МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ЯДЕРНЫЙ УНИВЕРСИТЕТ

«МИФИ»

Н.Г. Волчёнков

ОСНОВЫ ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ VISUAL BASIC ДЛЯ ОФИСНЫХ ПРИЛОЖЕНИЙ

МОСКВА 2018

УДК 004.42+004.432
ББК 32.973.26-018.2.75
В68

Волчёнков Н.Г. **Основы программирования на языке Visual Basic для офисных приложений**: Учебное пособие [Электронный ресурс]. – М.: НИЯУ МИФИ, 2018. − 166 с.

В пособие включены тексты шести лекций (главы 1-2, 4-5, 7-8 и 10) и материалы трёх лабораторных работ (главы 3, 6 и 9). Учебное пособие предназначено служить основой учебных курсов «Информатика. Основы программирования» и «Информационные технологии. Офисные приложения». Курсы с указанными названиями (или иные курсы по изучению основ современных информационных технологий с аналогичными разделами) читаются на младших курсах различных институтов НИЯУ МИФИ.

Тексты лекций включают освещение основных фундаментальных вопросов программирования на любом алгоритмически полном операторном языке программирования на примере языка офисного программирования VBA for Excel, который отвечает требованию алгоритмической полноты. Каждая лабораторная работа содержит вопросы по изучаемой теме, на которые студентам надо ответить пред её выполнением. На примерах программ, которые студенты создают и отлаживают в ходе выполнения лабораторных работ, демонстрируется технология создания разнообразных полезных приложений.

Учебное пособие предназначено студентам – будущим бакалаврам, осваивающим теоретические основы и практические навыки программирования при решении содержательных задач различного характера, а также преподавателям, читающим лекции и проводящим лабораторные работы по указанным курсам в компьютерных классах.

Рецензент канд. техн. наук, доц. *В.Б. Шувалов.*

ISBN 978-5-7262-2446-6

Редактор *Е.Е. Шумакова*

Подписано в печать 22.01.2018. Формат 60x84 1/8.
Печ. л. 9,8. Уч.-изд. л. 9,0. Изд. № 017-1.

Оглавление

Введение

В данном учебном пособии излагаются сведения из раздела «Программирование» курса «Информатика», читаемого студентам Экономико-аналитического института, а также других институтов НИЯУ МИФИ.

Основы программирования, по мнению автора, являются необходимым компонентом любого курса, связанного с освоением современных информационных технологий. Разумеется, умение писать и отлаживать компьютерные программы может и не пригодиться в работе многим бакалаврам после окончания четырёхгодичного обучения по выбранным ими специальностям. Кроме того, давно стало очевидным, что владение современными компьютерными технологиями указанными специалистами не предполагает выходить за рамки использования готовых программ, разработанных высококлассными программистами-профессионалами.

Тем не менее, опыт показывает, что даже простое владение распространёнными технологиями – такими как использование офисных приложений (Microsoft Word, Microsoft Excel и т.д.) – иногда, при решении задач определённого класса, например экономических, требует от их пользователей выходить за рамки имеющихся в этих приложениях готовых к употреблению средств. Автор имеет в виду желательность или даже необходимость создания пользователями собственных макрокоманд (так называемых «макросов»), улучшающих решение этих задач. И для написания этих макрокоманд пользователю необходимо владеть приёмами и навыками программирования.

В указанных офисных приложениях их создатели позаботились об обеспечении программными средствами создателей макрокоманд: в них встроена среда программирования на одном из самых простых, доступных и в то же время алгоритмически полных языков программирования – на языке Visual Basic. Автор данного пособия имеет 20-летний опыт преподавания на этом языке и многочисленные публикации, начиная с 1998 г.: [1], [2] и другие, вплоть до 2015 г.

В этом языке представлены все базовые понятия и технологии, которые используются в любом «уважающем себя» языке программирования операторного (императивного) типа. Изучению основам языка Basic (точнее, его диалекту VBA – Visual Basic for Application) и приёмам программирования на этом языке посвящено данное учебное пособие.

Следует отметить, что язык VBA весьма консервативен: он очень мало меняется с годами, в отличие, например, от языка Visual Basic для создания автономных Windows-приложений. Язык VBA, почти в современном своём виде появившийся около 20 лет тому назад, возник на основе (как диалект) языка Visual Basic 6 [1]. Из этого языка были удалены развитые графические средства (которых достаточно в арсенале самих офисных приложений) и добавлены объектно-ориентированные средства работы со специфическими объектами того или иного офисного приложения: например, с объектами Workbook, Worksheet, Cell или Chart приложения MS Excel. Автор рекомендует слушателям данного курса использовать дополнительно для более глубокого усвоения материала один из известных самоучителей, например, книгу Л.Д. Слепцовой [2].

Таким образом, изучение указанного выше раздела курса «Информатика» преследует двоякую цель: во-первых, изучение начальных основ программирования как такового и, во-вторых, изучение конкретного, весьма распространённого в мире языка Basic – языка программирования макросов для офисных приложений.

Данное пособие состоит из 10 глав, соответствующих последовательности изучения данного курса, условно состоящего из 4-х разделов:

- «Основные понятия операторного языка программирования и среды программирования в офисных приложениях» – 2 лекции и одна лабораторная работа;
- «Основные операторы, выражения и функции языка Basic для офисных приложений» – 2 лекции и одна лабораторная работа;

- «Организация повторений, циклические операторы и массивы в языке Basic для офисных приложений» – 2 лекции и одна лабораторная работа;
- «Элементы объектно-ориентированного программирования в языке Basic для офисных приложений» – одна лекция.

Каждая глава – это либо конспект одной 2-часовой лекции, либо материалы одной лабораторной работы, проведение которой рассчитано на два занятия по 3 ч каждое. Таким образом, курс рассчитан на 7*2 = 14 ч лекций и 3*(3 + 3) = 18 ч лабораторных работ. Весь курс занимает 14 + 18 = 32 ч на студента.

Глава 1

Данная глава – это текст первой (вводной) лекции по курсу программирования на языке Visual Basic для офисных приложений Microsoft Excel и Word, кратко аннотированному во введении к данному учебному пособию. Содержание лекции можно структурировать следующим образом:

- Алгоритмы и программы.
- Введение в язык программирования VBA (Visual Basic for Application).
- Программный модуль и программная процедура. Модуль как множество процедур-событий.
- Модуль экранной формы.

1.1. Алгоритмы и программы

Что такое «алгоритм»?

Прежде всего, рассмотрим понятие «исполнитель алгоритма» и только после этого дадим определение самого понятия «алгоритма».

В широком смысле, исполнитель – это природный или рукотворный объект, способный решать поставленные перед ним задачи. Природный объект – это живое существо (в частности, человек или собака). Рукотворный объект – это механизм (в частности, робот или компьютер). Разумеется, в данном курсе, относящемся к информационным технологиям, мы будем рассматривать только алгоритмы для компьютера.

Алгоритм – это точный набор инструкций, описывающих порядок действий исполнителя для получения результата решения поставленной перед исполнителем задачи. Важным при этом является то, что задача должна быть решена за конечное время.

В старой трактовке вместо слов «порядок действий» использовались слова «последовательность действий», но уже на заре компьютерной эры, по мере роста сложности в работе компьютеров, слово «последовательность» стали заменять более общим словом «порядок». Это связано с тем, что работа одних инструкций алгоритма может зависеть от результатов работы других инструкций.

Происхождение термина «алгоритм» относится к имени жившего более 1200 лет назад арабского математика Аль-Хорезми. Слово «альхорезми» трансформировалось в «алхоризм» и, затем, в «алгорифм» или «алгоритм» (рис. 1.1).

Рис. 1.1. Почтовая марка, выпущенная в связи с 1200-летием Абу Абдуллаха Мухаммада ибн Муса аль-Хорезми (перс. خوارزمی])

Для представления алгоритмов вот уже более полувека используют наглядный вид диаграмм, которые называют «блок-схемами».

При таком представлении инструкции алгоритма изображаются фигурами (овалами, прямоугольниками, ромбами и т.д.), которые соединяются между собой стрелками, обозначающими переходы от одних инструкций к другим.

На рис. 1.2 приведены примеры блок-схем двух алгоритмов: для решения задачи вычисления объёма параллелепипеда и задачи вычисления площади треугольника по формуле Герона.

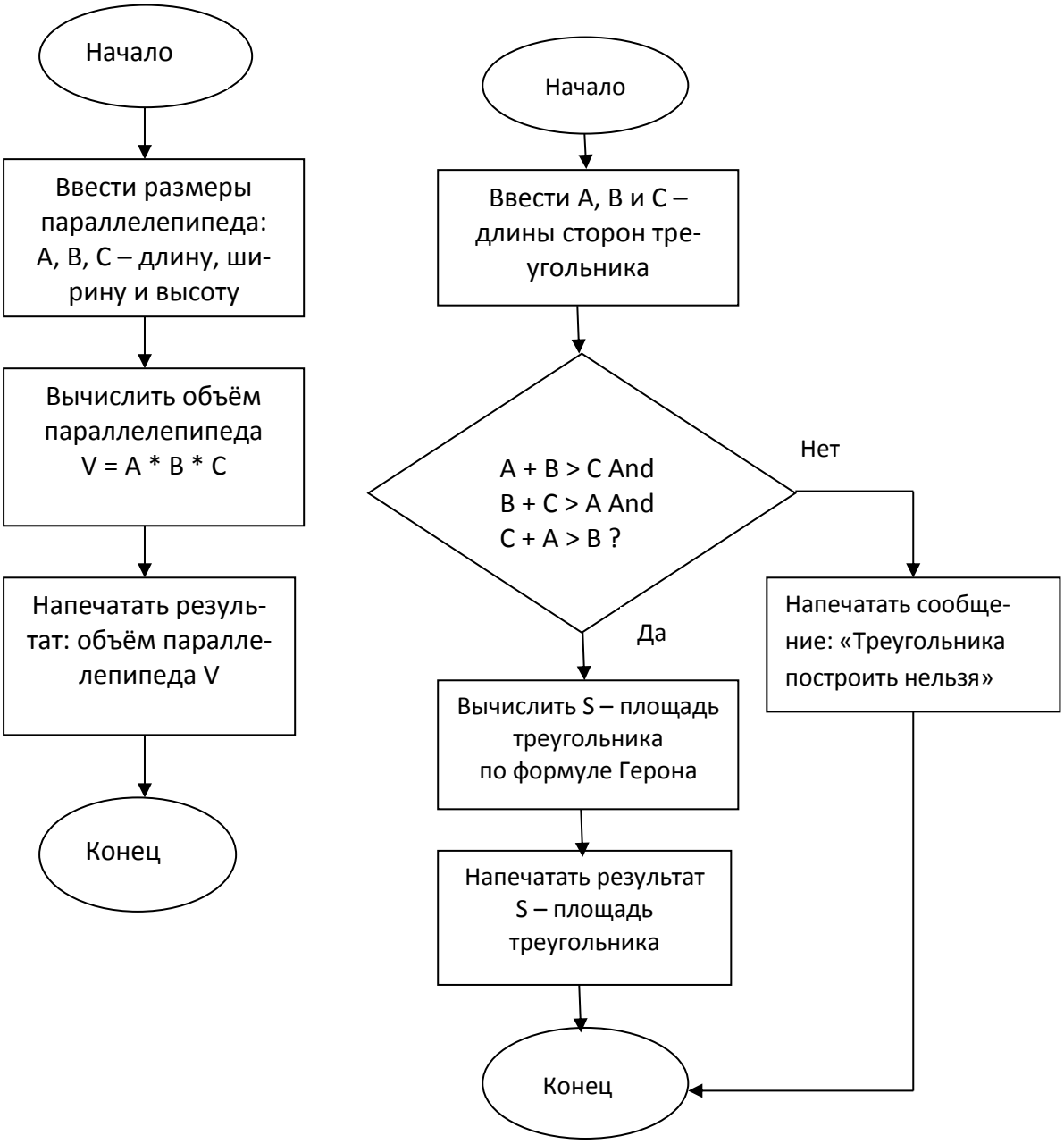


Рис. 1.2. Примеры блок-схем алгоритмов

Следующим после алгоритмизации этапом программирования на компьютере является собственно «программирование». Оно заключается:

- в создании так называемого «исходного текста» компьютерной программы на каком-либо языке программирования;
- в вводе этого текста в компьютер с использованием среды программирования для этого языка;
- в отладке программы на конкретных примерах (устранение ошибок при её написании) и, наконец,
- в передаче программы компьютеру с целью её компиляции (создание так называемого исполняемого кода).

Ниже приведены три определения компьютерной программы.

Определение 1. Компьютерная программа – набор инструкций, предназначенный для исполнения устройством управления вычислительной машиной. Программа – один из компонентов программного обеспечения. В зависимости от контекста рассматриваемый термин может относиться также и к исходным текстам программы. Компьютерные программы как объект интеллектуальной собственности относят к категории нематериальных активов.

(*цит.* – **http://ru.wikipedia.org/wiki/Компьютерная_программа**).

Определение 2. Компьютерная программа – это данные, предназначенные для управления конкретными компонентами системы обработки информации в целях реализации определённого алгоритма.

ГОСТ 19781-90 – Единая система программной документации (ЕСПД). Это комплекс государственных стандартов Российской Федерации, устанавливающих взаимосвязанные правила разработки, оформления и обращения программ и программной документации.

Определение 3. Компьютерная программа – представленная в объективной форме совокупность данных и команд, предназначенных для функционирования ЭВМ и других компьютерных устройств с целью получения определённого результата, включая подготовительные материалы, полученные в ходе разработки программы для ЭВМ, и порождаемые ею аудиовизу-

альные отображения (ст. 1261 «Программы для ЭВМ», ГК РФ, гл. 70 «Авторское право»).

Аббревиатура ЭВМ означает «электронная вычислительная машина». Это устаревший термин, теперь вместо ЭВМ говорят «компьютер».

1.2. Введение в язык программирования Visual Basic на примере языка VBA (Visual Basic for Applications)

Из всего множества различных алгоритмически полных языков программирования в данном курсе рассматривается ориентированный на офисное программирование язык Visual Basic (VB).

Более трёх десятилетий тому назад фирма Microsoft приняла решение оснащать популярные офисные приложения – такие как MS Excel, MS Word и другие – встроенными средствами высокоуровневого программирования. С помощью этого программирования предполагалось создавать так называемые макросы (макрокоманды). Назначение этих макросов – повысить функциональные возможности электронных таблиц Excel или документов Word. Макросы – это дополнительные инструменты, не входящие в стандартный арсенал приложений MS Excel, MS Word и других, которые позволяют более гибко, наглядно и быстро решать те или иные офисные задачи.

Простой пример: в бухгалтерское приложение могут быть добавлены кнопки, позволяющие переводить денежные величины, представленные в цифровой форме, в форму «прописи», и наоборот (23 133 500 р. 40 к.<=> «двадцать три миллиона сто тридцать три тысячи пятьсот рублей 40 коп.).

В современных версиях указанных приложений фирмы Microsoft используется язык VBA (Visual Basic for Applications) – ставший классическим диалект языка VB6 [1], разработанного этой фирмой в конце 90-х годов для создания автономных Windows-приложений. В языке VBA (в отличие от VB6) отсутствуют многие возможности, необходимые для автономных приложений, например графика. Эти возможности с лихвой заменяются собственным богатым арсеналом средств указанных приложений, не требующих никакого программирования, например средств построения 2D и 3D диа-

грамм. Но программирование может понадобиться даже в случае работы со «стандартным арсеналом», например, если появится необходимость нестандартным способом управлять указанными диаграммами, как то: вращать их, динамично изменять параметры функций и т.д.

Отметим, что цель, которая должна быть достигнута в ходе изучения данного курса, – это овладение его слушателями, не имеющими элементарных знаний в области алгоритмизации и программирования, необходимым набором таких знаний. К ним следует отнести такие классические понятия программирования как «переменная», «операторы условного и безусловного переходов», «циклические процессы», «одномерные и многомерные массивы данных», «чтение исходных данных из файлов», «запись данных в файлы», «элементы объектно-ориентированного программирования». Целью данного курса является также овладение навыками применения указанных знаний для решения простых задач, которые могут возникнуть при создании документов или электронных таблиц. Следует отметить, что указанное программирование принято называть «программированием высокого уровня», т.е. программированием, не требующим профессионального программистского мастерства. Язык VBA – прекрасное средство для достижения указанной цели.

1.3. Программный модуль и процедура

В простейшем случае, написать программу на языке VBA для приложения MS Excel – это создать так называемый программный модуль – отдельный «кирпичик», из которого в дальнейшем будет строиться «здание» – программный проект, соответствующий одному или нескольким макрокомандам (макросам). (Отметим, что, в частности, проект может и не содержать ни одного макроса!) В дальнейшем мы будем создавать более сложные программы, состоящие более чем из одного модуля. И не только для приложения MS Excel, но и для других приложений. В частности, для приложения MS Word. Но начнём с простого – с создания проекта, состоящего из одного модуля и не содержащего ни одного макроса.

Чаще всего программный модуль состоит из так называемых процедур-событий – одной или нескольких. Процедуры-события связаны с какими-то действиями, производимыми пользователем, или с чем-то, произошедшим в самом компьютере без участия пользователя. Событие вызывает (запускает) другое действие (чаще всего, виртуальное), которое начинается после того, как это событие произошло.

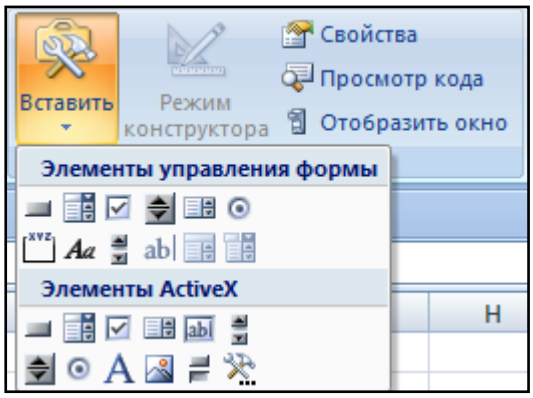
Самые яркие и распространённые события – это щелчки мышью «командных кнопок» – виртуальных объектов, установленных программистом, например, непосредственно на листах книги Excel (или на страницах документа Word). Эти события инициируют выполнение каких-либо процедур, например появление экранных форм, на которых, в свою очередь, тоже установлены виртуальные кнопки. Эти кнопки пользователь тоже может щёлкать, вызывая выполнение других процедур-событий, и т.д.

Рассмотрим пример создания программного модуля, который назовём «Приветствие». При этом проект не будет содержать ни одного макроса.

Пример 1.1. Пусть пользователь записал в ячейку «A1» листа List1 только что открытой книги Excel свои фамилию, имя и отчество, например: «Иванов Андрей Сергеевич». Затем он может щёлкнуть (а может и не щёлкать!) виртуальную кнопку, установленную программистом на этом же листе. (На этой копке установлена надпись «Щёлкни меня, чтобы получить приветствие».) Немедленно в ячейке «B1» этого листа появится фраза: «Привет, Иванов Андрей Сергеевич!».

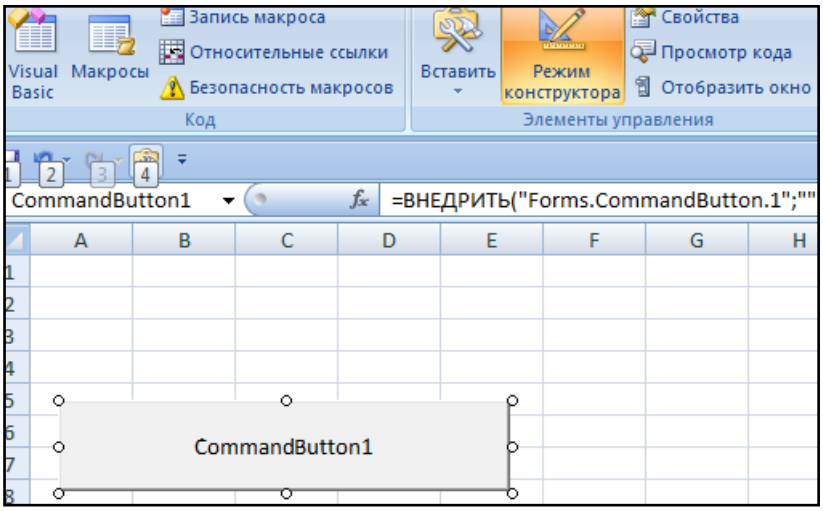
Модуль «Приветствие» для решения этой задачи может быть создан следующим образом.

На ленте меню приложения Excel выбирается вкладка «Разработчик». (Если при первом запуске приложения Excel вы этой вкладки не увидите, необходимо поместить её на ленту с помощью настройки ленты – отметить данную вкладку «галочкой» в разделе «Параметры Excel».)



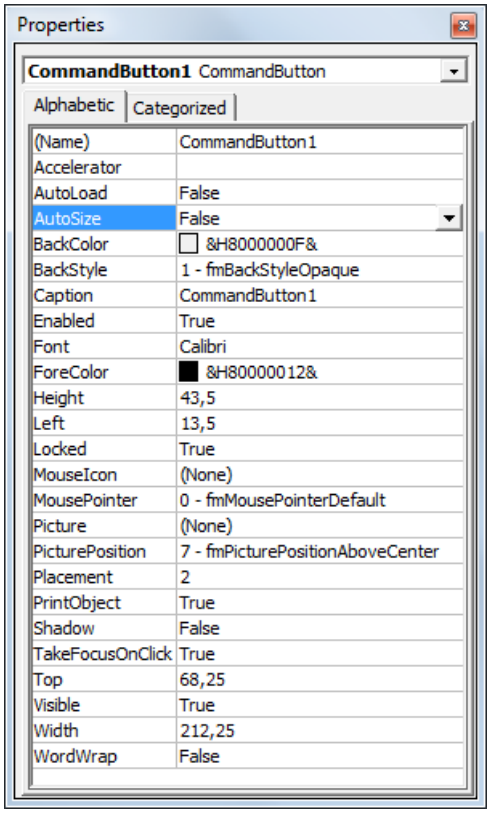
На вкладке «Разработчик» в группе «Элементы управления» выбирается команда «Вставить» (рис. 1.3) и в появившейся панели (в разделе «Элементы ActiveX») щёлкается значок «кнопка».

Рис. 1.3. Вставка элемента управления на экранную форму или на лист книги Excel



Управление передаётся мыши, с помощью которой на листе в произвольном месте устанавливается «макет» командной кнопки и активируется «Режим конструктора» (рис. 1.4).

Рис. 1.4. Активация «Режима конструктора» после вставки мандной кнопки



«Макет» командной кнопки (с надписью CommandButton1) имеет значения своих свойств, которые можно увидеть, открыв окно «Свойства» в этой же группе «Элементы управления» на вкладке «Разработчик» рис. 1.5). Следует отметить, что список свойств в данном окне относится только к «активному» элементу управления, который окружён белыми кружочками-маркерами.

Рис. 1.5. Окно свойств для «активного» элемента управления

Значения свойств в данной таблице можно менять. Например, надпись на кнопке (значение свойства Caption) заменить на «Показать приветствие пользователю», шрифт (Font), цвет кнопки (BackColor), перенос слов (Word-Wrap) и т.д.

После указанных замен получаем изменённый вид кнопки (рис. 1.6).

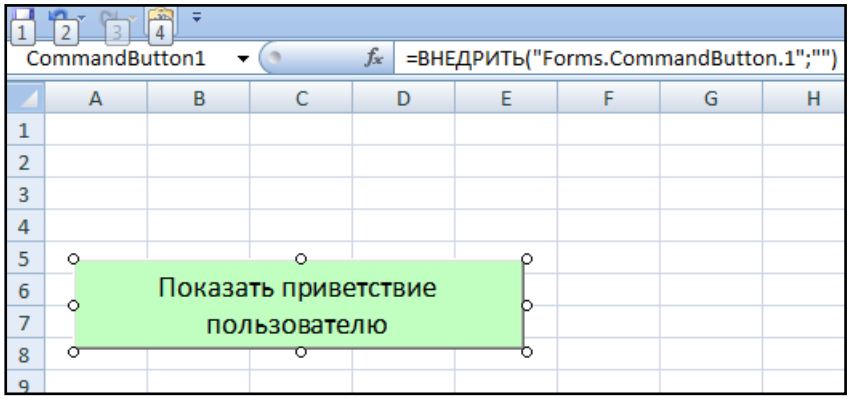


Рис. 1.6. Вид командной кнопки после изменения значений её свойств

Отметим, что изменение надписи на кнопке не означает, что её имя (значение свойства Name) изменилось: оно осталось прежним, CommandButton1.

Теперь приступим к написанию единственной (в данном случае) процедуры, которая должна запуститься после щелчка только что установленной на листе Excel кнопки. Можно сделать это по-разному.

Способ 1. Откроем среду проектирования Visual Basic – кнопка слева на вкладке «Разработчик» (рис. 1.7).

