

МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
имени М.В. ЛОМОНОСОВА

Факультет вычислительной математики и кибернетики

И.Г. Головин, Н.В. Баева

Языки управления приложениями

Учебно-методическое пособие

Москва – 2015

УДК 004.43(075.8)
ББК 32.973.26-018.1я73

*Печатается по решению Редакционно-издательского совета
факультета вычислительной математики и кибернетики
МГУ имени М.В. Ломоносова*

Рецензенты:

*А.Н. Терехин - к.ф.-м.н., доцент;
М.В. Орлов - к.ф.-м.н., доцент*

Головин И.Г., Баева Н.В.

Языки управления приложениями: Учебно-методическое пособие. – М.: Издательский отдел факультета ВМиК МГУ имени М.В. Ломоносова (лицензия ИД № 05899 от 24.09.2001); МАКС Пресс, 2015. – 92 с. [+4 стр.вкл.]

ISBN 978-5-89407-543-3

ISBN

В пособии рассмотрены основные понятия и концепции языков управления приложениями, а также основные приемы программирования на таких языках. В качестве основного примера, демонстрирующего эти понятия, выбран язык VBA (Visual Basic for Applications). Пособие предназначено для студентов-бакалавров 2 курса факультета ВМК МГУ. Пособие содержит материалы по курсу лекций «Языки управления приложениями», читаемому в 4 семестре, и по семинарским занятиям, соответствующим этому курсу.

Golovin I.G., Baeva N.V.

Embeddable programming languages: the guidelines. MAKS Press, Moscow, 2015. - 92 p. [+4 p.]

The textbook describes main notions and concepts of embeddable programming languages and basic programming techniques for these languages. VBA (Visual Basic for Applications) is chosen as the main example for illustration of these concepts and techniques. The textbook is targeted for the second-year student-bachelors of the MSU CMC faculty. It contains materials on the lecture course “Embeddable programming languages” read at 4th semester and on the practical lessons supporting the course.

УДК 004.43(075.8)
ББК 32.973.26-018.1я73

Учебно-методическое издание

ГОЛОВИН Игорь Геннадьевич
БАЕВА Наталия Валерьевна
ЯЗЫКИ УПРАВЛЕНИЯ ПРИЛОЖЕНИЯМИ
Учебно-методическое пособие

Издательский отдел

Факультета вычислительной математики и кибернетики МГУ имени М.В. Ломоносова
Лицензия ИД № 05899 от 24.09.2001

ISBN 978-5-89407-543-3
ISBN 978-5-317-05107-5

©Издательство "МАКС Пресс", 2015
©Факультет ВМК МГУ имени М.В.Ломоносова, 2015
©Головин И.Г., Баева Н.В., 2015

Содержание

1	Определение и основные свойства языков управления приложениями.....	4
1.1	Общее определение и свойства языков программирования	4
1.2	Определение, примеры и общие свойства языков управления приложениями	7
2	Основы языка VBA.....	10
2.1	Лексика и запись программ.....	10
2.2	Переменные, константы, основные типы данных	11
2.3	Массивы и структуры	22
2.4	Основные операторы	25
2.5	Процедуры и функции	32
2.6	Модули и структура проекта	34
2.7	Обработка ошибочных ситуаций.....	36
3	Понятие об объектной модели приложения.....	40
3.1	Объекты, их методы и свойства.....	40
3.2	Объявление, создание и уничтожение объектов	41
3.3	События и событийно-ориентированная архитектура	46
3.4	Коллекции	48
4	Основы объектной модели Excel.....	52
5	Задания практикума.....	59
5.1	Варианты первого практического задания (Microsoft Word) ..	67
5.2	Варианты второго практического задания (Microsoft Excel) ..	70
	Литература.....	92

1 Определение и основные свойства языков управления приложениями

1.1 Общее определение и свойства языков программирования

Одно из наиболее общих определений языков программирования разобрано в [3]. Оно гласит, **язык программирования – это инструмент для планирования поведения некоторого устройства-исполнителя.** В более узком смысле управляемое устройство – это компьютер (вычислительная система). Планы, управляющие поведением компьютеров, называются компьютерными программами. В данном пособии мы ограничимся рассмотрением специализированных языков программирования, в которых управляемым объектом служит некоторая прикладная программная система, например, MS Office, веб-браузер и другие. Такие языки будем называть языками управления приложениями (ЯУП). Прежде, чем разбирать свойства собственно ЯУП, поговорим об общих свойствах языков программирования, которые применимы и к подклассу ЯУП. Такими наиболее общими свойствами являются **данные, операции и связывание.**

Любая компьютерная программа обрабатывает некоторые данные, причем процесс обработки включает в себя выполнение некоторых операций (вычислений) над этими данными. Конкретная номенклатура данных и операций определяется каждым языком и может сильно варьироваться. Например, язык JavaScript содержит единственный числовой тип Number, который включает в себя всевозможные представимые в компьютере числа. Однако размер представимых чисел (а следовательно диапазон значений и точность представления), ограничены 64 битами. Язык Visual Basic (VB) имеет несколько числовых типов – два вещественных типа Single и Double (этот тип по множеству значений совпадает с типом Number из JavaScript), три целочисленных типа Byte, Integer и Long (они различаются диапазоном представимых значений) и два специальных типа Currency (для финансовых вычислений) и Decimal (для совместимости с некоторыми важными базами данных). В языке Python есть два вещественных типа данных, которые соответствуют Single и Double типам VB, стандартный целый, соответствующий Long, а также два интересных типа, аналогов которым в VB нет – комплексный тип, а также длинный целый тип, который может хранить потенциально бесконечный диапазон целочисленных значений (который на самом деле ограничен объемом доступной

оперативной памяти, но на практике это ограничение, конечно, несущественно).

Заметим, что данные и операции тесно связаны, иногда настолько, что мы говорим о *дуализме* данных и операций. Этот дуализм выражен в языке VB с помощью понятия «свойство» (property). Он выражается в том, что свойства объекта с точки зрения их использования выглядят как данные, которые можно установить (присвоить свойствам значения) и считать (скопировать присвоенные значения в другие данные). Однако на самом деле чтение и запись свойств означают вызов специальных функций, называемых функциями доступа (аксессорами). Таким образом, свойства объектов одновременно являются и данными (с внешней точки зрения), и операциями (с точки зрения реализации свойств).

Теперь рассмотрим понятие связывания, которое не менее важно, чем данные и операции, хотя и менее очевидно.

То, что мы коротко называем **связыванием**, это **процесс установления связи между элементом программы и конкретным атрибутом или характеристикой**. Как правило, связывание сводится к выбору атрибута из (конечного) набора атрибутов. Примеры связываний будут приведены ниже. **Время связывания** – момент установления этой связи. Нас, конечно, интересует не конкретное (астрономическое) время, а несколько основных видов времен связывания. Подробно они описаны в [3], а мы отметим самые важные:

- ✓ *Во время выполнения программы*. Этот вид связывания называется **динамическим**. Например, связывание значения и переменной – динамическое.
- ✓ *При трансляции*. Этот вид связывания называется **статическим**. Например, связывание значения и константы – статическое.

Понимание семантики и времени конкретного связывания часто является решающим при освоении языка. Рассмотрим, например, одно из ключевых связывания языков программирования – связывание переменной и типа данных. В большинстве языков такое связывание происходит в специальной конструкции – объявлении типа. Например, в языке VB эта конструкция имеет вид:

```
Dim ИМЯ as ТИП
```

Например,

```
Dim s as string, i as Integer, j as Integer
```

При таком объявлении связывание переменных с соответствующими типами данных происходит статически – в момент трансляции. Транслятор, следовательно, может

проконтролировать правильность значений, которые можно присвоить переменной. Например, присваивания $s = \text{"строка"} : i = 1 : i = i + j$ корректны, а присваивания $s = -3 : i = \text{"595"}$ некорректны, и будут диагностированы транслятором VB до начала выполнения программы (на самом деле – прямо в момент ввода некорректного присваивания).

В то же время объявление может иметь и такой вид:

Dim *ИМЯ*

что эквивалентно

Dim *ИМЯ* **as Variant**

Например, **Dim v,obj**

Тип данных **Variant** это не тип данных как таковой, а объединение всевозможных допустимых типов данных (в языке Си такое объединение называется `union`, а в языке Паскаль – запись с вариантами). Это означает, что в переменную вариантного типа можно загрузить произвольное значение. Поэтому в данном случае связывание переменной с типом данных – динамическое – в момент выполнения программы. Все приведенные ниже присваивания – корректны:

$v = 2 : obj = \text{"строка"} : v = obj : obj = i$

Время связывания переменной и типа важно с точки зрения еще одного важного связывания: связывания знака операции с семантикой (смыслом) этой операции. Не ограничивая общности можно считать, что семантика любой операции задается некоторой функцией. Поэтому связывание в данном случае сводится к выбору некоторой функции из заранее определенного конечного набора. Например, для переменной s , объявленной как **String**, знак операции $+$ в выражении $s + \text{"еще одна строка"}$ связывается с функцией, выполняющей конкатенацию строк. Это связывание происходит во время трансляции, поэтому транслятор может сразу вставить вызов этой функции или даже сразу подставить ее тело, что существенно повышает эффективность выполнения программы. То же самое можно сказать про выражение $i + j$ из примера выше, только связываемой семантикой будет функция сложения двух целых значений.

А вот про смысл операции $+$ из выражения $v + obj$ ничего сказать во время трансляции в общем случае нельзя. Поэтому транслятор вставляет вызов «обобщенной» операции сложения, которая в момент выполнения программы определяет (связывает) конкретную семантику операции сложения в зависимости от

текущих значений `v` и `obj`. Если в `v` и `obj` находятся строковые значения, то выполнится конкатенация, если целые значения, то – сложение целых чисел. А если типы значений `v` и `obj` различны, то перед выполнением нужной операции выполняется попытка приведения одного из значений к типу другого значения по достаточно сложным правилам, которые не всегда очевидны. При этом разные языки по-разному определяют эти правила. Например, Python вообще не будет ничего приводить, а в момент выполнения этой операции будет выдана ошибка времени выполнения, Visual Basic попытается выполнить приведение, однако не для всех пар допустимых типов данных это приведение возможно, так что ошибка времени выполнения может и не возникнуть, а JavaScript попытается «по максимуму» выполнить все приведения, так что ошибка времени выполнения возникает там еще реже, чем в VB (хотя и там подобного рода ошибки возможны). И вся эта запутанная логика вычисляется во время работы программы, а не до ее начала, что отрицательно влияет на эффективность программ.

Однако динамическое связывание удобно именно для языков управления приложениями (таких как Visual Basic for Applications). Какой тип данных может иметь значение, хранящееся в ячейке листа в MS Excel? Ответ – любой тип! Поэтому в выражении `Cell("A1") + Cell("B2")`, которое вычисляет сумму значений из ячеек A1 и B2 текущего листа, семантика операции `+` связывается именно динамически, и это очень удобно программисту.

1.2 Определение, примеры и общие свойства языков управления приложениями

Теперь дадим определение собственно языков управления приложениями. Язык управления некоторым приложением `A` – это нотация для записи программ, целью которых является расширение и специализация функциональности приложения `A` без какого-либо изменения исходного кода и структуры приложения.

Программы, управляющие приложениями, часто называются макросами или сценариями (или скриптами), а соответствующие языки – командно-скриптовыми.

Каждый ЯУП делится на две части: универсальное ядро, состоящее из конструкций – операторов, объявлений и тому подобного, аналогичных конструкциям из языков программирования общего назначения, а также часть, использующуюся для взаимодействия собственно с управляемым приложением. В

зависимости от принципа устройства этой второй части можно выделить несколько типов управления. Рассмотрим основные схемы управления и примеры ЯУП и приложений, использующих эти схемы.

Самая первая схема управления была разработана для управления приложениями в операционной системе UNIX. Она использует интерфейс системных вызовов для организации параллельных процессов (fork(), семейство вызовов exec() и т.д.) и перенаправление каналов ввода-вывода для связывания процессов в конвейеры. Впервые эта схема была реализована в программе shell [1], после чего различные варианты этой и других командных оболочек были реализованы в ОС UNIX и ее клонах (bash, c-shell, Korn-shell и другие).

Следующий тип управления – **объектный**, характеризуется тем, что в языке выделяется понятие объекта, и все сущности приложения представляются в виде **объектной модели приложения (ОМП)**. Типичный пример этой схемы – Microsoft Office и язык Visual Basic for Applications (VBA). Именно этой схеме уделено основное внимание в данном пособии. В профессиональной версии пакета MS Office каждое приложение пакета имеет встроенную систему программирования на языке VBA. Эта система содержит интерпретатор языка, единый для всех приложений MS Office, редактор программного текста, средства визуального проектирования, отладчик, обозреватель модулей и объектов. Каждое приложение имеет свою собственную объектную модель, хотя, где это возможно, модели унифицированы (например, в каждой модели доступен экземпляр приложения представлен объектом, именуемым Application). Одной из самых сложных (и исторически самой первой) является ОМП Excel. Основы этой модели рассмотрены в третьей главе пособия.

Известным примером управления на основе ОМП является крупнейшая система планирования ресурсов предприятия (ERP-enterprise resource planning) – SAP R/3 с языком ABAP компании Software AG. В России популярна система 1С в разнообразных вариантах. Эта система также имеет встроенный язык 1С-скрипт с ОМП 1С-приложений.

Третий вариант схемы управления – веб-управление – самый последний по времени изобретения и, пожалуй, самый интересный, поскольку объединяет в себе свойства предыдущих схем. Веб-управляемое приложение, или просто – веб-приложение, имеет

клиент-серверную архитектуру. Серверная часть занимается динамической генерацией веб-страниц, а клиентская часть ответственна за отображение и обновление информации и непосредственное взаимодействие с пользователем. При этом современные веб-приложения используют схемы управления, основанные на ЯУП, как в клиентской, так и в серверной части.

Клиентская часть веб-приложения реализуется браузером, который управляется сценариями, встроенными в веб-страницы. В настоящее время стандартным языком управления браузерами является JavaScript. Программы на JavaScript, встроенные в динамически генерируемые веб-страницы, используют объектный подход, аналогичный тому, что рассматривался выше. Соответствующая объектная модель приложения, включающая в себя модель веб-браузера и модель загруженного html-документа, также стандартизована и называется W3C DOM (WWW Consortium Document Object Model).

Серверная часть веб-приложения представлена веб-сервером, управляемым сценариями, которые чаще всего пишутся на командно-скриптовых языках, таких как Perl, Python, PHP, Ruby и других. В последнее время растет популярность пакета Node.JS [7], который позволяет разрабатывать обе части веб-приложения на едином языке – JavaScript. Однако для серверной части отсутствует единая объектная модель типа W3C DOM, так что каждый язык использует специфичные схемы управления и модели, которые в общем случае неприменимы к другим языкам.

2 Основы языка VBA

2.1 Лексика и запись программ

В отличие от таких языков программирования, как Паскаль, C/C++, Java, язык VBA накладывает ограничения на запись программ: каждое предложение языка (оператор или объявление) должно записываться на отдельной строке. Если мы хотим записать сразу несколько предложений в одной строке программы, то нужно разделять их двоеточием. Например,

```
I = I + 10  
J = I/2 - K
```

можно записать как:

```
I = I + 10 : J = I/2 - K
```

Если же, наоборот, требуется записать предложение на нескольких строках – очень длинное или из соображений лучшей читаемости – то это делается с помощью знака подчеркивания непосредственно в конце строки. В этом случае следующая строка «приклеивается к предыдущей»:

```
MsgBox "Ну очень длинное сообщение "  
& "можно записать вот таким образом "  
& "на нескольких строках"
```

Регистр букв не имеет значения, за исключением строковых литералов (см. в примере выше в двойных кавычках). Таким образом, предыдущие предложения можно было записать и так (хотя это не рекомендуется):

```
i = I + 10  
j = I/2 - k
```

```
msgBox "Ну очень длинное ..." 'полное предложение опущено
```

Желательно придерживаться одного стиля записи идентификаторов, а ключевые слова текстовый редактор VBA сам преобразует так, чтобы первая буква слова была в верхнем регистре, а остальные в нижнем: Function, Exit Sub, End If и тому подобное.

Предложения языка делятся на операторы, объявления и комментарии. Комментарии начинаются со знака апострофа и заканчиваются концом строки. Каждый оператор и объявление начинаются со своего ключевого слова, за исключением оператора вызова подпрограммы и оператора присваивания, которые начинаются с идентификатора (строго говоря, оператор присваивания тоже начинается с ключевого слова let или set, но слово let может быть опущено – подробнее см. ниже).

Лексемы (слова) языка делятся на ключевые слова (они зарезервированы и не могут совпадать с идентификаторами), идентификаторы (записываются аналогично идентификаторам языков Паскаль или Си с небольшими отличиями, которые мы пока не рассматриваем), разделители (например, двоеточие или запятая), знаки операций (например, плюс или минус), литералы (например, "строка", 25.7, -6).

2.2 Переменные, константы, основные типы данных

Переменные языка VBA делятся на два вида – типизированные переменные, которые могут принимать значения только одного типа (при попытке присвоить такой переменной значение другого типа либо фиксируется ошибка, либо происходит преобразование значения в нужный тип), и переменные, которые могут принимать значения любого типа. Про переменные второго вида говорят, что они имеют тип Variant. Типизированные переменные не могут стать вариантными, и наоборот.

Объявить тип переменной можно двумя способами (строго говоря, есть еще и третий способ – неявно по первой букве имени переменной, но мы его использовать не будем, так как он противоречит современным понятиям о хорошем стиле записи программ). Первый способ заключается в явном объявлении типа переменной в предложении объявления типа:

```
Dim i As Integer, s As String, k, d As Double
```

Неочевидным является тот факт, что переменная k не является типизированной переменной типа Double. Дело в том, что если в объявлении типа опущена часть «As Тип», как это сделано с переменной k, то переменная считается вариантной. Такого же эффекта можно добиться, если вообще не объявлять тип переменной (то есть по умолчанию все переменные – варианты), либо указать вариантность явно:

```
Dim k As Variant
```

С точки зрения хорошего стиля записи программ явное указание считается предпочтительным, поэтому в VBA есть специальное объявление, которое требует **всегда** явно указывать тип или вариантность переменной:

```
Option Explicit
```

Такое объявление нужно указывать один раз в модуле (о модулях и структуре программы см. ниже). В этом случае употребление необъявленной переменной будет рассматриваться транслятором как ошибка. Объявленные переменные получают

некоторое начальное значение, зависящее от типа, например, переменные числовых типов – нуль, строковые переменные – пустую строку и тому подобное. К сожалению, инициализировать переменную в объявлении типа каким-либо другим значением нельзя, это можно сделать далее путем присваивания.

Второй способ объявления типа переменной заключается в употреблении специального символа-маркера типа в имени переменной (точнее – в конце имени). Так, если имя переменной заканчивается на знак \$, то эта переменная имеет строковый тип, если на знак #, то – Double, % – Integer и т.д. Символ-маркер типа должен всегда указываться сразу после имени переменной, и он не может меняться, то есть одновременное употребление, например, k% и k#, либо k% и k – запрещено (правда, есть исключение, касающееся имен встроенных функций). Мы не рекомендуем использовать этот прием объявления переменных, однако помнить это правило надо, поскольку оно распространяется и на имена функций, в том числе встроенных. Все встроенные функции VBA, которые возвращают строковый тип, имеют две формы имени – первая форма имеет маркер строкового типа \$ в конце и возвращает значение типа строка, а вторая форма не имеет маркера типа и возвращает тип Variant.

Язык позволяет объявлять не только переменные, но и именованные константы, которые должны быть инициализированы значением сразу в объявлении, и после этого не могут поменять это значение:

```
Const BufferSize As Integer = 128  
Message As String = "Error in macro"
```

Остановимся на вопросе, стоит ли использовать типизированные переменные? С одной стороны, варианты переменные могут принимать любые допустимые в языке значения, поэтому использование типизированных переменных выглядит избыточным. Поэтому некоторые языки, такие как JavaScript, Python, Ruby не имеют типизированных переменных вообще. Однако использование типизированных переменных в отдельных фрагментах программы более эффективно (по причинам, рассмотренным в первой главе), а также позволяет найти некоторые (но отнюдь не все!) ошибки программы уже на стадии трансляции.

Теперь рассмотрим типы данных языка VBA. Можно выделить следующие группы:

- числовые типы
- логический (Boolean)
- строковые (String)
- дата и время (Date)
- объектные (Object)

Числовые типы, в свою очередь, делятся на следующие:

- целочисленные (Byte, Integer, Long)
- вещественные (Single, Double)
- валютный (Currency)
- десятичный (Decimal)

Значения целочисленных типов представляют собой целочисленные значения размером в 1 байт (Byte – диапазон значений 0 - 255), 2 байта (Integer – диапазон значений от -32768 до 32767) и 4 байта (Long – диапазон значений от -2147483648 до 2147483647). Заметим, что такие размеры и диапазоны целочисленных значений нехарактерны для архитектуры IA-32, на которой реализована ОС Windows и пакет MS Office. Они унаследованы от старых реализаций языка Visual Basic на 16-битных архитектурах. Это еще раз подчеркивает тот факт, что виртуальная машина языка может достаточно сильно отличаться от программно-аппаратной архитектуры вычислительной системы, на которой реализована машина.

Операции над целочисленными типами включают в себя обычные арифметические операции: сложение (+), вычитание (-), умножение (*), деление (/) – дает результат типа Double, деление нацело (\) – дает целую часть от результата деления, остаток от деления нацело (mod). Также имеется более «экзотическая» операция возведения в степень (^): 2^{10} даст 1024.

Также к целочисленным (как и любым числовым) типам применимы шесть обычных операций сравнения (обратите внимание на то, как записывается операция неравенства – <>): <, >, =, >=, <=, <>. Эти операции возвращают результат логического типа.

Вещественные типы представляют собой значения чисел, представленных в форме с плавающей точкой. Эти типы соответствуют спецификации представления вещественных типов IEEE-754. Тип Single соответствует 32-битному представлению

одинарной точности, а Double – 64-битному представлению двойной точности.

