## Лабораторная работа №1

## Основы программирования на языке VBA

**Цель работы.** Изучение основных операторов и реализации основных конструкций программирования в языке VBA.

#### 1.1. Основные этапы работы с программами на VBA в Excel.

Для подготовки и выполнения программы на языке VBA требуется выполнить следующее:

* в Excel выбрать из меню команду **Сервис – Макрос – Редактор Visual Basic;**
* в появившемся окне выбрать из меню команду **Insert – Module**. Создается модуль, т.е., упрощенно говоря, открывается окно, в котором можно вводить текст программы.

В начале модуля может указываться инструкция **Option Explicit**. Если она указана, то все переменные, используемые в программе, необходимо будет объявлять в операторе **Dim** (подробнее об этом см. в разделе 1.3).

В некоторых случаях, в зависимости от настройки среды VBA, инструкция **Option Explicit** указывается в начале модуля автоматически. Если программист желает использовать переменные, не объявляя их в операторе **Dim**, то инструкцию **Option Explicit** необходимо удалить.

Для запуска программы на выполнение необходимо выбрать из меню команду **Run – Run Sub/UserForm**.

#### 1.2. Простейший пример программы на языке VBA.

**Пример 1.1.** Программа, возводящаяуказанное число **a** в указанную степень **b**.

Sub primer1\_1()

‘Первый пример программы на VBA

a = InputBox(“Введите основание: ”)

b = InputBox(“Введите показатель степени: ”)

x = a^b ‘Возведение в степень

MsgBox(“Результат равен ” & x)

End Sub

Здесь слово **Sub** обозначает начало процедуры; ее имя в данном случае – **primer1\_1**. Программа на языке VBA всегда состоит из одной или нескольких процедур (в данном случае – из одной).

Символ ‘ (одиночная кавычка) обозначает начало комментария. Текст комментария может быть любым.

**InputBox** – функция для ввода значения переменной. Строка **a = InputBox (“Введите основание: ”)** означает, что вводится значение переменной **a**; при этом на экран выводится сообщение **“Введите основание:”**. Строка **x = a^b** – оператор присваивания: вычисляется значение правой части (в данном случае переменная **a** возводится в степень **b**), и результат присваивается переменной, указанной в левой части (в данном случае – переменной **x**). Строка **MsgBox(“Результат равен ” & x)** означает, что на экран выводится сообщение **“Результат равен ”** и значение переменной **x**.

**Примечание**. Знак **&** в функции **MsgBox** предназначен для сцепления нескольких элементов данных, которые требуется вывести на экран, в данном примере – строки **“Результат равен”** и переменной **x**. Аналогично знак **&** может использоваться в функции **InputBox** (примеры такого использования будут приведены далее). Перед знаком **&** и после него обязательно должны быть указаны пробелы.

В одной строке можно разместить несколько операторов языка VBA. Для этого они разделяются символами «двоеточие». Так, программу из примера 1.1 можно было записать, например, следующим образом:

Sub primer1\_1()

‘Первый пример программы на VBA

a = InputBox(“Введите основание: ”) : b = InputBox(“Введите показатель степени: ”)

x = a^b : MsgBox(“Результат равен ” & x) ‘Возведение в степень и вывод результата

End Sub

В рассмотренном примере использована процедура, называемая подпрограммой. Такая процедура начинается со слова **Sub**. В программе на VBA всегда имеется хотя бы одна процедура-подпрограмма. Кроме того, в языке VBA имеется еще один вид процедур – функции. Процедура-функция начинается со слова **Function**. Использование таких процедур будет рассмотрено в подразделе 1.9.

#### 1.3. Типы данных. Объявление переменных и констант.

#### 1.3.1. Типы данных.

Тип данных, указанный для переменной, определяет, какие значения может принимать эта переменная. Основные типы данных, используемые в VBA, приведены в таблице 1.1.

**Таблица 1.1. Основные типы данных в VBA**

|  |  |
| --- | --- |
| **Тип данных** | **Допустимые значения** |
| **Byte** (байт) | от 0 до 255 (только целые) |
| **Boolean** (логический) | **True** или **False** |
| **Integer** (целое) | от -32768 до 32767 (только целые) |
| **Long** (длинное целое) | от -2147483648 до 2147483647 (только целые) |
| **Single** (с плавающей точкой, обычной точности) | от -3,402823⋅1038 до 3,402823⋅1038 |
| **Double** (с плавающей точкой, двойной точности) | от -1,79769313486231⋅10308 до 1,79769313486232⋅10308 |
| **Date** (дата) | от 01.01.100 до 31.12.9999 |
| **String** (строка) | строки из любых символов, практически неограниченной длины |

**Примечание**. В данной таблице приведены только основные типы данных. Более подробные сведения о типах данных имеются в литературе по VBA, а также в справочной системе.

#### 1.3.2. Объявление переменных.

Тип переменной указывается при ее объявлении. Переменные обычно объявляются в начале программы, до их первого использования, как правило - сразу после оператора начала процедуры (**Sub** или **Function**). Основной оператор языка VBA для объявления переменных – оператор **Dim**.

Пусть, например, требуется, чтобы переменная **a** могла принимать любые числовые значения (как целые, так и дробные), переменные **b** и **c** – только целые (причем как положительные, так и отрицательные), переменная **d** – только положительные целые значения (причем небольшие), переменная **x** - строковые значения, переменная **y**– булевы значения (т.е. только **True** или **False**). Эти переменные можно объявить следующим образом:

Dim a As Single, b As Integer, c As Integer, d As Byte, x As String, y As Boolean

После этого, например, попытка присвоить переменной **d** любое значение, превышающее 255, вызовет сообщение об ошибке из-за несоответствия типов.

Если переменная не объявлена в операторе **Dim**, то ей назначается тип **Variant**. В этом случае тип переменной определяется автоматически в зависимости от значения, присваиваемого ей. Например, если переменная **z** не указана в операторе **Dim**, то в одной части программы ей может быть присвоено значение **z=”Минск”** (т.е. строковое значение), а в другой части программы - значение **z=5** (т.е. числовое). Использование переменных типа **Variant** приводит к увеличению затрат памяти и времени выполнения программы, а в некоторых случаях может приводить к ошибкам. Поэтому при разработке сложных программ, как правило, желательно объявлять с помощью оператора **Dim** все переменные.