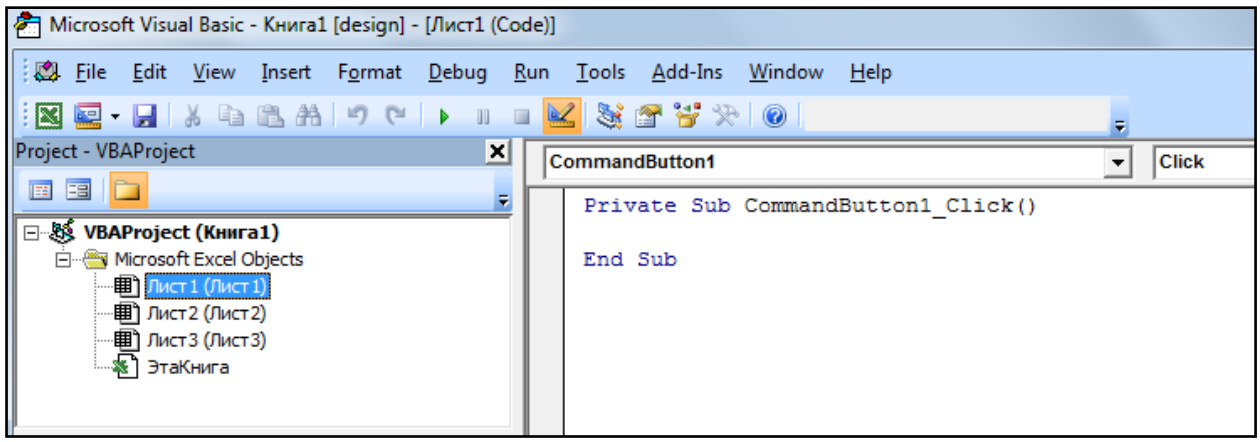


Рис. 1.7. Среда проектирования Visual Basic с открытым окном кода объекта – в данном случае, объекта CommandButton1

Слева – окно обозревателя проекта VBA (Project Explorer).

Справа – либо окно кода выделенного объекта, в данном случае, объекта Лист1 (View Code), либо окно самого этого объекта (View Object).

Переход от одного окна (View Code) к другому (View Object) осуществляется нажатием одной из двух кнопок в окне обозревателя проекта. Здесь открытым показано окно View Code. Над ним – два ниспадающих списка. Левый – список элементов на объекте. Правый – список событий, которые могут случиться с выбранным элементом. (Предугадывая выбор, система предлагает наиболее вероятное событие Click.) Автоматически формируются две строки процедуры, которую необходимо создать:

```
Private Sub CommandButton1_Click()

End Sub
```

Теперь достаточно вписать текст процедуры между этими строчками. Легко сообразить, что это будет всего одна строка, в которой запрограммирована запись в ячейку «B1» строки с приветствием:

```
Range("A2") = "Привет, " & Range("A1") & "!"
```

(Строка формируется из трёх строк, которые объединяются с помощью знака конкатенации &.)

Для запуска программы необходимо выйти из «режима конструктора» – отжать соответствующую кнопку меню на вкладке «Разработчик» – и щёлкнуть командную кнопку (не забыв предварительно ввести ФИО в ячейку «A1»!). Результат не заставляет долго себя ждать (рис. 1.8).

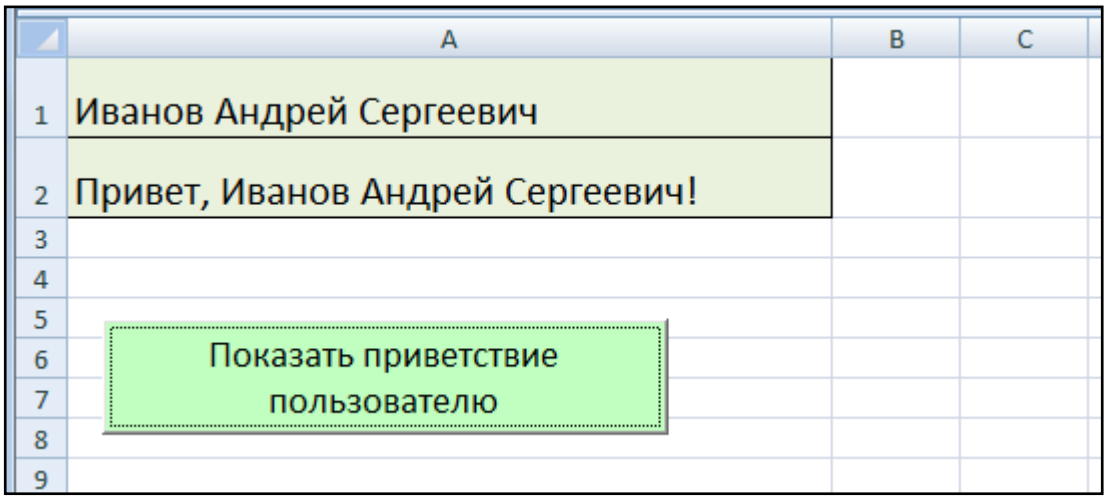


Рис. 1.8. Результат работы макроса после щелчка объекта CommandButton1

Способ 2. Открывать среду проектирования Visual Basic не обязательно – в режиме конструктора достаточно дважды щёлкнуть командную кнопку – и сразу открывается окно программного кода. Далее программируем щелчок кнопки, как это делалось 1-м способом

Пример 1.2. Слегка усложним задачу примера 1.1. Создадим макрос (макрокоманду) с именем «Привет». Единственная процедура этого макроса ничем, кроме имени, не будет отличаться от представленной выше процедуры с именем `CommandButton1_Click`:

```
Private Sub Привет()
   Range("A2") = "Привет, " & Range("A1") & "!"
End Sub
```

Принципиальная разница: макрос может вызываться из любого другого программного модуля нашего проекта. В частности, пусть одна и та же задача, описанная в примере 1.1, решается на всех трёх листах проекта. Макрос один, а результатов будет несколько. При этом командные кнопки на всех листах проектов будут иметь один и тот же код:

```
Private Sub CommandButton1_Click()
    Привет
End Sub
```

Все три указанные результата показаны на рис. 1.9.

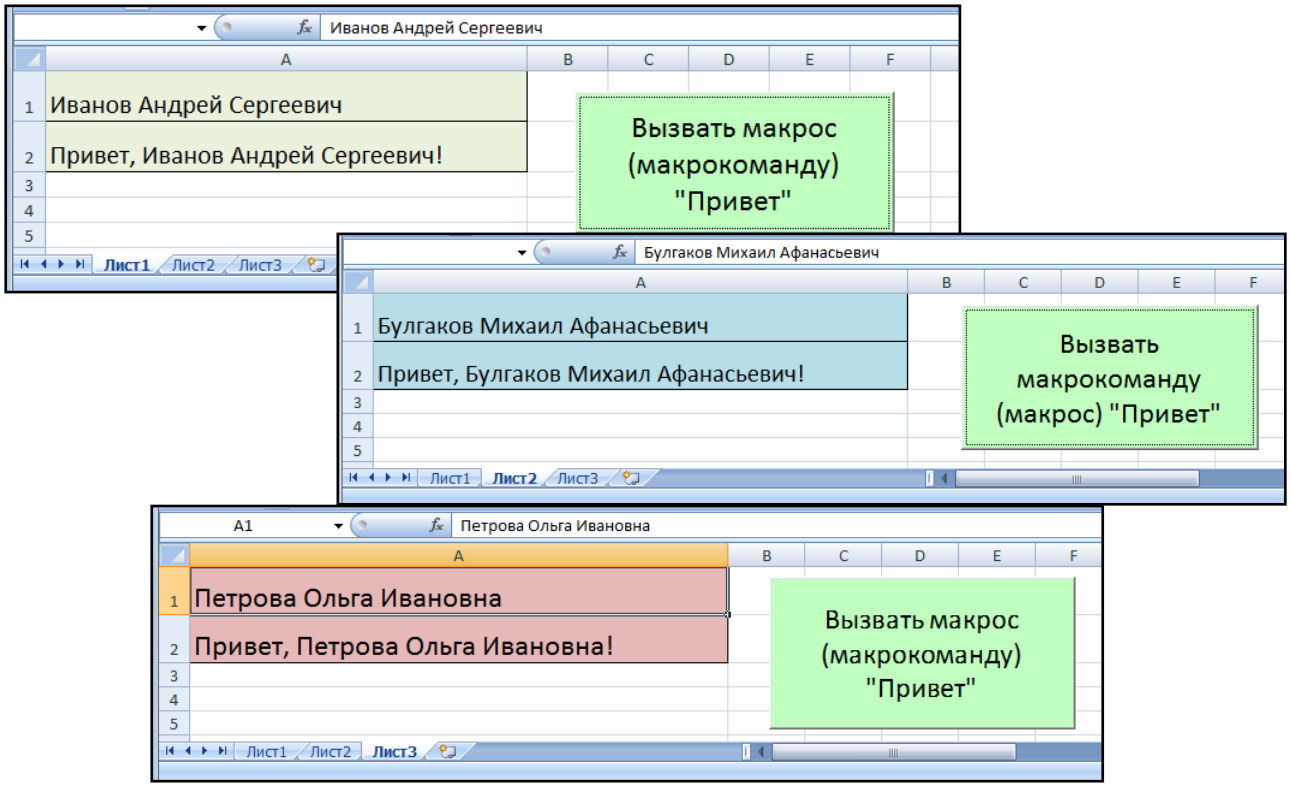


Рис. 1.9. Результаты работы макроса «Привет», вызываемого с трёх различных листов

Теперь уточним, как следует создавать макрос. Рассмотрим два способа.

Способ 1. На вкладке «Вид» ленты находим пункт «Макросы» и открываем панель «Макрос» (рис. 1.10).

Выбираем место «Эта книга» и имя макроса «Привет», после чего нажимаем кнопку «Создать».

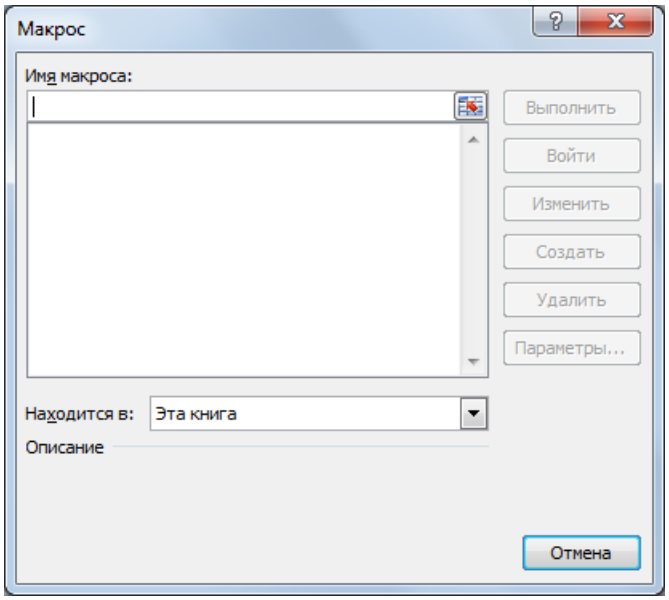


Рис.1.10. Панель для создания макроса и для работы с ним в дальнейшем

Открывается окно программного кода для написания одной или нескольких процедур (рис. 1.11).

Отметим, что ссылка на окно View Object в окне обозревателя модуля макроса погашена. А модуль макроса в виде файла Module1 появился автоматически в автоматически созданной папке Modules.

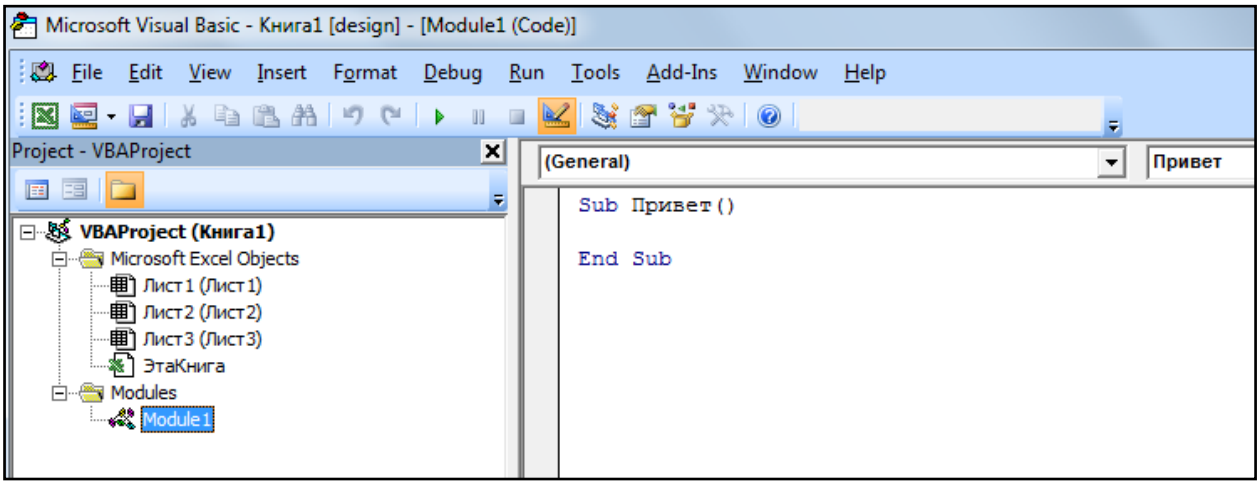


Рис. 1.11. Окно программного кода для создания макроса «Привет»

Между строками `Sub Привет()` и `End Sub` вставляем уже знакомую нам строку `Range("A2") = "Привет, " & Range("A1") & "!"`

На этом создание нового макроса заканчивается.

Способ 2. Пункт «Макросы» находим не на вкладке «Вид» ленты, а на вкладке «Разработчик» ленты. Вот и всё отличие 2-го от 1-го способа. Открывается та же панель (рис. 1.12).

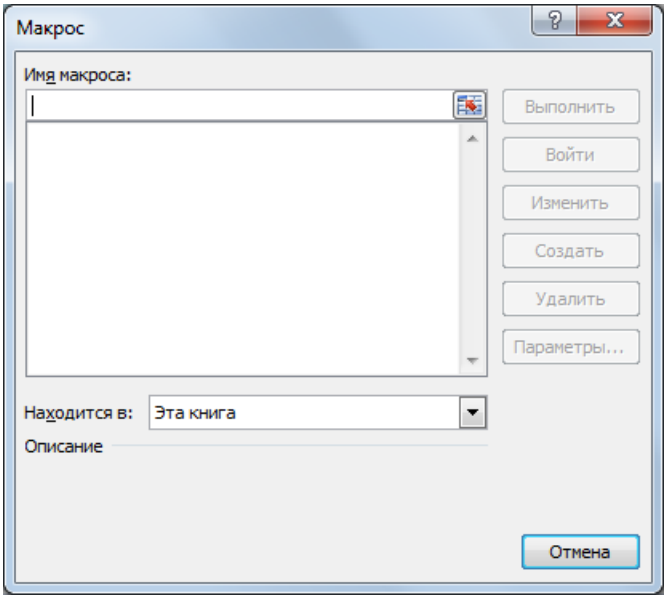


Рис. 1.12. Панель для создания макроса

Так же выбираем место «Эта книга» и имя макроса «Привет», после чего нажимаем кнопку «Создать». И т. д.

1.4. Модуль экранной формы

Очень удобным и наглядным способом программирования на языке VBA for Excel and for Word является способ с привлечением экранных форм. Экранная форма с разнообразными размещёнными на ней элементами управления может рассматриваться как программный объект, имеющий собственный программный модуль, который называют «модулем экранной формы».

Для включения в проект экранной формы необходимо использовать команды Insert → UserForm меню окна Microsoft Visual Basic, которое открывается с помощью вкладки «Разработчик» ленты офисного приложения (группа «Код»).

После указанного включения в проект экранной формы в обозревателе проекта появляется новая папка Forms, в которой мы видим файл UserForm1.

Щелчок по значку этого файла в обозревателе проекта позволяет открывать по желанию как окно данного объекта (Object View), так и окно кода данного объекта (View Code).

Окно объекта Object View изображено на рис. 1.13.

Серый прямоугольник с масштабными точками – заготовка будущей экранной формы. Рядом (справа внизу) – панель «инструментального ящика» (Toolbox). На ней значки элементов управления, которые можно разместить на экранной форме.

Рассмотрим пример программного модуля на основе введённой экранной формы для решения той же задачи, которая рассматривалась в предыдущих примерах.

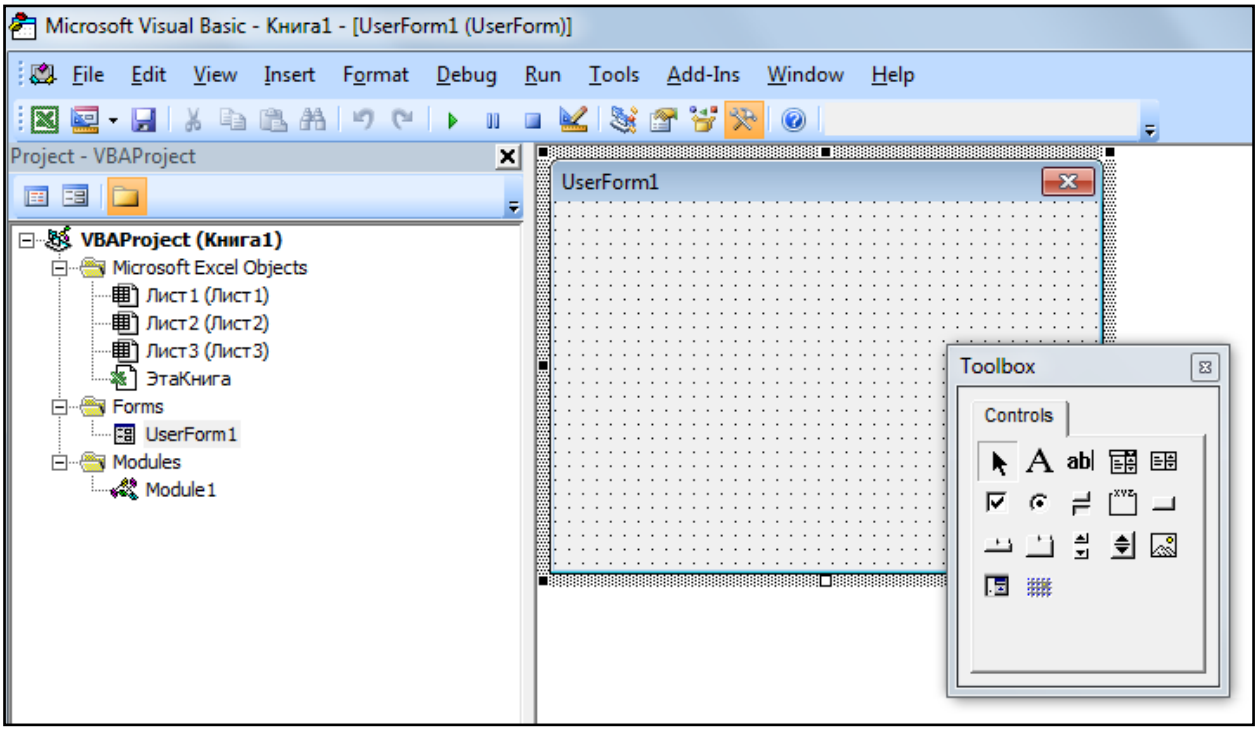


Рис. 1.13. Окно объекта Object View для создания экранной формы

Пример 1.3. После открытия экранной формы с помощью щелчка размещённой на листе книги Excel кнопки, процедура для которой имеет следующий вид:

```
Private Sub CommandButton1_Click()
    UserForm1.Show
End Sub
```

на экране появляется форма, готовая к работе (рис. 1.14).

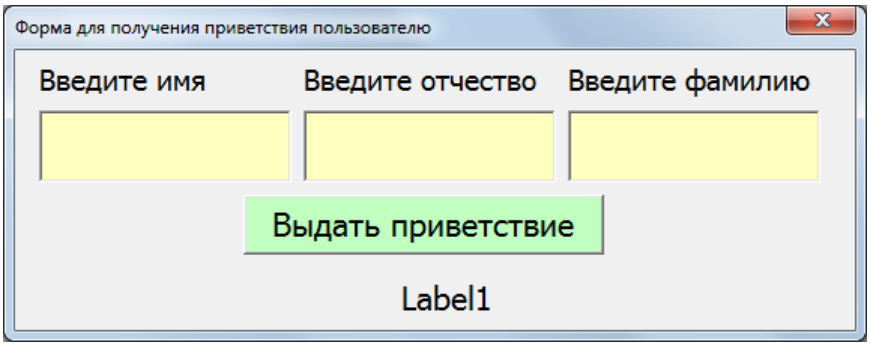


Рис. 1.14. Готовая к работе экранная форма

После ввода в текстовые поля экранной формы имени, отчества и фамилии и щелчка командной кнопки получается результат (рис. 1.15).

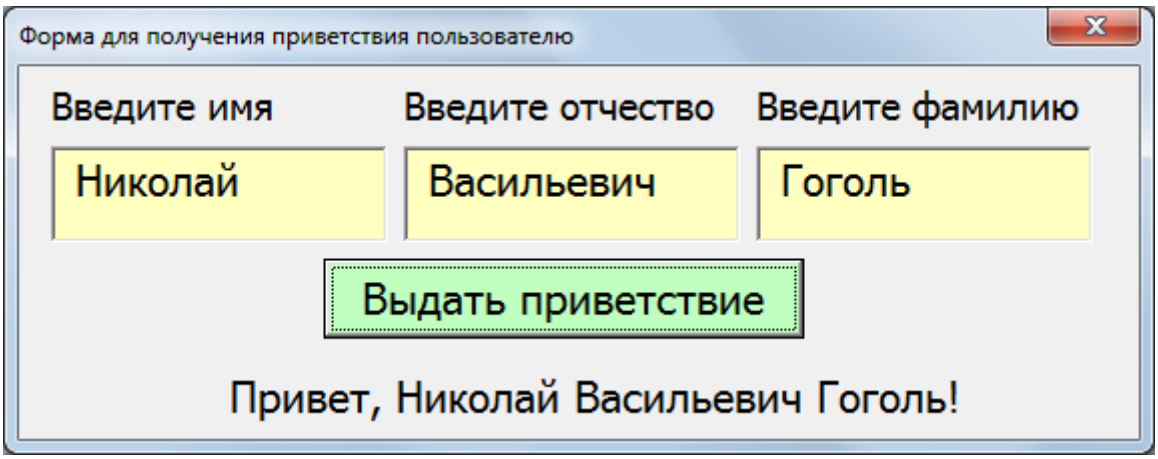


Рис. 1.15. Экранная форма с результатами работы

Модуль экранной формы состоит из единственной процедуры:

```
Private Sub CommandButton1_Click()
    Label1.Caption = "Привет, " & TextBox1.Text & _
    " " & TextBox2.Text & " " & TextBox3.Text & "!"
End Sub
```

Так как длина строки программного кода слишком велика, здесь использован перенос – с помощью символа «_» – подчёркивания.

Следует обратить внимание на то, что имена процедур для кнопки на листе книги Excel и для кнопки на экранной форме одинаковы, но это не играет никакой роли, так как эти кнопки относятся к разным объектам, т.е. указанные процедуры принадлежат к разным модулям и никак не связаны между собой.

Глава 2

Данная глава – это текст следующей, второй лекции по курсу программирования на языке Visual Basic. Содержание лекции можно структурировать следующим образом:

- Программа как инструмент обработки данных. Типы данных, рассматриваемые в языке VBA.
- Определение переменной и константы в программировании. Имя, тип, значение как характеристики переменной и константы.
- Декларация (объявление) переменной и константы в программном модуле и в процедуре.
- Оператор присваивания переменной значения определённого типа.
- Некоторые выражения и функции языка VBA.

2.1. Программа как инструмент обработки данных. Типы данных, рассматриваемые в языке VBA

Можно утверждать, что в большинстве случаев на языке VBA for Excel, VBA for Word и т.д. создаются программы, оформляемые как макрокоманды (макросы), предназначенные для обработки ***данных***. Данные могут быть разного ***типа*** – как числового, так и нечислового.

При этом, как правило, невозможно производить действия, предназначенные для данных одного типа, для данных другого типа. Например, нельзя применять арифметические операции к данным строкового типа; нельзя применять логические операции к числовым данным и т.д. Впрочем, «неожиданно» можно, например, удачно применить арифметическую операцию сложения не к числам, а к датам. Бывает и так, что результат какого-то действия выходит за пределы диапазона возможных значений, определяемых типом этого результата.

Поэтому перед началом преобразования алгоритма в программу важно внимательно изучить используемые в языке VBA типы данных.

Мы рассмотрим лишь 11 типов данных (реально, их 15[1]):

[1] «За кадром» остались ещё четыре типа данных: числовой **Decimal** (размер 14 байтов); два вариативных **Variant** (числовой и строковый); объектный **Object** [1].

(1) логический тип (**Boolean**);

(2 – 7) числовые типы (**Byte, Integer, Long, Single, Double, Currency**);

(8) тип времени-даты (**Date**);

(9) строковый тип (**String**) – для текстовых данных произвольной (нефиксированной) длины;

(10) строковый тип (**String * *N***) – для текстовых данных фиксированной длины *N*;

(11) тип, определяемый пользователем (**User defined type**).

Рассмотрим, какая память необходима для хранения данных некоторых из перечисленных типов и каков диапазон значений данных этих типов. Указанные сведения содержатся в табл. 2.1.