Помимо четырех стандартных арифметических операций к вещественным типам применима операция возведения в степень (^), а также операции сравнения.

Заметим, что формат представления вещественных чисел таков, что точность представления чисел относительна – чем больше модуль значения, тем меньше точность. Таким образом, для значений типа Single точность представления значений порядка 16 миллионов (что примерно совпадает с числом 2^{24}) составляет единицу. Так что представлять некоторые значения, требующие фиксированной точности для любых диапазонов значений, вещественными типами – крайне неудобно. Для некоторых диапазонов значений точность вещественных чисел слишком велика, для других – слишком мала. К таким значениям относятся, например, валютные значения, применяемые в финансовых расчетах. Они требуют фиксированной точности вычислений (четвертый знак после десятичной точки должен быть точным) для сколь угодно больших (в разумных пределах, конечно) диапазонов значений. Поэтому для финансовых расчетов, критично важных для приложений MS Office, в язык введен специальный валютный тип – Currency. Также в языке есть тип для представления десятичных чисел с фиксированной точкой – Decimal. В данном пособии мы не будем останавливаться на этих типах в силу их специфичности.

Логический тип данных имеет два значения – True и False. Эти значения можно свободно присваивать целым переменным. True представляется значением «-1» (то есть все единицы в битовом представлении), а False – нулем. К логическим переменным и значениям применимы следующие операции (известные нам из курса алгебры логики):

- And (логическое «И»)
- Or (логическое «ИЛИ»)
- Not (логическое «НЕ»)
- Xor (логическое «Исключающее ИЛИ»)
- Eqv (логическая эквивалентность)
- Imp (логическая импликация)

Строковые типы служат для представления текста, так что они, очевидно, являются одними из наиболее используемых. В VBA есть два строковых типа – с ограничением длины строки и с

неограниченной длиной строки. В строках первого типа память под буфер, где хранится текст, выделяется при объявлении переменной. Длина строки (константное выражение) указывается после имени типа через символ-звездочку:

```
Dim Name as String * 20, Surname as String * 40
```

При помещении текста в переменную-строку с ограничением длины текст, превышающий предельную длину, будет обрублен. Так, в следующем фрагменте будет выведено слово «hell»:

```
Dim s As String * 4  
s = "hello!!!"  
MsgBox s
```

Максимальное значение длины ограниченной строки: 65526, однако нужно помнить, что в эту длину включается еще и финальный символ-ноль, который VBA добавляет автоматически для того, чтобы строковые значения VBA были совместимы со строками языка C. Поэтому реальная длина текста, который можно поместить в ограниченные строки – 65525.

Если в объявлении переменной строкового типа не указано ограничение длины, то память под текст выделяется динамически, поэтому длина текста, который может поместиться в такую переменную, ограничена лишь объемом доступной кучи. Поскольку MS Office реализован в среде ОС Windows в 32-битной архитектуре, то теоретический максимум длины – 2Гбайт (поскольку 32-битным программам в пользовательском режиме ОС Windows доступно именно 2Гбайт памяти). На практике, конечно, этот лимит меньше по понятным причинам.

Достоинством неограниченных строк является автоматическая сборка мусора, то есть программисту на VBA не надо заботиться об отведении и освобождении памяти под строки, равно как и под другие объекты данных, размещаемые в динамической памяти (хотя определенные предосторожности при работе с динамически размещаемыми объектами соблюдать все-таки надо – подробнее об этом написано в главе, посвященной объектам).

Почему в языке существует два строковых типа? По двум причинам, первая из которых – совместимость с программами, разработанными для старых версий VB, которые работали в 16-битной виртуальной машине (отсюда и предельное ограничение длины – строка и служебная информация должны вмещаться в сегмент размера 64Кбайт = 65536 байт). Вторая причина состоит в том, что в реляционных СУБД (например, MS Access из состава пакета MS Office или MS SQL Server) в прямоугольных таблицах

удобно (эффективно) хранить строковые значения ограниченной длины. Конечно, неограниченные по длине тексты тоже можно хранить в современных СУБД (для этого в языке SQL есть специальные типы данных), но работа с такими значениями происходит менее эффективно.

Ограниченные и неограниченные строки полностью совместимы по набору операций с единственным ограничением на «обрубание» текста, помещаемого в ограниченные переменные. Кроме того, ограниченные строки не могут быть формальными параметрами процедур и функций, а также возвращаемыми значениями функций (как следствие, все встроенные функции VBA возвращают неограниченные строки). Таким образом, язык поощряет использование неограниченных строк везде, где это не противоречит соображениям производительности.

Основная операция, которую можно выполнять над строками, это конкатенация строк (&). Операция конкатенации строк s1 и s2 порождает новую строку s, в начало которой копируется строка s1, а затем в хвост s дописывается строка s2. Пример ниже выдает строку «Привет, Андрей!»:

```
Dim s As String
s = "Привет, " & "Андрей!"
MsgBox s
```

Знак операции конкатенации (&) обозначает только ее и не имеет никакого другого смысла. Но операция «плюс» (+) применительно к строковым значениям также обозначает конкатенацию. Для типизированных переменных это не вызывает проблем, однако для вариантных переменных действие, выполняемое операцией +, определяется только во время выполнения в зависимости от текущих значений операндов (то есть операция + связывается для вариантных переменных динамически – см. главу 1). Например, первый вызов подпрограммы MsgBox выдаст значение 3, а второй – 12:

```
Dim v1, v2 ' обе переменные имеют тип Variant
v1 = 1 : v2 = 2 ' обе переменные хранят значения целого типа
MsgBox v1 + v2 ' сложение целых значений
v1 = "1" : v2 = "2" ' обе переменные хранят
' значения строкового типа
MsgBox v1 + v2 ' конкатенация двух строк
```


А вот результат следующего фрагмента не так очевиден:

```
Dim v1, v2 'обе переменные имеют тип Variant
v1 = "1" ' переменная хранит значение строкового типа
           ' и это значение переводится в число
v2 = 2 ' переменная хранит значение целого типа
MsgBox v1 + v2 ' результат – строка "3"
v1 = "X" ' переменная хранит значение строкового типа
           ' но это значение не переводится в число
v2 = 2 ' переменная хранит значение целого типа
MsgBox v1 + v2 ' результат – ошибка
```

В первом случае интерпретатор VBA во время выполнения `v1 + v2` определит, что строковое значение представляет собой десятичную запись числа, и переведет это значение в число, после чего выполнит операцию сложения целых с результатом 3. Во втором случае "X" не является представлением никакого числа, поэтому будет выдана ошибка времени выполнения – «Type mismatch» – несоответствие типов.

Интересно, что замена в предыдущих фрагментах знаков операции + на знак операции конкатенации & даст совершенно другой результат для обоих фрагментов! Поскольку эта операция однозначна и требует **только** строковые операнды, то интерпретатор VBA во всех случаях переведет целые значения в строковое представления, и выполнит конкатенацию текстов `v1 & v2`. Поэтому в первом фрагменте будет оба раза выдан результат 12, а во втором фрагменте в первый раз – также 12, а во второй – X2.

Кроме операции конкатенации к строкам применимы 6 операций сравнения. Остановимся на них подробнее. Сравнение строк происходит в так называемом «лексикографическом» (то есть словарном) порядке: строки сравниваются на равенство посимвольно слева направо (от начала к концу строки) до тех пор, пока либо не кончатся обе строки одновременно (тогда они считаются равными), либо не кончится одна из строк (тогда та строка, которая кончилась, является префиксом второй, и более короткая строка считается меньше), либо будет достигнуто неравенство символов двух строк (в этом случае строки не равны, и результат сравнения строк есть результат сравнения найденных символов).

Таким образом, сравнение строк в итоге сводится к сравнению символов. Не вдаваясь в тонкости вопросов представления символов в виртуальной машине языка VBA, отметим только, что для европейских языков (включая языки с кириллическим алфавитом)

отдельные символы в строках VBA представлены в однобайтовой кодировке, зависящей от текущих региональных установок в ОС Windows. То есть каждый символ представлен одним байтом (для некоторых неевропейских языков, таких, как китайский или японский, это не так, но обсуждение этого вопроса выходит за рамки данного пособия). По умолчанию сравнение символов – это сравнение значений соответствующих байтов в текущей кодировке. Такой способ сравнения называется двоичным (binary), что апеллирует к тому факту, что сравниваются двоичные представления символов как числа. Однако такой способ сравнения не всегда удобен. Так, например, когда мы ищем слово в тексте, то нас может не интересовать тот факт, что слово может стоять в начале предложения и, следовательно, начинается с заглавной буквы. Но кодировки заглавной и строчной букв не совпадают (это же разные символы!), поэтому двоичное сравнение этих символов выдаст неравенство, так что вхождения слова в начало предложения не будут найдены. Обычно в таких случаях переводят сравниваемые тексты в один регистр (неважно, верхний или нижний), и ищут подстроки в «нормализованном» тексте. Однако такая операция требует дополнительное время и, что особенно важно для больших текстов, память. Поэтому в VBA есть возможность повлиять на механизм сравнения символов: есть специальный способ сравнения символов, который называется текстовым (text). При текстовом сравнении символы сравниваются независимо от регистра. В этом случае выражение "Ю" = "ю" будет истинным (True), в то время, как при двоичном сравнении оно будет ложным (False). Заметим, что этот способ также зависит от текущих региональных установок. Таким образом, если в системе установлены русские региональные значения, то русские символы будут сравниваться корректно. Если же установки соответствуют не русскоязычному региону (стране), а скажем, Франции или Греции, то текстовое сравнение для русскоязычных текстов будет некорректным.

Способ сравнения устанавливается специальным объявлением:

```
Option Compare Text
```

Это объявление должно быть помещено перед первой подпрограммой модуля. По умолчанию действует следующее объявление:

```
Option Compare Binary
```

Использование такого объявления не решает всех проблем, так как нельзя использовать одновременно оба способа в разных

подпрограммах одного модуля. Поэтому более универсальным подходом является использование встроенной функции `StrComp` вместо операций сравнения (`<`, `>`, `=`, `>=`, `<=`, `<>`). Первые два аргумента функции – строки. Функция выдает значение `-1`, если первый аргумент меньше второго, `0`, если они равны, `+1`, если первый аргумент больше второго. Если третий аргумент функции отсутствует, то сравнение выполняется по способу, установленному объявлением `Option Compare`, но можно установить нужный способ, указав третий аргумент: `vbBinaryCompare`, либо `vbTextCompare` (это встроенные константы языка VBA).

Еще одна операция над строками, напоминающая сравнение – операция подобия шаблону (`Like`):

```
S Like pattern
```

Операция возвращает результат `True`, если строка `S` соответствует шаблону `pattern`, и `False` в противном случае. Шаблон – это строка, которая определяет правила соответствия. Все символы, которые можно употреблять в строках VBA, делятся на два класса – обычные символы и метасимволы (иногда их называют подстановочными). При поиске соответствия (отождествления) символы строки и шаблона отождествляются по одному слева направо. Если символ в шаблоне – обычный, то при совпадении его с соответствующим символом из строки ищется соответствие для следующего символа из шаблона и так далее до конца шаблона. Если шаблон и строка окончились одновременно, и все символы отождествились, то операция возвращает `True`. Если же имело место несовпадение хоть по одному символу, то – `False`. Так что для обычных символов все очень просто и `Like` не отличается от обычной операции проверки на равенство. Если же в шаблоне встречается метасимвол, то он может быть отождествлен не сам с собой, а с одним или несколькими символами. Таблица 1 описывает метасимволы, допустимые в VBA.

Отметим, что отождествление (проверка на равенство) в операции `Like` проходит так же, как и при сравнении символов в операции `=`, то есть с учетом объявления `Option Compare`.

Нужно сказать, что более современный аппарат регулярных выражений (доступный и в VBA при подключении соответствующей библиотеки классов) является существенно более мощным и выразительным, чем операция `Like`, но его рассмотрение выходит за рамки настоящего пособия.

Таблица 1. Метасимволы, допустимые в VBA

Символ(ы)	Значение	Примеры
*	Отождествляется с любой последовательностью символов (в том числе и пустой)	"aa" Like "a*a" → True "abbaa" Like "a*a" → True "abcd" Like "a*a" → False
#	Отождествляется с любой цифрой от 0 до 9	"333-22-11" Like "###-##-##" → True
?	Отождествляется с любым одним символом	"a?" Like "a1" → True "a?" Like "ab" → True "a?a" Like "aa" → False
[a-f,0,2,4,D-K]	Отождествляется с любым одним символом из диапазона или списка	"axb" Like "a[x,y,z]b" → True "acb" Like "a[x,y,z]b" → False "Foo" Like "[A-Z,a-z]*" → True "foo" Like "[A-Z,a-z]*" → True "lfoo" Like "[A-Z,a-z]*" → False
[!a-f,0,2,4,D-K]	Отождествляется с любым одним символом НЕ из диапазона или списка	"axb" Like "a[!x,y,z]b" → False "acb" Like "a[!x,y,z]b" → True "Foo" Like "[!A-Z,a-z]*" → False "foo" Like "[!A-Z,a-z]*" → False "lfoo" Like "[!A-Z,a-z]*" → True

Следует сказать, что реальный набор операций над строками существенно расширен набором встроенных функций VBA – их несколько десятков. Так, кроме рассмотренной выше функции StrComp есть функции поиска подстроки InStr, форматирования, преобразования строк (например, выделения левого префикса – Left, окончания – Right и т.д.). Часть встроенных функций языка рассмотрена в приложении к настоящему пособию.

При изучении этих функций очень важно понимать, что строковые значения в VBA являются **неизменяемыми**, то есть ни одна операция или встроенная функция не могут изменить ни одного символа в строковом значении. Любые функции преобразования (например, преобразование всех символов строки в верхний регистр – UCase, удаление ведущих и хвостовых пробелов Trim и тому подобные) никогда не меняют собственно преобразуемые значения, вместо этого они возвращают преобразованную копию исходного значения. Такое ограничение позволяет более эффективно обрабатывать и хранить строковые значения в оперативной памяти, поэтому оно принято во многих других языках программирования, например, в JavaScript, C#, Java.

Тип `Date`, значения которого представляют собой даты (или только время суток) с точностью до секунды, также, как и строковые типы, обладает небольшим набором встроенных операций. Даты можно присваивать и сравнивать с помощью обычных операций сравнения. Однако, как и в случае строк, реальный набор операций «зашит» в большое количество встроенных функций, которые позволяют оперировать с датами и временными промежутками, форматировать даты и т.п. Например, очень полезна встроенная функция без параметров `Now`, которая возвращает значение текущей даты и времени. Внутреннее представление даты в интерпретаторе VBA – вещественное число, целая часть которого равна числу дней, прошедших с 1 января 1900 года, а дробная часть представляет время суток (0,5 соответствует 12:00:00, 0,1 – 2:24:00 и так далее). Поэтому датам можно присваивать числа (в том числе и отрицательные – тогда отсчет времени идет в обратную сторону от полуночи 1 января 1900 года), что не очень интересно. Более интересна возможность выполнения сложения и вычитания числовых значений с датами. Так `Now+1` соответствует «завтра», а `Now-2` – «позавчера». Вычитание двух дат дает временной промежуток, выраженный как вещественное число (целая часть – дни, дробная – доля времени суток в промежутке). Однако встроенные функции позволяют существенно более гибко работать с этими операциями. Так, можно прибавлять и вычитать временные промежутки, выраженные не только в днях, но и в годах, месяцах и даже в кварталах (равно как и в минутах, секундах и т.д.). К сожалению, объем пособия не позволяет нам подробно остановиться на этом типе.

Последний тип данных VBA, который мы рассматриваем – тип `Object`. В некотором смысле именно он является основным, поскольку работа с управляемым приложением в VBA происходит именно через переменные объектного типа. Подробнее об объектах мы будем говорить в главе 3, посвященной объектным моделям приложений. Сейчас отметим, что в VBA, как и в подавляющем большинстве других языков программирования с объектами, принята так называемая референциальная объектная модель. Она означает, что экземпляры любых объектов размещаются исключительно в динамической памяти, являются анонимными и доступны только через ссылки. Так что значениями объектного типа данных (то есть значениями переменных типа `Object`) являются не собственно объекты, а ссылки на них. Есть специальное обозначение для пустой ссылки, которая не ссылается ни на один объект – `Nothing`.

Референциальность означает, что при присваивании объектных значений не происходит копирования объектов – копируются только ссылки на один и тот же экземпляр объекта, так что к одному объекту можно обращаться посредством разных имен ссылок. Так, в примере ниже wrdObject и obj ссылаются на один и тот же объект, поэтому будет выведен заголовок "This is new title".

```
Dim wrdObject as Object
Set wrdObject = CreateObject("Word.Application")
Dim obj as Object
Set obj = wrdObject
obj.Caption = "This is new title"
MsgBox wrdObject. Caption
```

2.3 Массивы и структуры

В языке VBA массивы и структуры не имеют такого значения, как в универсальных языках программирования, поскольку объектные модели офисных приложений и модели из подключаемых библиотек объектов весьма богаты, что существенно снижает необходимость использования этих типов данных в макросах.

Структуры VBA по традиции, идущей от первых версий языка Basic, называют пользовательскими типами данных, хотя современная версия VBA дает возможность создавать существенно более общие и гибкие пользовательские типы данных с помощью механизма классов.

Пользовательский тип объявляется следующим образом:

```
Туре имя_типа
    поле1
    поле2
    . . .
End Туре
```

Объявление поля очень похоже на объявление переменной – только без начального слова (типа Dim, Public и т.п.):

```
Имя_поля As тип
```

Таким образом, объявление пользовательского типа данных очень близко к структуре языка C и записи (без вариантов) языка Паскаль. Поэтому далее мы будем использовать название «структура» для этих типов данных. Использование пользовательских типов полностью аналогично использованию записей и структур. Доступ к полю переменной пользовательского типа данных осуществляется, как и в других языках программирования, через точку: *имя_переменной.имя_поля*.

Наиболее часто структуры используются для хранения информации, полученной или записываемой в таблицу реляционной базы данных (например, MS Access из пакета MS Office). Для других применений рекомендуется использовать более адекватный механизм классов.

Рассмотрим массивы VBA. Как и строки, они делятся на два вида: статические, память под которые выделяется в период трансляции, и динамические, память под которые выделяется в куче во время выполнения программы. Но в отличие от динамических строк, память под которые выделяется в момент создания строки и более не перераспределяется, так как строки – неизменяемые сущности, память под динамические массивы выделяется и перераспределяется явно программистом с помощью специального оператора Redim.

Рассмотрим сначала объявление одномерного статического массива:

```
Dim имя_массива(размерность) As тип
```

Тип – тип элементов массива. Размерность указывает диапазон изменения индекса элементов массива и имеет вид: L To R, где L – левая граница, R – правая. Поскольку память под массив выделяется статически, то L и R должны быть целыми неотрицательными константами. L может принимать только два значения – 0 или 1. Левая граница может быть опущена, тогда размерность есть просто константа, а левая граница по умолчанию равна 0. Однако для совместимости со старыми версиями языка Basic это умолчание можно поменять на 1 с помощью объявления:

```
Option Base 1
```

Обращение к элементу массива в VBA (как и во всех версиях Basic), отличается от большинства языков программирования и синтаксически выглядит как обращение к функции, так как использует круглые скобки вместо квадратных:

```
Arr(I,J) = Arr(I,K)*Arr(K,J)
```

```
V(I+1) = V(N-I)
```

В качестве индексов могут использоваться только целочисленные выражения.

Примеры объявления статических массивов:

```
Dim WordList(1 To 3) As String
```

```
Vect(10) As Integer
```

Многомерные статические массивы объявляются аналогично одномерным, только вместо одной размерности указывается список размерностей через запятую:

```
Dim Matrix(3,3) As Double
ColorCube(255,255,255) As Long
```

Элементы в многомерных массивах, как говорят, хранятся «по строкам», то есть в оперативной памяти многомерный массив «вытягивается в цепочку», причем при проходе по такой последовательности элементов быстрее всего меняется последний индекс. То есть элементы массива $m(3,3)$ хранятся в оперативной памяти в следующем порядке: $m(0,0)$ $m(0,1)$ $m(0,2)$ $m(0,3)$ $m(1,0)$ $m(1,1)$. . . $m(2,3)$ $m(3,3)$

Число и длину размерностей в статических массивах изменить нельзя.

Динамические массивы предоставляют больше гибкости. Они позволяют менять все параметры массива – и длину, и число размерностей. Массив является динамическим, если в его объявлении опущена размерность:

```
Dim DynArr() as Double
```

Заметим, что реально память под массив при объявлении не отводится – имя динамического массива следует рассматривать только как ссылку, которую еще надо инициализировать реальным массивом. Это происходит с помощью оператора `Redim`:

```
Redim имя_динамического_массива (список_размерностей)
```

Заметьте, что `Redim` – оператор, а не объявление. Он интерпретируется при выполнении программы, поэтому границы в списке размерностей могут быть произвольными целочисленными выражениями, для левой границы вычисленное значение соответствующего выражения должно быть равно 0 или 1, для правой границы – не меньше левой.

```
Dim A() As Integer, N As Integer
N = InputBox("Введите размерность матрицы")
ReDim A(1 To N, 1 To N)
```

```
....
ReDim A(1 To N, 1 To N, 1 To N)
```

Заметим, что в динамическом массиве число размерностей может меняться при повторном выполнении оператора `Redim`, как в примере выше.