Объявление в операторе **Dim** обязательно для массивов (см. раздел 1.4), а также для некоторых сложных типов данных, не рассматриваемых в данном пособии.

Рекомендуется объявлять в операторе **Dim** все переменные, вводимые с клавиатуры, т.е. с помощью функции **InputBox**. Если, например, переменная **z** не указана в операторе **Dim**, и для нее с клавиатуры вводится числовое значение (например, 7), то оно может быть распознано программой не как число, а как строка символов (в данном случае – как строка “7”). Это может привести к ошибкам, например, в операциях сравнения. Такой случай будет показан в примере 1.6.

Следует обратить внимание, что в операторе **Dim** тип должен указываться для *каждой* переменной *отдельно*. Так, в примере, приведенном выше, объявление **b, c As Integer** было бы неправильным: тип **Integer** в этом случае относится только к переменной **c**, а переменная **b** остается необъявленной, и ей назначается тип **Variant**.

#### 1.3.3. Объявление констант.

Константы обычно указываются в начале процедуры, как правило – сразу после оператора начала процедуры (**Sub** или **Function**) или операторов объявления переменных (**Dim**). Для объявления констант используется оператор **Const**.

Пусть, например, в процедуре требуется использовать цену некоторого изделия, равную 250, и вес изделия, равный 1128. Эти величины удобно задать в начале процедуры в виде констант:

Const cena = 250, ves = 1128

Изменять константу в программе нельзя. В рассмотренном примере попытка указать в программе оператор **cena = cena + 10** или **cena = 220** приведет к сообщению об ошибке.

#### 1.3.4. Область видимости переменных.

Сложные программы практически всегда состоят из нескольких процедур, а иногда и из нескольких модулей. В зависимости от области видимости, т.е. от того, в какой части программы может использоваться переменная, различают три вида переменных:

* переменные уровня процедуры, используемые только в пределах той процедуры, в которой они объявлены. Такие переменные объявляются оператором **Dim** в соответствующей процедуре;
* переменные уровня модуля, используемые во всех процедурах модуля. Такие переменные объявляются оператором **Dim** или **Private** *в начале модуля* перед процедурами, т.е. до первого оператора **Sub** или **Function**;
* переменные уровня проекта, используемые во всех модулях проекта (т.е. программы, состоящей из нескольких модулей). Такие переменные объявляются оператором **Public** в начале любого из модулей или в отдельном модуле.

**Пример 1.2.** В модуле введена следующая программа:

Private x As Integer

Sub primer1\_2a()

Dim y As Integer

x = 5

y = 10

Call primer1\_2b

End Sub

Sub primer1\_2b()

MsgBox ("x=" & x)

MsgBox ("y=" & y)

End Sub

Здесь **x** – переменная уровня модуля, используемая во всех процедурах модуля; **y** – переменная уровня процедуры, используемая только в той процедуре, где она объявлена, т.е. в процедуре **primer1\_2a**.

Пусть запускается процедура **primer1\_2a**. Для этого требуется расположить курсор в пределах текста этой процедуры и выбрать из меню команду **Run – Run Sub/UserForm**. Переменной **x** присваивается значение 5, переменной **y** – значение 10. Затем вызывается процедура **primer1\_2b**. Для этого используется оператор **Call** (подробнее он будет рассмотрен в подразделе 1.8). В этой процедуре на экран выводятся значения переменных **x** и **y**. Так как **x** – переменная уровня модуля, ее значение, присвоенное в любой из процедур (в данном случае – в процедуре **primer1\_2a**), известно и во всех остальных процедурах модуля (в данном случае – в процедуре **primer1\_2b**). Поэтому на экран будет выведено **x=5**. Переменная **y** – переменная уровня процедуры, поэтому ее значение, заданное в процедуре **primer1\_2a**, неизвестно в процедуре **primer1\_2b**. На экран будет выведено **y=**, так как переменная **y** в процедуре **primer1\_2b** не имеет никакого значения (или, точнее, представляет собой пустую строку).

#### 1.3. Оператор If.

Общий вид условного оператора **If** следующий:

**If** условие1 **Then**

действия, выполняемые, если условие1 верно

**Elseif** условие2 **Then**

действия, выполняемые, если условие1 неверно, а условие2 верно

**Else**

действия, выполняемые, если и условие1, и условие2 неверны

**End If**

Части **Elseif** и **Else** необязательны, поэтому оператор **If** может иметь следующий вид:

**If** условие **Then**

действия, выполняемые, если условие верно

**Else**

действия, выполняемые, если условие неверно

**End If**

или следующий вид:

**If** условие **Then**

действия, выполняемые, если условие верно

**End If**

Если оператор **If** записывается в одну строку, то слова **End If** после него не указываются.

**Пример 1.3.** Программа запрашивает число и умножает его на 2, если оно меньше 5, или на 3, если оно больше или равно 5.

Sub primer1\_3()

x = InputBox(“Введите число: ”)

If x < 5 Then x=x\*2 Else x=x\*3

MsgBox(“Результат равен ” & x)

End Sub

Здесь оператор **If** записан в одну строку, поэтому слова **End If** не требуются.

Приведем еще один пример программы с использованием оператора **If**.

**Пример** **1.4.** Программа для решения квадратного уравнения.

Sub primer1\_4()

Dim a As Single, b As Single, c As Single

a = InputBox ("Введите коэффициент a", "Ввод коэффициентов уравнения")

b = InputBox ("Введите коэффициент b", "Ввод коэффициентов уравнения ")

c = InputBox ("Введите коэффициент c", "Ввод коэффициентов уравнения ")

d = b ^ 2 - 4 \* a \* c

If d > 0 Then

x1 = -b - Sqr(d) / (2 \* a): x2 = -b + Sqr(d) / (2 \* a)

MsgBox ("x1=" & x1)

MsgBox ("x2=" & x2)

Elseif d = 0 Then

x = -b / (2 \* a)

MsgBox ("x=" & x)

Else

MsgBox ("Вещественных корней нет")

End If

End Sub

Здесь оператор **a = InputBox ("Введите коэффициент a", "Ввод коэффициентов уравнения")** означает, что запрашивается значение переменной, указанной перед знаком равенства (т.е. переменной **a**). При этом строка **"Введите коэффициент a: "** выводится на экран в качестве подсказки для пользователя. Строка **"Ввод коэффициентов уравнения"** выводится на экран в качестве заголовка окна, в котором запрашивается указанная переменная.