Таблица 2.1

Тип (значение)	**Размер**	**Диапазон значений**
Boolean – логический тип (логическое значение)	2 байта	Всего 2 значения: Истина или Ложь (True или False)
Byte – байт	1 байт	Целое число от 0 до 255 (только положительные значения)
Integer – короткое целое	2 байта	От -32 768 до 32 767
Long – длинное целое	4 байта	От -2 147 483 648 до 2 147 483 647
Single – десятичное короткое	4 байта	От -3.4E+38 до -1.4E-45 и от 1.4E-45 до 3.4E+38 (7 значащих цифр)
Double – десятичное длинное	8 байт	От -1.8E+308 до -4.9E-324 и от 4.9E-324 до 1.8E+308 (15 значащих цифр)
Currency – денежное (масштабированное целое)	8 байт	От -922337203685477.5808 до 922337203685477.5807
Date – время-дата	8 байт	От 1 января 100 г. до 31 декабря 9999 г.

Окончание табл. 2.1

Тип (значение)	**Размер**	**Диапазон значений**
String – строка (текст) переменной длины	10 байт + длина строки	От 0 до 2 миллиардов символов
String – строка (текст) фиксированной длины	Длина строки	От 0 до 65 тысяч символов
User defined type – тип, определяемый пользователем – структура таблицы БД (базы данных)	Длина *R* строки таблицы БД	*N* * *R*, где *N* – число строк таблицы БД (диапазон значений не ограничен – определяется произвольным значением *N*)

В программировании на языке VBA типами обладают не только обрабатываемые данные, но и такие объекты, как переменные, константы, значения арифметических, логических и иных выражений. Далее будут даны определения этих понятий.

2.2. Определение переменной и константы в программировании. Имя, тип, значение переменной и константы

Не погрешив против истины, можно определить «переменную» очень коротко: «переменная – это изменяемая часть программы».

Говоря более строго, ***переменная*** – это поименованная область компьютерной памяти, в которой хранятся данные определённого типа. Эти данные могут меняться в ходе выполнения программы. В программе переменная может встречаться многократно – в формулах и операторах.

Близко к понятию переменной лежит понятие постоянной величины (или – ***константы***). Единственное отличие константы от переменной заключается в том, что значение этой величины не может меняться в ходе выполнения программы.

Любая переменная (а также любая константа) имеет три обязательные характеристики: (1) имя; (2) тип; (3) значение.

При составлении алгоритма и написании программы следует соблюдать правила создания имён переменных и констант:

1. Имя должно начинаться с буквы (латинской, русской – не важно!)
2. За 1-й буквой имени должны следовать буквы, цифры или знак подчёркивания – в любой комбинации.
3. Другие символы использовать или нельзя, или не рекомендуется.
4. Длина имени не должна превышать 255 символов.
5. Нельзя использовать ключевые (зарезервированные) слова языка.

Тип переменной (и константы) – это тип её значения, т.е. тип данных, которые являются значением данной переменной (константы). В языке VBA допускается тот случай, когда тип переменной не объявлен заранее, до присвоения этой переменной значения определённого типа. В этом случае при первом же действии с переменной, когда она получает определённое значение, ей автоматически приписывается тип этого значения. Но такая «вольность» хотя и допускается, является нежелательной. Предпочтительным считается объявление типа переменной заранее, до её использования в каких-либо действиях. Об этом – в следующем разделе.

2.3. Декларация (объявление) переменной и константы в программном модуле и в процедуре

Переменные и константы в начале программы, до того момента, когда начинается описание их использования, как было сказано в предыдущем разделе, желательно (в языке VBA – именно желательно, но не всегда обязательно) ***объявить***. То есть указать имя и тип переменной (или константы). Для этой цели используют различные ключевые слова: **Dim**, **Private, Const** и **Public**. Первые два слова – для объявления ***локальных*** переменных, третье слово – для объявления констант, четвёртое слово – для объявления ***глобальных*** переменных. Далее, для краткости, будем говорить только о переменных, имея в виду, что речь может идти и о константах.

Ключевое слово **Dim** используется при объявлении переменной внутри процедуры. При таком объявлении ***область видимости*** этой переменной –

только сама эта процедура. Из других процедур данная переменная не видна. Поэтому в других процедурах имя этой переменной может использоваться для любых иных целей.

Ключевое слово **Private** служит для объявления тоже локальной переменной, но используется не внутри процедуры, а снаружи. При этом объявляемая локальная переменная видна во всех процедурах данного модуля, но не видна из других модулей данного программного проекта.

На этом объявление переменной не заканчивается. После имени через пробел записывается ключевое слово **As**, после которого указывается тип данной переменной. Это ключевое слово, которое при написании программы можно выбрать из контекстного меню (рис. 2.1).

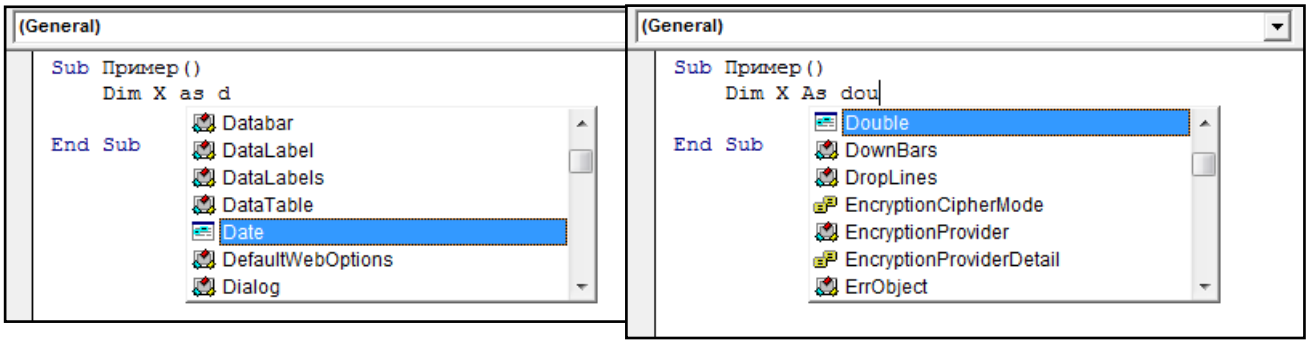


Рис. 2.1. Выбор ключевого слова из контекстного меню

Отдельно рассмотрим пример объявления двух ***констант***:

pi – числа **π;** **BDateTolstoy** – даты рождения Льва Толстого (по старому стилю):

```
    Const BDateTolstoy As Date = "28 августа 1828"

Sub Пример()
    Const pi As Double = 3.14159265358979
    MsgBox pi
    MsgBox BDateTolstoy
End Sub
```

Для проверки правильности объявления констант в процедуру макроса вставлен вызов встроенной процедуры «Окно сообщения» (MsgBox). После запуска макроса «Пример» получаем два окна сообщений (рис. 2.2).

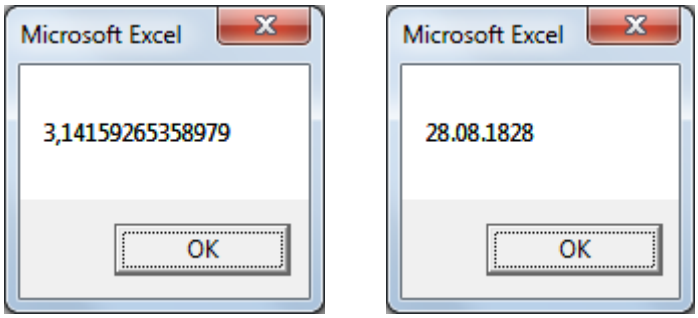


Рис. 2.2. Окна сообщений, появляющиеся после запуска макроса «Пример»

Ключевое слово **Public** служит для объявления глобальной переменной, область видимости которой простирается и на другие модули проекта. Для этого необходимо в других модулях обеспечить возможность ссылаться на эту переменную, указывая перед её именем имя модуля, в котором эта переменная объявляется.

Например, переменная X объявлена как глобальная в модуле **Module1**.

```
Public X As Integer
```

Чтобы сослаться на неё в модуле **Module2**, например прибавить к её значению единицу и результат возвести в квадрат, надо написать:

```
(Module1.X + 1) ^ 2.
```

Несколько переменных можно объявить на одной строке с помощью лишь одного ключевого слова **Dim** (или **Private**, или **Public**). Например:

```
Private x As Integer, y As Integer, N is Long
```

Особо отметим форму объявления пользовательского типа данных (на примере пользовательского типа **PersonData**):

```
Private Type PersonData
    INN As Double
    Fam As String * 30
    BDate As Date
    Tel As String * 20
    EMail As String * 50
End Type
```

Пользовательский тип данных, как было замечено выше, нужен для описания структуры таблицы базы данных. В данном случае, это таблица с данными о налогоплательщиках.

Строка этой таблицы состоит из пять полей:

- **INN** (8 символов для значения «ИНН» – 12-значного числа типа **Double**),
- **Fam** (30 знаков «фамилии» – строки типа **String**30***),
- **BDate** (8 символов «даты рождения» типа **Date**),
- **Tel** (20 символов «номера телефона» – строки типа **String**20***),
- **Email** (50 символов «адреса электронной почты» – строки типа **String**50***).

Таким образом, длина строки таблицы нашего примера составляет 116 символов. В программе, чтобы сослаться на какое-то поле этой таблицы, необходимо сначала объявить переменную типа **PersonData**, например:

```
    Dim Person As PersonData
```

Затем нужно обратиться к нужному полю. Например, напечатать ИНН и фамилию человека, данные о котором перед этим присвоены переменной **Person**:

```
    MsgBox Person.INN
    MsgBox Person.Fam
```

Как присвоить значение переменной, рассказано в следующем разделе.

2.4. Оператор присваивания переменной значения определённого типа

Оператор присваивания переменной значения – это «элементарное» действие исполнителя алгоритма (компьютера), при котором переменная становится «означенной», т.е. появление этой переменной в последующих инструкциях алгоритма будет означать, что мы имеем дело с этим значением. Для обозначения присваивания используется знак равенства «=», который ни в коем случае не означает математического равенства!

Например, запись программы X = X + 1 с математической точки зрения бессмысленна. А в программе на языке Visual Basic (VBA) эта строка означает, что переменной X присваивается новое значение, на единицу большее, чем старое значение.

Отметим, что можно «забыть» объявить переменную перед тем, как в программе будет применён оператор присваивания. В этом случае перемен-

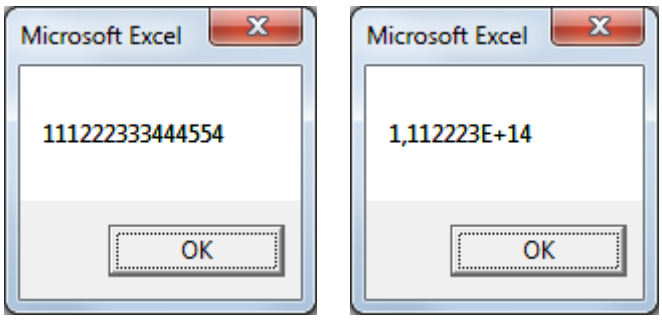
ной «по умолчанию» будет приписан тип значения, которое ей присваивается. Забегая вперёд, заметим, что часто «забывают» объявить тип различных индексов в операторах цикла (например, *i, j, k*). При задании начальных значений этих индексов, например, в строке: `For i = 1 To 10` - этому индексу по умолчанию приписывается тип целого числа.

Есть «устаревший» (редко применяемый сейчас) способ объявления типа переменной при первом её появлении в тексте программы – с помощью суффикса % (Integer), & (Long), ! (Single), # (Double), $ (String), @ (Currency).

Например, в упомянутой выше строке: `For i% = 1 To 10`.

Эти же суффиксы иногда автоматически приписываются значениям переменных, например, при переходе на новую строку после ввода строки с оператором присваивания. Рассмотрим **Пример 2.1**:

```
Sub Пример()
    N = 111222333444555#
    MsgBox N - 1
End Sub
```



а) *б*)

Рис. 2.3. Окна сообщений, появляющиеся после запуска двух вариантов макроса «Пример»

Переменной *N* при этом «по умолчанию» припишется тип **Double**, а запуск макроса «Пример» даст результат, показанный на рис. 2.3, *а*.

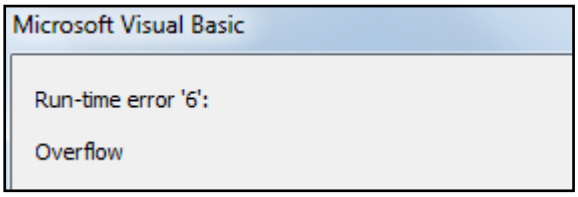
Если же переменной N заранее принудительно приписать другой тип, например, **Single**:

```
Sub Пример()
    N! = 111222333444555#
    MsgBox N - 1
End Sub
```

мы получим округлённый результат, показанный на рис. 2.3, *б*, который нас, возможно, и не устроит.

Попытка получить результат, приписав переменной N тип **Long**:

```
Sub Пример()
    N& = 111222333444555#
    MsgBox N - 1
End Sub
```



вообще заканчивается плачевно – при запуске макроса мы получим сообщение о «переполнении» (рис. 2.4).

Рис. 2.4. Окно с сообщением о переполнении при задании переменной типа **Long**

Действительно, этот тип переменной N не допускает её значения, которое превышает число 2 147 483 647.

Из приведенных примеров видно, что желательно (но не обязательно!), чтобы переменная и значение, которое ей присваивается, были одного типа.

2.5. Некоторые выражения и функции языка VBA

Подробно этот материал будет рассмотрен в следующей лекции, так как в языке VBA существует несколько видов выражений и функций. Здесь же будет рассмотрен только один вид выражений, которые называются ***строковыми*** (или ***текстовыми***), ***функции обработки строк*** и маленькое подмножество так называемых ***системных функций***: функция **InputBox** – ввода данных пользователем, а также функция **MsgBox** – выдачи сообщения пользователю. Отметим, что функция **MsgBox** может использоваться не только как функция, но и как встроенная процедура. В этой роли мы её уже демонстрировали в **Примере 2.1**, приведённом в предыдущем разделе.

2.5.1. Определение выражения (любого вида) и функции

Выражение – это строка на языке программирования, которая имеет то или иное значение. Значение обычно присваивается той или иной переменной. Выражение может быть простым или составным.

Простое выражение можно называть также ***термом***.

Составное выражение – это термы, связанные между собой ***операциями*** (иначе говоря, ***функциями***).

Термом можно считать и заключённое в скобки составное выражение.

При программировании макросов на языке VBA мы будем пользоваться тремя видами выражений: логическими; арифметическими; строковыми.

2.5.2. Строковые (текстовые) выражения

Важная операция над строками – это ***конкатенация*** (***склеивание***) строк. Обозначается эта операция знаком &, который читается как «амперсанд».

Пример 2.2. Пусть 5 переменных типа String получают следующие значения:

S1 = "Привет, ", S2 = "Иванов ", S3 = "Иван ", S4 = "Иванович ", S5 = "!".

В результате склеивания этих строк: S = S1 & S2 & S3 & S4 & S5 переменная S получает значение, которое появляется в окне сообщения после запуска макроса «Привет».

```
Sub Пример()
    S1 = "Привет, ": S2 = "Иванов "
    S3 = "Иван ": S4 = "Иванович ": S5 = "!"
    S = S1 & S2 & S3 & S4 & S5
    MsgBox S
End Sub
```

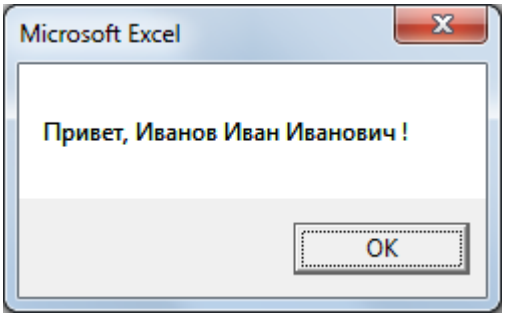


Рис. 2.5. Окно с сообщением о результате конкатенации (склеивания) строк

2.5.3. Функции обработки строк

Функции над строками (далеко не все, а только те, которые нам потребуются в этом курсе!) демонстрирует табл. 2.1.

Таблица 2.1

Функция	***Возвращаемое значение***
Len(S)	Длина строки. Пустая строка имеет длину 0
Mid(S, N, L)	Подстрока строки S, начиная с позиции N длиной L. Третий аргумент не обязателен
InStr(N, S1, S2)	Позиция начала подстроки S2 в строке S1. Поиск подстроки начинается с позиции N (этот аргумент не обязателен). Если S2 в S1 не найдена, функция возвращает 0

Пример 2.3. Создадим макрос, решающий следующую задачу.

Считаем, что исходная строка – это фамилия, имя и отчество – три подстроки, разделённые пробелами. Строка помещена в ячейку «A1» листа Excel.

Выделим в этой строке фамилию и поместим её в ячейку «A2»; имя поместим в ячейку «A3»; отчество поместим в ячейку «A4». Затем найдём длину исходной цепочки ФИО, длину фамилии, длину имени и длину отчества. Поместим результаты в ячейки «B1», «B2», «B3» и «B4».

Для решения используем функции обработки строк из приведённой выше таблицы. Используем «старинный» способ установки типов переменных – с помощью суффиксов $ и %.

Решение (текст макроса):

```
Sub ФИО()
    FIO$ = Range("A1")
        Range("B1") = Len(FIO)
    N1% = InStr(FIO, " ")
    Fam$ = Mid(FIO, 1, N1 - 1)
        Range("A2") = Fam
        Range("B2") = Len(Fam)
    IO$ = Mid(FIO, N1 + 1)
    N2% = InStr(IO, " ")
    Im$ = Mid(IO, 1, N2 - 1)
        Range("A3") = Im
        Range("B3") = Len(Im)
    Otch$ = Mid(IO, N2 + 1)
```

```
        Range("A4") = Otch
        Range("B4") = Len(Otch)
End Sub
```

Результат работы макроса «ФИО» показан на рис. 2.6.

Рис. 2.6. Фрагмент листа книги Excel с результатом работы макроса «ФИО»

2.5.4. Системная функция InputBox

Ввод строк пользователем реализуется с помощью системного окна ввода **InputBox**. Эта функция возвращает значение строкового типа, введенное пользователем после того, как появляется системное окно ввода (рис. 2.7).

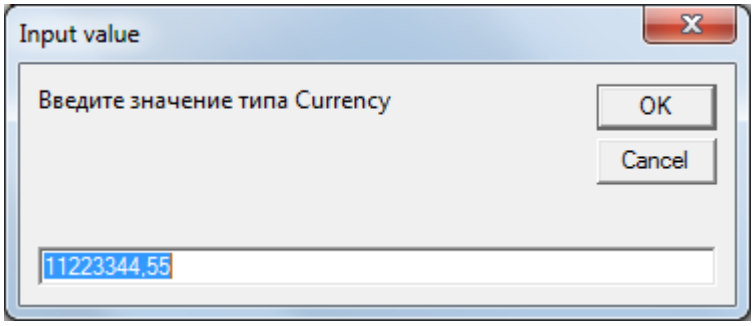


Рис. 2.7. Вид системного окна ввода

Синтаксис функции:

InputBox(<Подсказка> [, <Титул>, <Знач.по умолчанию>]).

2.5.5. Системная процедура MsgBox и системная функция MsgBox

Оператор вызова процедуры «по старинке» записывался в виде:

Call <имя процедуры>

С некоторых пор, уже лет 20 – 30, его записывают без ключевого слова Call, просто так:

<имя процедуры>

В примере кода:

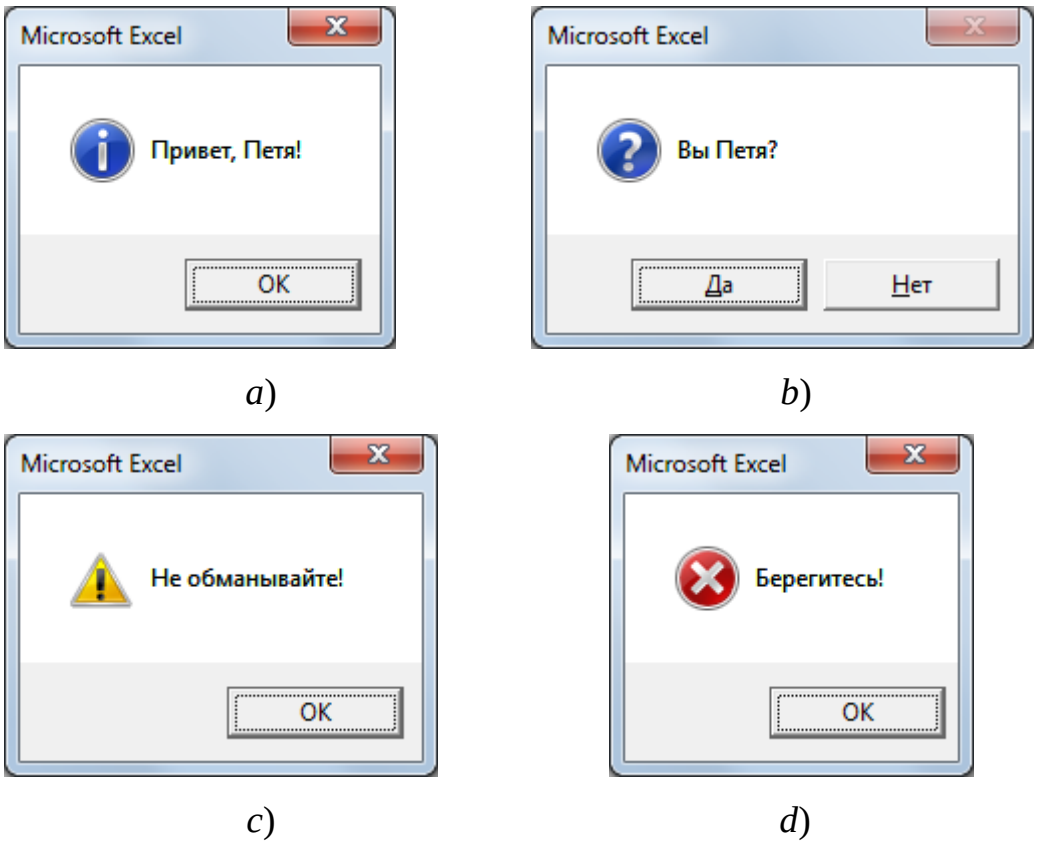
```
Строка = "Привет, Николай!"
  MsgBox  Строка
```

вызывается встроенная процедура MsgBox <выражение>, которая инициирует появление системного окна сообщения, и после того, как пользователь нажимает кнопку **OK**, это окно исчезает.

С помощью дополнительного (необязательного!) аргумента эту процедуру можно сделать более выразительной ☺ :

```
a) MsgBox   "Привет, Петя!",  vbInformation;
b) MsgBox   "Вы Петя?",  vbQuestion + vbYesNo;
c) MsgBox   "Не обманывайте!",  vbExclamation;
d) MsgBox   "Берегитесь!",  vbCritical
```

С использованием этих аргументов окна сообщения приобретают следующий, более выразительный вид:



a) *b*)

c) *d*)

Рис. 2.8. Варианты системных окон сообщения:

a) Information; *b)* Question; *c)* Exclamation; *d)* Critical

Кстати, ключевое слово языка Visual Basic **MsgBox** может выступать не в роли имени процедуры, а в роли имени функции. Например, в строке кода:

```
Ans = MsgBox("Вы Петя?", vbQuestion + vbYesNo)
```

эта функция возвращает значение в зависимости от набора кнопок – в данном случае, либо системную константу **vbYes**, либо системную константу **vbNo**.

Это значение в данном случае присваивается переменной **Ans.**

Отметим, что использование **MsgBox** в роли функции требует коррекции записи: аргументы этой функции записываются в скобках, в отличие от процедуры **MsgBox.**

Глава 3

Данная глава – это изложение материалов первой лабораторной работы, в которой закрепляется материал двух предыдущих лекций по курсу программирования на языке Visual Basic для офисных приложений. Содержание этих материалов можно структурировать следующим образом:

- Контроль знаний по теме лабораторной работы.
- Освоение VBE – среды программирования на языке VBA для офисного приложения Microsoft Excel.
- Освоение различных технологий программирования макросов (макрокоманд) на языке VBA for Excel.
- Примеры программирования процедур, использующих функции обработки строк – данных типа String – на языке VBA for Excel.

3.1. Контрольные вопросы по теме лабораторной работы

1. Что такое программный модуль и процедуры программного модуля?
2. Как с помощью вкладки «Разработчик» меню открытой новой книги приложения Excel войти в VBE – среду программирования «Visual Basic» и зайти в программный модуль одного из листов этой книги?
3. Как в открытом модуле листа Excel переходить из ***окна кода модуля*** (View Code) к окну ***модуля как объекта*** (View Object) и наоборот?
4. Как поместить на модуль как объект (т.е. непосредственно на лист книги Excel) элемент управления ActiveX (например, командную кнопку, метку, текстовое поле и т.д.)?
5. Какими способами можно начать создание процедуры для элемента управления ActiveX, размещённого на объекте модуля (в частности, непосредственно на листе книги Excel)?
6. Как создать процедуру для всей книги Excel без использования элементов управления – процедуру макрокоманды (макроса)?
7. Как включить в проект для данной книги Excel новый модуль – модуль пользовательской экранной формы (UserForm)?
8. Что такое переменная, используемая в программах процедур, и как она объявляется?

9. Приведите примеры типов данных, которые присваиваются переменным.
10. Как можно присвоить переменной значение, взятое из ячейки листа книги Excel?
11. Как поместить в ячейку листа книги Excel значение переменной?
12. Каковы функции обработки данных наиболее распространённого строкового типа?