Оператор `Redim` может сохранять предыдущие значения элементов:

```
Dim A() As Integer
ReDim A(1 To 10)
For i = 1 To 10
    A(i) = i
Next i
ReDim Preserve A(1 To 20)
```

В этом примере первые 10 элементов массива `A` сохраняют свои значения из-за употребления слова `Preserve`. Следует заметить, что в многомерном динамическом массиве слово `Preserve` нельзя употреблять, если меняются длины размерностей, кроме последней, то есть можно менять только последнюю размерность, в противном случае будет выдана ошибка времени выполнения. Это связано с тем, что, как уже было сказано, при хранении многомерного массива в памяти быстрее всего меняется последний индекс.

```
Dim A() As Integer
ReDim A(1 To 10, 1 To 10)
. . .
ReDim Preserve A(1 To 10, 1 To 20) 'допустимо – первые
    '100 элементов массива сохраняют значения
ReDim Preserve A(1 To 20, 1 To 20) 'недопустимо – будет
    'выдана ошибка: subscript out of range
```

Так же, как и в случае с динамическими строками, программисту не нужно заботиться о том, чтобы выделенная под динамические массивы память освобождалась – это сделает система сборки мусора автоматически.

2.4 Основные операторы

Как и в любом процедурном языке, основным оператором в VBA является оператор присваивания. Особенность языка Basic состоит в том, что он имеет две формы: простую (`let`) и объектную (`set`). `Let`-форма применима ко всем типам, кроме объектного, а `set` – только к присваиванию объектов (и обязательна). Форма указывается перед левой частью:

```
Let I = I + 10
Set obj = wrdObject
```

Если форма не указана, то по умолчанию принимается `Let` (как правило, это ключевое слово опускается). Использование `Let`-формы для типизированной объектной переменной приводит к выдаче ошибки при попытке присвоить значение. Так, в примере из конца предыдущего пункта, замена `Set obj = wrdObject` на

`obj = wrdObject` приведет к выдаче диагностического сообщения во время выполнения этого присваивания. Однако, если объявить `obj` как вариантную переменную (заменив `Dim obj as Object` на `Dim obj as Variant` или просто закомментировав это объявление), то присваивание выполнится, но любая попытка использовать переменную `obj` для ссылки на объект будет сопровождаться выдачей ошибки. Так, ошибка будет выдана при обращении к свойству `obj.Caption = "This is new title"`.

Это еще один аргумент в пользу использования типизированных переменных везде, где это возможно.

В остальном оператор присваивания ведет себя так же, как и в других языках. При присваивании значения типизированной переменной требуется, чтобы тип присваиваемого значения был совместим с типом переменной.

Кроме оператора присваивания в языке есть также операторы управления последовательностью выполнения и операторы ввода-вывода. Последние достались VBA «в наследство» от Visual Basic и для управляемых приложений используются достаточно редко, так как приложения MS Office сами хранят, считывают и записывают свои данные, так что нужда в операторах ввода-вывода возникает только при чтении-записи информации в «неродных» для MS Office форматах и журналировании (хотя журналирование также можно организовать через сервисы приложений MS Office). Так что в настоящем пособии мы не будем рассматривать эти операторы, а в оставшейся части этого раздела остановимся на основных управляющих операторах: ветвлениях и циклах. Оператор вызова процедуры будет рассмотрен в следующем разделе.

Прежде, чем рассматривать конкретные операторы, сделаем следующее замечание по синтаксису языка. И операторы цикла, и операторы ветвления относятся к классу сложносоставных операторов, то есть операторов, содержащих внутри себя другие операторы (в отличие от простых операторов, таких как присваивание или переход по метке, которые не имеют операторов внутри себя). Все языки программирования делятся на две группы по синтаксическому строению сложносоставных операторов. В языках первой группы отсутствуют явные завершители сложносоставных операторов. Вместо этого конец сложносоставного оператора определяется по структуре оператора, которая не может содержать более одного подряд идущего вложенного оператора. Если же требуется использовать несколько вложенных операторов подряд, то

в таких языках предусмотрена специальная конструкция – составной оператор (Паскаль, Модула 2, Оберон) или блок (C/C++, Java, C#), которая специально служит для объединения группы подряд идущих операторов в один. Потому там, где допустим только один оператор, можно всегда вставлять составной оператор (блок), в котором уже допустимы любые операторы в любом числе. Языки другой группы (к ним относится и VBA) более ортогональны, то есть они допускают любое количество вложенных операторов внутри сложносоставных. Как следствие, в этих языках отсутствует понятие составного оператора. Границы сложносоставных операторов в этих языках указываются явно, с помощью специальных завершителей (или терминаторов). В VBA таким терминатором служит конструкция из двух служебных слов – служебное слово End и слово, которое начинает сложный оператор (заметим, что каждый сложносоставной оператор начинается со своего служебного слова). При наборе программы достаточно указать только End, а нужное слово редактор подставит сам. Исключение из этого правила составляют операторы цикла, которые заканчиваются специфическими терминаторами (не End) – о них мы будем говорить в конце раздела. Терминатор рассматривается как отдельный оператор и, следовательно, должен быть записан на отдельной строчке. Наличие явного терминатора позволяет сделать структуру сложносоставных операторов более логичной, правда за счет некоторого увеличения числа строк в исходном тексте программы.

В языке есть два оператора ветвления – условный оператор и оператор ветвления. Условный оператор имеет следующий синтаксис:

```
If условие1 Then
    группа операторов1
ElseIf условие2 Then
    группа операторов2
...
Else
    группа операторовN
End If
```

Обязательными являются только первый оператор If (с группой операторов1), а также завершающий оператор End If. Остальные ElseIf и Else (с соответствующими группами операторов) являются необязательными. Таким образом наличие явного терминатора позволяет объединить все формы условного оператора (с развилкой на одно направление – «если-то», на два –

«если-то-иначе» и многовариантный выбор – «если-то-иначе если-то-иначе если ...») в одну конструкцию.

Второй оператор ветвления – оператор выбора, можно рассматривать как частный, но важный случай условного оператора. В случае, когда все условия в многовариантном операторе имеют вид сравнения одного и того же выражения с разными вариантами константных значений, вместо условного оператора можно (и нужно!) использовать оператор выбора. Его структура выглядит так:

```
Select Case выражение
Case список вариантов1
    группа операторов1
Case список вариантов2
    группа операторов2
    .
    .
    .
Case Else
    группа операторовN
End Select
```

Часть `Case Else` является необязательной. Отдельный вариант в списке вариантов имеет вид либо просто константы – «1» (тип константы должен быть совместим с типом выражения в `Select Case`), либо диапазона констант – «5 To 7», либо неравенства вида `Is знак-отношения константное выражение` – `Is > 10`. Список вариантов – любая непротиворечивая комбинация вариантов через запятую. Значение удовлетворяет списку вариантов, если оно удовлетворяет одному из вариантов списка (в силу непротиворечивости списка таких вариантов может быть не больше одного), то есть запятая в списке имеет смысл логической связки «ИЛИ». Различные списки вариантов в одном операторе не должны пересекаться, то есть значение может удовлетворять не более, чем одному списку. В процессе выполнения оператора выбора вначале вычисляется выражение. Если полученное значение не удовлетворяет ни одному из списков вариантов, то выполняется группа операторов после `Case Else`, если этот оператор присутствует. Если же его нет, то все ветви оператора выбора игнорируются, и выполняется следующий оператор. Если же какой-то список вариантов удовлетворяет значению (из описанных выше ограничений следует, что такой список может быть только один), то выполняется группа операторов, следующая непосредственно за `Case` с этим списком.

Приведем пример оператора выбора, который по количеству баллов `score` корректно определяет, какую форму слова «балл»

следует напечатать («три балла», «один балл», «пять баллов» и т.п.).

О функциях подробно говорится в следующем разделе.

```
Function score_to_str(ByVal score As Integer) As String
    Dim rmd100 As Integer, rmd10 As Integer
    rmd100 = score Mod 100
    rmd10 = score Mod 10
    Select Case rmd100
    Case 11 To 19
        score_to_str = score & " баллов"
    Case Else
        Select Case rmd10
        Case 2, 3, 4
            score_to_str = score & " балла"
        Case 1
            score_to_str = score & " балл"
        Case Else
            score_to_str = score & " баллов"
        End Select
    End Select
End Function
```

Теперь рассмотрим операторы цикла. VBA, видимо, относится к числу лидеров среди процедурных языков программирования по их количеству.

Вначале рассмотрим общий оператор цикла Do-Loop, который порождает несколько разновидностей.

```
Do
    группа операторов
Loop
```

Этот оператор повторяет группу операторов потенциально бесконечное число раз. Поскольку бесконечный цикл в макросах приводит к «зависанию» как программы на VBA, так и всего управляемого приложения, то, конечно, необходим способ выхода из этого цикла. Он обеспечивается оператором выхода из цикла: Exit Do (этот оператор очень похож на оператор break из языков с C-подобным синтаксисом). Рассмотрим пример:

```
I = 0
Do
    I = I + 1
    If I = 10 Then
        Exit Do
    End If
Loop
```

Цикл общего вида и оператор выхода позволяют организовать циклы с выходом из любого места цикла. В случае, когда выход происходит в начале или в конце цикла, а этот случай чаще всего встречается на практике, можно использовать условия выхода `While` или `Until`. Эти условия могут появляться в начале цикла – сразу после слова `Do` и в конце – сразу после слова `Loop`, порождая тем самым 4 дополнительных оператора цикла:

```
Do While условие  
      группа операторов  
Loop
```

```
Do Until условие  
      группа операторов  
Loop
```

```
Do  
      группа операторов  
Loop While условие
```

```
Do  
      группа операторов  
Loop Until условие
```

Если условие стоит в начале цикла, то оно проверяется перед началом каждой итерации, если же в конце – то после каждой итерации. Разница между условиями `While` и `Until` состоит в том, что истинность условия `While` означает продолжение цикла, а истинность `Until` – его окончание (сравните это с операторами цикла в С и Паскале).

Однако пяти форм цикла авторам языка показалось явно недостаточно. Так, в языке остался для совместимости со старыми версиями Basic оператор `While-Wend`.

```
While условие  
      группа операторов  
Wend
```

Этот цикл эквивалентен `Do While - Loop` циклу.

Также в языке есть две формы цикла `For` или цикла с параметром, которые предназначены для последовательного просмотра элементов массивов или последовательностей более общего вида – коллекций.

Первая форма очень похожа на аналогичный цикл из языка Паскаль, только позволяет еще и задавать шаг изменения параметра цикла:

```
For параметр = нач_значение To кон_значение Step шаг  
    группа операторов  
Next параметр
```

Пример цикла суммирования элементов массива A:

```
Sum = 0  
For i = 1 To Len(a)  
    Sum = Sum + A(i)  
Next i
```

Этот вид цикла не обязательно применять именно для массивов и коллекций, он удобен везде, где количество итераций цикла известно до начала его выполнения. Но вторая форма цикла For – For Each – специально и исключительно предназначена для последовательного доступа к элементам коллекций и массивов. Параметр цикла в For Each последовательно принимает значения элементов последовательности, в то время как в цикле For параметр рассматривается как номер (индекс) элемента:

```
For Each параметр in последовательность  
    группа операторов  
Next параметр
```

Предыдущий пример с суммированием массива принимает более наглядный (и более эффективный) вид:

```
Sum = 0  
For each elem in A  
    Sum = Sum + elem  
Next elem
```

Циклы, ветвления и аппарат подпрограмм вполне покрывают все потребности программиста в управляющих структурах, поэтому использование оператора перехода для формирования каких-то необычных программных конструкций совсем не поощряется современным стилем программирования. Однако, иногда оператор перехода все-таки приходится употреблять. В VBA этот оператор чаще всего применяется для обработки ошибок, поэтому коротко рассмотрим синтаксис оператора перехода:

```
Goto метка
```

метка – идентификатор, который помечает один (и только один) оператор внутри процедуры или функции:

```
метка: оператор
```

Область видимости метки – процедура или функция, нельзя использовать нелокальные переходы (из одной подпрограммы в другую), переходы внутрь сложных операторов. Таким образом, область применения операторов перехода в VBA – только

обработка ошибок. Нужно учесть, что операторы выхода позволяют обходиться без переходов в подавляющем большинстве случаев.

Мы уже рассматривали оператор выхода из цикла Do: Exit Do. Кроме выхода из цикла Do допускаются выходы из подпрограмм, циклов For, аксессоров свойств (в пособии не рассматриваются). Оператор выхода можно рассматривать как переход на оператор, стоящий сразу за конструкцией, из которой производится выход, а для подпрограмм оператор выхода очень похож на оператор return в языках типа C. Общее правило для синтаксиса оператора выхода: Exit *ключевое_слово*. *Ключевое_слово* обозначает конструкцию, из которой производится выход и может быть одним из: Do, For, Sub, Function, Property.

2.5 Процедуры и функции

В отличие от языка C и ему подобных, подпрограммы в VBA делятся на два класса: процедуры и функции, подобно тому, как это сделано в Паскале. Процедура является абстракцией действия, ее основным назначением является некоторый побочный эффект, то есть изменение каких-то переменных в модуле. Функция абстрагирует некоторое вычисление значения, ее основным назначением является вычисление какого-либо значения без изменения состояния переменных в модуле. Процедуры и функции отличаются как по способу объявления, так и по вызову: функции являются частью выражений, а для вызова процедур служит специальный оператор процедуры. Однако, подобно языку C, в VBA можно вызывать функции как процедуры, то есть через оператор процедуры. В этом случае вычисленное значение игнорируется.

Процедуры объявляются следующим образом:

```
Sub имя_процедуры (список_формальных_параметров)  
    группа операторов, представляющая тело процедуры  
End Sub
```

Выход из процедуры осуществляется при достижении оператора завершения процедуры – End Sub. Также можно выходить из процедуры, используя оператор выхода Exit Sub.

Список формальных параметров может отсутствовать. Такие процедуры без параметров называются макросами и играют особую роль в VBA (в случае, если они не объявлены как закрытые – о модулях, видимости и структуре программы см. следующий раздел). Макросы являются «точками входа» в программу на VBA. Они могут быть вызваны пользователем управляемого приложения либо

через меню «Сервис-Макрос-Макросы...», либо посредством «горячих клавиш», привязанных к макросам.

Объявление функции имеет вид:

```
Function имя_ф-ции (список_формальных_параметров) As тип  
    группа операторов, представляющая тело ф-ции  
End Function
```

Как и в процедуре, выход из функции осуществляется либо неявно посредством оператора End Function, либо явно – через оператор выхода Exit Function. Однако, в отличие от процедуры, до момента завершения функции должно быть выполнено хотя бы одно присваивание вида *имя_функции* = *выражение*. Последнее значение, присвоенное таким образом, и будет значением функции. Таким образом, имя функции можно рассматривать как локальную переменную функции, которую нужно обязательно инициализировать до выхода из функции. Тип значения, присвоенного переменной–имени функции, должен быть совместим с типом функции, объявленным с помощью конструкции As *тип* после списка формальных параметров. Так же, как и в случае с переменными, если эта конструкция отсутствует, то по умолчанию принимается тип Variant.

Список формальных параметров очень похож на объявление переменных, только без заглавного слова Dim: он представляет собой перечень разделенных запятыми объявлений параметров:

```
имя1 as тип1, имя2 as тип2, . . .
```

Как всегда, отсутствие конструкции As *тип* означает As Variant.

Перед объявлением параметра может стоять спецификатор способа передачи параметра. Спецификатор ByVal означает, что параметр передается по значению, то есть значение фактического параметра копируется в некоторую локальную переменную, обозначающую формальный параметр. Поэтому любые изменения формального параметра происходят с локальной копией и не отражаются на значении фактического параметра. Такой способ полностью аналогичен способу передачи параметров в языке С и передаче параметров-значений в языке Паскаль. Спецификатор ByVal означает, что параметр передается по ссылке, то есть копируется не значение фактического параметра, а ссылка на него. Поэтому любое изменение значения формального параметра автоматически означает изменение и фактического. Спецификатор ByVal принят по умолчанию. Для чего же нужно использовать

передачу параметров по значению? В случае типизированных параметров простых типов, которые не меняют своих значений, передача по значению может быть более эффективной. В остальных случаях, например, когда параметр должен изменяться, а также, когда эффективность не играет существенного значения, следует использовать передачу по ссылке (можно вообще не указывать спецификатор). Пример функции `score_to_str` рассмотрен в предыдущем разделе.

Первоначально вызовы функций и процедур в языке Basic синтаксически различались. Вызов процедуры `SubrFoo` с тремя аргументами выглядел так:

```
SubrFoo x, y-1, 2
```

В то же время вызов функции содержал круглые скобки:

```
let s = score_to_str(24)
```

Кроме того, так же, как в языке Паскаль и ряде других, нельзя было вызывать функцию как процедуру (то есть вне контекста выражения)

```
score_to_str(24) 'раньше – ошибка, теперь можно
```

В текущей версии VBA таких ограничений нет, можно вызывать функции как процедуры, в этом случае возвращаемое значение игнорируется. Также можно указывать круглые скобки при вызове процедуры: `SubrFoo(x, y-1, 2)`. Однако старый синтаксис для вызова процедур (и функций как процедур) сохраняется. Что точно нельзя делать и сейчас, так это вызывать функцию без скобок в контексте выражения:

```
s = score_to_str 24 'синтаксическая ошибка
```

При вызове подпрограмм типы фактических параметров должны соответствовать типам формальных. Транслятор проверяет это ограничение и выдает диагностическое сообщение при попытке запустить макрос с ошибкой.

2.6 Модули и структура проекта

Проект VBA состоит из модулей. В проекте есть один главный модуль и произвольное количество именованных модулей, модулей-классов и форм (специализированный модуль для описания интерфейса пользователя). Добавить модуль в проект можно посредством опции меню «Insert»-«Module/Class Module/User Form» в редакторе VBA. В интегрированной среде разработки VBA (которая вызывается из обычного меню управляемого приложения через опцию «Сервис-Макрос-Редактор Visual Basic»), а точнее – в обозревателе проектов

(Project Explorer) можно увидеть доступные проекты. Структура доступных проектов зависит от управляемого приложения. Например, для MS Word в каждом открытом документе есть свой проект, причем проекты для разных документов не доступны друг другу, то есть нельзя воспользоваться макросами одного документа из другого. Однако шаблон документа также имеет свой проект, и проект документа имеет доступ к проекту шаблона, на основе которого создан этот документ. Полное рассмотрение взаимодействия разных проектов в приложениях MS Office выходит за рамки настоящего пособия, так что мы сосредоточимся на вопросах видимости и доступа в пределах одного проекта.

Каждый модуль состоит из двух частей – общей (или декларативной) части и описания процедур и функций. В общей части описаны переменные, константы и структуры, видимые всему модулю или всему проекту. Видимость определяется спецификатором видимости, который указывается непосредственно перед объявлением сущности (переменной, константы, подпрограммы и так далее). В VBA существует два спецификатора: `public` (общий) и `private` (закрытый). Общий спецификатор означает, что объявленное имя видимо во всем проекте (его можно использовать в любом модуле проекта). В MS Word сущности, объявленные со спецификатором `public` в проекте шаблона, доступны также во всех проектах документов, созданных на основе этого шаблона. Заметим, что употребление спецификатора для сущностей из общей части синтаксически заменяет слово `Dim`:

```
Public i as Integer, s As String 'Dim уже не надо
```

Таким образом, слово `Dim` необходимо употреблять только для объявления локальных переменных и констант подпрограмм, в остальных случаях лучше использовать спецификатор видимости. Если же спецификатор опущен (например, перед объявлением подпрограммы), то по умолчанию устанавливается `private`.

```
Sub Foo 'эта процедура недоступна из других модулей
```

```
Public Function bar (v) As String 'видима везде в проекте
```

```
Dim MAXBUF as integer 'закрыта
```

К сожалению, не любые сущности могут быть открытыми. Так, константы, переменные-массивы или строки с фиксированной длиной не могут быть общими. Другими словами, открытыми (общими для всего проекта) могут быть простые переменные, динамические строки, подпрограммы.

Раздел описаний подпрограмм, как и следует из его названия, состоит только из описаний процедур и функций. Попытка добавить описание переменной или константы между описаниями подпрограмм вполне допустима, но редактор все равно перенесет эти описания в декларативный раздел.

Порядок описаний не имеет значения, это касается как общей части, так и описаний подпрограмм.

2.7 Обработка ошибочных ситуаций

В процессе функционирования любой программы возникают ошибки (аварийные ситуации), при которых невозможно продолжать выполнение программы обычным образом, а необходимо каким-либо образом отреагировать на ошибку с тем, чтобы последствия ошибки минимальным образом отразились бы на функционировании программы. В языке VBA принят способ реакции на ошибку, который называют локальным (как иногда говорят, принцип «ремонта на месте»). Его идея состоит в том, что, во-первых, реакция на ошибку всегда находится в той же подпрограмме, что вызвала ошибку, и, во-вторых, что после выполнения реакции на ошибку можно либо выполнить еще раз оператор, вызвавший ошибку, либо проигнорировать этот оператор и продолжить выполнение со следующего оператора, либо перейти на какой-либо другой оператор в этой же подпрограмме. Обычно в языках программирования ошибки, на которые можно реагировать программным образом, называют исключительными ситуациями (ИС).

Рассмотрим подробнее четыре аспекта обработки исключительных ситуаций и их реализацию в VBA:

- определение ИС;
- возникновение ИС;
- распространение ИС;
- реакция на ИС.

Определение ИС

В VBA с каждой ИС связано некоторое уникальное целое значение – код ошибки, однозначно характеризующий ошибку. Виртуальная машина VBA определяет некоторое количество встроенных ошибок, которые могут происходить при интерпретации программы на VBA, например, 6 – это переполнение (*overflow*), 13 – несоответствие типов при присваивании или передаче параметров подпрограммы и так далее. Возникшая при выполнении программы

ИС описывается специальным уникальным объектом `Err`. Свойства этого объекта описывают информацию о произошедшей ошибке:

- `Number` – код ошибки
- `Description` – строка с кратким словесным описанием (на английском языке)
- `Source` – строка, описывающая место, где произошла ошибка, например, проект, подпрограмму, номер строки и т.п.
- `HelpContext`, `HelpFile` – информация, нужная для вызова подсказки об ошибке (если, конечно, такая информация доступна в виде файла в специальном help-формате)

Кроме свойств объект `Err` обладает двумя методами: `Clear` (без параметров) для сброса ИС и всех свойств объекта `Err`, и методом `Raise` для возбуждения ИС, определяемой пользователем-программистом (об этом методе см. ниже).