#### 1.4. Массивы.

Массив в языке VBA, как и в других языках – объект программы, состоящий из нескольких элементов. Массивы необходимо объявлять в начале программы с помощью оператора **Dim**. Например, следующее объявление

Dim a(1 To 6) As Integer, b(1 To 3, 1 To 10) As Single, c (1 To 6, 1 To 3) As String

означает, что переменная **a** – одномерный массив, который может содержать не более шести элементов (целых чисел, так как указан тип **Integer**). Переменная **b**– двумерный массив из трех строк и десяти столбцов; элементы этого массива – вещественные числа (тип **Single**). Переменная **c** – массив из шести строк и трех столбцов; его элементы – строки символов.

Использование переменных для указания размеров массивов в операторе **Dim** не допускается. Например, объявление **Dim a(1 To m) As Integer** недопустимо, даже если переменной **m** уже присвоено некоторое значение.

Во многих случаях удобно задавать размеры массива не в операторе **Dim** (т.е. не в начале программы), а позже, в ходе выполнения программы. Для этого используется оператор **ReDim**. Массив в этом случае называется динамическим.

Пусть, например, требуется использовать в программе массив из целых чисел, но количество этих чисел сначала неизвестно (например, его нужно запрашивать у пользователя). Объявление массива в этом случае может быть следующим:

Dim a() As Integer

m = InputBox ("Введите количество элементов массива ")

ReDim a(1 To m)

Как видно из этого примера, для указания размеров динамического массива можно использовать переменные. Если массив объявлен как динамический, то изменять его размеры, используя оператор **ReDim**, можно неоднократно.

При обращении в программе к отдельным элементам массива номера элементов указываются в круглых скобках. Если массив двумерный, то сначала указывается номер *строки*, затем – номер *столбца*. Примеры:

a(2) = 15

b(2,7) = 8.3

c(1,4) = “Минск”

Здесь второму элементу массива a присвоено значение 15. Элементу массива **b**, расположенному во второй строке и седьмом столбце, присвоено значение 8,3. Элементу массива **c**, расположенному в первой строке и четвертом столбце, присвоено значение “Минск”.

Примеры задач, решаемых с использованием массивов, будут приведены ниже.

#### 1.5. Цикл ДО. Оператор For.

Цикл ДО (т.е. цикл, повторяющийся заданное количество раз) реализуется в VBA оператором **For**, имеющим следующий вид:

**For** переменная\_цикла = начальное\_значение **To** конечное\_значение **Step** шаг

операторы, выполняемые в цикле

**Next** переменная\_цикла

Это означает, что заданная переменная цикла изменяется от начального до конечного значения с заданным шагом. Если слово **Step** и величина шага не указаны, то переменная изменяется с шагом, равным 1. Для каждого значения переменной выполняются операторы в цикле.

**Пример 1.5.** Программа запрашивает количество строк и столбцов (переменные **m** и **n**), затем запрашивает элементы массива из **m** строк и **n** столбцов (имя массива – **a**). Вычисляются суммы столбцов массива. Из них составляется новый массив – одномерный массив **asum**.

Sub primer1\_5a()

Dim a(1 To 5, 1 To 10) As Single, asum(1 To 10) As Single

m = InputBox("Введите количество строк: ")

n = InputBox("Введите количество столбцов: ")

For i = 1 To m

For j = 1 To n

a(i, j) = InputBox("a(" & i & "," & j & "): ")

Next j

Next i

For j = 1 To n

For i = 1 To m

asum(j) = asum(j) + a(i, j)

Next i

Next j

For j = 1 To n

MsgBox ("Сумма " & j & "-го столбца = " & asum(j))

Next j

End Sub

Здесь оператор **Dim** – объявление переменных (в данном случае – массивов). Запись **Dim a(1 To 5, 1 To 10) As Single** означает, что переменная **a** – двумерный массив, который может содержать не более пяти строк и не более десяти столбцов. Массив будет состоять из вещественных чисел, которые могут быть как целыми, так и дробными (тип данных **Single**).

Для обработки массивов используются циклы. Например, для суммирования каждого столбца массива **a** применяется вложенный цикл:

For j = 1 To n

For i = 1 To m

asum(j) = asum(j) + a(i, j)

Next i

Next j

Здесь переменная **j** (номер столбца) принимает сначала значение 1. При **j=1** выполняется вложенный цикл:

For i = 1 To m

asum(j) = asum(j) + a(i, j)

Next i

т.е. переменная **i** (номер строки) принимает значения от 1 до **m**, и выполняется суммирование элементов **a(1,1), a(2,1),**…,**a(m,1)**. В результате вычисляется величина **asum(1)** – первый элемент нового массива **asum**, сумма элементов первого столбца массива **a**.

Затем переменная **j** (номер столбца) принимает значение 2. Снова выполняется вложенный цикл, т.е. переменная **i** (номер строки) принимает значения от 1 до **m**, и выполняется суммирование элементов **a(1,2), a(2,2)**,…,**a(m,2)**. В результате вычисляется величина **asum(2)**. Аналогичные действия выполняются для значений **j**=3,…,**n**, т.е. для каждого столбца массива **a**.

Для вывода элементов массива **asum** на экран используется следующий цикл:

For j = 1 To n

MsgBox ("Сумма" & j & "-го столбца = " & asum(j))

Next j

Здесь переменная **j** изменяется от 1 до **n** с шагом 1, т.е. принимает значения 1, 2, 3, …, **n**. В каждом цикле (т.е. **n** раз) выполняется оператор **MsgBox ("Сумма" & j & "-го столбца = " & asum(j))**, т.е. на экран выводится **j**-й элемент массива **asum**.

Примечание – В процедуре **primer1\_5a** массив **a** хранится в памяти компьютера как массив из пяти строк и десяти столбцов, а массив **asum** – как массив из десяти элементов (даже если фактические размеры массивов, т.е. значения переменных **m** и **n**, меньше). В то же время ввести значения переменных **m** и **n**, превышающие 5 и 10, нельзя: при выполнении цикла произойдет выход за объявленные границы массивов, и программа будет прервана с выдачей сообщения об ошибке. Разрешить эти проблемы можно, используя динамические массивы. В этом случае начало программы будет иметь следующий вид:

Sub primer1\_5b()

Dim a() As Single, asum() As Single

m = InputBox("Введите количество строк")

n = InputBox("Введите количество столбцов")

ReDim a (1 To m, 1 To n), asum(1 To n)

Теперь размеры массивов **a** и **asum** могут быть любыми, в зависимости от введенных значений переменных **m** и **n**.