3.2. Освоение VBE – среды программирования на языке VBA для офисного приложения Microsoft Excel

Задание 1. Откройте новую книгу MS Excel. С помощью вкладки «Разработчик» откройте окно «Microsoft Visual Basic». Другими словами, войдите в VBE (Visual Basic Environment) – среду программирования на языке Visual Basic. Затем зайдите в программный модуль одного из листов этой книги (рис. 3.1). В меню левой панели «Project – VBAProject» проверьте работу кнопок View Code и View Object (слева вверху).

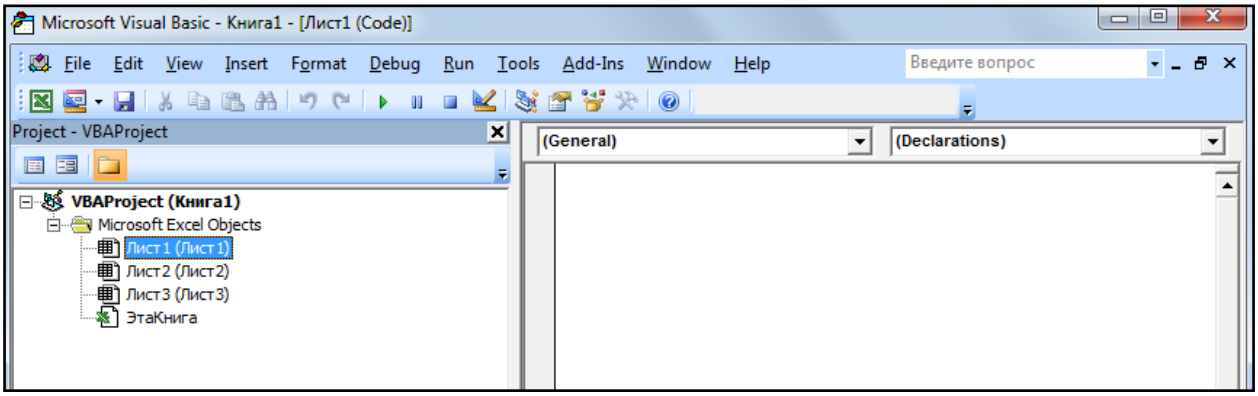


Рис. 3.1. Вид окна Microsoft Visual Basic для объекта Лист1 (нажата кнопка View Code)

Раскрывающиеся списки в окне View Code показаны на рис. 3.2.

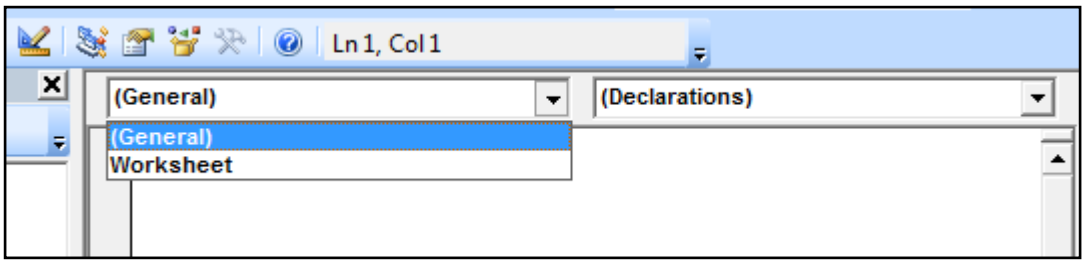


Рис. 3.2. Раскрывающиеся списки в окне View Code
(показано раскрытие списка объектов)

Вернувшись на ленту главного меню приложения, выберите вкладку «Разработчик» и нажмите кнопку «Вставить». На панели «Элементы управления» выберите вставляемый на Лист1 элемент управления. Например, окно списка ListBox1 (рис. 3.3). Разместите этот элемент в произвольном месте листа.

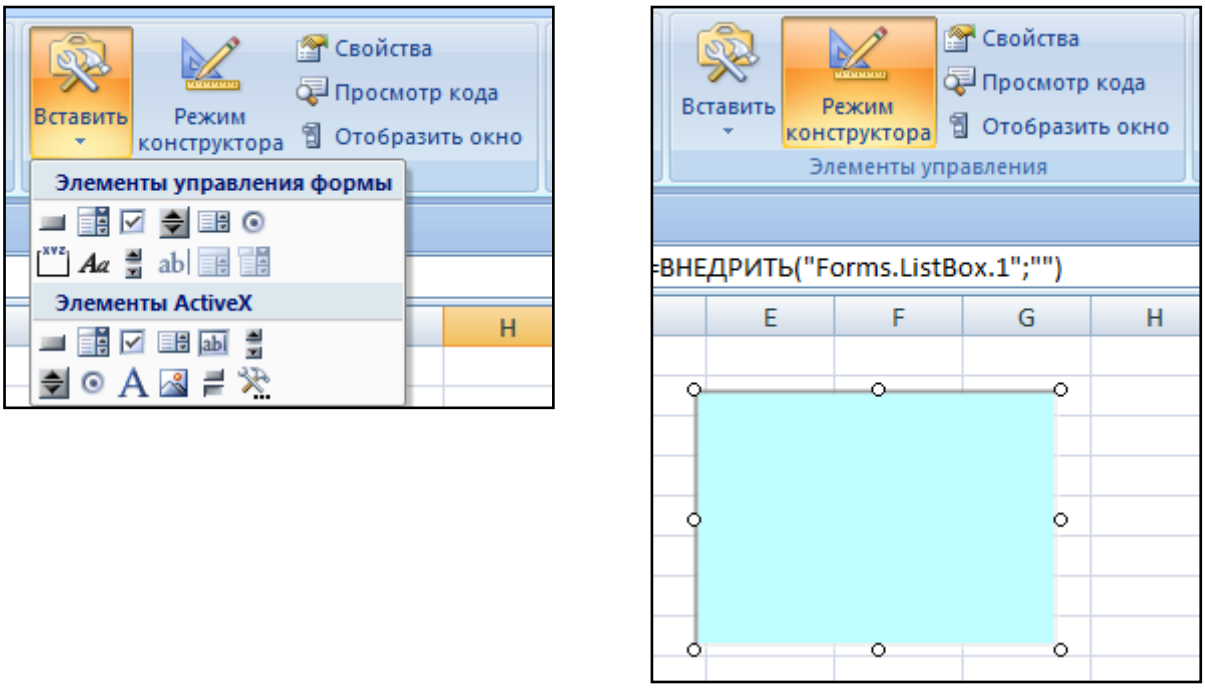


Рис. 3.3. Вставка элемента управления ListBox1 на Лист1

При этом, автоматически будет активирован «Режим конструктора». Нажав кнопку «Свойства», легко изменить некоторые свойства вставленного элемента ListBox1, например BackColor, как на рис. 3.3 справа.

3.3. Освоение различных технологий программирования макросов (макрокоманд) на языке VBA for Excel

Задание 2. В ячейки A1 – A3 Листа1 книги Excel, рассмотренной в задании 1, впишите ФИО троих человек, соответственно: «Исмаилов А.И.», «Кулиев Б.К.», «Сулейменов В.С.».

Создайте макрос (макрокоманду), выполнение которого (которой) заключается в следующем. Во-первых, окно списка на Листе1 очищается и, во-вторых, заполняется содержимым ячеек A1 – A3 Листа1.

Указание. Для записи в список очередного значения используется так называемый «метод» (встроенная процедура для данного элемента управле-

ния), который называется AddItem – добавить ещё одно значение в окно списка. Для очищения списка используется метод Clear.

Для создания макроса на вкладке «Разработчик» в разделе «Код» нажмите команду «Макросы». Появится панель «Макрос». Придумайте имя макроса, например «Список». Укажите, где находится макрос: «Эта книга». После чего на панели становится доступной кнопка «Создать». Щёлкнув её, вы попадаете в окно программного кода и записываете процедуру:

```
Sub Список()
    Лист1.ListBox1.Clear
    Лист1.ListBox1.AddItem Range("A1")
    Лист1.ListBox1.AddItem Range("A2")
    Лист1.ListBox1.AddItem Range("A3")
End Sub
```

В дальнейшем, после создания макроса, вид панели при её вызове будет таким как на рис. 3.4.

Выполнение макроса осуществляется с помощью кнопки «Выполнить» на этой панели.

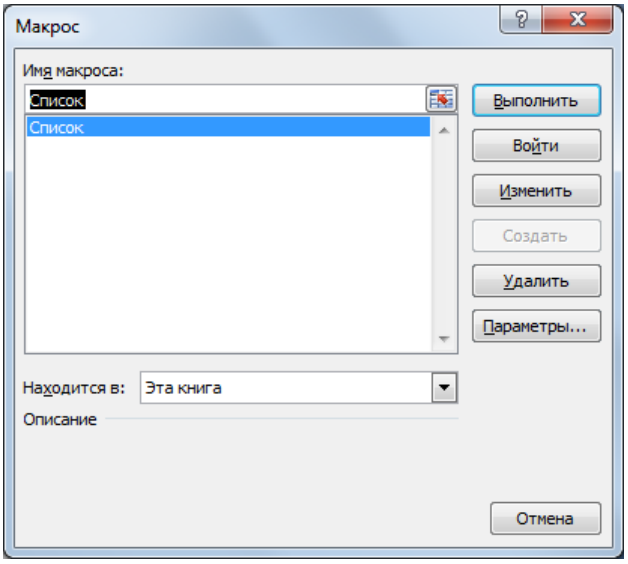


Рис. 3.4. Панель для работы с макросом

Но макрос может быть запущен и по-другому: с помощью вызова этой панели в разделе «Макросы» на вкладке «Вид» ленты главного меню.

И, наконец, для упрощения этого процесса можно реализовать вызов макроса «Список» просто щелчком по элементу ListBox1. (Разумеется, уже не в режиме «Конструктор» ☺.)

Для этого надо написать следующую процедуру в окне программного кода модуля Лист1:

```
Private Sub ListBox1_Click()
    Список
End Sub
```

Окно списка очищается (если в нём уже что-то было) и заполняется заново (рис. 3.5).

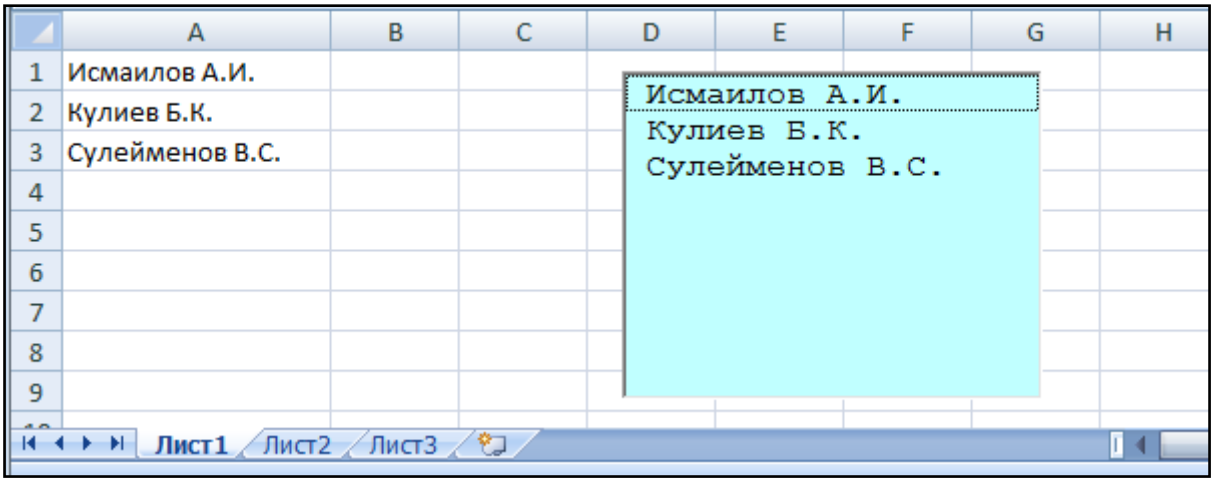
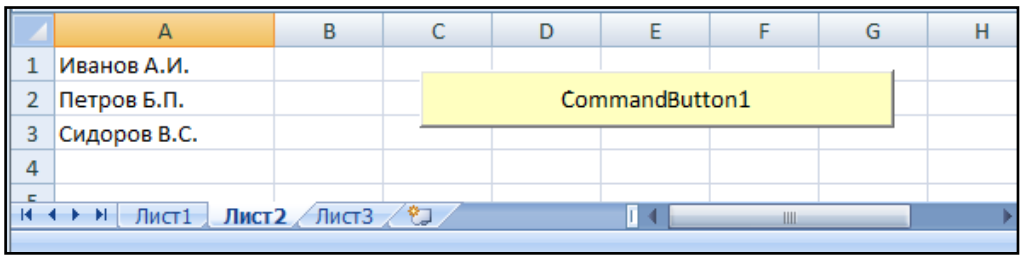


Рис. 3.5. Вид Листа1 после выполнения макроса «Список»

Задание 3. В ячейки А1 – А3 Листа2 книги Excel, рассмотренной в задании 1, впишите ФИО ещё троих человек, соответственно: «Иванов А.И.», «Петров Б.П.», «Сидоров В.С.» (рис. 3.6, *а*). В ячейки А1 – А3 Листа3 той же книги впишите ФИО также ещё троих человек, соответственно: «Михайлюченко А.М.», «Никоненко Б.Н.», «Романенко В.Р.» (рис. 3.6, *б*).

Установите на этих же двух листах (Лист2 и Лист3) книги Excel по одной командной кнопке с одинаковым именем CommandButton1.

а)



б)

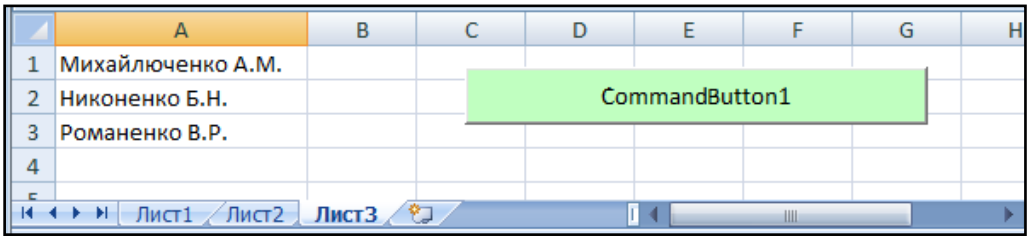


Рис. 3.6. Лист1 и Лист2 после установки на них командных кнопок

Отметим нижеследующее: несмотря на то, что имена командных кнопок одинаковы (CommandButton1), это разные объекты управления, так как они относятся к разным программным модулям: Лист2 и Лист3.

Создайте ещё два макроса: Список2 и Список3:

```
Sub Список2()
    Лист1.ListBox1.AddItem Лист2.Range("A1")
    Лист1.ListBox1.AddItem Лист2.Range("A2")
    Лист1.ListBox1.AddItem Лист2.Range("A3")
End Sub
```

```
Sub Список3()
    Лист1.ListBox1.AddItem Лист3.Range("A1")
    Лист1.ListBox1.AddItem Лист3.Range("A2")
    Лист1.ListBox1.AddItem Лист3.Range("A3")
End Sub
```

В соответствующих двух модулях (Лист2 и Лист3) напишите по одной процедуре.

```
Sub CommandButton1()
    Список2
End Sub
```

```
Sub CommandButton1()
    Список3
End Sub
```

Щелчки командных кнопок на каждом из двух этих листов будут вызывать выполнение макросов Список2 и Список3, которые будут «перебрасывать» данные из ячеек А1 – А3 каждого из двух этих листов в окно списка на 1-м листе.

Задание 4. Реализуйте ещё один способ обработки данных с помощью пользовательской экранной формы (UserForm). Пользовательская экранная форма – это новый объект, который можно включить в книгу Excel с помощью меню **Insert** в окне VBE (рис. 3.7).

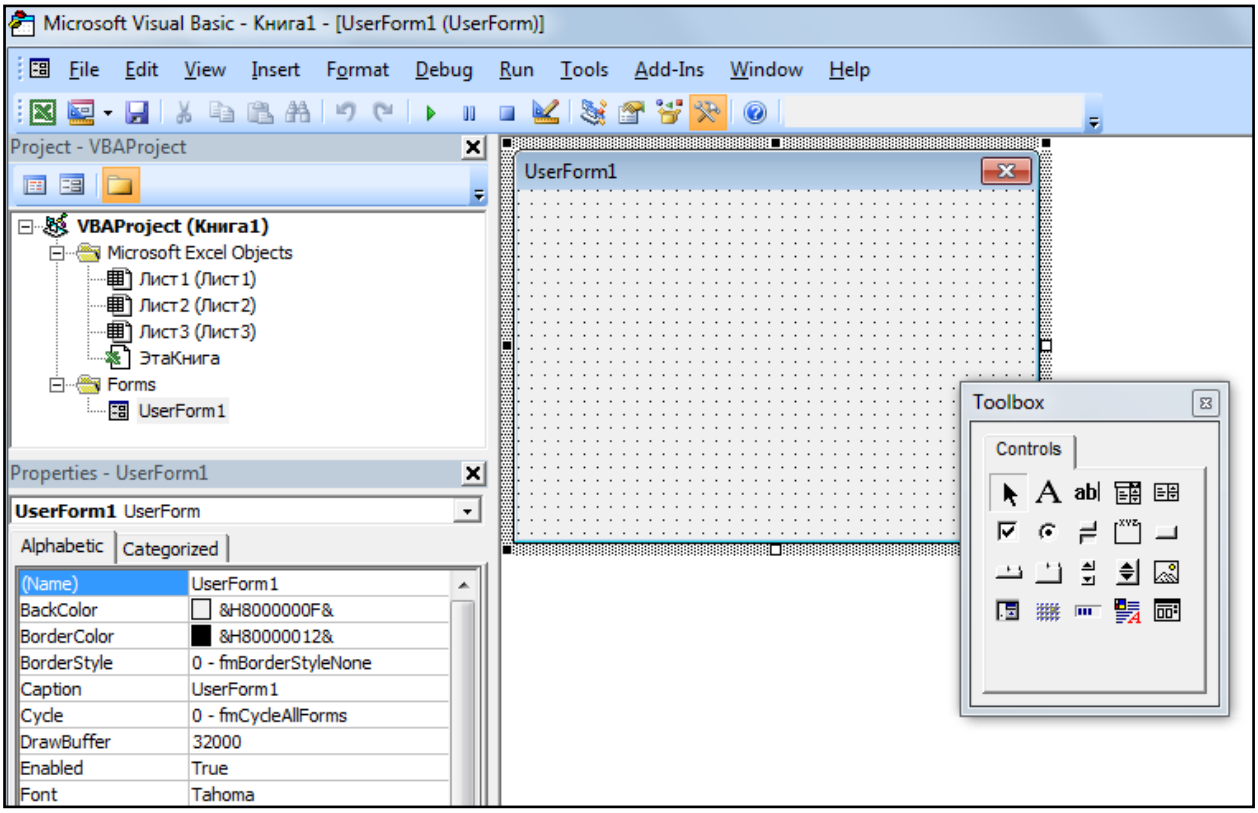


Рис. 3.7. Включение в проект модуля UserForm1

В задании 4 ставится задача вводить данные вручную с помощью Окна ввода, используя функцию InputBox. Например, это имена, отчества и фамилии произвольного числа русских писателей, которые в виде единого текста будут записаны в объекте «Текстовое поле» (TextBox1) на форме UserForm1.

Для ввода имён, отчеств и фамилий использовать три отдельные командные кнопки. После ввода данных об одном человеке следует указать, продолжится ли ввод о другом человеке (в этом случае будет введена запятая), или ввод закончится (в этом случае будет введена точка). Для этого нужно использовать ещё две командные кнопки.

С помощью окна свойств (Properties) для объекта TextBox1 (рис. 3.8) реализовать возможность многострочного представления получившегося текста (значение свойства MultiLine установить равным True).

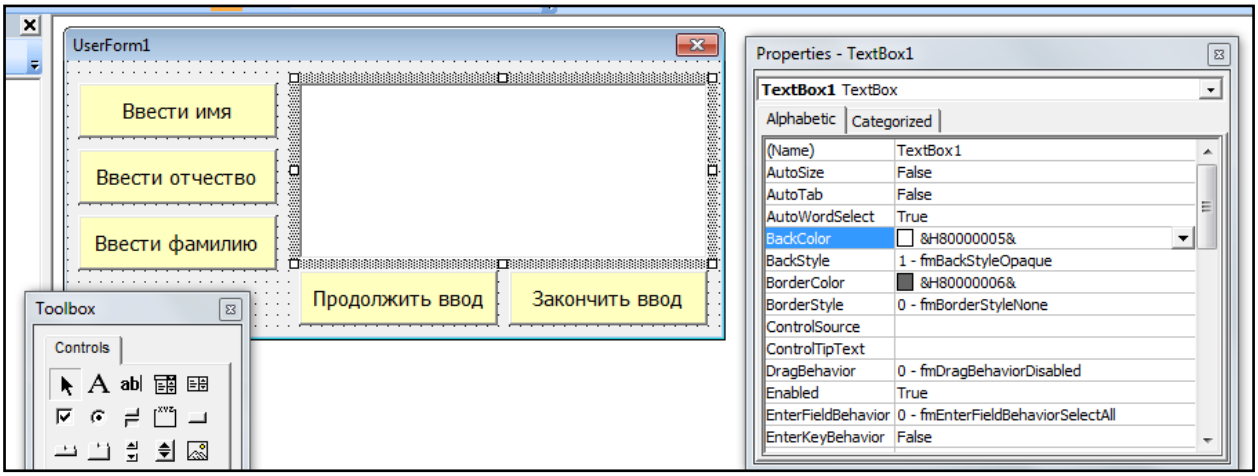


Рис. 3.8. Установка элементов управления на объекте UserForm1 и установка значений их свойств с помощью окна Properties

Для того чтобы начать работать с формой (разумеется, при отключении режима конструктора), следует её открыть. Например, с помощью макроса или с помощью кнопки, установленной на каком-либо листе книги:

```
Private Sub CommandButton1_Click()
    UserForm1.Show
End Sub
```

Ввод данных будет производиться после щелчков кнопок на форме UserForm1 под управлением процедур, которые следует создать в модуле этой формы:

```
Dim S As String
```

```
Private Sub UserForm_Activate()
    S = ""
    TextBox1.Text = S
End Sub
```

```
Private Sub CommandButton1_Click()
    S = InputBox("Введите имя")
    TextBox1.Text = TextBox1.Text & S & " "
End Sub
```

```
Private Sub CommandButton2_Click()
    S = InputBox("Введите отчество")
    TextBox1.Text = TextBox1.Text & S & " "
End Sub
```

```
Private Sub CommandButton3_Click()
    S = InputBox("Введите фамилию и " & _
    "щёлкните четвёртую или пятую кнопку")
    TextBox1.Text = TextBox1.Text & S
End Sub
```

```
Private Sub CommandButton4_Click()
    TextBox1.Text = TextBox1.Text & ", "
End Sub
```

```
Private Sub CommandButton5_Click()
    TextBox1.Text = TextBox1.Text & "."
End Sub
```

Получив результат (рис. 3.9), следует просто закрыть форму.

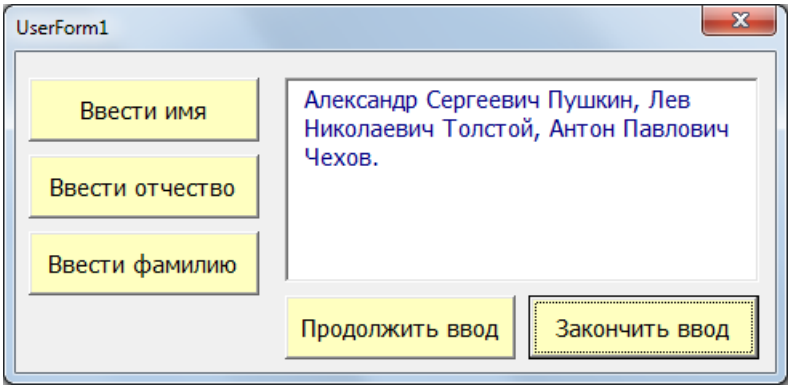


Рис. 3.9. Форма с полученным результатом – текстом в текстовом поле

Задание 5. Для сохранения полученного результата дополните процедуру щелчка кнопки CommandButton5 («Закончить ввод») строкой, в которой реализована запись текста в ячейку «A1» заданного листа данной книги. Результат должен быть следующим (рис. 3.10).

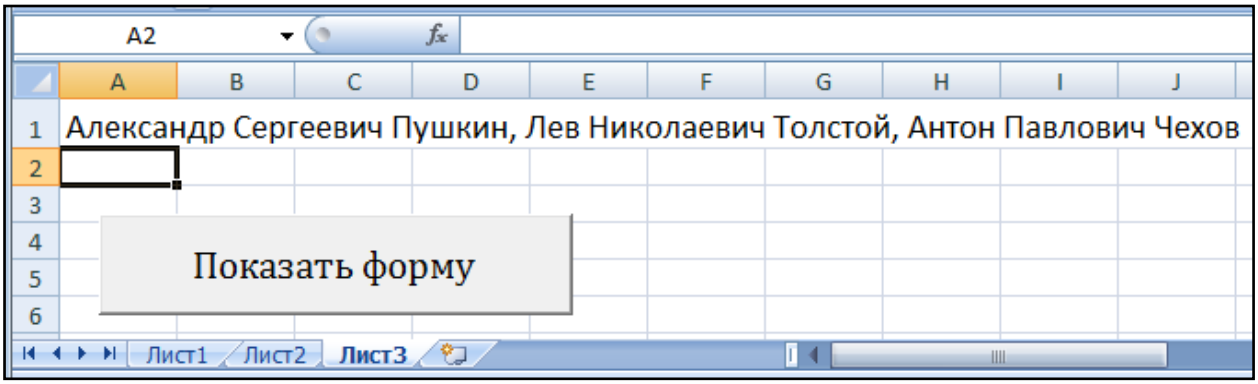


Рис. 3.10. Результаты, перенесённые с экранной формы на лист книги Excel

3.4. Примеры программирования процедур, использующих функции обработки строк – данных типа String – на языке VBA for Excel

Строковый тип данных – один из самых распространённых типов данных, используемых в программировании на языке VBA. Поэтому именно этот тип данных подробно рассматривается в 1-й лабораторной работе.