Возникновение ИС

ИС либо возбуждается интерпретатором языка (заметим, что синтаксические ошибки к ИС не относятся, так как возникают ДО выполнения программы), либо возбуждаются программистом с помощью метода `Err.Raise`:

```
Err.Raise Number, Source, Description, HelpFile, HelpContext
```

Параметры метода – значения, присваиваемые одноименным свойствам объекта `Err`. Все параметры, кроме первого, необязательные. Пример:

```
Err.Raise 10001, "sub ProcessUserInput", "Ошибка ввода числа"
```

Распространение ИС

Распространение ИС тесно связано с понятием стека вызовов подпрограмм.

Пусть исключительная ситуация (неважно, встроенная или пользовательская) возникла в некоторой подпрограмме А, которая была вызвана подпрограммой В, которая была вызвана подпрограммой С и так далее до самой первой подпрограммы Х, которая была активирована (макросом или обработчиком события).

Если в А отсутствует установленная реакция на ИС (реакция устанавливается оператором `On Error` – о нем мы будем говорить чуть ниже), то ИС **распространяется** на подпрограмму В, вызвавшую А. Если в В также нет установленной реакции, то ИС распространяется на подпрограмму С, и так далее до тех пор, пока либо в стеке вызовов не найдется подпрограмма с установленной реакцией, либо ИС распространится за пределы самой первой

вызванной подпрограммы X (в этом случае реакция не была установлена нигде в цепочке вызовов).

В первом случае срабатывает реакция на ошибку, которая зависит от формы проработавшего оператора On Error – см. ниже. Во втором случае интерпретатор выдает сообщение об ошибке (использующее информацию из свойств объекта Err), после которого можно либо завершить выполнение программы, либо перейти в режим отладки (это позволяет, в частности, увидеть оператор, вызвавший ошибку).

Реакция на ИС

Реакция на ИС устанавливается с помощью оператора On Error:

```
On Error реакция
```

Выполнение этого оператора приводит к установке реакции на ИС. После его выполнения любая исключительная ситуация (либо возбужденная интерпретатором, либо программистом с помощью Err.Raise) приводит к выполнению реакции.

Реакция имеет три вида:

```
On Error Goto 0  
On Error Resume Next  
On Error Goto метка
```

Первый вид реакции – это снятие установленной перед этим реакции. После выполнения такого оператора восстанавливается реакция по умолчанию (выдача сообщения с последующим завершением, либо отладкой). Вторая реакция устанавливает игнорирование ошибки: оператор, вызвавший ошибку, игнорируется, и выполняется следующий за ним оператор. Наконец, третья реакция имеет наиболее общий характер и позволяет установить программный обработчик ИС. После выполнения оператора On Error Goto *метка* возбуждение любой ИС приводит к выполнению оператора, помеченного соответствующей меткой. С точки зрения структуры подпрограмма с обработкой ошибок обычно выглядит так:

```
Sub Foo (аргументы)  
группа операторов, в которой ИС не обрабатывается  
On Error GoTo ErrorHandler  
группа операторов, в которой ИС обрабатывается  
Exit Sub  
ErrorHandler:  
группа операторов, обрабатывающая ошибку  
End Sub
```

Обработка исключительной ситуации завершается либо при достижении конца подпрограммы (или выполнении оператора выхода из подпрограммы – Exit), либо при выполнении специального оператора возобновления нормального выполнения программы – оператора Resume. Этот оператор указывает, какой именно оператор подпрограммы будет выполнен после завершения обработки ИС. Он имеет три формы:

```
Resume  
Resume Next  
Resume метка
```

Первая форма требует повторить оператор, вызвавший ошибку, вторая – игнорирует этот оператор и требует выполнения оператора, следующего за ним. Третья форма означает переход на оператор, помеченный меткой из оператора Resume. В силу локальности оператора перехода эта метка обязана быть объявлена в текущей подпрограмме.

Приведем пример функции, которая требует (и добивается) от пользователя ввода числа. Проблема в том, что функция ввода информации от пользователя (InputBox) позволяет ввести произвольную строку (не обязательно числовую). В случае, если эта строка – не число, то попытка присвоить имени функции результат ввода вызывает ошибку несоответствия типов (код – 13), которая приводит к переходу на обработчик ошибки. Обработчик выдает сообщение об ошибке и требует ввести информацию снова, пока не будет введено нужное число.

```
Function InpNumber(prompt As String) As Double  
On Error GoTo ErrorHandler  
InpNumber = InputBox(prompt, "Ввод числа", 1)  
Exit Function  
ErrorHandler:  
If Err.Number <> 13 Then 'не ошибка преобразования  
    InpNumber = -1  
    Exit Function  
Else  
    MsgBox "Ошибка ввода числа - попробуйте еще раз..."  
    Resume  
End If  
End Function
```

Обсуждение альтернативных подходов к обработке ошибок в программах можно найти в [3].

3 Понятие об объектной модели приложения

3.1 Объекты, их методы и свойства

Объект – ключевое понятие любого языка управления приложениями. С помощью объектов выражаются сущности как управляемого приложения, так и других приложений и сервисов. Объектная модель VBA похожа на модели объектно-ориентированных языков типа C# или Java и, фактически, является их подмножеством, поэтому в настоящем пособии мы не будем останавливаться на общем понятии объекта, а сосредоточимся на особенностях VBA.

Каждый объект VBA принадлежит какому-либо классу. Классы делятся на внутренние и внешние. Внутренние классы определяются на самом языке VBA в модулях-классах проекта. Внешние классы определяются в библиотеках классов. Имя внешнего класса выглядит как: `имя_библиотеки_классов.имя_класса_в_приложении`

Каждое приложение MS Office является библиотекой классов (точнее, содержит в себе библиотеку классов). Примеры классов: `Word.Application`, `Word.Document`, `Excel.Worksheet` и т.д.

По умолчанию всегда доступна библиотека классов управляемого приложения, будем называть ее текущей библиотекой. Кроме нее доступны зарегистрированные в системе библиотеки. Можно создавать объекты (экземпляры классов) как из текущей, так и из любой зарегистрированной библиотеки. Кроме того, текущая библиотека может предоставлять ряд уже существующих объектов, представляющих собой сущности управляемого приложения. Так, в любой объектной модели приложения MS Office существует (в единственном экземпляре) объект `Application`, принадлежащий классу `имя_приложения.Application`, например, `Excel.Application`. Транслятор VBA при поиске имени переменной вначале пытается отыскать его в таблице имен локальных переменных текущей подпрограммы, далее – в таблице глобальных имен текущего модуля, далее – в таблице глобальных публичных имен других модулей проекта. Если имени нет и там, то транслятор пытается отыскать его среди свойств объекта `Application`. Создание объектов из библиотек классов описано в следующем разделе.

Каждый объект обладает набором общедоступных методов и свойств. Кроме того, объекты внешних классов обладают аппаратом отклика на события, которые играют ключевую роль в привязке макросов к функциям управляемого приложения. События описаны в

нижеследующих разделах этой главы. Внутренние классы не имеют аппарата отклика на события.

Метод – процедура или функция, привязанная к классу или экземпляру класса. Методы VBA аналогичны функциям-членам классов в языках C# и Java. На них распространяются те же правила и ограничения, что и на подпрограммы VBA. Основное отличие состоит в том, что методы всегда вызываются через ссылку на объект с использованием операции "точка":

```
ссылка_на_объект.имя_метода (список_фактических_аргументов)
```

Как и в случае общих подпрограмм, скобки при вызове методов могут быть опущены.

Свойства объектов очень похожи на поля структур. С точки зрения программиста, использующего классы, свойства и выглядят как поля-данные объекта, аналогичные полям структурного типа. Существенное отличие свойств от обычных полей-данных состоит в том, что свойства могут быть ограничены с точки зрения чтения или записи, то есть некоторые свойства могут быть только считаны, а некоторые (такие встречаются значительно реже) допускают только изменение, но не чтение своих значений.

Рассмотрим фрагмент программы, который создает новый документ в MS Word. Предполагается, что переменная oWord ссылается на объект класса Word.Application.

```
Set oWordDoc = oWord.Documents.Add
```

Здесь сначала происходит чтение свойства Documents объекта oWord. Это свойство, доступное только для чтения, содержит ссылку на объект-коллекцию открытых документов MS Word. Далее происходит вызов метода-функции Add полученного объекта (скобки опущены), который создает новый объект-документ и добавляет его в коллекцию. Метод Add возвращает созданный объект, который присваивается переменной oWordDoc.

3.2 Объявление, создание и уничтожение объектов

Как уже говорилось в главе 2, в VBA есть специальный объектный тип Object, значением которого являются ссылки на объекты, которые создаются в динамической памяти. Таким образом, объявление переменной oWord создает ссылку на объект, однако самого объекта еще нет:

```
Dim oWord As Object
```

Создать внешний объект можно с помощью функции CreateObject, аргументом которой служит строка с именем класса

создаваемого объекта. Эта строка имеет вид: «имя_приложения.имя_класса_в_приложении». Имя приложения определяет, какую именно библиотеку классов надо загружать, а имя класса – какой именно класс из библиотеки нужно выбрать. Например, следующий оператор создает объект класса Word.Application. Как и следует из наименования класса, при создании экземпляра этого класса запускается приложение MS Word. Если это приложение уже запущено, то созданный объект ссылается на уже запущенный экземпляр приложения. Далее в приложении создается новый документ, и туда вводится текст «Hello, Word!»:

```
Set oWord = CreateObject("Word.Application")
```

' обратите внимание на Set!

```
Dim oWordDoc As Object
```

```
Set oWordDoc = oWord.Documents.Add
```

```
oWord.Selection.TypeText "Hello, Word!"
```

Новый документ можно создать и непосредственно без использования класса приложения (Application). В примере ниже создается объект класса Word.Document. При этом открывается новое окно MS Word (в случае необходимости приложение запускается). В новом окне в начальную позицию вводится текст «Hello, Word!»:

```
Dim oWordDoc As Object
```

```
Set oWordDoc = CreateObject("Word.Document")
```

```
oWordDoc.Range(0,0).Text = "Hello, Word!"
```

Однако есть еще один способ объявления и создания объектов – через явное объявление класса и операцию new. Оказывается, вместо общего типа Object можно использовать имя класса (в том виде, как оно указано в вызове функции CreateObject, только без кавычек), которое воспринимается как имя типа. Для создания объявленных таким образом объектов используется операция new. Приведенные выше примеры можно переписать так:

```
Dim oWord As Word.Application, oWordDoc As Word.Document
```

```
Set oWord = New Word.Application
```

```
Set oWordDoc = oWord.Documents.Add
```

```
oWord.Selection.TypeText "Hello, Word!"
```

или еще короче:

```
Dim oWord As New Word.Application, oWordDoc As Word.Document
```

```
Set oWordDoc = oWord.Documents.Add
```

```
oWord.Selection.TypeText "Hello, Word!"
```

Различия между двумя способами объявления, на первый взгляд, кажутся незначительными, но, на самом деле, они весьма

существенны. Эти различия касаются времени связывания свойств и методов с экземплярами класса, или, другими словами, времени связывания объекта с его конкретным типом данных. Если объект объявлен как Object, то говорят о позднем (или динамическом) связывании, а если через имя класса, то – о раннем (или статическом) связывании.

Дело в том, что тип Object, кроме операции присваивания Set, реально поддерживает только одну операцию – точку (".") – операцию доступа к методу или свойству. Эта операция состоит из двух этапов: сначала запрос на получение свойства или метода с заданным именем и характеристиками (чтение/запись, тип и т. д.), а затем, при успешном ответе на запрос – выполнение метода, чтение или запись свойства. Поэтому при позднем связывании при выполнении, например, оператора MsgBox oWord.Caption интерпретатор запрашивает объект oWord, поддерживает ли он свойство Caption, доступное для чтения. Если свойство поддерживается, то выполняется операция получения значения, и это значение передается функции MsgBox. Если же запрос не удовлетворяется, то интерпретатор возбуждает исключительную ситуацию, связанную с неверным свойством или методом (см. главу 2 – раздел об обработке ошибок во время выполнения). Если эта ситуация не обрабатывается, то выдается сообщение об ошибке, и интерпретатор либо завершает выполнение макроса, либо переходит в режим отладки. Например, ошибка возникнет при попытке обратиться к свойству Selection экземпляра объекта "Word.Document", так как документ не обладает таким свойством (им обладает объект класса "Word.Application"):

```
Dim oWordDoc As Object
Set oWordDoc = CreateObject("Word.Document")
oWordDoc.Selection.TypeText "Hello!"
'ошибка при выполнении операции oWordDoc.Selection
```

В случае раннего связывания транслятор VBA направляет запрос о свойстве или методе не конкретному объекту (ведь при трансляции объекты еще не созданы), а библиотеке классов, которая определяется из имени класса в объявлении объекта. Поэтому ошибки доступа к неподдерживаемому свойству или методу выявляются при трансляции до начала выполнения макроса:

```
Dim oWordDoc as new Word.Document
oWordDoc.Selection.TypeText "Hello!"
'ошибка при трансляции
```

Кроме удобства ранней диагностики ошибок статическое связывание более эффективно во время выполнения, поэтому его рекомендуется использовать везде, где это возможно. Отметим, что раннее (или позднее) связывание определяется объявлением объектной переменной, а не способом ее создания (через new или CreateObject). Поэтому присваивание статически связанного значения в переменную типа Object "теряет" информацию о типе:

```
Dim obj As Object, sel As Object
Dim oWordDoc As New Word.Document
Set obj = oWordDoc 'допустимо
Set sel = obj.Selection 'ошибка во время выполнения
Set sel = oWordDoc.Selection 'ошибка трансляции
Set oWordDoc = CreateObject("Word.Document")
    'создан новый документ
Set sel = oWordDoc.Selection 'ошибка трансляции
```

Также отметим, что присваивание типизированной объектной переменной значения неверного типа недопустимо.

Вышеописанным образом создаются объекты не только из приложений MS Office, но и других зарегистрированных в системе библиотек. Например, для доступа к файловой системе и получение возможности управления окнами и настройками Windows можно использовать библиотеку классов Shell. При выполнении следующего фрагмента программы минимизируются все окна верхнего уровня:

```
Dim oShell As New Shell
oShell.MinimizeAll
```

Рассмотрим теперь уничтожение (освобождение) объектов. В отличие от операций создания объектов, в VBA отсутствует операция их уничтожения. Вместо этого интерпретатор пытается сам отследить и уничтожить неиспользуемые объекты, подобно тому, как он автоматически уничтожает данные из динамической памяти – динамические массивы и строки. Однако, в случае объектов есть некоторые особенности. Для отслеживания неиспользуемых объектов интерпретатор использует счетчик ссылок на этот объект (точнее было бы сказать, что объект сам управляет подсчетом ссылок на себя, а интерпретатор просто вызывает соответствующие методы объекта, но мы не будем вдаваться в подобные тонкости). При создании объекта его счетчик ссылок равен 1. При присваивании объектных переменных Set o1 = o2 счетчик ссылок объекта, на который указывала переменная o1, уменьшается на 1 (если, конечно, o1 на что-либо указывала), а счетчик ссылок объекта, на который

указывала o2, наоборот, увеличивается. Если после присваивания счетчик ссылок какого-либо объекта стал равен 0, то объект уничтожается. Рассмотрим пример с одним объектом и двумя ссылками на него:

```
Sub test_obj()  
Dim doc1 As Object, doc2 As Object  
Set doc1 = CreateObject("Word.Document")  
    'объект создан, счетчик ссылок равен 1  
Set doc2 = doc1 'счетчик ссылок увеличивается на 1  
End Sub 'выход из подпрограммы.  
    'Локальные переменные doc1 и doc2 уничтожаются.
```

При вызове test_obj() создается объект-документ (если MS Word не запущен, то он запускается, затем открывается новое окно документа). Внутри подпрограммы есть две переменных, ссылающихся на этот объект (поэтому его счетчик ссылок равен 2). При выходе из подпрограммы локальные переменные уничтожаются. При уничтожении объектной переменной счетчик ссылок объекта, на который ссылается эта переменная, автоматически уменьшается на 1. Как только счетчик ссылок объекта обнуляется, этот объект уничтожается (окно документа закрывается, при этом возможны диалоги сохранения файла и т.п.). Эффект, аналогичный уничтожению локальной переменной при выходе из подпрограммы, можно достигнуть путем присваивания переменной значения "пустого" объекта. Это значение обозначается специальным служебным словом Nothing.

```
Set doc1 = Nothing 'doc1 указывал на документ.  
    'Счетчик ссылок на этот объект уменьшается на 1  
Set doc2 = Nothing 'то же самое. Счетчик ссылок теперь равен 0,  
    'так что документ уничтожается
```

Можно считать, что интерпретатор автоматически выполняет присваивание Set v = Nothing для каждой локальной переменной v при выходе из процедуры или функции. Ситуация меняется, когда речь идет о глобальных переменных, которые объявлены в декларативном разделе модуля. Эти переменные не уничтожаются автоматически, поэтому программист сам должен контролировать время жизни объектов, на которые ссылаются глобальные переменные. В случае, если объект уже не нужен, то всем переменным, которые ссылаются на него, необходимо присвоить значение Nothing. Поэтому следует крайне осторожно подходить к определению глобальных объектных переменных.

В заключение отметим, что описанная схема уничтожения объектов работает одинаково как для позднего (как в примерах выше), так и для раннего связывания.

3.3 События и событийно-ориентированная архитектура

Рассмотрим подробнее, как осуществляется вызов программ, написанных на VBA, из управляемого приложения. Первый и самый простой способ, описанный в главе 2, это явный вызов макроса пользователем. Напомним, что макросом называется любая публичная процедура без параметров. Все макросы доступны для вызова из меню. Кроме того, в меню можно присвоить макросам комбинации "горячих" клавиш, активация которых вызывает макрос. Также можно добавить в меню офисного приложения пункты, связанные с активацией конкретных макросов. Таким образом, достаточно просто добавлять в приложение новые функции, использующие или формирующие содержание текущего документа. Однако, наибольшая гибкость и мощность управляемого приложения достигается тогда, когда для активации программ на VBA используется аппарат событий. Современные прикладные программы представляют собой событийно-ориентированные системы. Событие – такое изменение состояния системы, на которое можно отреагировать программным способом, внося какие-либо изменения в поведение системы. Набор событий, способы установки программной реакции и возможности установленной реакции по модификации поведения системы определяются при разработке приложения и, как правило, являются неизменными. Программную реакцию на возникновение события называют "обработчиком события". Событийно-ориентированные программы представляют собой сложноорганизованный набор взаимодействующих обработчиков разнообразных событий, которые устанавливаются в процессе инициализации приложения и могут динамически изменяться, добавляться, удаляться в процессе функционирования приложения. В частности, пакет MS Office допускает установку обработчиков событий, написанных на VBA, для некоторых видов событий.

Перечень событий в офисных приложениях весьма разнообразен и богат, включает в себя десятки источников событий, то есть подсистем, генерирующих события. В частности, источником событий служит подсистема взаимодействия с пользователем – практически любое действие пользователя с системой: нажатие (или отпускание) клавиши или кнопки мыши, движение курсора мыши

являются событиями, на которые можно установить обработчик события. Также часто используется такой источник событий, как таймер, позволяющий вызывать обработчик через определенные промежутки времени. Каждое приложение дополнительно определяет набор прикладных событий, специфических для приложения, например, запуск и закрытие самого приложения, создание, открытие и закрытие документов и так далее. Полный набор событий определяется объектной моделью приложения и варьируется от одного приложения MS Office к другому.

С точки зрения VBA обработчик события – это процедура, которая вызывается в момент возникновения события, точнее, в момент обработки события приложением. Как уже говорилось, полный набор событий и способов реакции на них определяется приложением. При обработке определенного рода событий приложение может вызвать обработчики, установленные в проекте на VBA. Способ установки события зависит от приложения. Например, в MS Word нужно открыть проект на VBA и выбрать нужное событие из выпадающего списка в верхней панели текстового редактора. Если обработчик уже есть, то редактор позиционируется на нем, если обработчика нет, то он генерируется с пустым телом, но с нужным именем и параметрами. В других приложениях последовательность действий может быть немного другой, но суть остается прежней – с событием связывается единственный обработчик — процедура, набор параметров которой зависит от события. Вид процедуры-обработчика следующий:

```
Private Sub ИмяОбъекта_ИмяСобытия (список_аргументов)  
  группа_операторов (первоначально пустая)  
End Sub
```

Как видно, имя обработчика однозначно определяется событием и именем объекта, с которым связано событие. Рассмотрим пример события, которое генерируется для листа Excel в момент, когда пользователь нажимает правую кнопку мыши на диапазоне ячеек. Строго говоря, в этот момент генерируется два события (на самом деле, даже больше, но они связаны с другими объектами) – BeforeRightClick и AfterRightClick. Отличие между ними состоит в том, что первое вызывается до обработки приложением нажатия кнопки, а второе – после. Первое событие (BeforeRightClick) относится к так называемым отменяемым (discardable) событиям. Обработчики этих событий содержат в списке аргументов логическую переменную (обычно Cancel), и если

в теле обработчика этой переменной присвоено значение True, то обработка группы событий прекращается. Например, при прекращении обработки события BeforeRightClick, событие AfterRightClick не генерируется, и соответствующий обработчик не вызывается.

```
Private Sub Worksheet_BeforeRightClick _  
    (ByVal Target As Excel.Range, Cancel As Boolean)  
    If Target.Count > 1 Then  
        Cancel = True  
    End If  
End Sub
```

Первый аргумент обработчика – диапазон ячеек, на котором пользователь нажал правую кнопку. Приведенный пример игнорирует нажатие пользователем (зачем-то) на выделенный диапазон, состоящий из более, чем одной ячейки, а при нажатии на одиночную ячейку – выполняются стандартный обработчик.