**Пример 1.6.** Программа запрашивает элементы массива из пяти строк и трех столбцов (массив **a**), а также некоторое число (переменная **x**). Затем программа подсчитывает в каждой строке массива **a** количество элементов, равных переменной **x**. Если строка полностью состоит из чисел **x**, то номер строки выводится на экран.

Sub primer1\_6a()

Dim a(1 To 5, 1 To 3) As Single, x As Single

m = 5 : n = 3

MsgBox(“Вводите массив”)

For i = 1 To m

For j = 1 To n

a(i, j) = InputBox("a(" & i & "," & j & "): ")

Next j

Next i

x = InputBox("Введите число x: ")

For i = 1 To m

kol = 0

For j = 1 To n

If a(i,j) = x Then kol = kol + 1

Next j

If kol = n Then MsgBox (i & "-я строка состоит из чисел " & x)

Next i

End Sub

Здесь для подсчета количества элементов строки, равных числу **x**, используется переменная **kol**. Цикл **For i = 1 To m** используется для перебора строк, цикл **For j = 1 To n** – для перебора элементов строки. В начале перебора очередной (**i**-й) строки выполняется оператор **kol=0**, т.е. переменная **kol** обнуляется. Затем каждый элемент **i**-й строки проверяется на равенство переменной **x**, и если равенство выполняется, то переменная **kol** увеличивается на единицу (**If a(i,j) = x Then kol = kol + 1**). Таким образом, после завершения цикла
**For j = 1 To n** (т.е. по окончании перебора элементов **i**-й строки) переменная **kol** оказывается равной количеству элементов строки, значение которых совпало с переменной **x**. В операторе **If kol = n Then MsgBox (i & "-я строка состоит из чисел " & x)** проверяется значение переменной **kol**. Если оно равно количеству столбцов массива (т.е. количеству элементов в строке), значит, вся строка состояла из переменных, равных числу **x**. В этом случае номер строки выводится на экран. Затем выполняется возврат к началу цикла **For i = 1 To m**, т.е. переменная **i** увеличивается на единицу, и проверяется очередная строка.

Следует обратить внимание, что переменная **x**, вводимая с клавиатуры, объявлена в операторе **Dim** с типом **Single**, т.е. с тем же типом, что и элементы массива. В данном случае такое объявление обязательно. Если не сделать этого, то при сравнении элемента массива **a(i,j)** с переменной **x** (**If a(i,j) = x Then …**) эти величины всегда будут распознаваться как разные, так как элементы массива имеют тип **Single** (т.е. представляют собой десятичные числа), а переменная **x** будет рассматриваться как строка символов, а не как число (например, значение 7 будет распознано как строка “7”).

Рассмотрим еще один способ решения данной задачи. Чтобы определить, состоит ли вся строка массива из чисел **x**, воспользуемся логической переменной **vse**.

Sub primer1\_6b()

... См. программу primer1\_5a …

For i = 1 To m

vse = True

For j = 1 To n

If a(i,j) <> x Then vse = False

Next j

If vse = True Then MsgBox (i & "-я строка состоит из чисел " & x)

Next i

End Sub

Здесь в начале перебора каждой строки переменной **vse** присваивается значение **True**. Затем каждый элемент строки проверяется на равенство переменной **x**, и если равенство *не* выполняется, то переменная **vse** получает значение **False**. Таким образом, по окончании перебора элементов строки переменная **vse** будет иметь значение **False**, если хотя бы один элемент строки будет иметь значение, отличное от **x** (и останется равной **True**, если все элементы строки будут равны переменной **x**). В операторе **If vse = True Then …** проверяется значение переменной **vse**. Если эта переменная равна **True**, значит, вся строка состояла из переменных, равных числу **x**. В этом случае номер строки выводится на экран.

**Пример 1.7.** В программу вводится двумерный массив. Программа определяет в каждом столбце массива максимальное число и выводит его на экран.

Sub primer1\_7()

Dim a(1 To 3, 1 To 5) As Single

m = 3 : n = 5

MsgBox(“Вводите массив”)

For i = 1 To m

For j = 1 To n

a(i, j) = InputBox("a(" & i & "," & j & ")")

Next j

Next i

For j = 1 To n

maximum = a(1,j)

For i = 1 To m

If a(i,j) > maximum Then maximum = a(i,j)

Next i

MsgBox ("В " & j & "-м столбце максимальный элемент равен " & maximum)

Next j

End Sub

Здесь цикл **For j = 1 To n** используется для перебора столбцов, цикл
**For i = 1 To m** – для перебора элементов столбца. Переменная **maximum** используется для запоминания максимального элемента столбца. Сначала она принимается равной первому элементу столбца: **maximum = a(1,j)**. Если в ходе перебора столбца обнаруживается элемент, превышающий текущее значение переменной **maximum**, то он присваивается этой переменной. В результате по окончании перебора столбца переменная **maximum** будет равна его максимальному элементу.

**Пример 1.8.** В программу вводится двумерный массив. Программа определяет в каждом столбце массива максимальное число и меняет его местами с первым элементом данного столбца. Измененный массив выводится на экран.

Sub primer1\_8()

Dim a(1 To 3, 1 To 5) As Single

m = 3 : n = 5

MsgBox(“Вводите массив”)

For i = 1 To m

For j = 1 To n

a(i, j) = InputBox("a(" & i & "," & j & ")")

Next j

Next i

For j = 1 To n

maximum = a(1,j)

nomer = 1

For i = 1 To m

If a(i,j) > maximum Then

maximum = a(i,j)

nomer = i

End If

Next i

x = a(1,j)

a(1,j) = a(nomer,j)

a(nomer,j) = x

Next j

For i = 1 To m

For j = 1 To n

MsgBox("a(" & i & "," & j & ") = " & a(i,j))

Next j

Next i

End Sub

Поиск максимального элемента столбца выполняется аналогично предыдущему примеру. В переменной **maximum** запоминается максимальный элемент столбца, а в переменной **nomer** – номер этого элемента (или, другими словами, номер строки, в которой находится максимальный элемент данного столбца). В следующей группе операторов первый и максимальный элемент **j**-го столбца меняются местами:

x = a(1,j)

a(1,j) = a(nomer,j)

a(nomer,j) = x

Здесь **x** – вспомогательная переменная, используемая для промежуточного хранения первого элемента столбца.