Задание 6. Рассматриваются основные (главные) функции обработки значений строкового типа (табл. 3.1).

Таблица 3.1

Функция	***Возвращаемое значение***
S1 & S2	Результат конкатенации (слияния) строк S1 и S2
Len(S)	Длина строки S. Пустая строка имеет длину 0
InStr(N, S1, S2)	Позиция начала подстроки S2 в строке S1. Поиск подстроки начинается с позиции N (этот аргумент не обязателен). Если S2 в S1 не найдена, функция возвращает 0
Mid(S, N, L)	Подстрока строки S, начиная с позиции N, длиной L (третий аргумент не обязателен)

Для закрепления смысла этих функций предлагается продемонстрировать на листе книги Excel непосредственное вычисление их значений так, как показано на рис. 3.11.

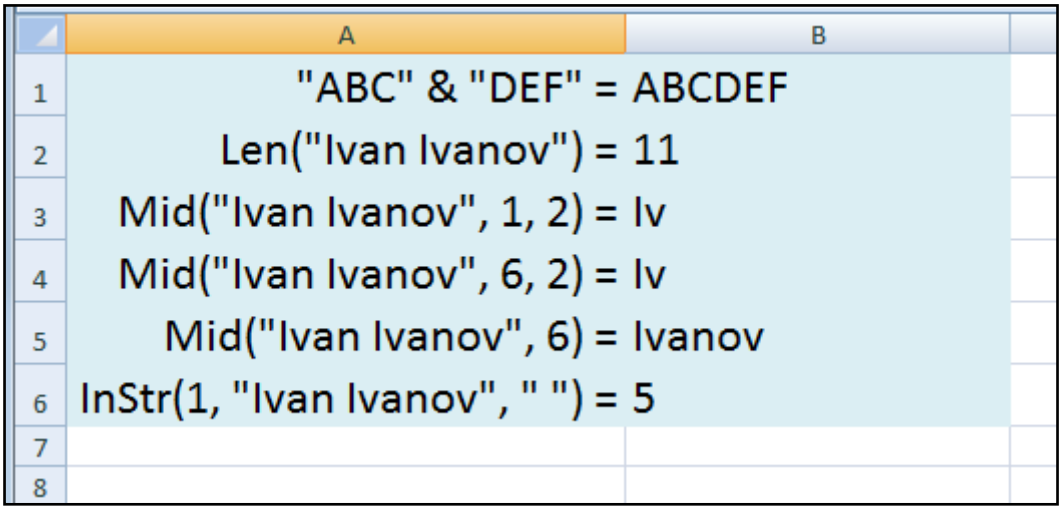
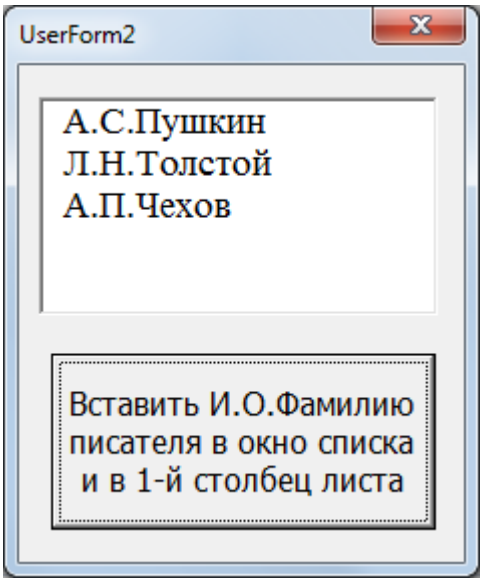


Рис. 3.11. Демонстрация работы функций обработки строк

Здесь в ячейках A1–A6 записаны функции, а в ячейках B1–B6 – возвращаемые значения, получаемые с помощью макроса:

```
Sub ФункцииОбработкиСтрок()
    Range("B1") = "ABC" & "DEF"
    Range("B2") = Len("Ivan Ivanov")
    Range("B3") = Mid("Ivan Ivanov", 1, 2)
    Range("B4") = Mid("Ivan Ivanov", 6, 2)
    Range("B5") = Mid("Ivan Ivanov", 6)
    Range("B6") = InStr(1, "Ivan Ivanov", " ")
End Sub
```



Задание 7. Используя полученный в результате выполнения ***заданий 4 и 5*** текст, проанализируйте этот текст и выделите в нём инициалы и фамилии русских писателей, которые поместите: в Окно списка на объекте UserForm1 (рис. 3.12), а также в столбец «А» листа книги (рис. 3.13).

Рис. 3.12. Результат в Окне списка на форме

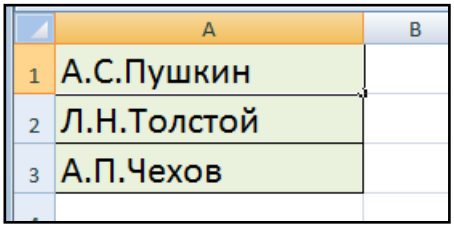


Рис. 3.13. Результат в столбце листа книги Excel

Далее – фрагмент процедуры, решающей данную задачу только для одного писателя. Остальные части кода (для двух других писателей) легко создаются путём копирования и небольшой коррекции данного фрагмента (последняя фамилия заканчивается не запятой, а точкой).

```
Dim S As String, S1 As String
Dim S2 As String, S3 As String
Dim N As Integer

Private Sub CommandButton1_Click()
    S = Лист1.Cells(1, 1)
```

```
    MsgBox S
        S1 = Mid(S, 1, 1) & "." ' Первая буква имени
        N = InStr(1, S, " ")    ' Позиция пробела
        S = Mid(S, N + 1)
        S2 = Mid(S, 1, 1) & "." ' Первая буква отчества
        N = InStr(1, S, " ")    ' Позиция пробела
        S = Mid(S, N + 1)
        N = InStr(1, S, ", ")   ' После Фамилии запятая
        S3 = Mid(S, 1, N - 1)   ' Фамилия (целиком)
        S = Mid(S, N + 2)
        ListBox1.AddItem S1 & S2 & S3
        Cells(1, 1) = S1 & S2 & S3
   …  <другие фрагменты – для других писателей>
End Sub
```

Линейный рост длины программы от числа фамилий в заполняемых списках, конечно же, наивен. Напрашивается вывод о разумности и необходимости перехода от линейных алгоритмов к нелинейным – к использованию условных и безусловных переходов, циклических операторов (операторов повторений) типа For … Next, Do … Loop и т.д. и т.п. Что и будет темой дальнейших лекций и практических занятий по данному курсу.

Глава 4

Данная глава – это текст следующей, третьей лекции по курсу программирования на языке Visual Basic для офисных приложений. Содержание лекции можно структурировать следующим образом:

- Синтаксис и семантика оператора условного перехода в языке Visual Basic (VBA for Excel).
- Примеры использования оператора условного перехода в многострочной форме (задача о пенсионном возрасте) и в однострочной форме (нахождение максимального из нескольких чисел).
- Оператор безусловного перехода GoTo; его использование, совместно с оператором условного перехода If … Then … Else, для реализации повторяющихся (циклических) действий.
- Использование оператора безусловного перехода совместно с оператором условного перехода для реализации повторяющихся (циклических) действий – на примере обработки текста с произвольным числом фамилий с инициалами.
- Оператор Select Case, использующийся для выбора альтернативных операторов.

4.1. Синтаксис и семантика оператора условного перехода в языке Visual Basic (VBA for Excel)

В прошлой лекции был рассмотрен лишь один оператор языка Visual Basic, в частности, языка VBA – оператор присваивания, обозначаемый знаком равенства =. С помощью только одного этого оператора можно программировать лишь *линейные алгоритмы*, когда инструкции выполняются строго одна за другой.

Таким алгоритмам, как говорится, «грош цена». Ведь даже для очень простых «практических» задач программирования приходится составлять *нелинейные алгоритмы*, когда выполнение многих инструкций зависит от выполнения или невыполнения каких-то условий.

При этом чаще всего используется ***оператор условного перехода***.

Форма записи (синтаксис) этого оператора такова:

If <*условное выражение*> **Then** <*оператор*> [**Else** <*оператор*>],

где **If, Then** и **Else** – *ключевые слова* языка VBA, а квадратные скобки указывают на то, что конструкция внутри них необязательна.

Это так называемая *однострочная форма* записи оператора условного перехода. Если внутри условного оператора после слов **Then** и **Else** операторов много, тогда лучше использовать *многострочную форму*:

If <*условное выражение*> **Then**

<*оператор*>

[<*другие операторы*>]

[**Else**

<*оператор*>

[<*другие операторы*>]]

End If

Внутри оператора перед словом **Else** допускается использование любого числа дополнительных проверок с помощью ключевого слова **ElseIf:**

[**ElseIf** <*условное выражение*> **Then**

<*оператор*>

[<*другие операторы*>]]

(**ElseIf** – так же, как и слово Else, является ключевым словом языка.)

Теперь о *семантике* (т.е. о *смысле*) условного оператора.

1. Если условное выражение после слова **If** (или слова **ElseIf**) имеет значение **True** (условие выполнено), то выполняются операторы после слова **Then**. После чего работа условного оператора заканчивается. Если же условное выражение имеет значение **False** (условие не выполнено), то проверяем следующее условие (если оно есть) – переходим к пункту 1. Если же все проверки сделаны, т.е. нет больше строк, начинающихся со слова **ElseIf**, то
2. Выполняются операторы после слова **Else** (если, конечно, это слово есть). После чего работа условного оператора заканчивается. Если слова **Else** нет, то работа условного оператора тоже сразу заканчивается.

4.2. Примеры использования оператора условного перехода в многострочной форме (задача о пенсионном возрасте) и в однострочной форме (задача нахождения максимального из нескольких чисел)

Пример 4.1. Создадим макрос, который решает следующую задачу: определяет, пенсионер ли пользователь[2]. Для пользователя удобно вводить данные и получать результат на экранной форме (рис. 4.1).

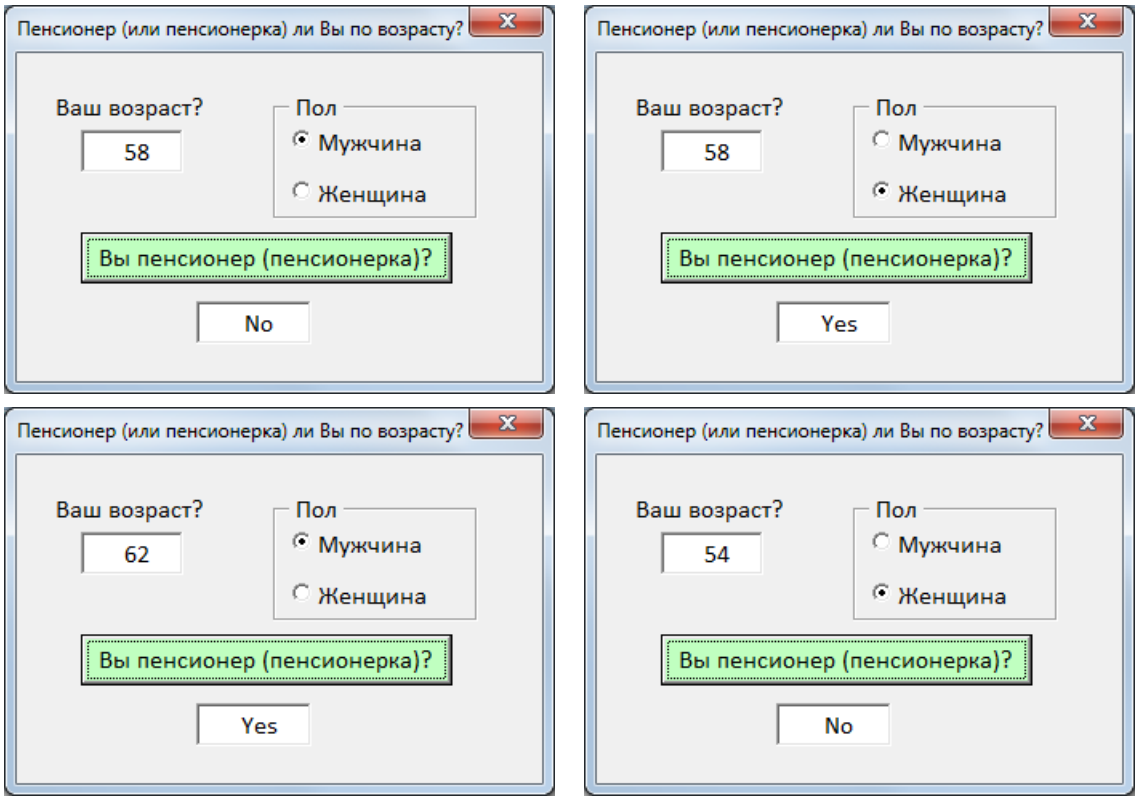


Рис. 4.1. Форма для получения ответа на вопрос, является ли пользователь пенсионером по возрасту

Далее, текст макроса – процедуры для щелчка единственной командной кнопки:

```
Private Sub CommandButton1_Click()
    Dim Age As Byte        ' Возраст: тип Byte
    Dim Female As Boolean  ' ЖенПол: тип Boolean,
                ' Famale = True, если нажата
                ' 2-я опциональная кнопка
    Age = Val(TextBox1.Text)
        ' Val - преобразование строки в число
      TextBox2.Text = "No"
      Female = OptionButton2.Value
```

[2] Отметим, что в Российской Федерации у мужчин пенсионный возраст составляет 60 лет; у женщин – 55 лет. (Данные на 2017 год.)

```
            ' Значение переменной Female = True,
            ' если нажата 2-я опциональная кнопка
        If Female And Age > 55 Then
            TextBox2.Text = "Yes"
        ElseIf Age > 60 Then
            TextBox2.Text = "Yes"
        End If
End Sub
```

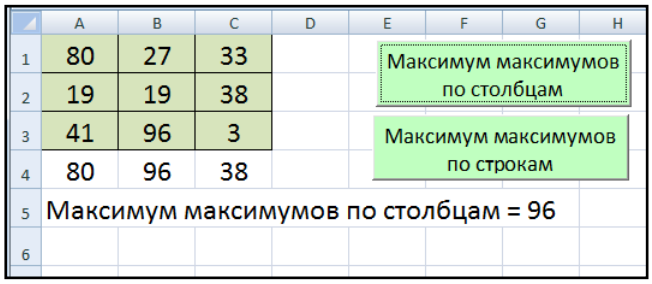
Здесь использована многострочная форма записи оператора условного перехода, требующая завершающей строки **End If**.

Пример 4.2. Решим задачу нахождения максимального из 9 чисел, находящихся в матрице 3x3 – ячейках диапазона A1 – C3 листа книги Excel, используя однострочную форму оператора условного перехода.

Будем использовать два алгоритма: (1) поиска максимальных элементов в каждом столбце матрицы 3x3, а затем – поиска максимального значения из трёх найденных; (2) поиска максимальных элементов в каждой строке матрицы 3x3, а затем – поиска максимального значения из трёх найденных.

Результаты будем фиксировать на листе книги Excel в виде, показанном на рис. 4.2 (*а*, *б*).

а)



б)

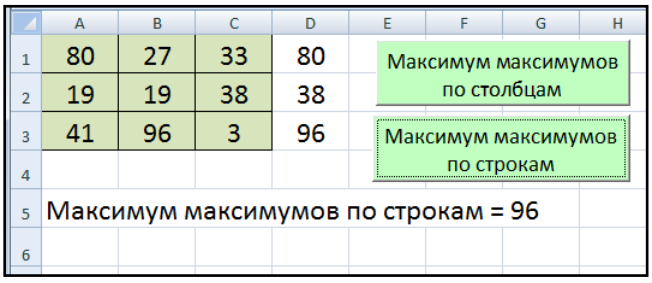


Рис. 4.2. Вычисление максимального значения как максимума максимумов:
а) по столбцам матрицы 3x3 и б) по строкам матрицы 3x3

Оба алгоритма можно реализовать в виде двух процедур для командных кнопок, показанных на рис. 4.2:

```
Private Sub CommandButton1_Click()
       Dim Max As Integer, Max1 As Integer, _
       Max2 As Integer, Max3 As Integer
    Max1 = Cells(1, 1)
        If Cells(2, 1) > Max1 Then Max1 = Cells(2, 1)
        If Cells(3, 1) > Max1 Then Max1 = Cells(3, 1)
        Cells(4, 1) = Max1
    Max2 = Cells(1, 2)
        If Cells(2, 2) > Max2 Then Max2 = Cells(2, 2)
        If Cells(3, 2) > Max2 Then Max2 = Cells(3, 2)
        Cells(4, 2) = Max2
    Max3 = Cells(1, 3)
        If Cells(2, 3) > Max3 Then Max3 = Cells(2, 3)
        If Cells(3, 3) > Max3 Then Max3 = Cells(3, 3)
        Cells(4, 3) = Max3
    Max = Max1
        If Max2 > Max Then Max = Max2
        If Max3 > Max Then Max = Max3
       Cells(5, 1) = _
      "Максимум максимумов по столбцам = " & Max
End Sub
```

```
Private Sub CommandButton2_Click()
       Dim Max As Integer, Max1 As Integer, _
       Max2 As Integer, Max3 As Integer
    Max1 = Cells(1, 1)
        If Cells(1, 2) > Max1 Then Max1 = Cells(1, 2)
        If Cells(1, 3) > Max1 Then Max1 = Cells(1, 3)
        Cells(1, 4) = Max1
    Max2 = Cells(2, 1)
        If Cells(2, 2) > Max2 Then Max2 = Cells(2, 2)
        If Cells(2, 3) > Max2 Then Max2 = Cells(2, 3)
        Cells(2, 4) = Max2
    Max3 = Cells(3, 1)
        If Cells(3, 2) > Max3 Then Max3 = Cells(3, 2)
        If Cells(3, 3) > Max3 Then Max3 = Cells(3, 3)
        Cells(3, 4) = Max3
    Max = Max1
        If Max2 > Max Then Max = Max2
        If Max3 > Max Then Max = Max3
       Cells(5, 1) = _
      "Максимум максимумов по строкам = " & Max
End Sub
```

4.3. Оператор безусловного перехода GoTo; его использование совместно с оператором условного перехода If … Then … Else для реализации повторяющихся (циклических) действий

Иногда (хотя это бывает крайне редко) в программировании на языке Visual Basic используется *оператор безусловного перехода*:

GoTo <метка>

Основное его предназначение – это обеспечение экстренного выхода из процедуры при возникновении критической ситуации, которая может привести к аварийной остановке работы программы.

Он может быть вписан в произвольное место программы и обеспечивает безусловный переход к тому месту в программе, перед которым поставлена специальная <метка>. Это, чаще всего, какая-нибудь буква или буква с номером. После метки ставится двоеточие.

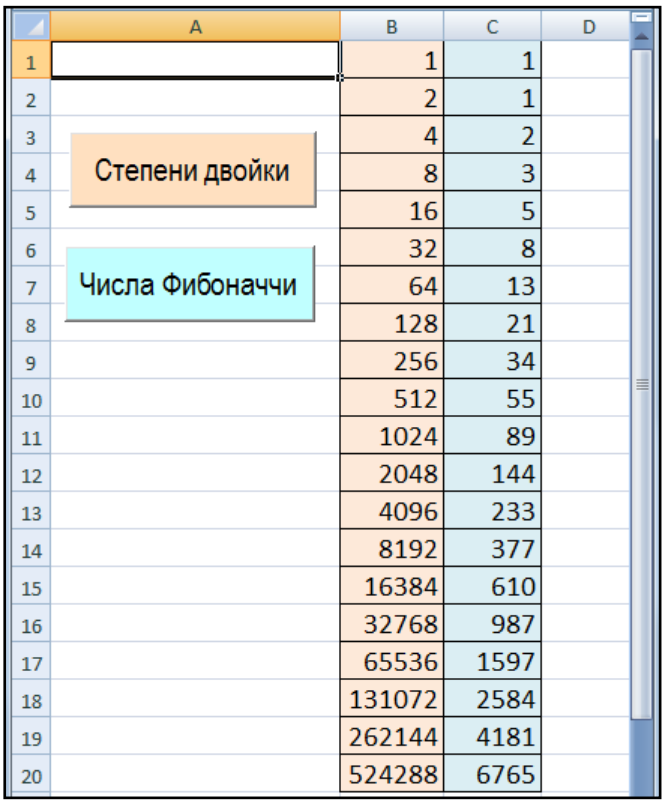
С помощью оператора безусловного перехода можно реализовать «движение по замкнутому кругу». Два примера приведены ниже.

Пример 4.3. Будем вписывать в ячейки какого-нибудь столбца листа книги Excel все возрастающие значения степеней двойки (2, 4, 8, 16 и т.д.). Условием циклического возобновления этого процесса можно сделать, например, следующее: «Продолжим процесс, если очередное значение 2^N, допустим, не превышает миллион».

```
        ' Выдача степеней двойки от 0-й (2^0 = 1) до
        ' предельного значения (2^N > 10000000):
Sub БезусловныйПереход2()
    Dim N As Integer, X As Long
    N = 1
    X = 2 ^ 0
M1:
    Cells(N, 2) = X      ' Или так: Range("B" & N) = X
    X = 2 ^ N
    N = N + 1
    If X <= 1000000 Then GoTo M1
End Sub
```

Пример 4.4. Программно будем вписывать в ячейки какого-нибудь столбца листа книги Excel числа *ряда Фибоначчи*[3] (1, 1, 2, 3, 5, 8 и т.д.).

```
    ' Выдача чисел Фибоначчи от 1-го номера (N = 1) до
    ' предельно допустимого номера N (N = 20):
Sub БезусловныйПереход1()
    Dim N As Integer, X1 As Long, X2 As Long
    N = 1
    X1 = 1: X2 = 1
M1:                     ' Метка для безусловного прехода:
    Cells(N, 3) = X1 ' Или так: Range("C" & N) = X
    Cells(N + 1, 3) = X2
    X1 = X1 + X2
    X2 = X2 + X1
    N = N + 2
    If N < 20 Then GoTo M1
End Sub
```



	A	B	C	D
1		1	1	
2		2	1	
3		4	2	
4		8	3	
5		16	5	
6		32	8	
7		64	13	
8		128	21	
9		256	34	
10		512	55	
11		1024	89	
12		2048	144	
13		4096	233	
14		8192	377	
15		16384	610	
16		32768	987	
17		65536	1597	
18		131072	2584	
19		262144	4181	
20		524288	6765	

Результаты работы двух процедур для обоих примеров (4.3 и 4.4) представлены на рис. 4.3.

Рис. 4.3. Результаты работы процедур примеров 4.3 и 4.4

[3] Для тех, кто не знает: ряд Фибоначчи строится следующим образом. Первые два числа, по определению, это 1 и 1. Каждое следующее число равно сумме двух предыдущих чисел: 2 = 1 + 1, 3 = 1 + 2, 5 = 2 + 3 и т.д. Использование ряда Фибоначчи в физике, технике, биологии, архитектуре и т.д. очень широко.

4.4. Использование оператора безусловного перехода совместно с оператором условного перехода для реализации повторяющихся (циклических) действий на примере обработки текста с произвольным числом фамилий с инициалами

Пример 4.5. Рассмотрим более сложный пример совместного использования условного и безусловного переходов.

Пусть в текстовом поле TextBox1, установленном непосредственно на листе книги Excel, размещён текст, представляющий собой перечень русских писателей в форме: <*И.О.Фамилия*{,|.}>. Фигурные скобки здесь означают, что после каждой фамилии стоит либо запятая, либо (в конце текста) точка. Кроме того, отметим, что число пробелов в тексте может быть любым.

На рис. 4.4 в текстовом поле справа вверху пример такого текста.