Не все события являются отменяемыми. Например, событие Worksheet_Activate, которое вызывается в том момент, когда пользователь активизирует лист Excel. Обработчик этого события не имеет аргументов и вызывается после того, как лист уже активирован. Ссылка на активированный лист находится в свойстве ActiveSheet объекта Application. Как уже говорилось в первом пункте этой главы, свойство ActiveSheet доступно непосредственно, без необязательного уточнения Application.ActiveSheet.

3.4 Коллекции

При обсуждении массивов в главе 2 мы отмечали, что в VBA значение этой программной конструкции невелико. Дело в том, что в VBA есть мощная и гибкая абстракция динамически изменяющегося набора объектов произвольного типа – классы коллекций.

Объектные модели любого приложения MS Office не используют массивы. Любая последовательность объектов, например, Application.Documents – набор открытых документов MS Word, представляется коллекцией. Основное отличие коллекций от массивов – возможность вставки, добавления и удаления любых элементов коллекции. Кроме того, некоторые коллекции обеспечивают прямой доступ к элементам не только по индексу (как для массивов), но и по другому ключу, в роли которого чаще всего выступает строка. Например, к элементам коллекции Application.Documents можно обращаться как по индексу

документа в коллекции, например `Documents(1)` – ссылка на первый открытый документ (напомним, что префикс "Application." можно опускать), так и по имени открытого документа – `Documents("ЯУП-3-глава.doc")`. Доступ по индексу реализован во всех коллекциях, а по другим ключам – в зависимости от конкретной коллекции, точнее от того, как именно элементы добавлялись в коллекцию.

Библиотека встроенных классов VBA содержит класс `Collection`, в котором есть следующие методы и свойства:

- метод `Add` – добавление объекта в коллекцию
- метод `Remove` – удаление объекта из коллекции
- метод `Items(index)` – возвращает объект с заданным индексом `index` (индекс первого элемента – 1)
- свойство `Count` – количество объектов в коллекции

Опишем подробнее метод `Add`, поскольку он имеет несколько форм. Первая форма – `Add(obj)` – используется для добавления объекта `obj` в конец коллекции. Получить доступ к этому элементу можно только по индексу (начиная с 1). Вторая форма – `Add(obj, key)` – добавляет объект `obj` с заданным уникальным в коллекции строковым ключом `key`. Для доступа к таким элементам можно использовать форму метода `Item(key)`. Заметим, что такая форма часто предпочтительнее, поскольку индекс (номер) элемента может меняться в процессе вставки и удаления элементов коллекции, поэтому работа с элементом через индекс не всегда удобна. Однако, существуют еще формы `Add` с нестандартным синтаксисом, использующие служебные слова `before` и `after`:

`Add(obj, before index)` и `Add(obj, after index)`

Очевидно, первая форма вставляет объект `obj` перед элементом с заданным индексом `index` (так что индекс вставленного элемента становится равным `index`, а индексы следующих элементов увеличиваются на единицу), а вторая – после элемента с заданным индексом `index` (так что индекс добавленного элемента станет равным `index+1`).

Заметим, что при вставке и добавлении элементов лучше либо использовать только индексы, либо только ключи, поскольку добавление элемента по ключу производится так, что элементы в коллекции упорядочены по ключу (для быстроты поиска, выполняемого методом `Item(key)`), поэтому предсказать, как изменятся индексы остальных элементов – затруднительно. Однако,

вполне можно запутать себя (и других), используя обе формы. Так, в before- и after- формах можно вместо целого индекса использовать ключ:

```
myColl.Add someObj, after "Name"
```

Этот оператор добавляет элемент someObj без ключа непосредственно после элемента с ключом "Name" (так что целочисленный индекс добавленного элемента на единицу больше элемента с ключом "Name").

Теперь ясно, что метод Remove имеет две формы – по индексу и по ключу. В первой задается индекс элемента, во второй – ключ. Точно так же, как и Remove, метод Item тоже имеет две формы: с индексом и с ключом. Заметим, что явный вызов Item не обязателен: можно применять операцию индексирования непосредственно к переменной-коллекции. Например, следующие операторы делают одно и то же (предполагая, что имя первого документа в списке открытых есть "Doc1"), а именно: активируют (устанавливают) клавиатурный фокус на окно и делают окно верхним в "стопке" окон) первого документа в списке открытых:

```
Documents.Item(1).Activate  
Documents(1).Activate  
Documents("Doc1").Activate
```

Вспомним, что для последовательного просмотра элементов коллекций (а также массивов) в VBA введен специальный оператор цикла For Each. В примере ниже формируется коллекция открытых документов, автором которых является некто "GIG". Чтобы получить значение свойства "Автор документа" используется еще одна встроенная коллекция – Document.BuiltInProperties, которая содержит значения полей из краткого описания документа (Summary). Для доступа к элементам этой коллекции нужно использовать встроенные константы-ключи, например, wdPropertyAuthor для автора, wdPropertyTitle для заголовка, wdPropertyCompany для компании и т.д.

```
Dim GIGDocs As New Collection, doc As Word.Document  
For Each doc In Documents  
    If doc.BuiltInProperties(wdPropertyAuthor) = "GIG" Then  
        GIGDocs.Add(doc, doc.Name)  
    End If  
Next doc
```

Коллекции, "встроенные" в объектные модели приложений MS Office, расширяют класс Collection в нескольких аспектах. Во-первых, эти коллекции типизированы, то есть содержат элементы

одного типа. Попытка добавить элементы другого типа приводит к выдаче диагностики, что позволяет избежать ряда ошибок. Во-вторых, в силу типизированности коллекции появляется еще одна форма метода Add – без параметров, например:

```
Set oDoc = Documents.Add
```

Эта форма создает новый объект заданного в коллекции типа (в примере – Word.Document), добавляет его в коллекцию и возвращает ссылку на созданный объект. В нетипизированной коллекции такая форма метода невозможна, поскольку непонятно, объект какого именно типа нужно создавать. Заметим, что, кроме добавления документа в коллекцию, этот метод выполняет еще и много другой работы, в частности, открывает окно нового документа и активизирует его. Заметим также, что добавление документа происходит не в конец коллекции, а в начало, так что индексы открытых документов изменяются. Это еще один аргумент в пользу того, что с подобного рода коллекциями лучше работать по ключам, а не по индексам.

4 Основы объектной модели Excel

В объектной модели Excel представлено более 100 объектов [6]. Вершиной иерархии объектной модели Excel является объект Application (Приложение). На следующем уровне иерархии находится объект Workbook (Рабочая книга), который совпадает с файлом рабочей книги Excel. Объект Workbook содержит объекты более низкого уровня, в частности объект worksheet (Рабочий лист). Объект worksheet, в свою очередь, состоит из других объектов, среди которых выделим Range (Диапазон) и т.д. Ниже описаны основные объекты Excel и их часто используемые методы, свойства и события.

Application

Объект Application в MS Excel представляет собой всё приложение Excel и находится на самом верхнем уровне объектной модели. В открытой Excel книге объект Application доступен всегда. При обращении к какому-либо свойству без указания вышестоящего объекта, VBA будет считать, что идет обращение к свойству объекта Application. Например эти строки равнозначны:

```
Application.Workbooks.Add  
Workbooks.Add
```

Методы

DoubleClick	Событие двойного клика мышкой. Worksheet ("Лист1").Activate 'сделать активным первый лист книги Application.DoubleClick 'вызвать метод двойного клика на активную ячейку листа
FindFile	Позволяет пользователю открыть файл с помощью стандартного диалога. В случае успешного открытия файла возвращает TRUE, в противном случае FALSE Application.FindFile
OnKey	При нажатии клавиши на клавиатуре вызывает указанную процедуру. Application.OnKey "+^a", "MySub" 'при нажатии SHIFT+CTRL+a вызвать процедуру MySub
Save	Сохранить текущую книгу
Quit	Закреть Microsoft Excel For Each w In Application.Workbooks w.Save 'сохранить все открытые Книги Next w Application.quit 'закреть Excel

Свойства

ActiveCell	Активная ячейка. MsgBox ActiveCell.Value ' вывести диалог со значением текущей активной ячейки
ActiveSheet	Активный Лист MsgBox "The name of the active sheet is " & ActiveSheet.Name ' вывести диалог с названием текущего активного листа
Charts	Коллекция всех диаграмм активной книги Charts("Chart1").ChartTitle.Text = "Прибыль" ' дать название "Прибыль" диаграмме Chart1
Cells	Все ячейки активного листа Excel книги Cells(1, 1).Value = 5 ' изменить значение активной ячейки активного листа текущей книги Excel на 5
Columns	Все столбцы активного или указанного листа книги Excel Selection.Rows.Count ' посчитать количество столбцов в выделенном диапазоне на активном Листе
Range	Ячейка или диапазон ячеек Range(Cells(1, 1), Cells(5, 3)).Font.Italic = True ' у указанного диапазона ячеек сделать шрифт наклонным (курсивом)
Rows	Все строки активного или указанного листа книги Excel Worksheets("Sheet1").Rows(3).Delete ' удалить третью строку листа с именем Sheet1
Selection	Выделенный объект на активном листе Selection.Clear ' удаляет содержимое выделенных ячеек (в том случае, если был выделен диапазон ячеек)
Workbooks	Коллекция всех открытых книг Excel For Each w In Workbooks w.Close savechanges:=True Next w ' сохранить изменения и закрыть все открытые книги Excel
Worksheets	Коллекция Листов активной или указанной Excel книги Set newSheet = Worksheets.Add newSheet.Name = "Отчет" ' в активную Книгу добавить новый Лист с именем "Отчет"

События

SheetBeforeDoubleClick	Возникает при двойном клике на рабочем листе до стандартного обработчика этого события Параметр Sh является листом, на котором был двойной клик, Target - ближайшая к месту клика ячейка, параметр Cancel позволяет отменить стандартный обработчик двойного клика
SheetBeforeRightClick	Возникает при клике правой кнопкой мыши на рабочей книге до стандартного обработчика этого события. Параметры аналогичны SheetBeforeDoubleClick

Коллекция Workbooks и объект Workbook

Объект Workbook представляет собой книгу Excel и находится на уровень ниже относительно объекта Application. Все открытые книги Excel объединены в коллекцию Workbooks.

Методы

Activate	Сделать активной указанную Книгу Excel Workbooks("Отчет.xls").Activate 'активирует книгу с именем "Отчет". Если у этой книги несколько Листов, то активным становится первый.
Add	Создать новую Книгу Excel Dim oWbk As Workbook Set oWbk = Workbooks.Add()
Close	Закреть Книгу. Можно указать следующие параметры: SaveChanges - нужно ли сохранять изменения FileName - под каким именем сохранить файл Workbooks("Отчет.xls").Close SaveChanges:=False 'закреть книгу "Отчет" без сохранения изменений
Open	Открытие существующей Книги Excel Dim oWbk As Workbook Set oWbk = WorkBooks.Open("C:\mybook1.xls")
Save	Сохранить сделанные изменения ActiveWorkbook.Save
SaveAs	Сохранить Книгу под другим именем. Можно указать следующие параметры: Filename - под каким именем сохранить Книгу FileFormat - какого формата будет сохраняемый файл ActiveWorkbook.SaveAs Filename:="Отчет" 'сохранить книгу под именем "Отчет"

Свойства

ActiveSheet	Активный лист текущей книги Excel MsgBox "The name of the active sheet is" & ActiveSheet.Name ' в диалоге будет показано имя активного листа
Count	Возвращает количество объектов в Workbooks Workbooks.Count
Item	Возвращает Книгу Excel. Параметр Index указывает какую Книгу нужно вернуть Set wb = Workbooks.Item("myaddin.xla")
Saved	Проверить, были ли сделаны какие-то изменения в Книге с момента последнего сохранения. Результат True/False If Not ActiveWorkbook.Saved Then MsgBox "This workbook contains unsaved changes." End If 'показать сообщение, если Книга содержит несохраненные изменения
Worksheets	Коллекция всех Листов Книги For Each ws In Worksheets MsgBox ws.Name Next ws 'поочередно показать названия всех листов активной Книги

События

Activate	<p>Книга, Лист или диаграмма становятся активными.</p> <pre>Private Sub Worksheet_Activate() Range("A1:A9").Sort Key1:=Range("A1"), Order1:=xlAscending End Sub 'если Лист становится активным, диапазон A1:A9 сортируется</pre>
BeforeClose	<p>Книга закрывается, событие происходит до того, как пользователя спрашивают, хочет ли он сохранить изменения. Параметр Cancel позволяет прервать процесс закрытия книги.</p> <pre>Private Sub Workbook_BeforeClose(Cancel as Boolean) If Me.Saved = False Then Me.Save End Sub 'изменения ВСЕГДА будут сохраняться перед закрытием Книги</pre>
Deactivate	<p>Книга, Лист или диаграмма становятся неактивными.</p> <pre>Private Sub Workbook_Deactivate() Application.Windows.Arrange xlArrangeStyleTiled End Sub 'если Книга стала неактивной, упорядочить для просмотра все открытые Книги Excel</pre>
Open	<p>Возникает при открытии Книги</p> <pre>Private Sub Workbook_Open() Application.WindowState = xlMaximized End Sub 'при открытии Книги максимизировать окно Excel</pre>

Коллекция Sheets и объект Worksheet

Следующим объектом в иерархии является объект Worksheet (лист). Все листы книги объединены в коллекцию Sheets.

Методы

Activate	<p>Сделать лист активным</p> <pre>Worksheets("Отчет").Activate 'Лист "Отчет" станет активным</pre>
Add	<p>Создать новый Лист. Параметр Before указывает перед каким Листом Книги нужно добавить новый. Параметр After - после какого Листа. Count указывает, каким по счету должен быть создаваемый Лист. Type - тип Листа (обычный или диаграмма)</p> <pre>Worksheets.Add Count:=2, Before:=Sheets(1)</pre>
Copy	<p>Копирует Листы в Книгу. Параметры Before и After указывают после или до какого Листа произвести копирование.</p> <pre>Worksheets("Sheet1").Copy After:=Worksheets("Sheet3")</pre>
Delete	Удалить Листы
Move	<p>Переместить Листы. Параметры аналогичны методу Add</p> <pre>Worksheets("Sheet1").Move After:=Sheets(Sheets.Count)</pre>
Paste	<p>Вставить содержимое буфера. Параметр Destination указывает на место вставки</p> <pre>Worksheets("Sheet1").Range("C1:C5").Copy ActiveSheet.Paste Destination:=Worksheets("Sheet1").Range("D1:D5")</pre>

Свойства

Cells	<p>Диапазон ячеек листа</p> <pre>Worksheets("Sheet1").Cells(5, 3).Font.Size = 14</pre> <p>' изменить размер шрифта ячейки (5, 3) на 14</p>
Columns	<p>Диапазон всех столбцов листа</p> <pre>Worksheets("Sheet1").Columns(1).Font.Bold = True</pre> <p>' в ячейках первого столбца сделать шрифт жирным</p>
Count	Возвращает количество объектов в Worksheets
Item	Возвращает Лист Excel. Параметр Index указывает какой Лист нужно вернуть.
Range	<p>Ячейка или диапазон ячеек</p> <pre>Worksheets("Sheet1").Range(Cells(1,1), Cells(5,3))._Font.Italic = True</pre> <p>' в указанном диапазоне сделать шрифт курсивом</p>
Rows	<p>Все строки листа</p> <pre>Worksheets("Sheet1").Rows(3).Delete</pre> <p>' удалить третью строку</p>
Visible	<p>Устанавливает видимый ли указанный Лист</p> <pre>Worksheets(1).Visible = False</pre>

События

Activate	<p>Книга, Лист или диаграмма становятся активными.</p> <pre>Private Sub Worksheet_Activate() Range("A1:A9").Sort Key1:=Range("A1"), Order1:=xlAscending End Sub</pre> <p>' если Лист становится активным, диапазон A1:A9 сортируется</p>
BeforeDoubleClick	<p>Возникает после двойного клика на листе, до запуска стандартного обработчика. Параметр Cancel позволяет отменить стандартный обработчик двойного клика. Параметр Target указывает на ближайшую к точке клика ячейку.</p> <pre>Private Sub BeforeDoubleClick(Target, Cancel) ActiveCell.Interior.ColorIndex = 1 Cancel=true End Sub</pre> <p>' покрасить активную ячейку в черный цвет ' и отменить стандартный обработчик события</p>
BeforeRightClick	Аналогично BeforeDoubleClick
Change	<p>Возникает, если ячейка рабочего листа была изменена</p> <pre>Private Sub Worksheet_Change(ByVal Target as Range) Target.Font.ColorIndex = 5 End Sub</pre> <p>' сделать фон измененной ячейки синей</p>
SelectionChange	<p>Возникает, если изменена область выделения</p> <pre>Private Sub Worksheet_SelectionChange(ByVal Target As Range) ActiveCell.Next.Activate End Sub</pre> <p>' сделать активной следующую ячейку</p>

Range

Объект Range представляет собой либо одну ячейку, либо несколько ячеек (которые могут быть несмежными), либо диапазон смежных ячеек, либо набор несмежных диапазонов, либо целый лист. Любые действия с ячейками должны производиться через объект Range.

Методы

Activate	Внутри выделенного диапазона делает ячейку активной. <code>Range("A1:C3").Select 'выделяем диапазон A1:C3</code> <code>Range("B2").Activate 'делаем ячейку B2 активной</code>
Clear	Очистить содержимое ячеек <code>Range("A1:G37").Clear 'очистить содержимое диапазона A1:G37</code>
Select	Выделяет указанный объект <code>Range(Cells(1,1),Cells(9,9)).Select 'выделить указанный диапазон</code>

Свойства

Address	Строка, определяющая адрес ячейки или диапазона <code>Cells(1, 1).Select 'выделить ячейку (1, 1)</code> <code>MsgBox Selection.Address() '\$A\$1</code>
Borders	Возвращает коллекцию рамок диапазона <code>With Worksheets("Sheet1").Range("B2").Borders(xlEdgeBottom)</code> <code>.LineStyle = xlContinuous</code> <code>.Weight = xlThin</code> <code>.ColorIndex = 3</code> <code>End With 'у ячейки B2 сделать нижнюю границу красной и тонкой</code>
Cells	Диапазон ячеек <code>Range(Cells(1, 1), Cells(5, 3)).Interior.ColorIndex=6</code> ' в указанном диапазоне сделать фон ячеек желтым
Characters	Ссылка на конкретный символ в тексте <code>Range("A1").Value = "Мой текст"</code> <code>Range("A1").Characters(1, 1).Font.Color = vbRed</code> ' изменить цвет первой буквы текста на красный
Column	Номер первого столбца указанного диапазона <code>For Each col In Worksheets("Sheet1").Columns</code> <code> If col.Column Mod 2 = 0 Then</code> <code> col.ColumnWidth = 4</code> <code> End If</code> <code>Next col 'установить ширину четных столбцов в 4 пункта</code>
Columns	Столбцы указанного диапазона <code>Range("myRange").Columns(1).Value = 0</code> ' в диапазоне с именем myRange ячейки первого столбца сделать равными 0

ColumnWidth	Возвращает или устанавливает ширину столбца <pre>With Worksheets("Sheet1").Columns("A") .ColumnWidth = .ColumnWidth * 2 End With 'увеличить ширину столбца в 2 раза</pre>
Count	Возвращает количество указанных объектов в диапазоне <pre>Selection.Columns.Count 'количество столбцов в выделенном диапазоне</pre>
Font	Возвращает объект типа Font <pre>Range("A1").Font.Name = "Arial" 'установить в ячейке A1 шрифт Arial</pre>
Hidden	Скрывает или показывает столбец/строку <pre>Worksheets("Sheet1").Columns("C").Hidden = True</pre>
Interior	Позволяет изменять внутреннее форматирование ячейки <pre>Range("A1").Interior.ColorIndex = 8 'изменить цвет заливки ячейки A1 на голубой</pre>
Next	Следующая ячейка <pre>ActiveCell.Next.Activate 'сделать активной следующую ячейку</pre>
Previous	Предыдущая ячейка <pre>ActiveCell.Previous.Select 'выделить предыдущую ячейку</pre>
Range	Диапазон ячеек <pre>Worksheets("Sheet1").Range(Cells(1,1), Cells(5,3))._ Font.Italic = True 'установить наклонный шрифт в указанном диапазоне</pre>
Row	Аналогично Column
Rows	Аналогично Columns
RowHeight	Высота строки. Аналогично ColumnWidth
Value	Значение ячейки <pre>For Each c in Worksheets("Sheet1").Range("A1:D10") If c.Value < ,001 Then c.Value = 0 End If Next c 'если значение ячейки меньше 0,001, заменить его на ноль</pre>
Worksheet	Лист, на котором находится активный объект <pre>MsgBox ActiveCell.Worksheet.Name 'показать имя Листа, на котором находится активная ячейка</pre>

5 Задания практикума

Семинар 1

Базовый синтаксис и основные конструкции языка VBA. Свойства простейших объектов приложения Microsoft Excel. Изменение формата ячеек с помощью макросов [5].

Материалы семинара:

Конструкции языка VBA: For ... Next, If ... Then ... Else

Свойства объекта ячейка:

Cells(i, j).Value – значение ячейки

Cells(i, j).Font – шрифт текста ячейки

Cells(i, j).Font.ColorIndex – цвет шрифта

Cells(i, j).Interior.ColorIndex – цвет фона

Свойство ColorIndex может принимать значения от 1 до 56, если у ячейки отсутствует фон, то значение ColorIndex равно – 4142.

Задание семинара:

Написать макрос, который в столбец А заносит числа от 1 до 56, цвет шрифта (ColorIndex) должен совпадать с записанным числом. Соседнюю ячейку в столбце В требуется залить цветом с этим же ColorIndex (см. Рисунок 1).

Пример кода:

```
Sub ttt()  
  For i = 1 To 56  
    Cells(i, 1).Value = i  
    Cells(i, 2).Interior.ColorIndex = i  
    Cells(i, 1).Font.ColorIndex = i  
  Next i  
End Sub
```

Рисунок 1

Задание для самостоятельной работы:

Модифицировать написанный макрос таким образом, чтобы пара столбцов цифра-цвет располагалась в прямоугольнике высотой 10 и шириной 12 (см. Рисунок 2).