**Пример 1.9.** В программу вводится двумерный массив. По каждой строке вычисляется среднее значение. Составляется массив из средних значений, превышающих некоторую заданную величину (эта величина вводится с клавиатуры и обозначается как переменная **predel**). Этот массив выводится на экран.

Sub primer1\_9()

Dim a() As Single, sred() As Single, predel As Single

m = InputBox("Введите количество строк ")

n = InputBox("Введите количество столбцов ")

redim a(1 To m, 1 To n), sred (1 To m)

For i = 1 To m

For j = 1 To n

a(i, j) = InputBox("a(" & i & "," & j & ")")

Next j

Next i

predel = InputBox(“Введите минимально допустимое среднее”)

k = 0

For i = 1 To m

sum = 0

For j = 1 To n

sum = sum + a(i,j)

Next j

sred\_stroki = sum/n

If sred\_stroki > predel Then

k=k+1 ‘номер нового элемента массива

sred(k) = sred\_stroki

End If

Next i

For i = 1 To k

MsgBox (sred(i))

Next i

End Sub

Здесь **sred** – массив из средних значений, превышающих заданную величину **predel**. Каждый раз, когда среднее значение строки превышает переменную **predel** (т.е. выполняется условие **sred\_stroki > predel**), вычисляется новое значение переменной **k** - номер очередного элемента массива **sred**. Для этого переменная **k** увеличивается на единицу (**k=k+1**). Затем создается новый (**k**-й) элемент массива **sred**: **sred(k) = sred\_stroki**.

#### 1.6. Цикл ПОКА. Операторы While и Until.

Кроме цикла ДО, который повторяется заданное количество раз, применяется также цикл ПОКА, который повторяется до тех пор, пока верно некоторое заданное условие (или, наоборот, пока некоторое условие не станет верным). В VBA имеется четыре основных способа реализации цикла ПОКА:

* оператор **While** с проверкой в начале цикла;

**Do While** условие

операторы, выполняемые в цикле

**Loop**

Цикл повторяется, пока условие верно (и прекращается, когда оно становится неверным). Условие проверяется перед началом цикла. Если оно сразу неверно, то цикл не выполняется ни разу;

* оператор **While** с проверкой в конце цикла:

**Do**

операторы, выполняемые в цикле

**Loop While** условие

Цикл повторяется, пока условие верно. Условие проверяется в конце цикла. Поэтому цикл выполняется, как минимум, один раз;

* оператор **Until** с проверкой в начале цикла:

**Do Until** условие

операторы, выполняемые в цикле

**Loop**

Цикл повторяется, пока условие *неверно* (и прекращается, когда оно становится верным). Условие проверяется перед началом цикла;

* оператор **Until** с проверкой в конце цикла:

**Do**

операторы, выполняемые в цикле

**Loop Until** условие

Цикл повторяется, пока условие *неверно* (и прекращается, когда оно становится верным). Условие проверяется в конце цикла, поэтому цикл выполняется, как минимум, один раз.

**Пример** **1.10.** В программе вводится одномерный массив из восьми чисел, и выполняется суммирование его элементов (начиная с первого), пока сумма не превысит 20.

Sub primer1\_10a()

Dim a(1 To 8) As Integer

MsgBox(“Вводите массив”)

For i = 1 To 8

a(i) = InputBox("a(" & i & ")")

Next i

sum = 0 : i = 0

Do While sum <= 20

i = i + 1

sum = sum + a(i)

Loop

MsgBox ("Сумма достигнута на " & i & "-м элементе и равна" & sum)

End Sub

Здесь переменная **sum** сначала равна нулю, поэтому условие **sum<=20** верно. Выполняются операторы цикла:

i = i + 1

sum = sum + a(i)

Таким образом, переменная **sum** становится равной первому элементу массива **a**. Если он превышает 20, то цикл завершается, так как условие **sum<=20** оказывается неверным. Выполняется оператор, следующий за циклом, т.е. выводится сообщение о найденной сумме. Если же элемент **a(1)** не превышает 20, то условие **sum<=20** остается верным, и цикл повторяется снова: переменная *i* принимает значение 2, и к переменной **sum** прибавляется величина **a(2)**. Цикл повторяется, пока переменная **sum** не превысит 20.

Эту же задачу можно решить с помощью оператора **Until**. Для этого следует заменить строку **Do While sum <= 20** на следующую: **Do Until sum > 20**.

Следует обратить внимание, что программа, приведенная в примере 1.8, не вполне правильная: в ней не предусмотрено, что сумма элементов массива может оказаться меньше 20. В этом случае программа завершится с выдачей сообщения об ошибке, когда переменная **i** превысит объявленное количество элементов массива (в данном примере оно равно 8). Чтобы исправить эту ошибку, можно изменить программу, как показано ниже.

Sub primer1\_10b()

Dim a(1 To 8) As Integer

MsgBox ("Вводите массив")

For i = 1 To 8

a(i) = InputBox("a(" & i & ")")

Next i

sum = 0: i = 0

Do While (sum <= 20) And (i < 8)

i = i + 1

sum = sum + a(i)

Loop

If sum > 20 Then

MsgBox ("Сумма достигнута на " & i & "- м элементе и равна " & Sum)

Else

MsgBox ("Сумма не достигнута")

End If

End Sub

Здесь цикл **While** выполняется до тех пор, пока верны два условия: переменная **sum** не превышает 20, и переменная **i** меньше 8. Если хотя бы одно из этих условий нарушается, то цикл прекращается. Поэтому, если в ходе выполнения цикла переменная **sum** достигает значения, превышающего 20, то цикл прекращается, так как нарушается условие **sum <= 20**. Если же оказывается, что сумма всех элементов массива меньше 20, то цикл прекращается из-за нарушения условия **i < 8** (так как переменная **i** увеличивается на единицу в каждом цикле и достигает значения 8, когда вычислена сумма всех элементов массива). После прекращения цикла в операторе **If** проверяется достигнутое значение переменной **sum**, и выводится соответствующее сообщение.