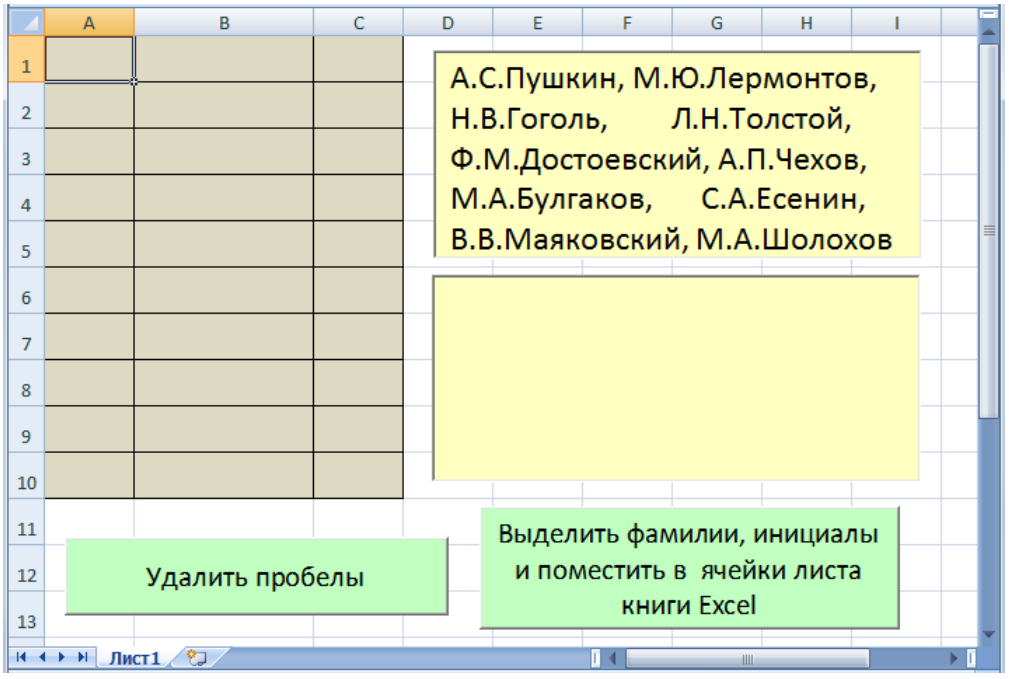


Рис. 4.4. Лист книги Excel, подготовленный для работы, предусмотренной примером 4.5

Задача состоит в том, чтобы (1) убрать все пробелы из текста и (2) поместить фамилии писателей в ячейки столбца «B» листа, а их инициалы – в ячейки столбца «C». Столбец «A» будет содержать номера в списке писателей, отсортированных по алфавиту средствами приложения Excel.

Разумеется, для решения этой задачи необходимо применить функции обработки строк, рассмотренные на предыдущих занятиях.

После щелчка кнопки «Удалить пробелы» второе текстовое поле должно стать таким, как показано на рис. 4.5.

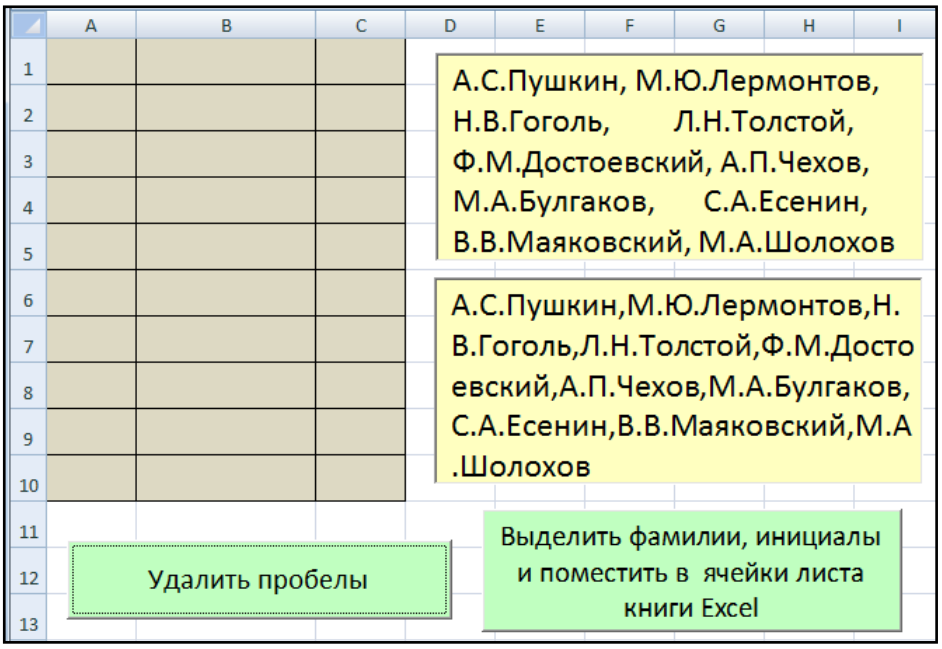


Рис. 4.5. Лист книги Excel после щелчка кнопки «Удалить пробелы»

Текст процедуры макроса для щелчка кнопки «Удалить пробелы»:

```
Private Sub CommandButton1_Click()
    S = TextBox1.Text
    S1 = ""
M1:
    N = InStr(1, S, " ")
    If N = 0 Then
        S1 = S1 & S
        GoTo M2
    End If

    S1 = S1 & Mid(S, 1, N - 1)
    S = Mid(S, N + 1)
    GoTo M1
M2:
    TextBox2.Text = S1
End Sub
```

Отметим, что оператор безусловного перехода здесь используется дважды!

Щелчок второй командной кнопки инициирует заполнение ячеек листа столбцов «B» и «C»:

```
Private Sub CommandButton2_Click()
    K = 0
```

```
    S = TextBox2.Text
    S1 = ""
    ' MsgBox S
M1:
    K = K + 1
    N = InStr(1, S, ",")

    If N = 0 Then
        S1 = S1 & S
        Cells(K, 2) = Mid(S, 5)
        Cells(K, 3) = Mid(S, 1, 4)
        GoTo M2
    End If
    S0 = Mid(S, 1, N - 1)
    Cells(K, 2) = Mid(S0, 5)
    Cells(K, 3) = Mid(S0, 1, 4)
    S1 = S1 & S0
    S = Mid(S, N + 1)
    GoTo M1
M2:
End Sub
```

Хотя это и не обязательно, обе процедуры желательно дополнить объявлением общих переменных.

```
Dim S As String, S0 As String, S1 As String
Dim N As Integer, K As Integer
```

Напомним, что часть этих переменных, хотя они и являются общими для указанных двух процедур, не следует относить к глобальным, так как обе процедуры принадлежат к одному программному модулю Лист1.

После щелчка кнопки «**Выделить фамилии, инициалы и поместить в ячейки листа книги Excel**» текстовое поле **TextBox2** должно стать таким, как показано на рис. 4.6.

	A	B	C
1	7	Булгаков	М.А.
2	3	Гоголь	Н.В.
3	5	Достоевский	Ф.М.
4	8	Есенин	С.А.
5	2	Лермонтов	М.Ю.
6	9	Маяковский	В.В.
7	1	Пушкин	А.С.
8	4	Толстой	Л.Н.
9	6	Чехов	А.П.
10	10	Шолохов	М.А.

Рис. 4.6. Лист книги Excel после щелчка второй командной кнопки

4.5. Оператор Select Case, использующийся для выбора альтернативных операторов

Select Case – это еще один условный оператор. Он используется в том случае, когда в зависимости от значения какого-то выражения выполняются альтернативные операторы.

«Альтернативные операторы» в данном случае – это просто присваивание переменной, например, R разных значений.

Синтаксис данного оператора следующий.

Select Case *<выражение>*

Case *<диапазон значений>*

<операторы>

Case *<диапазон значений>*

<операторы>

…

Case Else

<операторы>

End Select

Выражением может быть любое арифметическое или иное выражение. Его значение и будет определять выбор дальнейшей работы программы. Чаще всего в роли выражения выступает единственная переменная.

Диапазон значений – это либо просто одно из значений выражения, либо границы некоторой области значений. Многоточие «…» в записи синтаксиса оператора **Select Case** означает, что две строки

Case <*диапазон значений*>

<*операторы*>

могут повторяться произвольное число раз.

Семантика оператора **Select Case** такова. Когда в программе очередь доходит до выполнения данного оператора, прежде всего, вычисляется значение выражения. Его значение начинает по очереди сравниваться с диапазонами значений, записанными вслед за ключевым словом Case. Под «сравнением» понимается проверка того, попадает или нет значение выражения в ту область, которую определяет диапазон значений. В общем случае диапазоном значений может быть специальное выражение, которое указывает на границы области возможных значений.

Далее на примерах приведены наиболее распространенные специальные выражения (табл. 4.1).

Таблица 4.1

Диапазон значений (выражение после слова Case)	**Что может входить в диапазон значений**
-10, 10, 30, 50	Одно из указанных чисел
50 To 100	Любое число между 50 и 100
Is > 200	Любое число, превышающее 200
Is <> «Путин В.В.»	Любая строка, отличная от «Путин В.В.»
-10, 10, 30, 50 To 100, Is > 200	Любое число из первых трёх примеров

Лучше всего показать действие оператора выбора **Slect Case** на конкретном примере.

Пример 4.6. Чтобы получить значение лингвистической переменной «Рост» по введенному значению роста взрослого мужчины в сантиметрах, можно воспользоваться следующей процедурой:

```
Sub Рост()
    S = "Вы человек "
    i = 1
M1:
  V = Val(InputBox("Введите ваш рост в сантиметрах"))
    i = i + 1
     Select Case V
        Case Is < 150
            r = S & "маленького роста"
            Cells(i, 3) = r & ": " & V
        Case 151 To 178
            r = S & "среднего роста"
            Cells(i, 3) = r & ": " & V
        Case 179 To 250
            r = S & "высокого роста"
            Cells(i, 3) = r & ": " & V
        Case Else
            r = S & "неправдоподобно высокого роста"
            Cells(i, 3) = r & ": " & V
  End Select
  If MsgBox("New?", vbQuestion + vbYesNo) = vbYes  Then
     GoTo M1
  End If
End Sub
```

Результат работы программы показан на рис. 4.7.

	A	B	C	D
1			<Значение лингвистической переменной "Рост">: <Рост в см>	
2			Вы человек маленького роста: 149	
3			Вы человек среднего роста: 165	
4			Вы человек высокого роста: 188	
5			Вы человек высокого роста: 219	
6			Вы человек неправдоподобно высокого роста: 280	
7				
8				
9				
10				

Рис. 4.7. Лист книги Excel с результатом работы процедуры «Рост»

Глава 5

Данная глава – это текст четвёртой лекции по курсу программирования на языке Visual Basic для офисных приложений. Тема лекции «Выражения и функции языка VBA». Содержание лекции можно структурировать следующим образом:

- Наиболее распространённые виды выражений и функций в языке Visual Basic для офисных приложений.
- Логические выражения и логические (булевы) функции.
- Арифметические выражения и математические функции.
- Функции преобразования типов данных.
- Функции обработки и представления значений дат и времени.
- Функции, используемые при программировании финансовых операций.
- Функции, определяемые пользователем (user defined functions).

5.1. Наиболее распространённые виды выражений и функций в языке Visual Basic для офисных приложений

В прошлых лекциях был подробно рассмотрен лишь один вид выражений и функций – для представления и обработки строк. И лишь вкратце упоминались примеры арифметических (а точнее, математических) функций, например, операции сложения для увеличения номера шага **i** в повторяющемся процессе: **i = i + 1**, а также логических функций, например, операции сравнения числовых значений в условном операторе **If … Then … Else …**, например, **If** a > b **Then** Max = a **Else** Max = b.

В настоящей лекции более подробно будут рассмотрены наиболее распространенные логические и математические выражения и функции, а также и некоторые другие типы выражений и функций.

Прежде всего, напомним определение выражения, данное в лекции 2 (раздел 2.5.1).

Выражение – это строка на языке программирования, которая имеет то или иное значение. Значение обычно присваивается той или иной переменной. Выражение может быть простым или составным. Простое выражение

называют также термом. Составное выражение – это термы, связанные между собой операциями (иначе говоря, ***функциями***). Термом можно считать и заключённое в скобки составное выражение.

При программировании макросов на языке VBA мы будем пользоваться, в основном, тремя видами выражений: уже рассмотренными в лекции 2 строковыми выражениями, а также логическими и арифметическими выражениями. Соответственно, рассматриваются три основные типа функций: строковые (текстовые), логические (булевы) и арифметические (математические) функции.

Но в «арсенале» языка VBA есть и другие достаточно важные типы функций, которые мы также рассмотрим в данной лекции. В частности, это функции преобразования типов данных; функции обработки дат и времени; финансовые функции и функции, определяемые пользователем. Они также будут рассмотрены в настоящей лекции.

5.2. Логические выражения и логические (булевы) функции

Логическое выражение – это выражение, которое имеет только одно из двух возможных значений: **True** (Истина) или **False** (Ложь). Термы в логических выражениях – это:

- константы **True, False**;
- переменные, имеющие эти значения;
- логические выражения в скобках.

Термы соединяются логическими операциями (***логическими*** или ***булевыми*** функциями). Наиболее популярные из этих функций: **Not, And, Or, Imp –** *отрицание, конъюнкция, дизъюнкция, импликация*.

Как было упомянуто в п. 5.1, в алгоритмах и программах на языке VBA условные выражения используются для проверки условий, влияющих на порядок выполнения шагов алгоритма (программы).

Значение каждой логической операции определяется таблицами истинности (рис. 5.1), в которых **Истина** обозначается единицей («1»), а **Ложь** – нулём («0»).

A	Not A
0	1
1	0

A	B	A And B
0	0	0
0	1	0
1	0	0
1	1	1

A	B	A Or B
0	0	0
0	1	1
1	0	1
1	1	1

A	B	A Imp B
0	0	1
0	1	1
1	0	0
1	1	1

Рис. 5.1. Таблицы истинности для основных логических операций

Пример 5.1. Пусть **X = True**, тогда выражение **Not (X Imp (Not X Or False))** имеет значение **True**.

Разумеется, кроме представленных логических выражений условные выражения могут содержать обычные ***операции сравнения*** значений переменных со значениями арифметических выражений, которые будут рассмотрены в разделе 5.3. Эти сравнения также могут иметь только одно из двух значений: **Истина** или **Ложь**.

Операции сравнения перечислены в табл. 5.1.

Таблица 5.1

Операция сравнения	Пояснение
X = Y	Значение X равно значению Y
X <> Y	Значение X не равно значению Y
X > Y	Значение X больше значения Y
X < Y	Значение X меньше значения Y
X >= Y	Значение X больше или равно значению Y
X <= Y	Значение X меньше или равно значению Y

Пример 5.2. Пусть **X** = 3, **Y** = 5. Реализуем программную проверку на языке VBA всех перечисленных в табл. 5.1 операций сравнения. Две последние операции >= и <= проверим и для равных значений: **X** = 4, **Y** = 4.

На рис. 5.2 показан результат работы следующего макроса, оформленного в виде процедуры:

```
Sub Сравнение()
        X = 3: Y = 5
  Cells(2, 3) = X & " = " & Y:  Cells(2, 4) = (X = Y)
  Cells(3, 3) = X & " <> " & Y: Cells(3, 4) = (X <> Y)
  Cells(4, 3) = X & " > " & Y:  Cells(4, 4) = (X > Y)
  Cells(5, 3) = X & " < " & Y:  Cells(5, 4) = (X < Y)
  Cells(6, 3) = X & " >= " & Y: Cells(6, 4) = (X >= Y)
  Cells(7, 3) = X & " <= " & Y: Cells(7, 4) = (X <= Y)
        X = 4: Y = 4
  Cells(8, 3) = X & " >= " & Y: Cells(8, 4) = (X >= Y)
  Cells(9, 3) = X & " <= " & Y: Cells(9, 4) = (X <= Y)
End Sub
```

Книга1

	A	B	C	D	E
1	X	Y	Сравнение	Результат	
2	3	5	3=5	ЛОЖЬ	
3	3	5	3<>5	ИСТИНА	
4	3	5	3>5	ЛОЖЬ	
5	3	5	3<5	ИСТИНА	
6	3	5	3>=5	ЛОЖЬ	
7	3	5	3<=5	ИСТИНА	
8	4	4	4>=4	ИСТИНА	
9	4	4	4<=4	ИСТИНА	
10					

Рис. 5.2. Результат тестирования работы операций сравнения

5.3. Арифметические выражения и математические функции

Термами в арифметических выражениях служат числа и переменные, имеющие числовые значения, а также арифметические выражения в скобках.

Значение арифметического выражения – это число.

Кроме семи общепринятых функций (возведения в степень, сложения, вычитания, умножения, деления, целочисленного деления и остатка от целочисленного деления, соответственно: ^, +, -, *, /, \, mod), в арифметических выражениях используются разнообразные «встроенные» в язык VBA математические функции, например, представленные в табл. 5.2.

Таблица 5.2

Функция	Значение, возвращаемое функцией
Abs(X)	Абсолютная величина числа X
Cint(X)	Целое число, ближайшее к X
Cos(X)	Косинус числа X
Fix(X)	Целое число, равное X без дробной части
Int(X)	Наибольшее целое число, не превышающее X
Sin(X)	Синус числа X
Sqr(X)	Квадратный корень из числа X
Round(X [, N])	Округлённое число X, N – число знаков после десятичной точки; при отсутствии 2-го аргумента округление до целого

Есть ещё функция без аргумента **Rnd**, которая возвращает «псевдослучайное» число между 0 и 1. Число называется «псевдослучайным», так как при повторном пуске программы эта функция будет возвращать то же значение, что и при предыдущем пуске. Чтобы при повторном пуске программы значение, возвращаемое функцией **Rnd**, было непредсказуемым, перед оператором, в котором выполняется эта функция, следует вставить специальный оператор **Randomize**.

Пример 5.3. Пусть **X = 2017**, тогда выражение

3 * 2 ^ ((X mod 1000) – 14) имеет значение **24**,

и выражение **3 * 2 ^ ((X \ 1000) + 1)** также имеет значение **24**.

Пример 5.4. Пусть числовой переменной **X** должно быть присвоено случайное целое значение между 0 и 100. Предлагается такой оператор:

X = Round(100 * Rnd).

Если при первом пуске программы переменная получает значение **24**, то при повторном пуске она также получит значение **24**. Значение будет непредсказуемым целым числом K ∈ [0, 1, …, 100], если перед указанным оператором поместить оператор **Randomize**.

Пример 5.5. На рис. 5.3 показано тестирование работы функций **Int(X), CInt(X), Fix(X)**:

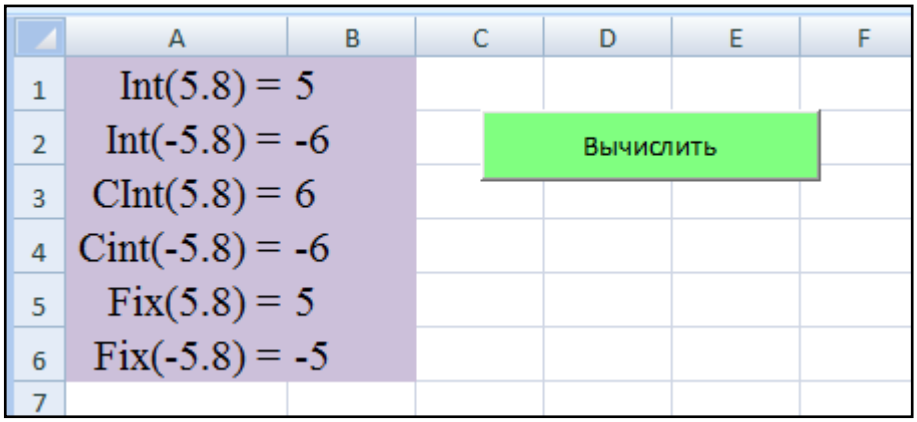


Рис. 5.3. Результат тестирования работы функций **Int(X), CInt(X), Fix(X)**

Проверка была сделана программно так:

```
Cells(1, 1) = "Int(5.8) = " :  Cells(1, 2) = Int(5.8)
```

или с помощью более наглядной функции **Range** вместо **Cells**:

```
Range("A1") = "Int(5.8) = " :  Range("B1") = Int(5.8)
```

и т.д. – всего 6 раз.

5.4. Функции преобразования типов данных

В программах на языке Basic часто возникает необходимость преобразования данных из одного типа в другой. Например, преобразовывать строку в число. Или – наоборот. В этом разделе ограничимся лишь девятью функциями преобразования типов данных:

1. CBool(S) – преобразование выражения S в тип Boolean – логический тип данных тип,
2. CByte(S) – преобразование выражения S в тип Byte,
3. CDbl(S) – преобразование выражения S в тип Double,
4. CSng(S) – преобразование выражения S в тип Single,
5. CInt(S) – преобразование выражения S в тип Integer,
6. CLng(S) – преобразование выражения S в тип Long,
7. CCur(S) – преобразование выражения S в тип Currency,
8. CStr(S) – преобразование выражения S в тип String,
9. CDate(S) – преобразование выражения S в тип Date.

Следует обратить внимание на то, что аргументом каждой функции заявлено *выражение* – без уточнения, какое выражение имеется в виду: строковое, арифметическое и т.д.

На рис. 5.4 на примерах показаны примеры работы всех этих функций.

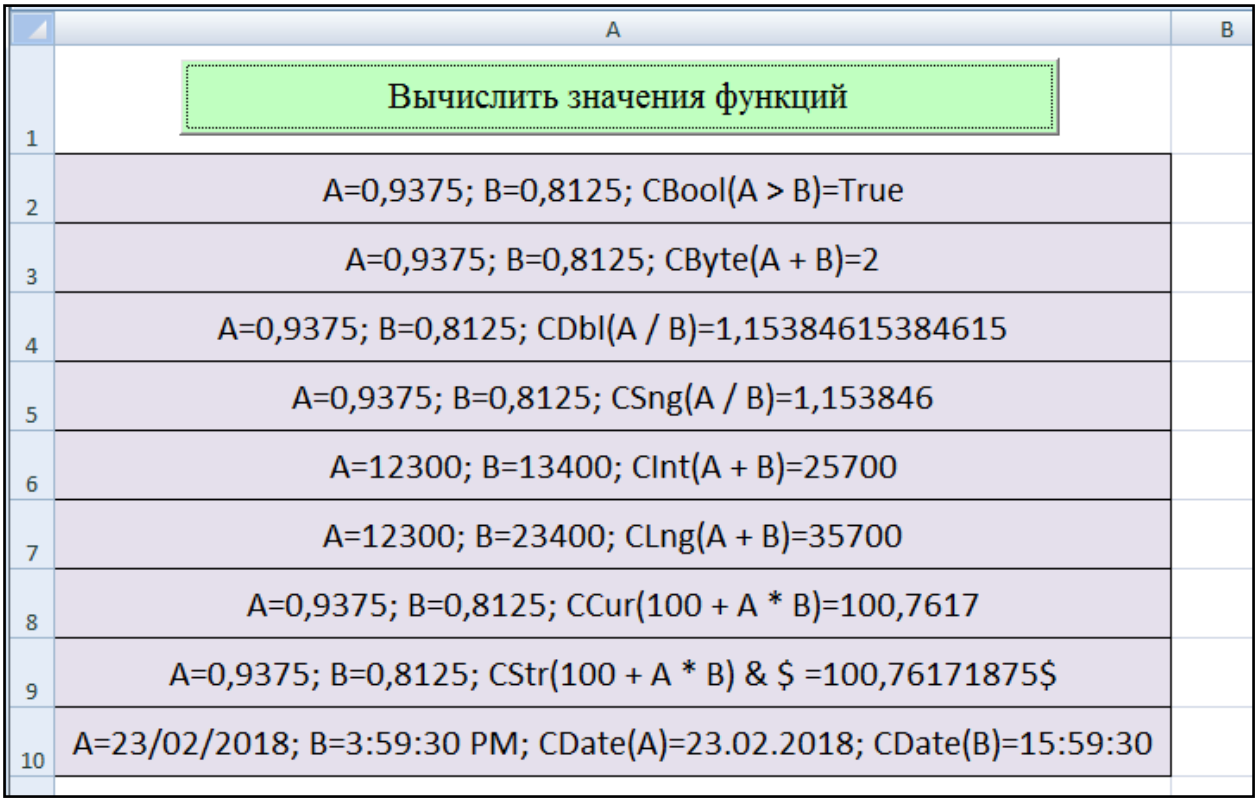


Рис. 5.4. Результат тестирования работы функций преобразования типов данных

Программа, с помощью которой проводилось тестирование, приведена в следующем коде:

```
Private Sub CommandButton1_Click()
    A = 15 / 16: B = 13 / 16
    Range("A2") = "A=" & A & "; " & "B=" & B & "; " & _
                  "CBool(A > B)=" & CBool(A > B)
    Range("A3") = "A=" & A & "; " & "B=" & B & "; " & _
                  "CByte(A + B)=" & CByte(A + B)

    Range("A4") = "A=" & A & "; " & "B=" & B & "; " & _
                  "CDbl(A / B)=" & CDbl(A / B)
    Range("A5") = "A=" & A & "; " & "B=" & B & "; " & _
                  "CSng(A / B)=" & CSng(A / B)

    A = 12300:  B = 13400
    Range("A6") = "A=" & A & "; " & "B=" & B & "; " & _
                  "CInt(A + B)=" & CInt(A + B)
```

```
    A = 12300:  B = 23400
    Range("A7") = "A=" & A & "; " & "B=" & B & "; " & _
                  "CLng(A + B)=" & CLng(A + B)

    A = 15 / 16: B = 13 / 16
    Range("A8") = "A=" & A & "; " & "B=" & B & "; " & _
         "CCur(100 + A * B)=" & CCur(100 + A * B)
    Range("A9") = "A=" & A & "; " & "B=" & B & "; " & _
         "CStr(100 + A * B) & $ =" & CStr(100 + A * B) & "$"

    A = "23/02/2018":  B = "3:59:30 PM"
    Range("A10") = "A=" & A & "; " & "B=" & B & "; " & _
                   "CDate(A)=" & CDate(A) & "; " & _
                   "CDate(B)=" & CDate(B) & " - OK!"
End Sub
```

В следующем примере, как частный случай, демонстрируется преобразование строковых выражений (S1, S2, S3) в числа (рис. 5.5).