Рисунок 2

Семинар 2

Диалоги, создание форм. Элементы управления, их возможности и свойства. Обработка событий [2, 4].

Материалы семинара:

Элементы управления: Label, TextBox, CheckBox, RadioButton, Frame, CommandButton, ScrollBar, Image.

Активная ячейка и ее свойства:

ActiveCell – текущая активная ячейка

ActiveCell.Value – значение текущей активной ячейки

ActiveCell.Interior.ColorIndex – цвет фона активной ячейки

ActiveCell.Next.Activate – сделать активной следующую ячейку

ActiveCell.Row – номер строки активной ячейки

ActiveCell.Column – номер столбца активной ячейки

Cells(1,1).Activate – сделать ячейку (1,1) активной

Последовательность псевдослучайных чисел, их использование в программе. Randomize n, Randomize Second(Now), Col = Int((Rnd * 56) + 1)

Задание семинара:

Написать макрос, который выводит форму с тремя кнопками (см. Рисунок 3).

Кнопка **Покрасить** – красит активную ячейку в синий цвет и переходит к следующей ячейке.

Кнопка **Пропустить** – переходит к следующей ячейке, ничего не делая с текущей

Кнопка **Выход** – заканчивает работу макроса.

Пример кода:

Код макроса:

```
Sub test1()  
UserForm1.Show  
End Sub
```

Код формы:

```
Private Sub CommandButton1_Click()  
ActiveCell.Interior.ColorIndex = 11  
ActiveCell.Next.Activate  
End Sub  
  
Private Sub CommandButton2_Click()  
ActiveCell.Next.Activate  
End Sub  
  
Private Sub CommandButton3_Click()  
End  
End Sub
```

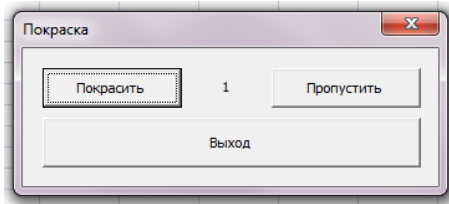


Рисунок 3

Задание для самостоятельной работы:

Изменить макрос, написанный в первом задании следующим образом: ячейки требуется красить не в синий цвет, а в случайный, причем в ячейку требуется вписывать номер этого цвета. Предлагаемый цвет для покраски показывается на форме в элементе Label. При каждом нажатии на кнопку Покрасить или Пропустить случайный цвет меняется (см. Рисунок 4).

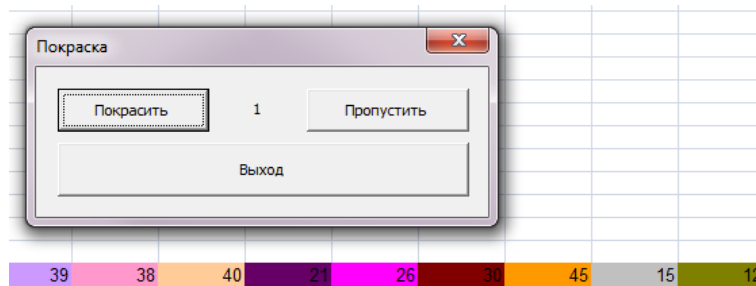


Рисунок 4

Семинар 3

Collection, Range, Selection. Построение гистограмм [2].

Материалы семинара:

Collection. Свойство Count, методы Add, Item, Remove.

Объявление новой коллекции:

```
Dim mycoll As Collection  
Set my coll = New Collection
```

Range, Selection, Cells, их свойства и методы

Цикл For Each ... Next

Построение гистограмм

Стандартное определение:

```
Dim ch As Excel.ChartObject  
Set ch = ActiveSheet.ChartObjects.Add(Left, Top, Width, Height)
```

Добавление категории:

```
ch.Chart.SeriesCollection.NewSeries  
ch.Chart.SeriesCollection(i).Name = i  
ch.Chart.SeriesCollection(i).Values = mycoll.Count
```

Легенда внизу:

```
ch.Chart.SetElement(msoElementLegendBottom)
```

нет подписей по оси x:

```
ch.Chart.SetElement(msoElementPrimaryCategoryAxisWithoutLabels)
```

Массивы. Определение и использование массива. Динамические массивы.

Задание семинара:

Пользователь заполняет ячейки цифрами от 1 до 9. Потом выделяет произвольный фрагмент получившейся таблицы и запускает макрос. Макрос должен в выделенной области пройтись по всем ячейкам, составить коллекцию ячеек, значение которых равно 1 и покрасить их в желтый цвет (см. Рисунок 5).

Пример кода:

```
Sub coll()  
Dim mycoll As Collection  
Set mycoll = New Collection  
For Each c In Selection.Cells  
    If c.Value = 1 Then  
        mycoll.Add c  
        c.Interior.ColorIndex = 6  
    End If  
Next c  
End Sub
```

	A	B	C	D	E
1					
2		4	3	4	3
3		6	3	4	2
4		7	8	9	1
5		1	1	1	2
6					

Рисунок 5

Модифицируем алгоритм следующим образом: ячейки будем красить не в желтый цвет, а в тот цвет, ColorIndex которого равен количеству ячеек с единицей. Рядом построим гистограмму, отражающую количество встретившихся единиц (см. Рисунок 6).

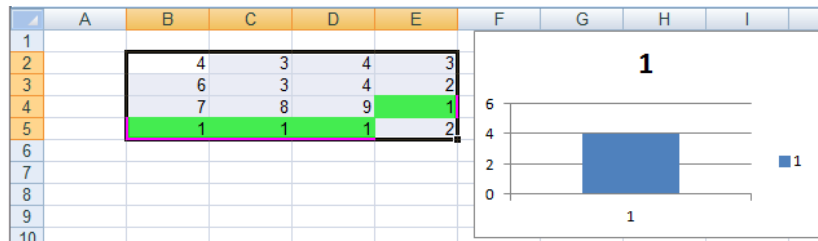


Рисунок 6

Пример кода:

```
Sub coll1()  
Dim mycoll As Collection  
Set mycoll = New Collection  
For Each c In Selection.Cells  
    If c.Value = 1 Then  
        mycoll.Add c  
    End If  
Next c  
For Each c In mycoll  
    c.Interior.ColorIndex = mycoll.Count  
Next c  
  
Dim ch As Excel.ChartObject  
Set ch = ActiveSheet.ChartObjects.Add(250, 1, 250, 150)  
ch.Chart.SeriesCollection.NewSeries  
ch.Chart.SeriesCollection(1).Name = 1  
ch.Chart.SeriesCollection(1).Values = mycoll.Count  
End Sub
```

Задание для самостоятельной работы:

Пользователь заполняет ячейки цифрами от 1 до 9. Потом выделяет произвольный фрагмент таблицы и запускает макрос. Макрос должен в выделенной области Selection пройтись по всем ячейкам и выбрать те, которые содержат цифру 1, и составить коллекцию этих ячеек. С помощью метода count понять, сколько было всего единиц, и покрасить все эти ячейки в ColorIndex, совпадающий с количеством единиц. Аналогично для остальных цифр от 2 до 9. Соответственно, если каких-то цифр в выделенном фрагменте было одинаковое количество, то ячейки с этими цифрами в итоге будут одинакового цвета.

На активном листе построить две гистограммы, показывающие распределение цифр в выделенном диапазоне. В первой гистограмме должно быть 9 категорий, в каждой по одному числу. Во второй гистограмме должна быть одна категория, состоящая из 9 чисел (см. Рисунок 7).

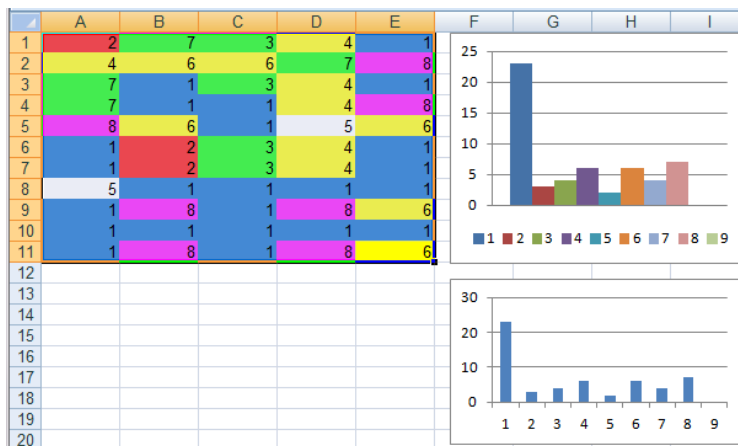


Рисунок 7

Семинар 4.

Реакция на события [8].

Материалы семинара:

События книги (workbook). События листа(worksheet).

Задание семинара:

На первом листе книги при клике на ячейку она меняет цвет следующим образом: из белой в черную, из черной в белую.

Пример кода:

```
Private Sub Worksheet_SelectionChange(ByVal Target As Range)
If ActiveCell.Interior.ColorIndex = -4142 Then
'отдельно рассматриваем случай, когда ячейку еще ни разу не красили
ActiveCell.Interior.ColorIndex = 1
Else
ActiveCell.Interior.ColorIndex = (ActiveCell.Interior.ColorIndex + 1) Mod 2
End If
End Sub
```

Немного изменим задание: пусть на втором листе книги ячейки меняют цвета не только на черный и белый, а добавим туда еще два цвета. Для оптимизации кода создадим процедуру, которую поместим в код объекта ЭтаКнига:

```
Sub ColCh(Num As Integer)
If ActiveCell.Interior.ColorIndex = -4142 Then
ActiveCell.Interior.ColorIndex = 1
Else
ActiveCell.Interior.ColorIndex = (ActiveCell.Interior.ColorIndex+1) Mod Num
End If
End Sub
```

Теперь из первого листа обратимся к функции:

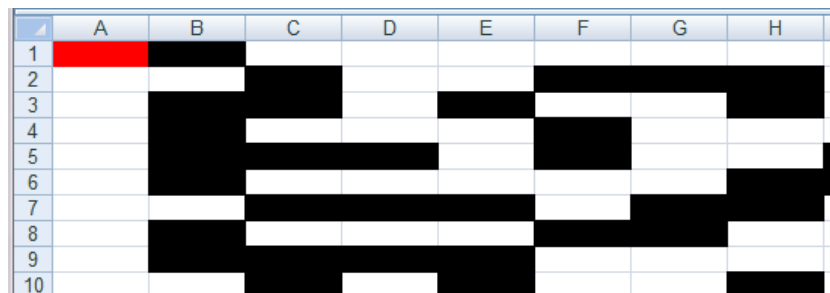
```
Private Sub Worksheet_SelectionChange(ByVal Target As Range)
ЭтаКнига.ColCh 2
End Sub
```

А из второго:

```
Private Sub Worksheet_SelectionChange(ByVal Target As Range)
ЭтаКнига.ColCh 4
End Sub
```

Задание для самостоятельной работы:

В объекте ЭтаКнига написать обработчик событий Workbook_SheetBeforeDoubleClick, который будет отменять стандартный обработчик события двойного клика на ячейке, а вместо этого в текущем активном листе менять цвет ячейки (1,1) циклично от 1 до 56 (см. Рисунок 8).



	A	B	C	D	E	F	G	H
1	Red	Black	White	White	White	White	White	White
2	White	Black	Black	White	White	Black	Black	Black
3	White	Black	White	White	Black	White	White	White
4	White	Black	White	White	White	Black	White	White
5	White	Black	Black	White	White	Black	Black	Black
6	White	Black	White	White	White	Black	Black	Black
7	White	Black	Black	White	White	Black	Black	Black
8	White	Black	White	White	White	Black	Black	Black
9	White	Black	Black	White	White	Black	Black	Black
10	White	Black	White	White	Black	White	White	White

Рисунок 8

Семинар 5.

WORD. Объектная модель. Изменения свойств объектов из макросов [8].

Материалы семинара:

Выделение текста целиком

```
Selection.Start = ActiveDocument.Range.Start
Selection.End = ActiveDocument.Range.End
```

Выделение с 1 по 3 абзац

```
Selection.Start = ActiveDocument.Paragraphs(1).Range.Start
Selection.End = ActiveDocument.Paragraphs(3).Range.End
```

Работа с формой

```
Form.Show
Form.Hide
```

Задание семинара:

Во всем тексте найти слова, начинающиеся на букву “а”. Перед текстом вставить абзац, состоящий из найденных слов. Слова пронумеровать по шаблону: 1:ага 2:аба... .

1 шаг. Собираем коллекцию слов, начинающихся с буквы “а”.

2 шаг. Перед текстом вставляем абзац, в который по заданному шаблону записываем все слова из коллекции.

Пример кода:

```
Sub Searchmini()
Dim coll As Collection
Set coll = New Collection
Selection.Start = ActiveDocument.Range.Start
Selection.End = ActiveDocument.Range.End
For Each w In Selection.Words
    If w.Characters(1) = "a" Then
        coll.Add w
        w.Font.Bold = True
    End If
Next
Selection.Move wdParagraph, -1 'перемещаем курсор в начало текста
Dim i As Integer
i = 1
For Each w In coll
    Selection.TypeText i & ":" & w & " " 'печатаем слова по заданному шаблону
    i = i + 1
Next
Selection.InsertParagraph 'добавляем символ конца абзаца
End Sub
```

Задание для самостоятельной работы:

Во всем тексте найти слова, начинающиеся с букв, заданных пользователем в диалоге (см. Рисунок 9). Перед текстом вставить абзацы, состоящие из найденных слов. Слова пронумеровать по шаблону: 1:ага 2:аба... .

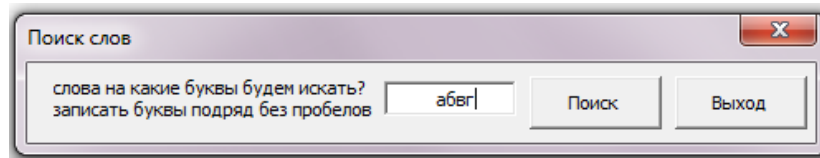


Рисунок 9

Семинар 6.

Поиск и замена с помощью подстановочных знаков [9].

Материалы семинара:

Использование подстановочных знаков для поиска и замены:

?, *, <, >, [], [в-к], [!x-z], {n}, {n,}, {n,m}, @, (), \1 \2, ^2, ^13, ^12, space {1,}.

Использование встроенной функции поиска и замены

Selection.Find, Selection.Find.Replacement

Задание семинара:

Заменяем все предлоги к на латинскую букву k, напишем ее жирным шрифтом и покрасим 4 цветом палитры

```
Sub Zamenak()  
Selection.Start = ActiveDocument.Range.Start  
Selection.End = ActiveDocument.Range.End  
Selection.Find.MatchWholeWord = True 'поиск слова целиком  
Selection.Find.MatchWildcards = True 'поиск по шаблону  
Selection.Find.Text = "к" 'поиск вхождения  
Selection.Find.Replacement.Text = "k" 'заменить с учетом регистра  
' Selection.Find.Replacement.Text = "" 'если текст надо оставить как есть  
Selection.Find.Replacement.Font.Bold = True 'курсив - Italic  
Selection.Find.Replacement.Font.ColorIndex = 4  
Selection.Find.Execute Replace:=wdReplaceAll 'заменять все найденные  
End Sub
```

Задание для самостоятельной работы:

В первых 3 абзацах текста найти все слова, начинающиеся на букву "в", отметить такие слова курсивом и окрасить их в красный цвет.

5.1 Варианты первого практического задания (Microsoft Word)

1. **PCO.** Дан текст в Microsoft Word. Написать макрос, выделяющий (цветом или стилем шрифта) определенные слова исходного текста. В первом абзаце выделяются все слова, начинающиеся с буквы, заданной пользователем, во втором – с буквы, следующей за ней по алфавиту и т.д. В диалоге с пользователем уточняются: 1) тип выделения (различные цвета и/или стиль шрифта), 2) буква алфавита, используемая для выделения слов первого абзаца. Следует помнить, что палитра цветов *ColorIndex* ограничивается 16 цветами, а букв в алфавите 33. Если текст большой, то после 33-й буквы алфавита и после 16-го цвета палитры следует продолжать выделение, используя 1 букву алфавита и 1 цвет палитры.

2. **SB.** Дан текст в Microsoft Word. Написать макрос, который по заданному тексту собирает коллекции слов, начинающихся с одной и той же буквы. В диалоге с пользователем уточняется диапазон букв, используемых при составлении коллекций, например *б-к*. Исходный текст заменяется на новый, в котором в первом абзаце расположены все слова исходного текста, начинающиеся с первой буквы заданного диапазона, во втором – со следующей буквы диапазона и т.д. Например, для диапазона *б-к* первый абзац будет состоять из слов, начинающихся на букву *б*, во втором – на *в*, всего получится 11 абзацев.

3. **PCM.** Дан текст в Microsoft Word. Написать макрос, собирающий коллекции из всех слов, начинающихся с заданных пользователем букв. Результатом работы макроса является исходный текст, в котором выделены (цветом или шрифтом) те слова, коллекция которых оказалась наибольшей. В диалоге с пользователем уточняются: 1) тип выделения (различные цвета и/или стиль шрифта), 2) начальные буквы слов для сбора коллекций.

4. **SC.** Дан текст в Microsoft Word. Написать макрос, собирающий коллекции слов в зависимости от их длины, например коллекция слов длины 7. В диалоге с пользователем задаётся диапазон интересующих его длин, например 3-5 (должна быть предусмотрена возможность выбора «все длины») Исходный текст заменяется на новый, в котором в первом абзаце расположены слова, длина которых равна первому числу диапазона (в нашем примере 3), во втором абзаце слова, длина которых на единицу больше (в нашем примере 4) и т.д.

5. **PZP.** Дан текст в Microsoft Word. Написать макрос, выделяющий (цветом или стилем шрифта) те слова, после которых стоит один из заданных пользователем знаков препинания, а также подсчитывающий количество знаков препинания разного вида. В диалоге с пользователем уточняется тип выделения (различные цвета и/или стиль шрифта), а также выделяемые и подсчитываемые знаки препинания.

6. **PCZ.** Дан текст в Microsoft Word. Написать макрос, выделяющий (цветом или стилем шрифта) те слова, которые написаны с большой буквы, но не являются началом предложения (перед этим словом не стоит точка, вопросительный знак, восклицательный знак или многоточие). В диалоге с пользователем уточняется тип выделения (различные цвета и/или стиль шрифта).

7. **PBL.** Дан текст на русском языке в Microsoft Word. Написать макрос, который находит и выделяет (цветом или стилем шрифта) латинские буквы или цифры, входящие в этот текст. Особо надо выделить слова, целиком состоящие из латинских букв или цифр. В диалоге с пользователем уточняется тип выделения (различные цвета и/или стиль шрифта).

8. **ZMX.** Дан текст в Microsoft Word. Написать макрос, изменяющий случайным образом порядок букв каждого слова, при этом первые N и последние M букв слова остаются неизменными. Например, при начальных параметрах N=2, M=1 слово “случайным” преобразуется в “слйауынчм”. В диалоге с пользователем уточняются параметры N и M.

9. **PBM.** Дан текст в Microsoft Word. Написать макрос, который подсчитывает количество вхождений каждой буквы в текст и выделяет (цветом или стилем шрифта) те слова, в которые входит самая часто встречающаяся буква. В диалоге с пользователем уточняется тип выделения (различные цвета и/или стиль шрифта).

10. **PBI.** Дан текст в Microsoft Word. Написать макрос, подсчитывающий, сколько раз каждая буква, из числа заданных пользователем, встретилась в тексте, и подсвечивающий во всем тексте эти буквы цветом, зависящим от частоты их встречаемости. Например, если какая-то буква встретилась в тексте 14 раз, то она должна быть выделена во всем тексте цветом с $\text{ColorIndex} = 14$, если буква встретилась 18 раз, то цвет должен иметь индекс $18 \bmod 16 = 2$. В диалоге с пользователем уточняется диапазон подсвечиваемых букв, например *б-е*.

11. **ZBZ.** Дан текст в Microsoft Word. Написать макрос, подсчитывающий частоту встречаемости букв в тексте. Те буквы,

частота встречаемости которых превышает некоторое заданное пользователем число, должны быть заменены на аналогичные им большие (заглавные) и выделены (цветом или стилем шрифта). В диалоге с пользователем уточняется порог частоты встречаемости и способ выделения.

12. **ZCF**. Дан текст в Microsoft Word. Написать макрос, находящий все числа в тексте и подсчитывающий значение некоторой функции от этих чисел, а затем увеличивающий найденные числа на подсчитанное значение. Функция выбирается пользователем в диалоге из 6-8 возможных, например: сумма чисел, произведение, минимум, максимум, среднее арифметическое и т.д.

13. **ZCM**. Дан текст в Microsoft Word. Написать макрос, просматривающий заранее выделенный пользователем фрагмент текста и собирающий коллекцию встретившихся в нем букв. Затем во всем тексте выделяются слова, удовлетворяющие заданному пользователем условию, например: выделяются слова, состоящие только из букв коллекции, или слова, в которых есть буквы, не входящие в собранную коллекцию, и т.д. В диалоге с пользователем ему предоставляются на выбор 3-4 условия.

14. **ZAF**. Дан текст в Microsoft Word. Написать макрос, просматривающий заранее выделенный пользователем фрагмент текста и подсчитывающий значение некоторой функции от встретившихся в этом фрагменте цифр. Функция выбирается пользователем в диалоге из 6-8 возможных, например: сумма чисел, их произведение, минимум, максимум, среднее арифметическое и т.д. Подсчитанное значение используется для преобразования исходного текста, оно определяет максимальное число слов в абзаце: слова, выходящие за эту границу, из исходного текста удаляются.

15. **ZAU**. Дан текст в Microsoft Word. Написать макрос, подсчитывающий количество слов в указанном пользователем абзаце и оставляющий в тексте только те абзацы, количество слов в которых удовлетворяет некоторому условию. Условие выбирается пользователем в диалоге из 5-6 возможных, например: количество слов не должно превышать подсчитанное число, количество слов равно подсчитанному числу и др...