**Пример 1.11.** Вычислить сумму ряда  с точностью до 0,0001.

Sub primer1\_11a()

n=0 : element = 1 : sum = 0

Do While Abs (element) >= 0.0001

sum = sum + element

n=n+1

factorial = 1

For i=1 To n

factorial = factorial \* i

Next i

element = ((-1)^n)/(factorial\*(2\*n+1))

Loop

MsgBox(“Сумма ” & sum & “достигнута при n = ” & n-1)

End Sub

Здесь **Abs** – встроенная (стандартная) функция языка VBA, вычисляющая модуль. Для вычисления факториала использован цикл **For**, рассмотренный ранее. Вычитание при выводе результата (**n-1**) требуется из-за того, что в цикле переменная **n** увеличивается, и только после этого проверяется условие окончания цикла (поэтому на момент окончания цикла переменная **n** будет иметь значение, на единицу превышающее количество элементов вычисленной суммы).

Эту же программу можно реализовать и другими способами, например, используя цикл ПОКА в форме **Until**:

Sub primer1\_11b()

n=0 : element = 1 : sum = 0

Do

sum = sum + element

n=n+1

factorial = 1

For i=1 To n

factorial = factorial \* i

Next i

element = ((-1)^n)/(factorial\*(2\*n+1))

Loop Until Abs (element) < 0.0001

MsgBox(“Сумма ” & sum & “достигнута при n = ” & n-1)

End Sub

Здесь цикл повторяется, пока условие **Abs (element) < 0.0001** неверно (и завершается, когда это условие становится верным).

#### 1.7. Безусловный переход. Оператор GOTO.

Для безусловного перехода на желаемый оператор в программе на VBA используется оператор **Goto**, за которым указывается метка оператора, к которому требуется перейти. Метка указывается перед оператором. Между меткой и оператором указывается двоеточие.

**Пример 1.12.** Программа запрашивает у пользователя число и вычисляет его квадратный корень. Работа программы продолжается, пока не будет введено отрицательное число.

Sub primer1\_12()

nachalo: x = InputBox("Введите число")

If x < 0 Then Goto konec

koren = Sqr(x)

MsgBox ("Корень равен " & koren)

Goto nachalo

konec: End Sub

Здесь **nachalo** и **konec** – метки.

#### 1.8. Подпрограммы.

Во многих случаях удобно выделить часть программы в отдельную процедуру (подпрограмму) и вызывать ее для выполнения из другой процедуры. Подпрограмма может иметь входные и выходные параметры. Входные параметры – это величины, передаваемые в подпрограмму из другой (вызывающей) процедуры и являющиеся для вызываемой подпрограммы «исходными данными». Этими величинами могут быть как переменные, так и константы (например, конкретные числа). Выходные параметры – это переменные, являющиеся результатами выполнения вызываемой подпрограммы.

**Пример** **1.13.** Требуется определять в заданной строке двумерного массива максимальный элемент этой строки, а также его номер. Поиск максимального элемента в строке требуется реализовать как подпрограмму.

Sub primer1\_13()

Dim a() As Single

m = InputBox("Введите количество строк массива")

n = InputBox("Введите количество столбцов массива")

ReDim a (1 To m, 1 To n), asum(1 To n)

For i = 1 To m

For j = 1 To n

a(i, j) = InputBox("a(" & i & "," & j & ")")

Next j

Next i

nomer\_stroki = InputBox(“Укажите, в какой строке требуется найти максимум”)

Call poisk\_max(a, n, nomer\_stroki, max\_elem, nomer\_max\_elem)

MsgBox(“Максимальный элемент “ & i & “-й строки равен “ & max\_elem)

MsgBox(“Он находится в “ & nomer\_max\_elem & “-м столбце“)

End Sub

Sub poisk\_max(massiv, kol\_stolb, i, max, nom\_max)

max=massiv(i,1) : nom\_max=1

For j=1 To kol\_stolb

If massiv(i,j)>max Then

max = massiv(i,j)

nom\_max = j

Next j

End Sub

Здесь подпрограмма **primer1\_13** является вызывающей (или основной) подпрограммой, а **poisk\_max** – вызываемой подпрограммой. Таким образом, данная программа (в целом) состоит из двух подпрограмм. При ее запуске курсор должен находиться в пределах текста основной подпрограммы.

В начале основной подпрограммы выполняется ввод массива (с именем **a**). Вводится также переменная **nomer\_stroki** – номер строки, в которой требуется найти максимальный элемент. Все действия, выполняемые при этом, рассмотрены ранее.

Оператор **Call** – вызов подпрограммы. В данном примере он имеет следующий вид:

Call poisk\_max(a, n, nomer\_stroki, max\_elem, nomer\_max\_elem)

Здесь **poisk\_max** – имя вызываемой подпрограммы. Переменные **a**, **n** и **nomer\_stroki** – входные параметры, так как при вызове подпрограммы их значения уже заданы. Переменные **max\_elem** и **nomer\_max\_elem** – выходные параметры, так как они должны быть определены в результате выполнения подпрограммы.

Следующая строка представляет собой заголовок подпрограммы:

Sub poisk\_max(massiv, kol\_stolb, i, max, nom\_max)

Конечно, количество параметров в вызове подпрограммы (т.е. в операторе **Call**) и в ее заголовке должно совпадать, и порядок параметров в этих операторах должен соответствовать друг другу.

В данном примере при вызове подпрограммы **poisk\_max** переменная **massiv** получает значение переменной **a** (т.е. значение массива, введенного в основной программе), переменная **kol\_stolb** – значение **n**, переменная **i** – значение **nomer\_stroki**. Переменные **max** и **nom\_max** получают значения переменных **max\_elem** и **nomer\_max\_elem**; так как этим переменным в основной программе не было присвоено никакого значения, они равны нулю.

Затем выполняется подпрограмма **poisk\_max.** В результате ее выполнения определяются значения переменных **max** и **nom\_max**. Эти же значения получают соответствующие переменные основной программы, в данном случае – переменные **max\_elem** и **nomer\_max\_elem**.

По окончании выполнения подпрограммы происходит возврат в основную подпрограмму. Выполняется оператор, следующий за оператором **Call**. В данном случае это оператор **MsgBox**, выводящий результат на экран.

В качестве входных параметров могут использоваться не только переменные, но и константы. Пусть, например, в рассматриваемой задаче не требуется запрашивать у пользователя номер строки, в которой необходимо найти максимум; вместо этого задано, что необходимо найти максимум в третьей строке. Тогда вызов подпрограммы мог бы иметь следующий вид:

Call poisk\_max(a, n, 3, max\_elem, nomer\_max\_elem)

Никаких изменений в самой подпрограмме **poisk\_max** при этом не требуется. При вызове подпрограммы переменная **i** получит значение 3. Все остальные действия выполняются точно так же, как описано выше.