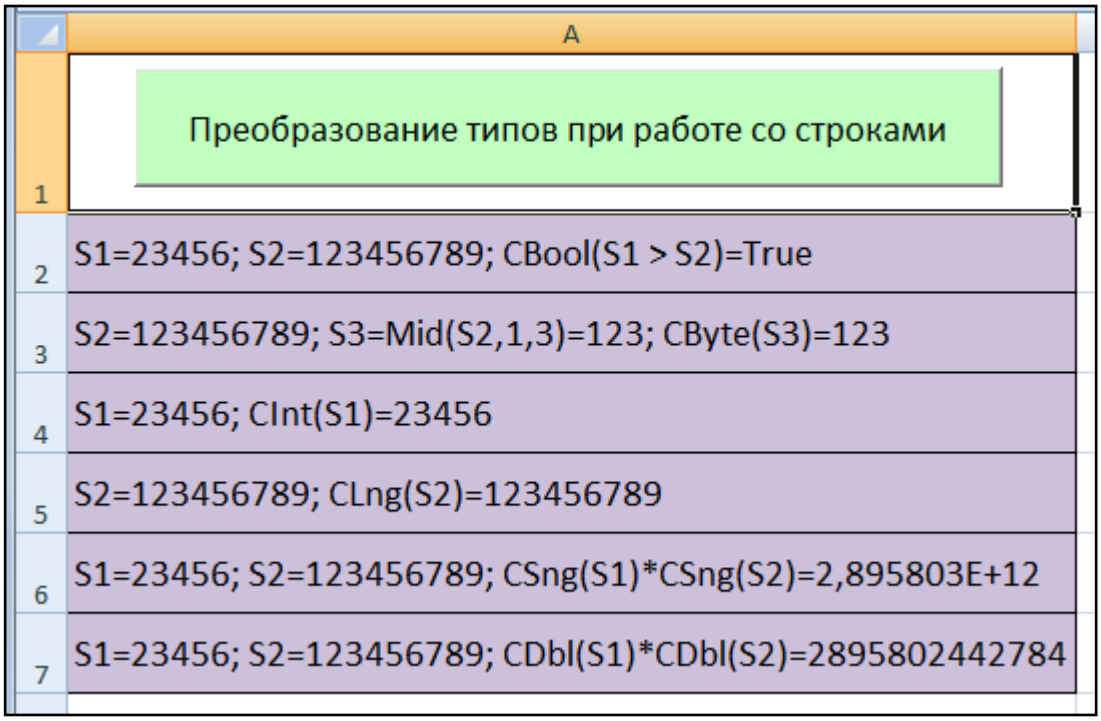


Рис. 5.5. Преобразование текстовых данных в логическое выражение и в числа

Программа, с помощью которой проводилось тестирование, приведена в следующем коде:

```
Private Sub CommandButton1_Click()
    Dim S1 As String, S2 As String, S3 As String
    S1 = "23456": S2 = "123456789"
    Range("A2") = "S1=" & S1 & "; " & "S2=" & S2 & "; " & _
                  "CBool(S1 > S2)=" & CBool(S1 > S2)
    Range("A3") = "S2=" & S2 & "; " & "S3=Mid(S2,1,3)=" & _
```

```
                    Mid(S2, 1, 3) & "; " & _
                   "CByte(S3)=" & CByte(Mid(S2, 1, 3))
    Range("A4") = "S1=" & S1 & "; " & _
                   "CInt(S1)=" & CInt(S1)
    Range("A5") = "S2=" & S2 & "; " & _
                   "CLng(S2)=" & CLng(S2)
    Range("A6") = "S1=" & S1 & "; " & "S2=" & S2 & "; " & _
                   "CSng(S1)*CSng(S2)=" & CSng(S1) * CSng(S2)
    Range("A7") = "S1=" & S1 & "; " & "S2=" & S2 & "; " & _
                   "CDbl(S1)*CDbl(S2)=" & CDbl(S1) * CDbl(S2)
End Sub
```

5.5. Функции обработки и представления значений дат и времени

Перечислим лишь самые распространённые функции из указанной группы:

- **Date** – возвращает текущее значение системной даты;
- **Time** – возвращает текущее значение системного времени;
- **Now** – возвращает текущие значения системной даты и системного времени;
- **DateAdd**(интервал, число, дата) – возвращает значение даты, равное сумме исходной даты и заданного числа ***указанных интервалов***;
- **DateDiff**(интервал, дата1, дата2) – возвращает число ***указанных интервалов*** между указанными датами;
- **Day**(дата) – возвращает день указанной даты;
- **Month**(дата) – возвращает месяц указанной даты;
- **Year**(дата) – возвращает год указанной даты;
- **Timer** – возвращает число секунд после полуночи;
- **WeekDay**(дата) – возвращает число (от 1 до 7) – номер дня недели, начиная с понедельника (по-русски ☺);
- **WeekDayName**(номер) – по номеру дня недели устанавливает его название.

Рассмотрим подробнее функции, где используется так называемый «интервал»:

- **DateAdd**(интервал, число, дата) – возвращает значение даты, равное сумме исходной даты и заданного числа указанных интервалов;
- **DateDiff**(интервал, дата1, дата2) – возвращает число указанных интервалов между указанными датами.

Интервал – это константа. Их много. Рассмотрим лишь некоторые:

«YYYY» – годы;
«Q» – кварталы;
«M» – месяцы;
«Ww» – недели;
«H» – часы;
«N» – минуты;
«S» – секунды.

Пример демонстрирует рис. 5.6.

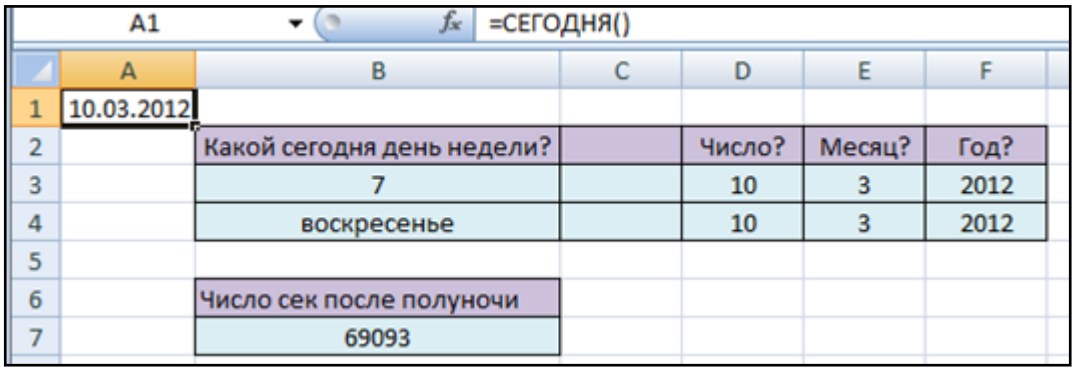


Рис. 5.6. Пример обработки и представления значений дат и времени на языке VBA

```
Sub Сегодня()
    Dim d As Date
    d = Range("A1")
    Range("B4") = WeekdayName(Weekday(d))
    Range("D4") = Day(d)
    Range("E4") = Month(d)
    Range("F4") = Year(d)
    Range("B7") = Timer
End Sub
```

Здесь программа заполняет ячейки 4-й и 7-й строки. А 3-я строка заполняется с помощью функций, которые не требуют программирования. Пример представлен на рис. 5.7.

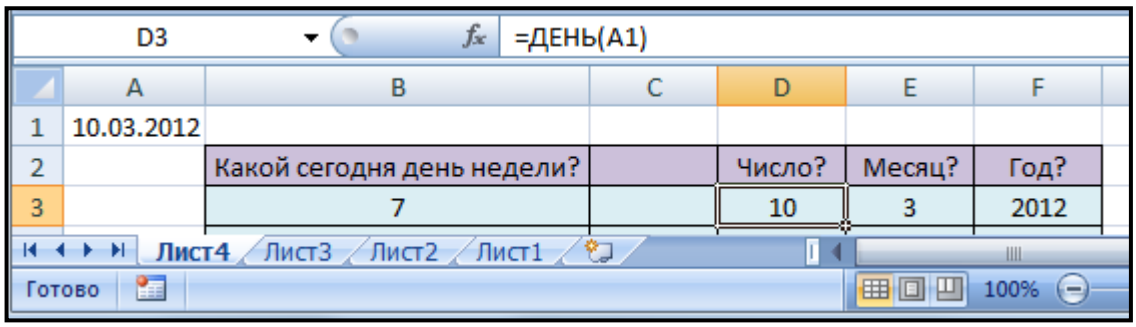


Рис. 5.7. Пример обработки и представления значений дат и времени с помощью функций Excel

5.6. Функции, используемые при программировании финансовых операций

Финансовые функции языка Visual Basic (в частности языка VBA):

- **Pmt**(rate, nper, pv [, fv [, type]]) – периодическая выплата банку по кредиту;
- **Rate**(nper, pmt, pv [, fv [, type]]) – банковская процентная ставка;
- **Nper**(rate, pmt, pv [, fv [, type]]) – число периодических выплат банку;
- **PV**(rate, nper, pmt [, fv [, type]]) – величина кредита (инвестиции);
- **FV**(rate, nper, pmt [, pv [, type]]) – величина накопления.

В каждой функции три обязательных аргумента и два необязательных – в квадратных скобках. Имена аргументов говорят сами за себя ☺. Кроме последнего: этот аргумент type имеет значение 1 или 0 – в зависимости от того, когда клиентом производится выплата – в начале (1) или в конце (0) периода выплаты.

Важное замечание. Финансовые функции могут быть использованы и без программирования – с помощью только формульной строки на листе книги Excel.

На рис. 5.8 представлена страница справочной системы приложения Excel с описанием одной из финансовых функций. Это функция выплаты по кредиту **Pmt**, которая в русскоязычном приложении Excel имеет «кириллическое» название **ПЛТ**. На рис.5.9 – пример работы данной функции.

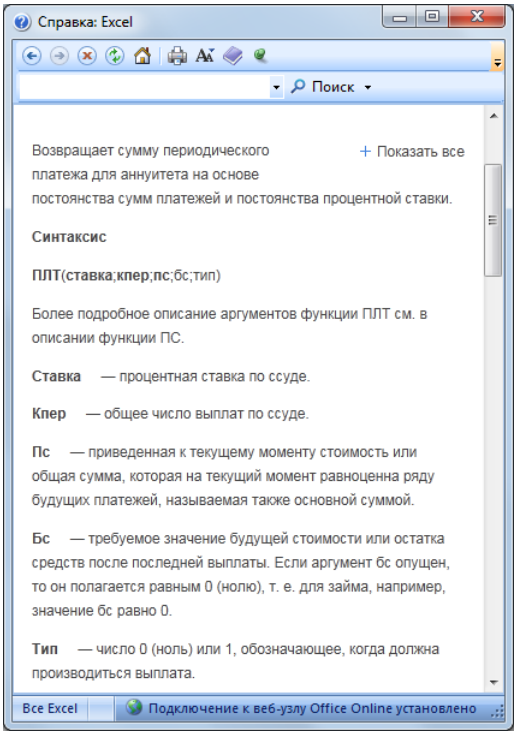


Рис. 5.8. Справочная страница Excel
с описанием функции ПЛТ

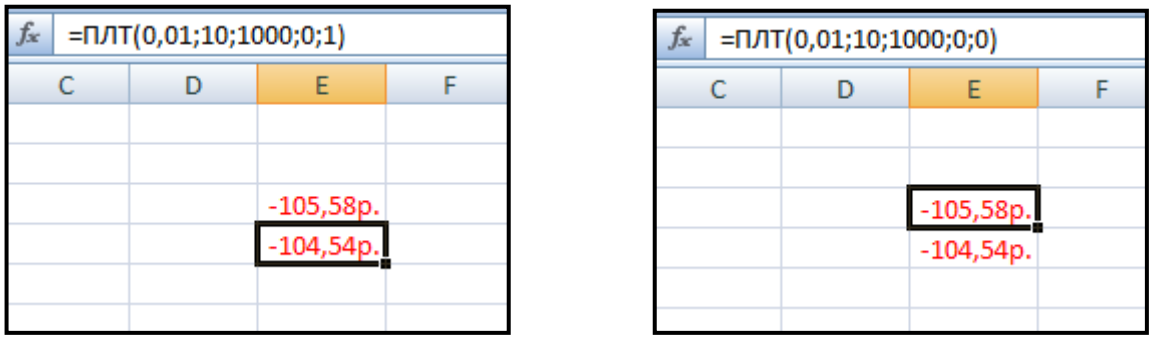


Рис. 5.9. Пример работы функции ПЛТ

Значения всех пяти аргументов функции ПЛТ:

- 0,01 – ставка (в долях на период выплаты – при 12 % годовых);
- 10 – количество выплат (взят кредит на 10 месяцев);
- 1000 – величина кредита;
- 0 – остаток после последней выплаты;
- 0 или 1 – выплата в начале или в конце периода.

Отметим, что значение выплаты клиента банку (а не банка клиенту) всегда берётся со знаком «минус».

На приведенном выше примере показано различие между значениями функции ПЛТ для разных значений пятого аргумента.

Теперь продемонстрируем использования всех перечисленных финансовых функций с помощью единой экранной формы.

По технологии, рассмотренной в 1-й лабораторной работе (***Задание 4***), можно создать форму, на которой следует установить пять текстовых полей – по числу перечисленных в начале данного раздела финансовых показателей:

- банковская процентная ставка;
- число периодических выплат банку;
- величина кредита (инвестиции);
- величина накопления;
- периодическая выплата банку по кредиту.

Для указания того, когда – в начале или в конце периода выполняется выплата – следует поместить на форму пару опциональных кнопок.

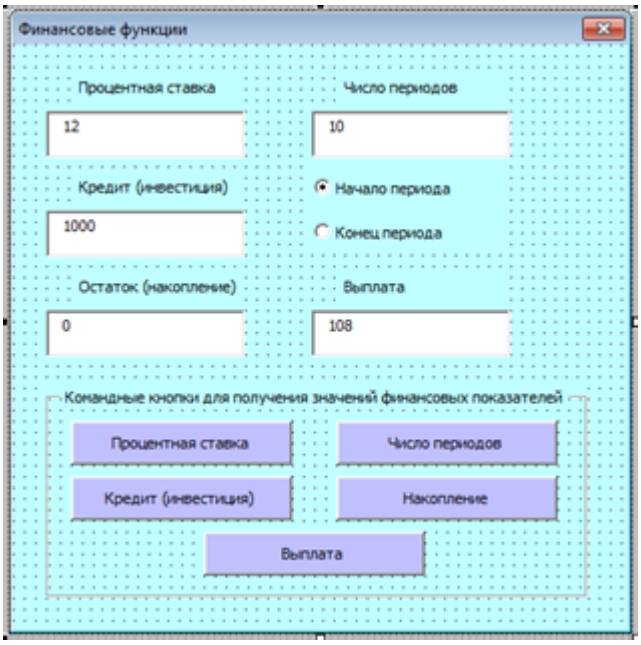


Рис. 5.10. Пример предлагаемой формы для демонстрации работы финансовых функций

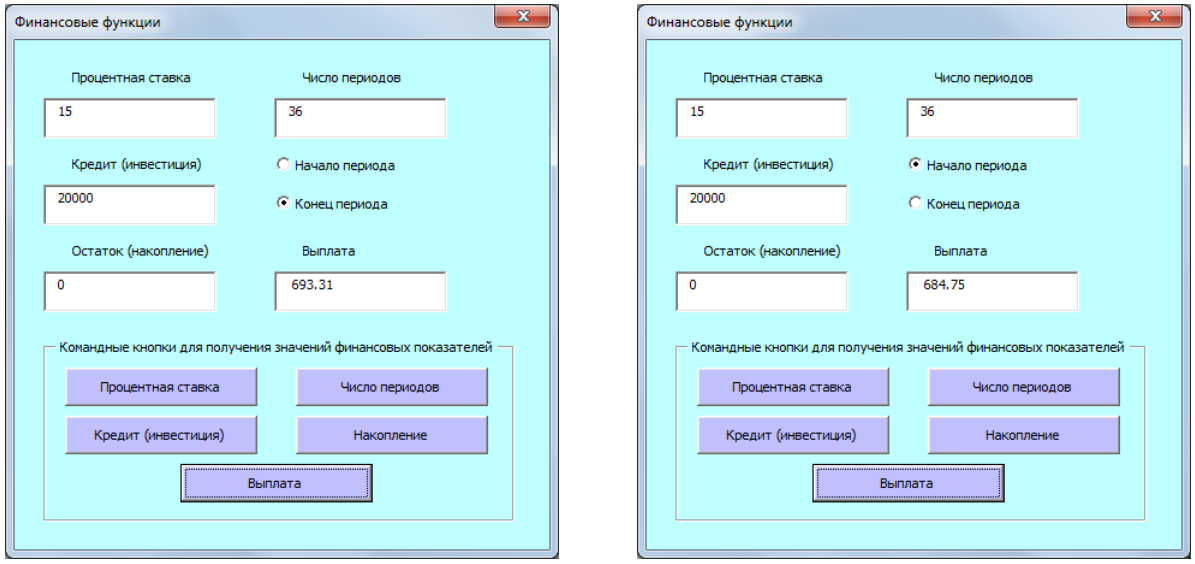
На форме следует установить также пять командных кнопок по числу соответствующих финансовых функций. Назовём форму «Финансовые функции». На рис. 5.10 показан пример этой формы.

Теперь рассмотрим только одну из пяти процедур, соответствующих пяти командным кнопкам экранной формы «Финансовые функции»:

```
Dim r As Double, n As Double
Dim v1 As Double, v2 As Double
Dim p As Double, t As Double, s As String
```

```
Private Sub CommandButton5_Click()
   r = Val(TextBox1.Text) / (12 * 100)
   n = Val(TextBox2.Text)
   v1 = Val(TextBox3.Text)
   v2 = Val(TextBox4.Text)
       If OptionButton1.Value Then t = 1 Else t = 0
   p = Pmt(r, n, v1, v2, t)
   s = Str(-Round(p, 2))
   TextBox5.Text = s
End Sub
```

Рассмотрим результат решения задачи вычисления выплаты по кредиту с помощью данной экранной формы (рис. 5.11).



а) *б*)

Рис. 5.11. Демонстрация работы функции Pmt (выплаты по кредиту) на экранной форме: *а*) выплата в конце периода и *б*) выплата в начале периода

Дополнительно можно оформить решение рассмотренной задачи не только с помощью экранной формы, но и записывая исходные данные и результаты в ячейки листа книги Excel (рис. 5.12).

Рис. 5.12. Демонстрация работы функции Pmt (выплаты по кредиту) помимо экранной формы – непосредственно в ячейках листа Excel

Для указанного оформления решения следует добавить в рассмотренную процедуру `CommandButton5_Click` следующие строки:

```
Range("A" & 3) = "Исходные данные:"
Range("A" & 4) = "Исходные данные:"
Range("A" & 5) = "Исходные данные:"
Range("A" & 6) = "Исходные данные:"
Range("A" & 7) = "Исходные данные:"
Range("A" & 8) = "Результат :”
Range("C" & 3) = TextBox1.Text  & " %"
Range("C" & 4) = TextBox2.Text
Range("C" & 5) = TextBox3.Text  & " $"
Range("C" & 6) = TextBox4.Text  & " $"
If t = 1 Then
      Range("C" & 7) = "начало"
Else
      Range("C" & 7) = "конец"
End If
      Range("C" & 8) = TextBox5.Text  & "$"
```

Заметим, что столбец «B» на листе книги Excel должен быть заполнен заранее вручную и не будет меняться в ходе работы программы.

5.7. Функции, определяемые пользователем (user defined functions)

Таких функций нет в арсенале встроенных функций языка VBA. Они программируются пользователями – фактически программистами – на языке VBA.

В программный модуль необходимо вписать код, реализующий алгоритм интересующей вас функции. Например, вычисление площади треугольника по трём сторонам a, b и c – по формуле Герона:

$$S = \sqrt{p(p-a)(p-b)(p-c)},$$

где $p = (a + b + c)/2$ – полупериметр треугольника.

```
Private Function ПлощТреуг(a As Single, b As Single, _
                           c As Single) As Single
    Dim p As Single
    p = (a + b + c) / 2
    ПлощТреуг = Sqr(p * (p - a) * (p - b) * (p - c))
End Function
```

Очевидно, что данную пользовательскую функцию можно использовать как обычную системную (встроенную) функцию в процедурах программного модуля, в котором она определена. А если заменить ключевое слово **Private** в начале определения функции ключевым словом **Public**, то её можно использовать и в других программных модулях.

Определяемые пользователем функции могут быть рекурсивными. Например, функция реверсирования строки символов:

```
Private Function Reverse(S As String) As String
    If S = "" Then
        Reverse = S
    Else
        Reverse = Mid(S, Len(S), 1) & _
                  Reverse(Mid(S, 1, Len(S) - 1))
    End If
End Function
```

Это рекурсивная функция. Она возвращает цепочку, первый символ которой – это последний символ исходной цепочки, к которому справа присоединяется реверсированная укороченная цепочка после отделения от неё последнего символа. Пример её работы показан на рис. 5.13 как результат работы следующего макроса:

```
Sub Реверс()
    Dim S As String
    S = Range("A1"): Range("A2") = Reverse(S)
End Sub
```

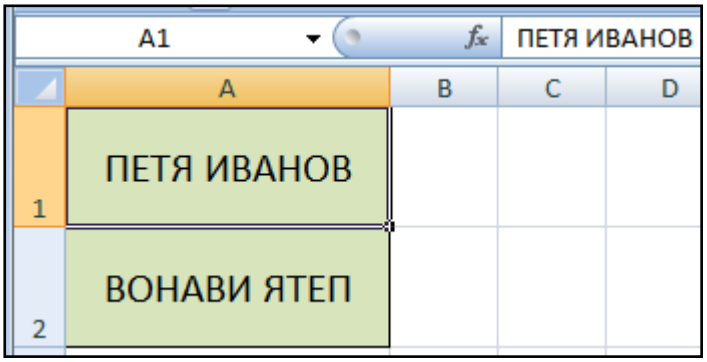


Рис. 5.13. Исходные данные и результат работы процедуры «Реверс»

Глава 6

Данная глава – это изложение материалов второй лабораторной работы, в которой закрепляется материал двух предыдущих лекций по курсу программирования на языке Visual Basic для офисных приложений. Содержание этих материалов можно структурировать следующим образом:

- Контроль знаний по теме лабораторной работы.
- Пример программирования процедуры с использованием оператора условного перехода («задача о треугольнике»).
- Пример программирования построения числовых рядов с использованием оператора безусловного перехода совместно с оператором условного перехода («геометрическая прогрессия», «ряд Фибоначчи»).
- Пример программирования процедуры с использованием финансовых функций («выплата по кредиту», «накопление» и других).
- Пример программирования процедуры с использованием функции, определяемой пользователем (расширение «задачи о треугольнике», «палиндром»).

6.1. Контрольные вопросы по теме лабораторной работы

1. Что такое условный оператор **If … Then … Else** на операторном языке программирования, в частности на языке Basic? Как изображается условный оператор на блок-схеме алгоритма?
2. Чем отличаются однострочная и многострочная формы синтаксиса условного оператора на языке Basic (VBA)?
3. Что представляет собой оператор безусловного перехода **Go To**? Как можно использовать этот оператор совместно с оператором условного перехода для организации повторений?
4. С какой целью целесообразно использовать оператор выбора **Select Case** на языке Basic?
5. Как рекурсивно определить понятие выражения на языке Basic? Какие примеры логических и математических (арифметических) выражений вы можете привести?
6. В каких случаях на языке Basic необходимо применять логические (булевы) функции?

7. В каких программах на языке Basic необходимо применять математические функции? (Приведите примеры.)
8. Зачем нужны функции преобразования типов данных? Какие ограничения накладываются на аргументы этих функций? Приведите примеры функций, преобразующих данные строкового типа в числа данного типа (**Byte, Integer, Long, Single, Double**).
9. Перечислите приведенные на предыдущей лекции функции, используемые при программировании финансовых операций.
10. Приведите примеры функций для представления и обработки значений дат и времени.
11. Что такое «функция, определяемая пользователем»? Приведите пример такой функции.
12. Что такое рекурсивная функция? В качестве примера напишите определение рекурсивной функции **Reverse(S)**, возвращающей значение реверса строки S.

6.2. Пример программирования процедуры с использованием оператора условного перехода («задача о треугольнике»)

Задание 1. Пусть *a, b, c* – случайные целые положительные числа из интервала [0, 100]. Напишите программу на языке VBA, которая отвечает на вопрос: «Можно ли эти числа считать сторонами треугольника?». Программу оформить в виде макроса с именем «Треугольник» для приложения MS Excel.

Макрос «Треугольник» может вызываться щелчком командной кнопки, которую необходимо разместить на листе книги Excel (рис. 6.1).