16. **SA**. Дан текст в Microsoft Word. Написать макрос, подсчитывающий длину каждого абзаца (количество слов в нем) и сортирующий в исходном тексте эти абзацы в зависимости от их длины. В диалоге с пользователем уточняется способ сортировки, например: по возрастанию количества слов, по убыванию количества слов и т.д.

5.2 Варианты второго практического задания (Microsoft Excel)

При желании можно повысить уровень сложности заданий. Элементы, повышающие уровень сложности, обсуждаются с преподавателем.

Реверси (Отелло) (сложность высокая)

С помощью макроса реализовать в Excel интерфейс игры **реверси**.

Правила игры:

Игра проходит на доске размером 8x8. Играют два игрока обычно чёрным и белым цветом. На каждом ходу игрок имеет право занять свободную клетку – покрасить ее в свой цвет. При этом все клетки соперника, которые находятся на одной линии (по вертикали, горизонтали и диагонали) между новой (только что покрашенной) клеткой и другой его клеткой окрашиваются в его цвет. Допустимым ходом является выбор только такой клетки, при котором закрашивается хотя бы одна клетка соперника.

Например, Вы играете белым цветом, и есть ряд, занятый черными, причем с краю от этого ряда ваше белое поле. Тогда Вы можете занять поле с другого края ряда, и захватить этот ряд чёрных:

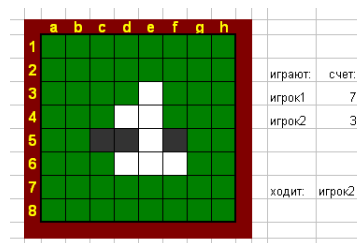


Заметим, что в результате одного хода игрок может окрасить в свой цвет клетки по нескольким направлениям.

На рисунке при ходе чёрных они могут сделать следующие ходы: D3, D7, E7, F2, F3, F7. При ходе чёрных на D3, поля D4 и E4 окажутся захваченными.

Если один из игроков не может сделать свой ход, то ход передаётся сопернику.

Игра заканчивается, когда ни у одного из игроков нет хода. Обычно это происходит, когда поле полностью заполнено. Выигрывает тот, кто окрасил в свой цвет большее количество клеток игрового поля. В случае равенства засчитывается ничья.



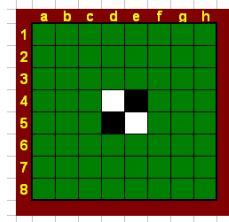
Постановка задачи:

На активном листе должно быть подсвечено игровое поле 8*8, на котором в каждый момент времени отображается текущее состояние игры.

Перед началом игры в диалоге уточняются имена игроков и цвета, которыми они хотят играть. В начале игры

должна быть выставлена начальная игровая позиция (см. рисунок). В течение игры рядом с полем показывается текущий счет и имя игрока, который ходит в данный момент. По окончании игры должно быть выведено сообщение с именем победителя и счетом.

Ходы осуществляются кликом мышкой в ячейки игрового поля. Автоматически проверяется допустимость данного хода.



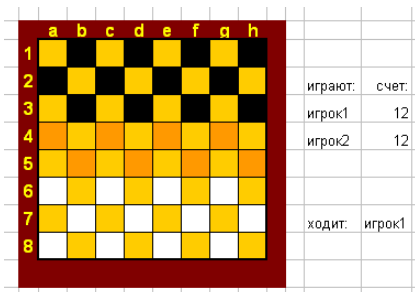
Шашки (сложность высокая)

С помощью макроса реализовать в Excel интерфейс игры **русские шашки**.

Правила игры:

Игра проходит на доске размером 8x8 только на чёрных клетках. Играют два игрока обычно чёрным и белым цветом. У каждого игрока в начальной позиции по 12 фишек — так называемых «простых» шашек, которые занимают первые три ряда с каждой стороны. В ходе игры шашки движутся по черным полям и могут вставать только на незанятые клетки. «Простая» шашка может ходить по диагонали вперёд на одну клетку или бить вперед или назад. При бое шашка движется по диагонали на две клетки, перепрыгивая через шашку или дамку соперника, которая при этом снимается с доски (съедается). Если из нового положения бьющей шашки можно побить другую шашку соперника, то ход продолжается, и т. д. При прохождении любого поля из последнего от игрока горизонтального ряда, «простая» шашка превращается в дамку и продолжает бой по правилам дамки. Дамка может ходить по диагонали в любом направлении на любое число полей. Игрок на своём ходе обязан побить шашку противника, если у него есть такая возможность. Если есть несколько вариантов хода с боем, то игрок выбирает любой из них. Пропуск хода не допускается. Цель игры — съесть или «запереть» (лишить возможности хода) все шашки противника.

Постановка задачи:



На активном листе должно быть подсвечено игровое поле 8*8, на котором в каждый момент времени отображается текущее состояние игры. Вместо фишек используется соответствующая раскраска клеток игрового поля.

Перед началом игры в диалоге уточняются имена игроков и цвета, которыми они хотят играть.

В начале игры должна быть выставлена начальная игровая позиция (см. рисунок). В течение игры рядом с полем показывается текущий счет и имя игрока, который ходит в данный момент. По окончании игры должно быть выведено сообщение с именем победителя.

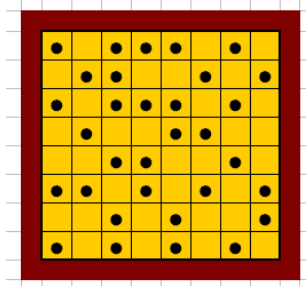
Ходы осуществляются кликом мышкой в ячейки игрового поля: клик в шашку означает ее выделение, клик в пустое поле обозначает ячейку, в которую игрок хочет переместить выделенную шашку. Автоматически проверяется допустимость данного хода.

Ним (сложность низкая)

С помощью макроса реализовать в Excel интерфейс игры **ним**.

Правила игры:

При игре в «ним» два соперника поочередно берут фишки с игрового поля, на котором в начальный момент находится несколько рядов фишек, причем в каждом ряду имеется хотя бы одна фишка. При каждом ходе игрок может взять произвольное количество фишек из одного ряда (но не менее одной). Выигрывает тот игрок, кто берет последнюю фишку.



Постановка задачи:

На активном листе должно быть подсвечено игровое поле, на котором в каждый момент времени отображается текущее состояние игры. До начала игры в диалоге пользователь указывает размер игрового поля. В начале игры случайным образом расставляются фишки, количество которых зависит от размера поля. В течение игры рядом с полем показывается текущий счет и имя игрока, который ходит в данный момент. По окончании игры должно быть выведено

сообщение с именем победителя. Ходы осуществляются кликом мышкой в ячейки игрового поля. Клик в фишку выделяет ее, клик в выделенную фишку отменяет выделение. После выделения игроком фишек, которые он хочет взять на своём ходе, требуется тем или иным образом подтвердить свой ход, после чего автоматически проверяется допустимость этого хода.

Тик-такс (сложность низкая)

С помощью макроса реализовать в Excel интерфейс игры **тик-такс**.

Правила игры в «тик-такс» отличаются от правил игры в «ним» лишь тем, что на каждом ходе можно брать только рядом стоящие фишки одного ряда.

Требование к интерфейсу см. в варианте для игры «ним».

Нижняя левая (сложность низкая)

С помощью макроса реализовать в Excel интерфейс игры **нижняя левая**.

Имеется прямоугольное поле $N \times N$, ($5 \leq N \leq 7$) из клеток, заполненных фишками. Каждый из двух игроков по очереди выбирает фишку из оставшихся на поле и убирает ее вместе со всеми фишками, расположенными правее и выше выбранной. Проигрывает тот, кто забирает фишку, стоящую в нижнем левом углу.

Требование к интерфейсу см. в варианте для игры «ним».

Цветовая экспансия (сложность средняя)

С помощью макроса реализовать в Excel интерфейс игры **цветовая экспансия**.

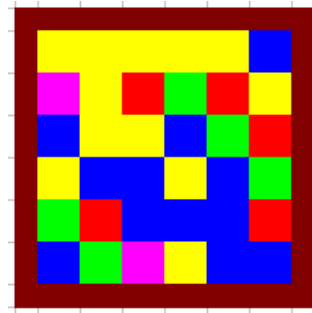
Правила игры:

Игра проходит на поле размером $N \times N$ ($N \geq 2$). В начале игры все клетки поля раскрашены в M цветов ($M \geq 3$) так, что соседние клетки имеют разные цвета. При этом необходимо, чтобы клетки с координатами $(1,1)$ и (N,N) были окрашены в разные цвета. Играют два игрока, строя свои «империи» из углов поля, т.е. клеток с координатами $(1,1)$ первый игрок и (N,N) — второй. В начале игры «империя» каждого игрока состоит из одной угловой клетки. Игроки ходят по очереди. Ход игрока заключается в выборе одного из M цветов, при этом выбранный цвет должен отличаться от текущего цвета игрока (цвета его угловой клетки) и от текущего цвета противника. После того, как цвет выбран, вся «империя» игрока

перекрашивается в выбранный цвет. При этом клетки, граничащие с «империей» игрока и имеющие выбранный на текущем ходу цвет, присовокупляются к ней. После каждого хода подсчитываются очки каждого игрока — размеры их «империй». Игра заканчивается, когда на поле не останется клеток, не входящих в какую-либо империю. Выигрывает тот, чья «империя» к концу игры будет больше. При равенстве объявляется ничья.

Постановка задачи:

Перед началом игры в диалоге уточняются имена игроков, размер поля и количество цветов, после чего в соответствии с правилами игры генерируются начальные цвета ячеек, и на активном листе появляется игровое поле. В течение игры рядом с полем показывается текущий счет и имя игрока, который ходит в данный момент.



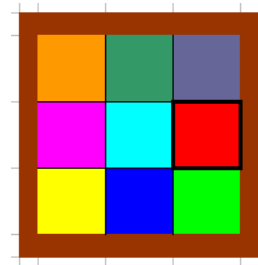
Ходы могут осуществляться кликом мышкой в ячейки, расположенные рядом с игровым полем и раскрашенные цветами, используемыми в игре. После каждого хода игрока проверяется его допустимость. В случае недопустимого хода игроку выдается сообщение об ошибке и предоставляется возможность сделать ход еще раз. Если игрок сделал допустимый ход, то после него происходит смена цвета «империи» и ход переходит к противнику.

Тренировка памяти (сложность низкая)

С помощью макроса реализовать в Excel интерфейс игры **тренировка памяти**.

Правила игры:

Игра проходит на поле размером NxM ($3 \leq N, M \leq 7$). Каждая ячейка игрового поля покрашена в свой цвет, цвета не повторяются. Играет один человек. Компьютер показывает игроку некоторую последовательность ячеек (по очереди выделяет их). Игрок должен запомнить и повторить эту последовательность. После трех ошибок игра останавливается.



Постановка задачи:

Перед началом игры в диалоге уточняется размер поля и длина последовательности K ($K \geq 3$), с которой игрок хочет начать. После этого компьютер случайным образом выбирает K ячеек игрового поля и запоминает их последовательность. Затем демонстрирует ее (сначала жирным обводится первая ячейка, потом выделение убирается, затем вторая и т.д.). После этого ход переходит к игроку. Он должен, кликая по ячейкам, выделять их в той же последовательности, как показывал компьютер. Если это удастся, то ход считается успешным. В этом случае компьютер добавляет еще одну ячейку в конец последовательности и снова демонстрирует ее. Если игрок в свой ход ошибся, то появляется сообщение и компьютер демонстрирует последовательность еще раз, не изменяя ее. Игра останавливается после трех ошибок. Результатом игры является длина последней успешно повторенной последовательности.

Прямоугольники (сложность средняя)

С помощью макроса реализовать в Excel интерфейс игры прямоугольники.

Правила игры:

На поле $N \times M$ ($N \geq 2$, $M \geq 2$) положили K ($K \geq 2$) прямоугольников один поверх другого в случайном порядке. Длины сторон прямоугольников выражаются целым числом клеток. Прямоугольники не выходят за границы поля.

Получившуюся ситуацию записали в таблицу чисел (каждой клетке поля соответствует клетка таблицы). Если клетка поля не закрыта прямоугольником, то в соответствующую клетку таблицы записали число 0. Если же клетка закрыта одним или несколькими прямоугольниками, то в соответствующую клетку таблицы записали число, соответствующее самому верхнему прямоугольнику, закрывающему эту клетку. Два разных прямоугольника могут состоять из ячеек с одинаковыми числами.

По содержимому таблицы требуется определить положение и размеры прямоугольников.

Постановка задачи:

Перед началом игры в диалоге уточняются размер поля и количество

0	2	2	2	2	0	0
0	2	2	2	2	0	0
1	1	2	4	4	4	4
1	1	0	4	4	4	4
1	1	0	0	0	0	0
1	1	0	0	0	0	0

прямоугольников, после чего в соответствии с правилами на игровом поле расставляются цифры.

В течение игры пользователь выделяет прямоугольники, которые, по его мнению, были загаданы компьютером, и, после подтверждения хода, этот прямоугольник убирается с доски. В этот момент требуется проверить, что ход игрока корректен, т.е. выделенные ячейки содержат одинаковые числа. Если это так, то выделенные числа убираются с поля, а освободившиеся ячейки заполняются числами, которые стояли бы в них, если бы убранный прямоугольник перекрывал собой другие числа, теперь они станут видны, если под ним чисел не было, ячейки заполняются нулями). После этого игрок может делать следующий ход. Если ход, сделанный игроком, некорректен, то показывается сообщение об ошибке и дается возможность сделать новый ход. В ходе игры рядом с полем показывается, сколько прямоугольников осталось отгадать. Цель игры убрать с поля все большие нуля цифры.

Игра считается успешно законченной, если игровое поле заполнено нулями, а количество отгаданных игроком прямоугольников не больше количества загаданных компьютером в начале игры.

Найди пару (сложность низкая)

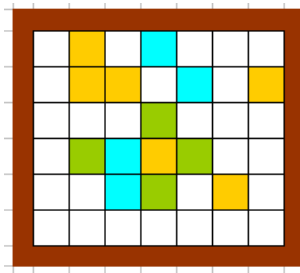
С помощью макроса реализовать в Excel интерфейс игры **найди пару**.

Правила игры:

Игра проходит на поле размером $N \times M$ ($N \geq 2$, $M \geq 2$, M – четное). В начале игры каждой клетке поля случайным образом приписывается свой цвет (но игроку цвета не показываются). Количество полей одинакового цвета – четно. Количество цветов K ($K \geq 2$) определено заранее. Игрок за ход может посмотреть цвета двух ячеек. Если они совпадут, то цвет ячеек сохранится, если нет – то цвет ячеек опять скроется. Задача игрока за минимальное количество ходов открыть цвета всех ячеек.

Постановка задачи:

Перед началом игры в диалоге уточняются размер поля и количество цветов, в которые будут краситься



ячейки. После этого игроку показывается игровое поле белого цвета и предоставляется возможность сделать ход. После каждого хода происходит проверка открытых на этом ходе цветов. Если цвета не совпали, то ячейки опять становятся белыми. В ходе игры рядом с полем показывается, сколько ходов было сделано на текущий момент.

Усложнение задачи (плюс один балл):

Ввести ограничение по времени: пользователю дается 5 минут на игру. Если игрок открыл все поле, а время еще не истекло, то создается новое игровое поле, в котором количество участвующих в раскраске цветов увеличивается на один. При успешном открытии пары время, оставшееся игроку на игру, увеличивается на 10 секунд. При истечении времени игра останавливается. Результат игрока – количество пар, которые удалось отгадать за все время игры.

Прятки (сложность низкая)

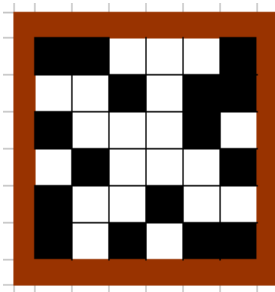
С помощью макроса реализовать в Excel интерфейс игры **прятки**.

Правила игры:

Игра проходит на поле размером NxM ($M, N \geq 6$). Играют двое, ходят по очереди. В начале игры поле пустое. В свой ход игрок может перекрасить одну белую клетку. Клетка считается «видной» по горизонтали, если в одной строке с ней на поле уже есть хотя бы одна перекрашенная клетка. Аналогичным образом клетка может быть «видна» по вертикали и обоим диагоналям. Если игрок в свой ход покрасил ячейку, которая «не видна» по всем четырем направлениям, он получает 4 балла. Если клетка видна по одному направлению, то количество начисленных баллов будет 3, если по двум, то 2, если по трем, то 1. Если клетка «видна» во всех четырех направлениях, то она «плохая», ее перекрашивать нельзя. Игра идет до тех пор, пока все белые ячейки не станут «плохими». Выигрывает тот, кто в процессе игры набрал большее количество баллов.

Постановка задачи:

Перед началом игры в диалоге уточняются имена игроков, размер поля и цвет, которым будут закрашиваться ячейки. После этого игроки делают ходы по очереди. В свой ход игрок кликает на клетку, которую хочет перекрасить. Программа проверяет, не является ли клетка «плохой». Если клетка «плохая», то выдается сообщение и право



хода переходит к сопернику, баллы игроку не добавляются. Если клетку перекрасить можно, то программа добавляет игроку баллы, исходя из правил игры, перекрашивает ячейку, проверяет, остались ли поля, которые могут быть закрашены в следующий ход. Если такие поля есть, то передает право хода сопернику. Если нет, то игра заканчивается, и сообщается имя победителя.

Крестики-нолики (сложность низкая)

С помощью макроса реализовать в Excel интерфейс игры **крестики-нолики**.

Правила игры:

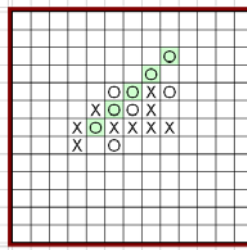
Имеется прямоугольное поле NxN ($10 \leq N \leq 30$). Играют двое, ходят по очереди. Фишки одного игрока – «крестики» (или поля одного цвета), а другого – «нолики» (или поля другого цвета). В начале игры все клетки доски пустые.

При своем ходе игрок ставит свою фишку в любую свободную клетку.

Выигрывает тот, кто первым выстроил 5 своих фишек в (сплошной) ряд по горизонтали, вертикали или диагонали. Возможна и ничья, если никто не выиграл и на доске не осталось пустых клеток.

Постановка задачи:

На активном листе должно быть подсвечено игровое поле, на котором в каждый момент времени отображается текущее состояние игры. До начала игры в диалоге пользователь указывает размер игрового поля. В течение игры рядом с полем показывается имя игрока, который ходит в данный момент. По окончании игры должно быть выведено сообщение с именем победителя. Ходы осуществляются кликом мышкой в ячейки игрового поля. После того, как игрок сделал ход, проверяется, не стал ли он победителем в результате этого хода. Если нет, то ход переходит другому игроку.



Додж (to dodge – увертываться) (сложность средняя)

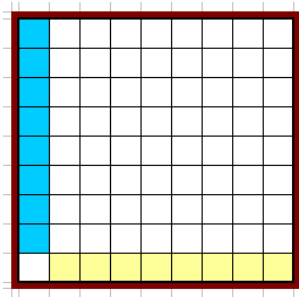
С помощью макроса реализовать в Excel интерфейс игры **додж**.

Правила игры:

Имеется прямоугольное поле NxN ($4 \leq N \leq 10$). Играют двое, ходят по очереди. Один игрок играет желтыми фишками, другой

синими. Вместо фишек используется соответствующая раскраска клеток игрового поля. В начале игры на доске уже выставлено по (N-1) фишке каждого цвета – так, как показано на рисунке.

При каждом ходе желтая фишка может перейти в соседнюю свободную клетку влево, вверх или вправо (но не вниз), а синяя – вверх, вправо или вниз (но не влево). Подойдя к верхнему (правому) краю доски, желтая (синяя) фишка может уйти с доски. Запрещается «запирать» фишки противника (не давать ему ходить); тот, кто сделает это, сразу проигрывает.



Выигрывает тот игрок, который быстрее уведет свои фишки с доски.

Постановка задачи:

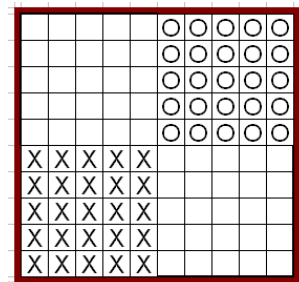
На активном листе должно быть подсвечено игровое поле, на котором в каждый момент времени отображается текущее состояние игры. До начала игры в диалоге пользователь указывает размер игрового поля. В течение игры рядом с полем показывается имя игрока, который ходит в данный момент. По окончании игры должно быть выведено сообщение с именем победителя. Ходы осуществляются кликом мышкой в ячейки игрового поля. После того, как игрок сделал ход, проверяется, не стал ли он победителем в результате этого хода. Если нет, то ход переходит к другому игроку.

Уголки (сложность низкая)

С помощью макроса реализовать в Excel интерфейс игры уголки.

Правила игры:

Имеется прямоугольное поле NxN ($6 \leq N \leq 16$). Играют двое, ходят по очереди. Фишки одного игрока – «крестики» (или поля одного цвета), а другого – «нолики» (или поля другого цвета). В начале игры фишки одного игрока уже выставлены в его «базе» (в квадрате из MxM клеток), расположенной в левом нижнем углу доски, а фишки другого игрока – в другой базе, расположенной в правом верхнем углу (M заранее фиксировано и должно равняться примерно N/2).



При своем ходе игрок может:

- сделать «шаг» – передвинуть любую свою фишку в любую соседнюю (по горизонтали или вертикали) свободную клетку;
- сделать «прыжок» – переставить любую свою фишку через любую соседнюю (свою или чужую) фишку, если за ней находится свободная клетка;
- сделать любое число «прыжков» подряд, если это возможно.

Замечание: нельзя в одном ходе сочетать «шаги» и «прыжки».

Выигрывает тот, кто быстрее переведет свои фишки в базу противника.