#### 1.9. Функции.

Функция, как и подпрограмма – это отдельная процедура, вызываемая из другой процедуры. Отличие функции от подпрограммы состоит в том, что в результате выполнения функции определяется значение одной переменной, в то время как подпрограмма может определять значения нескольких переменных.

Имя переменной, определяемой в результате выполнения функции, должно совпадать с именем самой функции.

Функция, как и подпрограмма, имеет входные параметры, т.е. величины, передаваемые в функцию из вызывающей процедуры. Этими величинами могут быть как переменные, так и константы.

Следует обратить внимание, что способы вызова подпрограммы и функции существенно различаются. Подпрограмма вызывается с помощью специального оператора **Call**. Функция обычно вызывается в правой части оператора присваивания.

**Пример** **1.13.** Требуется разработать функцию для вычисления расстояния между двумя точками в пространстве любой размерности. Точки задаются координатами.

Sub primer1\_13()

Dim a() As Single, b() As Single

n = InputBox("Введите размерность пространства")

ReDim a(1 To n), b(1 To n)

MsgBox ("Вводите координаты первой точки")

For i = 1 To n

a(i) = InputBox("Координата " & i & ": ")

Next i

MsgBox ("Вводите координаты второй точки")

For i = 1 To n

b(i) = InputBox("Координата " & i & ": ")

Next i

d = rasst(a, b, n)

MsgBox ("Расстояние равно " & d)

End Sub

Function rasst(x, y, razmernost)

rasst = 0

For i = 1 To razmernost

rasst = rasst + (x(i) - y(i)) ^ 2

Next i

rasst = Sqr(rasst)

End Function

Здесь **rasst** – функция. Она вызывается в операторе **d = rasst(a, b, n)**. Таким образом, переменные **a** и **b** (массивы с координатами точек), а также переменная **n** (размерность пространства, т.е. количество координат у каждой точки) являются входными параметрами функции **rasst**. В самой функции **rasst** значения этих переменных присваиваются переменным **x**, **y** и **razmernost** соответственно. Функция вычисляет расстояние между точками. Результат присваивается переменной **rasst**. Следует обратить внимание, что имя этой переменной обязательно должно совпадать с именем самой функции (в данном случае – **rasst**). По окончании выполнения функции **rasst** выполняется возврат в основную программу: выполняется оператор, где была вызвана функция, т.е. оператор **d = rasst(a, b, n)**. Таким образом, значение функции **rasst** присваивается переменной **d**. Затем эта переменная выводится на экран оператором **MsgBox**.

#### 1.10. Область видимости процедур.

В начале заголовка процедуры, т.е. перед словом **Sub** или **Function**, может указываться слово **Private** или **Public**, задающее область видимости процедуры:

* **Private**: процедура может вызываться только из процедур того модуля, в котором она находится;
* **Public**: процедура может вызываться из процедур как того модуля, в котором она находится, так и других модулей.

Если область видимости не указана, то по умолчанию предполагается видимость процедуры во всех модулях (т.е. область видимости, соответствующая слову **Public**). Если программа состоит только из одного модуля, то область видимости процедур этой программы никак не влияет на ее работу.

#### 1.11. Варианты заданий.

**Вариант 1**. В программе вводится двумерный массив. Вычисляется среднее по каждой строке, а также среднее по всему массиву. Составляются два новых массива. Первый из них включает те строки исходного массива, для которых среднее значение превышает среднее по массиву, второй – все остальные строки исходного массива. Вычисление среднего по строке массива должно быть реализовано в виде отдельной процедуры.

**Вариант 2**. В программе вводится двумерный массив. Из него должны удаляться все строки и столбцы, состоящие только из нулей. Проверка строк и столбцов (с целью выяснить, состоит ли данная строка или столбец только из нулей) должна выполняться с помощью отдельных процедур.

**Вариант 3**. В программе вводятся два массива: двумерный (обозначим его как **a**) и одномерный (**b**), причем количество элементов в одномерном массиве должно быть равно количеству строк в двумерном массиве. Для каждого элемента массива **b** определяется ближайший к нему элемент в соответствующей строке массива **a** (т.е. элемент, разность с которым минимальна). Поиск ближайшего элемента должен быть реализован в виде отдельной процедуры. Из номеров этих ближайших элементов составляется одномерный массив.

**Вариант 4**. В программе вводится двумерный массив. Из него должен составляться новый массив, включающий только те строки исходного массива, которые состоят из одинаковых чисел. Проверка строк (с целью выяснить, состоит ли данная строка из одинаковых чисел) должна выполняться с помощью отдельной процедуры.

**Вариант 5**. В программе вводится двумерный массив. В каждой строке определяется максимальное и минимальное число, и вычисляется их разность. Строка, где эта разность максимальна, должна меняться местами со строкой, где разность минимальна. Определение минимального и максимального значения в строке должно быть реализовано в виде отдельной процедуры.

**Вариант 6**. В программе вводится двумерный массив. Из него составляются два новых массива: в первый из них включаются те строки исходного массива, где нет ни одного отрицательного числа, во второй – те строки, где есть хотя бы одно отрицательное число. Проверка строки на наличие отрицательного числа должна быть реализована в виде отдельной процедуры.

**Вариант 7**. В программе вводится двумерный массив. В каждой строке вычисляется среднее значение. Составляется новый массив, содержащий столько же строк, что и исходный массив. Строки нового массива составляются из элементов соответствующей строки исходного массива, превосходящих среднее значение по своей строке. Так как количество элементов, отбираемых из каждой строки, оказывается при этом разным, недостающие элементы в конце строк нового массива остаются равными нулю. Вычисление среднего по строке массива должно быть реализовано в виде отдельной процедуры.

**Вариант 8**. В программе вычисляется значение функции *sh x* (гиперболический синус) на основе ее представления в виде ряда Тейлора: . Значение переменной *x*, а также требуемая точность вводятся с клавиатуры. Вычисление ряда Тейлора должно быть реализовано в виде функции (вызывающая процедура должна содержать только ввод исходных данных, вызов функции и вывод результата).

