив, содержащий 1000 элементов, можно представить в виде куба.

Dim MyАггау(1 To 10, 1 To 10, 1 To 10) as Integer

Чтобы обратиться к элементу внутри массива, используются три индекса.

MyАггау(4, 8, 2) = 0

Объявление динамических массивов

Динамический массив не имеет предопределенного количества элементов.

Он объявляется с незаполненными значениями в скобках.

Dim Myarray() as Integer

Прежде чем динамический массив можно будет использовать в программе, необходимо обратиться к оператору ***ReDim***, указывающему VBa, сколько элементов находится в массиве.

Для этого часто применяется переменная, значение которой неизвестно до тех пор, пока процедура не будет запущена на выполнение.

Например, если переменной *x* присвоено число, размер массива определяется с помощью следующего оператора:

ReDim Myarray (1 to x)

Оператор *ReDim* можно использовать повторно, изменяя размер массива по мере необходимости.

При изменении количества размерностей массива имеющиеся значения *утрачиваются*. Если нужно сохранить эти значения, используется оператором *ReDim Preserve*.

ReDim Preserve Myarray (1 to y)

Задание 18. Создать программу, вводящую массив из положительных целых чисел A(1), A(2), A(3), расположенных в ячейках A1, B1, C1.

Считая их массами гирь, выяснить, можно ли все эти гири как-либо положить на обе чаши весов таким образом, чтобы весы оказались в равновесии.

Если да, то расположить на рабочем листе Эксель гири левой и правой чаши в отдельных ячейках и подписать их. Если нет — вывести сообщение, что чаши весов уравновесить невозможно.

Это задание **ОТЧЁТНОЕ** — программный код переписать в тетрадь и сдать преподавателю.