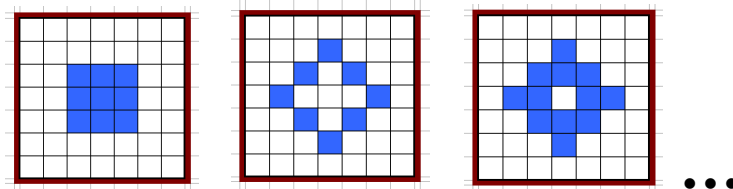
Игра «Жизнь» (сложность средняя)

С помощью макроса реализовать в Excel интерфейс игры **жизнь**.

Место действия этой игры — размеченная на клетки поверхность, ограниченная или замкнутая. Каждая клетка на этой поверхности может находиться в двух состояниях: быть живой или быть мёртвой. Клетка имеет восемь соседей. Распределение живых клеток в начале игры называется первым поколением. Каждое следующее поколение рассчитывается на основе предыдущего по таким правилам:

- пустая (мёртвая) клетка рядом с тремя живыми клетками-соседями оживает;
- если у живой клетки есть две или три живые соседки, то эта клетка продолжает жить; в противном случае (если соседок меньше двух или больше трёх) клетка умирает (от «одиночества» или от «перенаселённости»).

Игрок не принимает прямого участия в игре, а лишь расставляет «живые» клетки, которые взаимодействуют согласно правилам уже без его участия.



Постановка задачи:

На активном листе должно быть подсвечено игровое поле, на котором в каждый момент времени отображается текущее состояние игры. До начала игры в диалоге пользователь указывает размер игрового поля и его тип (замкнутое или нет), после чего ему дается

возможность отобразить на заданном поле первое поколение. В процессе игры пользователь должен иметь возможность просмотра как развития колонии по шагам, так и непрерывного развития на несколько шагов вперед. В случае гибели колонии игра останавливается.

Пятнашки (сложность средняя)

С помощью макроса реализовать в Excel интерфейс игры **пятнашки**.

Правила игры:

Имеется прямоугольное поле $N \times M$ ($2 \leq N, M \leq 8$). Все клетки, кроме одной, заполнены фишками с нарисованными на них числами от 1 до $N \cdot M - 1$. Одно поле остается пустым. Ход игрока заключается в том, чтобы на пустое место сдвинуть одну из соседних с ним фишек. В начале игры фишки располагаются произвольным образом. Задача игрока расставить их по порядку.

Постановка задачи:

На активном листе должно быть подсвечено игровое поле, на котором в каждый момент времени отображается текущее состояние игры. До начала игры в диалоге пользователь указывает размер игрового поля. В течение игры рядом с полем показывается имя игрока и количество сделанных им ходов. Ходы осуществляются кликом мышкой в ячейки игрового поля.

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

Сложность заключается в том, что при произвольной расстановке фишек часть позиций является «неправильными», т.е. неразрешимыми. Пользователю надо давать только правильные позиции. Одним из вариантов решения этой проблемы является составление начальной расстановки фишек из правильно сложенной головоломки путем случайных сдвигов. Количество сдвигов должно зависеть от размеров игрового поля.

Быки и коровы (сложность низкая)

С помощью макроса реализовать в Excel интерфейс игры **быки-коровы**.

Правила игры:

За 10 попыток отгадать число, состоящее из 4-х неповторяющихся цифр, которое автоматически создается при загрузке игры. Ход игрока состоит в том, что он вводит четырехзначное число и после подтверждения хода получает

Постановка задачи:

На активном листе должно быть подсвечено игровое поле, на котором в каждый момент времени отображается текущее состояние игры. Размер игрового поля указывается пользователем в диалоге перед началом игры. Клетки, в которых расположены мины, определяются случайным образом. Рядом с игровым полем показывается, сколько мин осталось обнаружить. Когда количество клеток, помеченных игроком как заминированные, становится равным количеству мин, автоматически проверяется, совпадают ли помеченные клетки с теми, в которых реально расположены мины. Пользователь получает сообщение о результате игры и причинах ее окончания.

Змейка (сложность средняя)

С помощью макроса реализовать в Excel интерфейс игры змейка.



Правила игры:

Игра проходит на прямоугольном поле произвольного размера. Игрок управляет движением змейки на игровом поле. На поле определенным цветом выделена бонусная ячейка. При прохождении змейки через бонусную ячейку длина змейки увеличивается, а бонусная ячейка меняет свои координаты.

При увеличении длины змейки ее скорость увеличивается. Игра заканчивается, когда змейка наткнется на границу игрового поля, или когда она напозаает на саму себя.

Постановка задачи:

На активном листе должно быть подсвечено игровое поле, на котором в каждый момент времени отображается текущее состояние игры. До начала игры в диалоге пользователь указывает размер игрового поля и начальную длину змейки. Начальная длина не должна превышать половину длины или ширины игрового поля. В ходе игры рядом с полем показываются текущая длина змейки и ее скорость. Координаты бонусной ячейки определяются случайным образом. Игрок с помощью клавиатуры изменяет направление движения змейки. При окончании игры пользователю должны быть

показаны итоговая длина змейки, ее скорость и причина окончания игры.

Морской бой (сложность средняя)

С помощью макроса реализовать в Excel интерфейс игры **морской бой**.

	1	2	3	4	5	6	7	8	9	10	
а			•	•	•						а
б			•	X	•			X			б
в			•	X	•			X			в
г			•	X	•						г
д			•	•	•						д
е		•	•	•	•	•	•				е
ж		•	X	•	•	•			•		ж
з		•	•	•							з
и		•						•			и
к											к
	1	2	3	4	5	6	7	8	9	10	

Правила игры:

Игра проходит на поле размером 10x10. До игры на поле расставляются корабли: 1 четырехпалубный, 2 трехпалубных, 3 двухпалубных и 4 однопалубных. При размещении корабли не могут касаться друг друга даже углами. Расстановка кораблей игроку не показывается. Игрок последовательно делает выстрелы по полю (открывая

клетки поля), при этом проверяется, попал ли игрок в корабль. Если да, то клетка помечается крестом, и, если при этом все остальные клетки корабля тоже помечены крестом, то корабль считается подбитым. Если при выстреле корабль не был поражен, то в это поле ставится точка (промах). Цель игры – подбить все корабли за как можно меньшее число выстрелов.

Постановка задачи:

На активном листе должно быть подсвечено игровое поле 10*10, на котором в каждый момент времени отображается текущее состояние игры. До начала игры случайным образом создается расстановка кораблей. Пользователь делает выстрелы кликом на клетках поля. При гибели корабля автоматически вокруг него все поля помечаются точками (промахами), а сам он подсвечивается особым образом. По окончании игры показывается, сколько выстрелов сделал игрок. Желательно указывать рядом с полем во время игры, сколько и каких кораблей осталось подбить.

Охота на лис (сложность средняя)

С помощью макроса реализовать в Excel интерфейс игры **охота на лис**.

Правила игры:

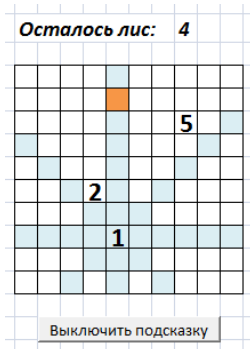
Игра проходит на поле размером 10x10. В пяти клетках поля спрятаны пять лис, по одной в клетке. Цель игры: найти всех лис. Для поиска игрок указывает ячейку поля и получает ответ:

- поле окрашивается рыжим цветом (т.е. лиса найдена)
- в поле появляется цифра от 0 до 5 в зависимости от того, сколько лис видно из этой ячейки.

Лиса считается видной из ячейки, если она находится в одной вертикали, горизонтали или диагонали от нее.

Постановка задачи:

На активном листе должно быть подсвечено игровое поле 10x10, на котором в каждый момент времени отображается текущее состояние игры. До начала игры случайным образом создается расстановка лис. Пользователь делает ходы кликом на клетках поля. При нахождении лисы поле подсвечивается рыжим цветом, в противном случае появляется число видимых лис.



Требуется предусмотреть подсказку: если по текущему состоянию игры можно точно сказать, что в каких-то ячейках лисы быть не может, такие ячейки требуется подсветить другим цветом. Подсказку можно включать и выключать кнопкой, расположенной на игровом поле. По окончании игры показывается, сколько ходов сделал игрок.

Solitaire (сложность низкая)

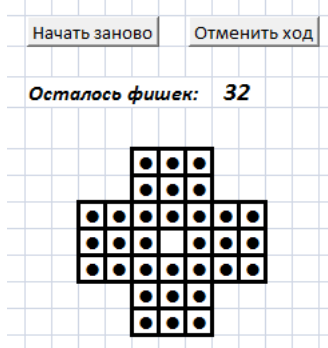
С помощью макроса реализовать в Excel интерфейс головоломки **solitaire**.

Правила игры:

Исходное положение головоломки показано на рисунке. Ход заключается в перепрыгивании по горизонтали или по вертикали некоторой выбранной фишки через соседнюю фишку на свободное поле, при этом фишка, через которую перепрыгнули, снимается с поля. Головоломка считается решенной, если осталась одна фишка в центре игрового поля.

Постановка задачи:

На активном листе должно быть подсвечено игровое поле, на котором в каждый момент времени отображается текущее состояние игры. До начала игры фишки расставляются таким образом, как показано на рисунке. Ходы осуществляются кликом мышкой в



ячейки игрового поля: клик в поле с фишкой означает ее выделение, клик в пустое поле обозначает ячейку, в которую игрок хочет переместить выделенную фишку. Автоматически проверяется допустимость данного хода. Если ход возможен, то фишка перемещается, а с поля снимается фишка, через которую был произведен прыжок. Головоломка считается решенной, если осталась одна фишка, и она расположена в центре игрового поля. Рядом с полем необходимо указывать текущее количество фишек. Желательно, чтобы на поле были кнопки, позволяющие отменить последние N ходов или начать игру заново.

Сумма (сложность средняя)

С помощью макроса реализовать в Excel интерфейс игры **сумма**.

Правила игры:

Игра проходит на поле размером NxN, где N – нечетно. В центральной клетке расположено число 0, сумма всех чисел равна нулю. В центре находится фишка. Играют два игрока, поочередно делая ходы. За один ход игрок может переместить фишку в любую соседнюю клетку, в которую еще не было сделано хода. После хода в ячейку она закрывается и в нее больше нельзя ходить. При каждом ходе вычисляется счет игры, который равен текущему счету плюс значение ячейки, в которую был сделан ход. Игра заканчивается, когда нельзя сделать ни одного хода. Если в результате игры счет получился отрицательным, то побеждает первый игрок. Если счет положительный, то второй.

Постановка задачи:

На активном листе должно быть подсвечено игровое поле, на котором в каждый момент времени отображается текущее состояние игры. До начала игры в диалоге пользователь указывает размер игрового поля. В начале игры в центр поля записывается число 0, и туда ставится игровая фишка (поле можно подсветить каким-то цветом). Остальные цифры расставляются случайным образом, сумма всех цифр поля должна быть равна 0. В течение игры рядом с полем показывается текущий счет и имя игрока, который ходит в данный момент. Ход осуществляется кликом в ячейку игрового поля, которая находится

Текущий счет: -10						
ходит		игрок1				
1	-13	2	-22	17	-3	-11
22	-5	-7	3	-25	-7	11
14	44	-15	-7	-22	-4	-22
21	-7	5	0	-14	-12	17
2	-17	3	-7	-13	20	-5
-1	-4	14	-9	25	3	-8
22	14	35	-22	-6	-5	-2

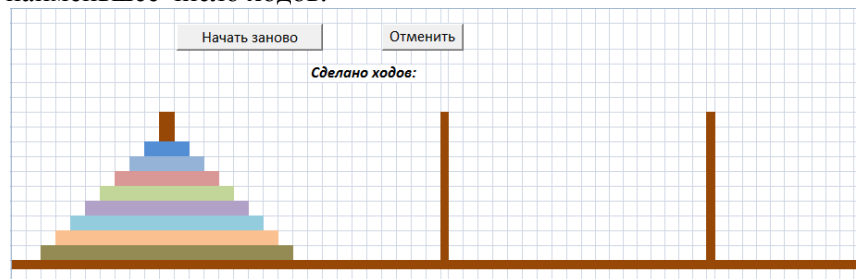
по соседству с текущим положением фишки, после чего автоматически проверяется допустимость этого хода, и меняется текущий счет игры. Если допустимых ходов нет, то объявляется победитель.

Ханойская башня (сложность низкая)

С помощью макроса реализовать в Excel интерфейс головоломки ханойская башня.

Правила игры:

Даны три стержня, на один из которых нанизаны восемь колец, причем кольца отличаются размером и лежат меньшее на большем. Кольца можно переносить с одного стержня на другой. За один раз разрешается переносить только одно кольцо, причём нельзя класть большее кольцо на меньшее. Задача состоит в том, чтобы перенести пирамиду из восьми колец с одного стержня на другой за наименьшее число ходов.



Постановка задачи:

На активном листе должно быть подсвечено игровое поле, на котором в каждый момент времени отображается текущее состояние игры. До начала игры все кольца находятся на самом левом стержне. Пользователь перемещает кольца, кликая сначала на тот стержень, с которого снимает кольцо, а потом на тот стержень, куда хочет кольцо переместить. Программа проверяет, допустим ли такой ход по правилам игры, и, если да, то отрисовывает кольцо на новом месте. Головоломка считается решенной, как только на втором или третьем стержне соберется пирамида из восьми колец. Рядом с полем необходимо указывать, какое количество ходов произведено на текущий момент. Желательно, чтобы на поле были кнопки, позволяющие отменить последние N ходов или начать игру заново.

Lines (сложность высокая)

С помощью макроса реализовать в Excel интерфейс игры **lines**.

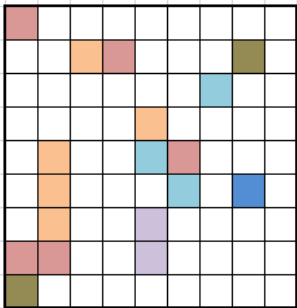
Правила игры:

Игра проходит на поле размером 9x9. В игре участвуют фишки семи цветов. До игры на поле расставляются три фишки произвольного цвета. За один ход игрок может передвинуть одну фишку, выделив ее и указав ее новое местоположение. Для совершения хода необходимо, чтобы между начальной и конечной клетками существовал путь из свободных клеток. Цель игры состоит в удалении максимального количества фишек, которые исчезают при выстраивании фишек одного цвета по пять и более в ряд (по горизонтали, вертикали или диагонали). При исчезновении ряда фишек новые три фишки не выставляются. В остальных случаях каждый ход выставляются новые три фишки. Игрок может видеть заранее три фишки, которые появятся в следующем ходу. Чем длиннее собранная линия, тем больше очков получает игрок. Чем больше очков у игрока, тем меньше времени ему даётся на ход. Игра заканчивается, когда поле полностью заполнится фишками.

Постановка задачи:

На активном листе должно быть подсвечено игровое поле 9x9, на котором в каждый момент времени отображается текущее состояние игры. До начала игры случайным образом на поле выставляются три фишки. Пользователь делает ход, выбирая фишку, которую он хочет переместить, а затем поле, куда фишка должна попасть. Если такой ход возможен, то фишка перемещается, и происходит проверка, не собралась ли линия. Если нет, то программа выставляет на поле еще 3 фишки и опять проверяет, не собралась ли линия. Количество очков, которые добавляются игроку за собранную линию, зависит от длины этой линии. В ходе игры рядом с игровым полем показывается, сколько ходов сделал игрок, сколько линий собрал и сколько очков получил на текущий момент.

Текущий счет:	22
Сделано ходов:	7
Собрано линий	3



время на ход	1:45
--------------	------

При желании можно выбрать свою игру, но выбор ОБЯЗАТЕЛЬНО обсудить с преподавателем.

Бегущая информационная строка (сложность низкая)

С помощью макроса реализовать в Excel эмуляцию бегущей строки на информационном табло (подобная строка есть в некоторых автобусах и в вагонах метро). Эмуляция реализуется путем циклического сдвига текста строки справа налево, при этом исчезающие слова появляются через некоторое время справа.

Перед началом работы в диалоге пользователь имеет возможность задать:

- ширину табло (количество символов на нем);
- стиль и цвет шрифта;
- текст рекламного сообщения произвольной длины;
- скорость движения текста по табло.

о	б	р	о	п	о	ж	а	л	о	в	а	т	ь	в	н	а	ш	м
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Рекламное табло (сложность средняя)

С помощью макроса реализовать в Excel эмуляцию показа текста на рекламном табло из нескольких строк. Показ текста осуществляется путем циклического сдвига строк снизу вверх, при котором верхняя строка табло исчезает, но появляется через некоторое время снизу.

Перед началом работы в диалоге пользователь имеет возможность задать:

- ширину табло (количество символов в строке и количество строк на нем);
- стиль и цвет шрифта;
- рекламный текст произвольной длины;
- скорость движения текста по табло.

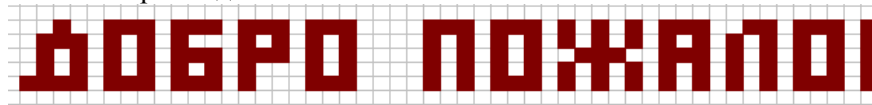
Д	о	б	р	о	п	о	ж	а	л	о	в	а	т	ь	в	н	а	ш	
м	а	г	а	з	и	н	!												
Л	е	т	н	и	е	к	а	н	и	к	у	л	ы	н	а	м	о	р	е
О	т	к	р	ы	т	ы	п	р	о	д	а	ж	и	л	е	т	н	и	х
т	у	р	о	в	!	Д	н	а	п	а	,	С	о	ч	и	,			

Световая матрица (сложность высокая)

С помощью макроса реализовать в Excel эмуляцию показа текста в виде бегущей строки на информационном табло. Показ осуществляется сдвигом подсветки ячеек справа налево, при этом исчезающий слева текст появляется через некоторое время справа. Каждая буква изображается с помощью нескольких подсвечивающихся ячеек табло, на фиксированной для нее матрице (на рисунке для букв А и О размер матрицы 5x3, а для буквы Ж 5x5).

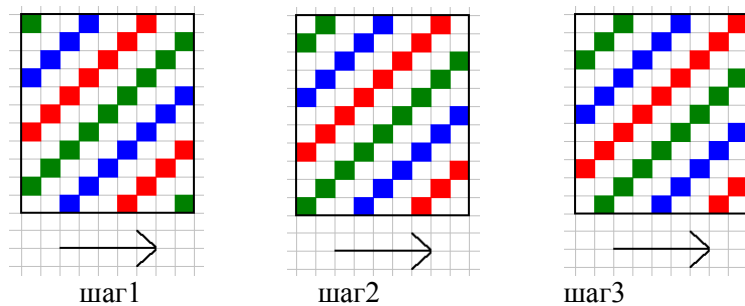
Перед началом работы в диалоге пользователь имеет возможность задать:

- ширину табло (количество ячеек в табло);
- текст информационного сообщения;
- скорость движения текста по табло.



Иллюминация (сложность низкая)

С помощью макроса реализовать в Excel эмуляцию праздничной иллюминации. Перед началом работы в диалоге пользователь задает размер светового табло (количество ячеек по горизонтали и вертикали). После этого на поле соответствующего размера ему дается возможность с помощью стандартных средств Excel создать рисунок (изменить цвет некоторых ячеек) и выбрать направление движения рисунка (по вертикали, по горизонтали или по диагонали). По мере исчезновения рисунка с одной стороны он должен появляться с другой. На примере показано последовательное движение рисунка по табло в направлении слева направо.

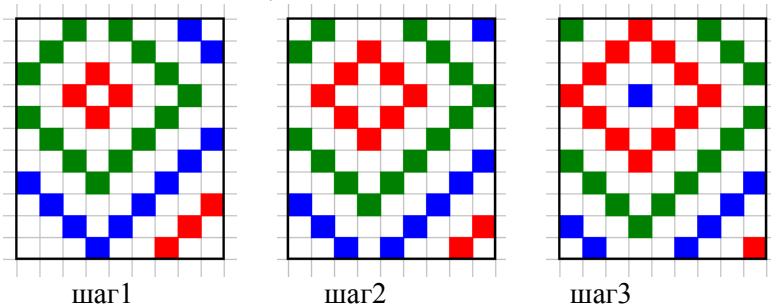


Иллюминация в разбег (сложность высокая)

С помощью макроса реализовать в Excel эмуляцию праздничной иллюминации. Движение рисунка происходит от точки внутри светового табло к его краям. Перед началом работы пользователь выбирает:

- размер светового табло;
- тип рисунка из 5-6 предложенных на выбор (ромбы, круги, квадраты, лучи т.д.);
- количество используемых цветов;
- координату точки, в которой будет расположен центр рисунка.

На примере показано движение ромбов от точки с координатой (4,4) с использованием 3-х цветов.



Литература

1. Баурн С. Операционная система UNIX. – М.: Мир, 1986.
2. Гарнаев А.Ю. Самоучитель VBA. – СПб.: БХВ-Петербург, 2004.
3. Головин И.Г., Волкова И.А. Языки и методы программирования. – М.: ИЦ Академия, 2012.
4. Король В.И. Visual Basic 6.0, Visual Basic for Application 6.0. Язык программирования. Справочник с примерами. – М.:КУДИЦ-ОБРАЗ, 2000
5. Кузьменко В.Г. VBA. – М: ООО "Бином-Пресс", 2015
6. Михеев Р.Н. VBA и программирование в MS OFFICE для пользователей. – СПб.: БХВ-Петербург, 2006.
7. Хэррон Д. Node.js Разработка серверных веб-приложений на JavaScript. – М.: ДМК Пресс, 2012.
8. Эйткен, Питер. Разработка приложений на VBA в среде Office XP.: Пер. с англ. – М.:Издательский дом «Вильямс», 2003
9. Visual Basic for Applications Language Reference [Электронный ресурс]. – Электрон. дан. – URL: [http://msdn.microsoft.com/en-us/library/ee441138\(v=office.12\).aspx](http://msdn.microsoft.com/en-us/library/ee441138(v=office.12).aspx) (дата обращения: 08.09.2015)