**Вариант 9**. В программе вводится квадратная матрица (обозначим ее как **a**) и некоторое число ***s***. Элемент матрицы **a(1,1)** сравнивается с числом **s**. Если **a(1,1) > s**, то программа завершает работу. В противном случае вычисляется сумма элементов квадратной матрицы 2x2 из левого верхнего угла матрицы **a**. Если эта сумма снова оказывается меньше **s**, то вычисляется сумма матрицы 3x3, и т.д., пока не будет достигнута сумма, превышающая **s**. Вычисление суммы квадратной матрицы, выделенной из исходной матрицы **a**, должно быть реализовано в виде функции. Программа должна выводить на экран размерность квадратной матрицы, сумма которой превысила **s**. Если число **s** не превышено при суммировании всей матрицы **a**, то должно выводиться сообщение об ошибке.

**Вариант 10.** В программе вводится одномерный массив **a**, состоящий из целых чисел, не превышающих 100. Вводятся также два числа: **g** (вводится значение 100) и **k** (некоторое произвольное целое число). В программе подсчитывается количество элементов массива **a**, превышающих величину **g**. Если это количество составляет не меньше **k**, то программа завершает работу; при этом из чисел, превышающих **g**, составляется новый массив. В противном случае величина **g** уменьшается на 0,5, и проверка повторяется. Подсчет количества элементов массива, превышающих заданную границу **g**, должен быть реализован в виде функции.

**Вариант 11**. В программе вводится одномерный массив **a**, и некоторое число **s**. Вычисляется среднее значение элементов массива (если в массиве есть элементы, равные нулю, то при вычислении среднего они не учитываются). Если среднее значение превышает **s**, то программа завершает работу; при этом из ненулевых элементов массива **a** составляется новый массив. В противном случае в массиве определяется минимальное число, оно заменяется на ноль, снова вычисляется среднее, и проверка повторяется. Если заданное среднее значение не достигается после исключения всех элементов массива, то выводится соответствующее сообщение.

**Вариант 12**. В программе вычисляется интеграл от некоторой функции *y* = *f*(*x*) на отрезке [*a*, *b*] на основе метода левых прямоугольников. Для этого отрезок [*a*, *b*] разбивается на *k* участков одинаковой длины *d*=(*b*-*a*)/*k*, т.е. определяются точки *x*0 = *a*, *x*1 = *a*+*d*, *x*2=*a*+2*d*, …, *xk*-1 = *a*+(*k*-1)*d*. Вычисляются значения функции *y* в этих точках: *y*0 = *f*(*x*0), *y*1 = *f*(*x*1), …, *yk*-1 = *f*(*xk*-1). Оценка интеграла вычисляется как (*y*0+*y*1+…+*yk*-1)·*d*. Затем *k* увеличивается на единицу, и вычисление повторяется. Процесс завершается, когда разность оценок интеграла при двух последовательных значениях *k* оказывается меньше заданной точности. Требуется разработать программу для вычисления интеграла от функции  этим методом. В программе должны запрашиваться границы отрезка (*a* и *b*), а также требуемая точность. Начальное значение *k* задается в программе (например, *k*=2). Непосредственно алгоритм вычисления интеграла должен быть реализован в виде функции.

**Вариант 13**. В программе определяется решение некоторого уравнения *f(x)*= 0 на отрезке [*a*, *b*] методом дихотомии. Для применения этого метода необходимо выбрать точки *a* и *b* таким образом, чтобы величины *f(a)* и *f(b)* имели разные знаки. Определяется середина этого отрезка *c* = (*b*+*a*)/2, и в этой точке вычисляется значение *f*(*c*). Если это значение отличается от нуля на величину, не превышающую заданной точности, то задача решена (*c* – решение). В противном случае отрезок [*a*, *b*] разбивается пополам. Выбирается тот из двух полученных отрезков, на концах которого исследуемая функция *f*(*x*) принимает разные знаки, и для этого отрезка процесс повторяется. Требуется разработать программу для решения уравнения *sin x / x* = 0 этим методом. В программе должны запрашиваться границы отрезка (*a* и *b*), а также требуемая точность. Непосредственно алгоритм решения уравнения должен быть реализован в виде функции.

**Вариант 14** – В программе вводятся два одномерных массива (обозначим их как **a** и **b**), причем известно, что сумма массива **a** меньше суммы массива **b** (если это не так, то должно выводиться сообщение об ошибке). Составляется новый одномерный массив **c**. Сначала в него включаются все элементы массива **a**; затем к нему добавляются элементы массива **b**, начиная с **b(1)**, до тех пор, пока сумма массива **a** не превысит сумму оставшихся элементов массива **b**. Суммирование элементов массива должно быть реализовано в виде функции. На экран выводится полученный массив **c**.

# Контрольные вопросы.

1. Что представляет собой программа VBA?
2. Что включает в себя структура среды разработки IDE (редактор VBA)?
3. Какие области включает в себя каждый модуль программы VBA?
4. Дайте определение линейной программы VBA.
5. Поясните, как используются комментарии в программах.
6. Изложите структурупрограммы-процедуры на VBA.
7. Изложите структурупрограммы–функции на VBA.

### Оформление отчета

Отчет по лабораторной работе должен содержать:

#### Титульный лист.

1. Цели работы.
2. Краткие теоретические сведения в виде ответов на контрольные вопросы.
3. Результаты лабораторной работы (графики, таблицы, значения параметров измеренных в процессе работы).
4. Необходимые расчеты (если они оговариваются в практической части лабораторной работы).
5. Выводы по работе (что получено в процессе работы, почему получены данные результаты и т. д.).
6. Отчет для защиты представляется преподавателю в отпечатанном или электронном виде.

Литература

1.  Гарнаев А. Ю. Самоучитель VBA. / А. Ю. Гарнаев. – СПб.: БХВ-Петербург, 2003. – 512с.

2.  Демидова Л. А.  Программирование в среде Visual Basic for Applications: Практикум. / Демидова Л.А. , Пылькин А.Н.  – М.: Горячая линия – Телеком, 2004. – 176 с.

3.Воробьева Ф.И., Воробьев Е.С. Приемы программирования в среде Visual Basic for Application MS Office. – Казань: Изд-во Казан. гос. технол. ун-та, 2010. – 104 с.

4. Любин А.Н. Создание пользовательских форм в Visual Basic for Applications. – М.: Изд-во Московского государственного технического университета «МАМИ», 2012. – 60 с.