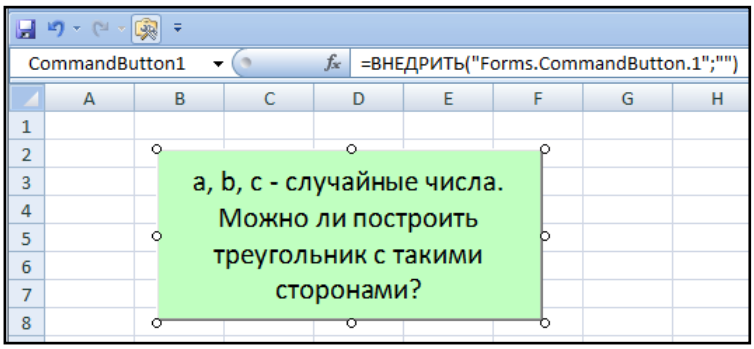


Рис. 6.1. Кнопка для вызова макроса «Треугольник», проверяющего допустимость значений трёх переменных

Щелчок командной кнопки, показанной на рис. 6.1, должен запускать процедуру, которая программно создаёт случайные числа *a*, *b*, *c* и выдаёт пользователю одно из двух «окон сообщения», показанных на рис. 6.2.

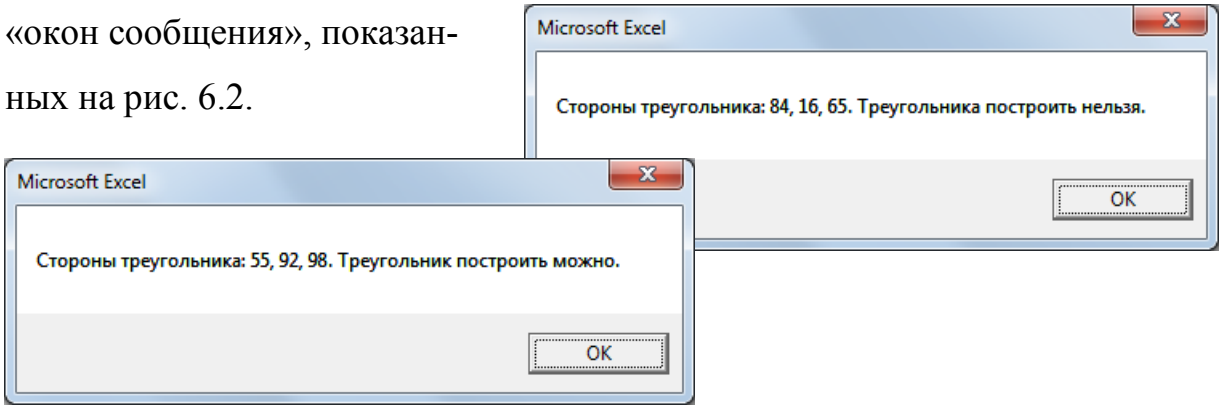


Рис. 6.2. Окна сообщения, которые выдаёт макрос проверки значений трёх переменных на их допустимость для создания треугольника

Указания:

1. Числа должны быть ***случайными*** и ***целыми*** – от 0 до 100. Функция **Rnd** возвращает случайное десятичное число от 0 до 1. Целое число из данного интервала можно получить путём умножения значения **Rnd** на 100 и округления: **Round(100 * Rnd)**.
2. Так как сумма двух из этих чисел не превышает 200, можно использовать тип данных **Byte**.
3. При проверке допустимости этих чисел для создания треугольника в операторе **If … Then … Else** можно использовать следующее условное выражение: $a + b > c$ **And** $b + c > a$ **And** $c + a > b$.

Решение задачи – на следующих двух рисунках (рис. 6.3 и 6.4).

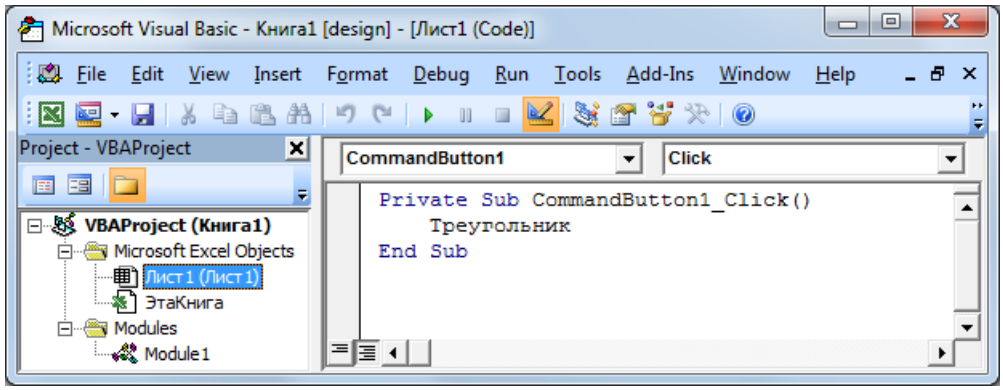


Рис. 6.3. Процедура щелчка командной кнопки на листе Excel с вызовом макроса «Треугольник»

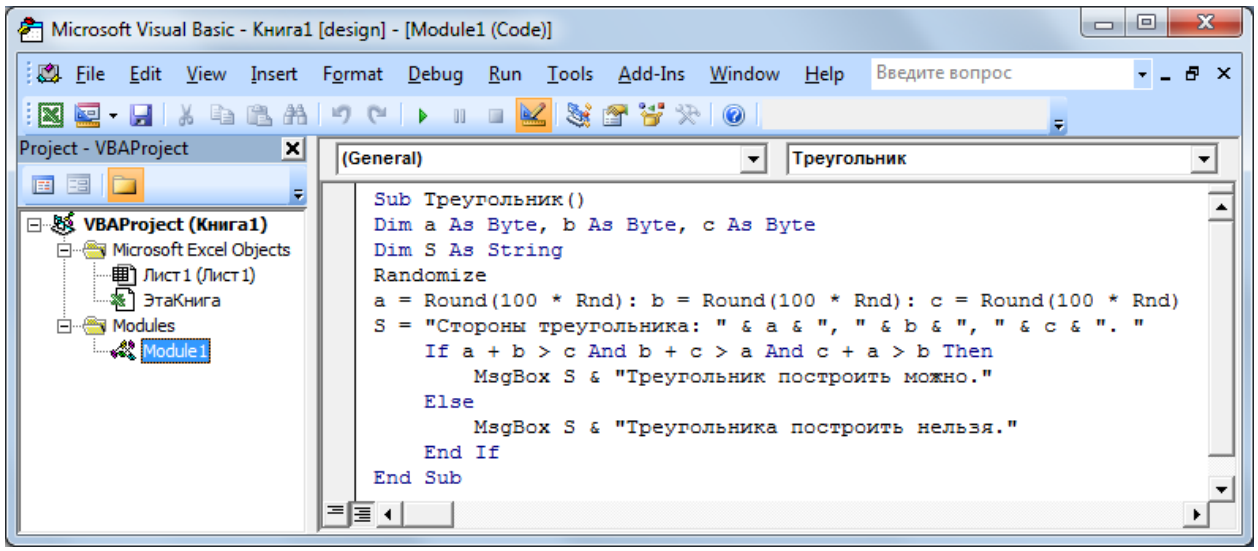


Рис. 6.4. Процедура макроса «Треугольник»

Задание 2. Реализовать другой вариант решения задачи предыдущего задания: случайные числа и результат надо помещать в ячейки Листа1 книги Excel (рис. 6.5).

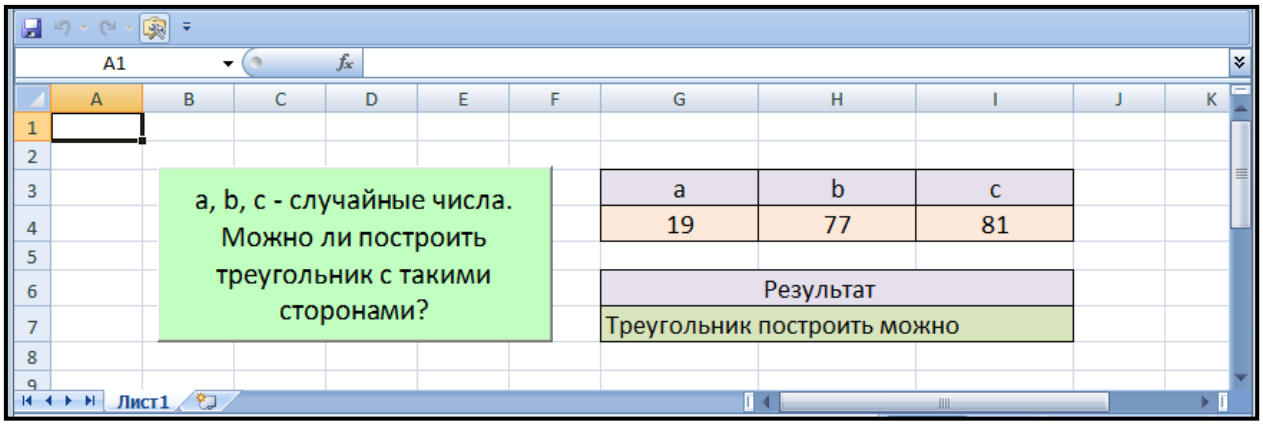


Рис. 6.5. Лист1 книги Excel с результатами работы макроса «Треугольник»

В изменённом варианте решения данной задачи добавлена (легко догадаться, куда) одна строка для записи значений *a*, *b*, *c* в ячейки Листа1:

```
Cells(4,7) = a : Cells(4,8) = b : Cells(4,9) = c
```

и исправлены две строки программного кода:

строка

```
MsgBox S & «Треугольник построить можно»
```

заменена на строку

```
Cells(7,7) = «Треугольник построить можно»,
```

а строка

```
MsgBox S & «Треугольника построить нельзя»
```

заменена на строку

```
Cells(7,7) = «Треугольника построить нельзя».
```

6.3. Пример программирования процедур построения числовых рядов с использованием оператора безусловного перехода совместно с оператором условного перехода («геометрическая прогрессия» и «ряд Фибоначчи»)

Рассмотрим применение оператора безусловного перехода **GoTo**.

Отметим, что этот оператор рекомендуется, по возможности, *не использовать* при создании программ – особенно таких, которые отличаются значительной структурной сложностью. (Безусловные переходы часто сильно затрудняют попытки разобраться в работе программ, написанных другими программистами.) Но в некоторых задачах оператор **GoTo** всё же применяется. Например, в случаях, когда необходим экстренный выход из программы при возникновении «внештатной ситуации». Здесь мы рассмотрим другой случай, когда оператор **GoTo** совместно с оператором условного перехода **If … Then … Else** может (без специальных *операторов цикла*) реализовать многократно повторяющийся процесс.

Задание 3. Напишите макрос, целью запуска которого является построение числовой последовательности неизвестной заранее длины *k*, состоящей из степеней двойки – начиная с 1-й степени двойки (2^1) и заканчивая значением 2^k, которое превысило один миллион.

Все члены этой последовательности должны быть записаны в столбец «А» Листа1 книги Excel (рис. 6.6.).

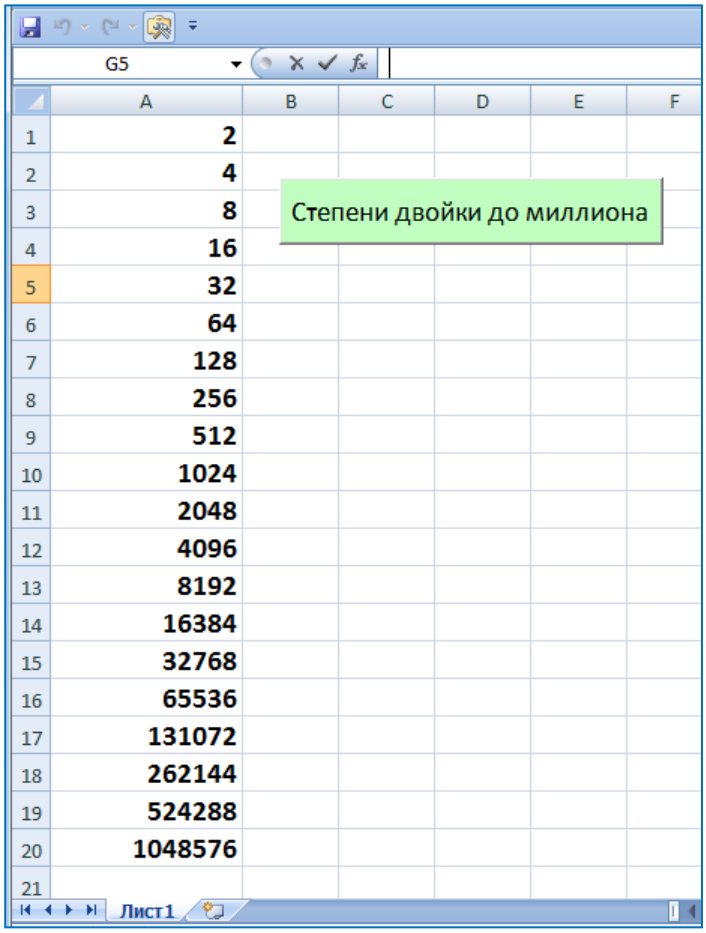


Рис. 6.6. Результат работы макроса «Степени двойки»

Массив и оператор цикла при выполнении данного задания использовать не разрешается – необходимо ограничиться лишь операторами условного и безусловного переходов.

Решение данной задачи демонстрирует следующий код:

```
Private Sub CommandButton1_Click()
  Dim X As Long
  N = 1         ' Выдача степеней двойки от 1-й до
                ' предельно допустимой:
M1:
  X = 2 ^ N
  Cells(N, 1) = X     ' Или так: Range("A" & N) = X
  N = N + 1
  If X <= 1000000 Then GoTo M1 ' Однострочная форма
End Sub
```

Задание 4. Напишите макрос, целью запуска которого является построение так называемого «ряда Фибоначчи» – начиная с чисел F_1=1, F_2=1, F_3=2, F_4=3, F_5=5,… и заканчивая значением F_k, которое первым превысило один миллион. Все члены этой последовательности должны быть записаны в столбец «A» Листа1 (рис. 6.7).

Вопрос студентам: «Каким является номер *k* последнего числа в полученном ряду?»

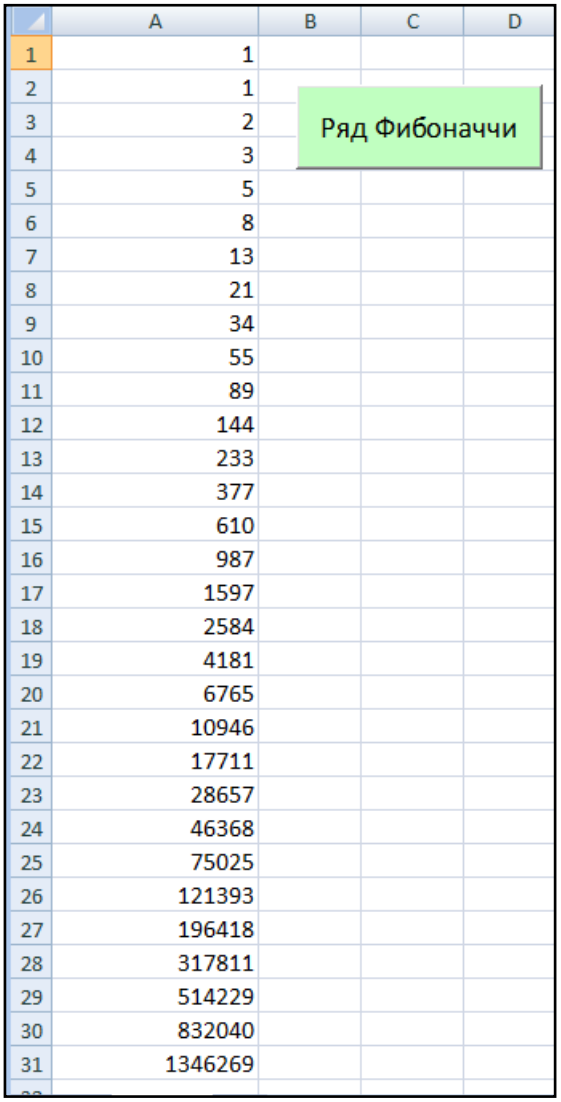


Рис. 6.7. Результат работы макроса «Ряд Фибоначчи»

Как и в предыдущем случае, массив и оператор цикла при выполнении данного задания использовать не разрешается – необходимо ограничиться лишь операторами условного и безусловного переходов.

Решение данной задачи демонстрирует следующий код:

```
Private Sub CommandButton1_Click()
    Dim X1 As Long, X2 As Long
    N = 1
      ' Выдача чисел Фибоначчи  (1,1,2,3,5,8,13,…) до
      ' предельно допустимого (здесь 1000000).
    X1 = 1: X2 = 1
M1:
    Cells(N, 1) = X1      'Или так: Range("A" & N) = X1
    Cells(N + 1, 1) = X2 'Или: Range("A" & N + 1) = X2
    X1 = X1 + X2
    X2 = X2 + X1
    N = N + 2
    If X1 <= 1000000 Then GoTo M1
    Cells(N, 1) = X1      'Или: Range("A" & N) = X1
End Sub
```

Отметим, что в построенном ряду Фибоначчи первым числом, превышающим 1 миллион, является F_{31} (31-е число полученного ряда).

6.4. Пример программирования процедуры с использованием финансовых функций («выплата по кредиту», «накопление» и других)

Напомним: в лекции были упомянуты пять наиболее популярных финансовых функций языка VBA:

- **Pmt**(rate, nper, pv [, fv [, type]]) – периодическая выплата по кредиту;
- **Rate**(nper, pmt, pv [, fv [, type]]) – банковская процентная ставка;
- **Nper**(rate, pmt, pv [, fv [, type]]) – число периодических выплат банку;
- **PV**(rate, nper, pmt [, fv [, type]]) – величина кредита (инвестиции);
- **FV**(rate, nper, pmt [, pv [, type]]) – величина накопления.

В каждой функции три обязательных аргумента и два необязательных – в квадратных скобках. Имена аргументов, благодаря мнемонике, говорят сами за себя. Кроме последнего: этот аргумент type имеет значение 1 или 0 – в

зависимости от того, когда клиентом производится выплата – в начале (1) или в конце (0) периода выплаты.

Задание 5. Каждый студент, выполняющий данную лабораторную работу, получает один из пяти вариантов. Каждый из этих вариантов заключается в создании процедуры для щелчка одной из пяти командных кнопок, установленных на экранной форме, показанной на рис. 6.8.

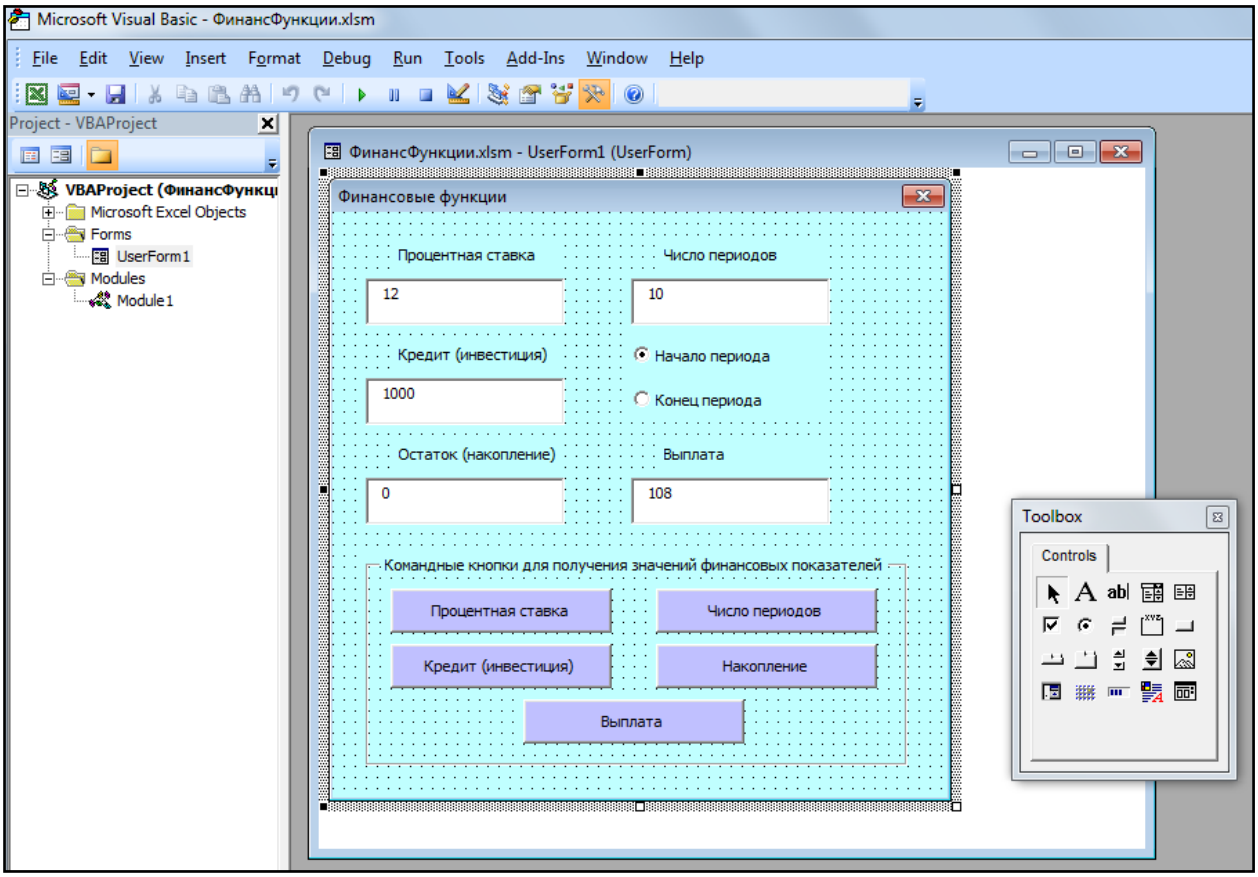


Рис. 6.8. Экранная форма «Финансовые функции» на этапе проектирования

Каждый студент «с чистого листа» должен создать свою собственную экранную форму в своём проекте – книге Excel. Напомним, что новая экранная форма **UserForm1** создаётся после отработки цепочки меню **Insert → UserForm** в окне Microsoft Visual Basic, показанном на рис. 6.8.

Следующий фрагмент программного кода – объявление переменных – является одним и тем же для всех пяти вариантов:

```
Dim r As Double, n As Double, v1 As Double, v2 As Double
Dim p As Double, t As Double, i As Integer, s As String
```

Далее – процедуры для отдельных вариантов:

Вариант 1. Процедура для щелчка кнопки «Процентная ставка»

```
Private Sub CommandButton1_Click()
n = Val(TextBox2.Text)
v1 = Val(TextBox3.Text)
v2 = Val(TextBox4.Text)
p = -Val(TextBox5.Text)
    If OptionButton1.Value Then t = 1 Else t = 0
r = Rate(n, p, v1, v2, t)
s = Str(Round(r * (12 * 100), 1))
TextBox1.Text = s
          For i = 3 To 8
             Range("A" & i) = "Исходные данные:"
          Next
        Range("A" & 3) = "Результат:"
        Range("C" & 3) = TextBox1.Text & " %"
        Range("C" & 4) = TextBox2.Text
        Range("C" & 5) = TextBox3.Text & " $"
        Range("C" & 6) = TextBox4.Text & " $"
        If t = 1 Then Range("C" & 7) = "начало" Else _
                      Range("C" & 7) = "конец"
        Range("C" & 8) = TextBox5.Text & " $"
End Sub
```

Вариант 2. Процедура для щелчка кнопки «Число выплат»

```
Private Sub CommandButton2_Click()
r = Val(TextBox1.Text) / (12 * 100)
v1 = Val(TextBox3.Text)
v2 = Val(TextBox4.Text)
p = -Val(TextBox5.Text)
    If OptionButton1.Value Then t = 1 Else t = 0
n = Round(NPer(r, p, v1, v2, t), 2)
s = Str(n)
TextBox2.Text = s
          For i = 3 To 8
            Range("A" & i) = "Исходные данные:"
          Next
        Range("A" & 4) = "Результат:"
        Range("C" & 3) = TextBox1.Text & " %"
        Range("C" & 4) = TextBox2.Text
        Range("C" & 5) = TextBox3.Text & " $"
        Range("C" & 6) = TextBox4.Text & " $"
        If t = 1 Then Range("C" & 7) = "начало" Else _
                      Range("C" & 7) = "конец"
        Range("C" & 8) = TextBox5.Text & " $"
End Sub
```

Вариант 3. Процедура для щелчка кнопки «Кредит (инвестиция)»

```
Private Sub CommandButton3_Click()
r = Val(TextBox1.Text) / (12 * 100)
n = Val(TextBox2.Text)
v2 = Val(TextBox4.Text)
p = -Val(TextBox5.Text)
    If OptionButton1.Value Then t = 1 Else t = 0
v1 = PV(r, n, p, v2, t)
s = Str(Round(v1, 2))
TextBox3.Text = s
          For i = 3 To 8
             Range("A" & i) = "Исходные данные:"
          Next
        Range("A" & 5) = "Результат:"
        Range("C" & 3) = TextBox1.Text & " %"
        Range("C" & 4) = TextBox2.Text
        Range("C" & 5) = TextBox3.Text & " $"
        Range("C" & 6) = TextBox4.Text & " $"
        If t = 1 Then Range("C" & 7) = "начало" Else _
                       Range("C" & 7) = "конец"
        Range("C" & 8) = TextBox5.Text & " $"
End Sub
```

Вариант 4. Процедура для щелчка кнопки «Величина накопления»

```
Private Sub CommandButton4_Click()
r = Val(TextBox1.Text) / (12 * 100)
n = Val(TextBox2.Text)
v1 = Val(TextBox3.Text)
p = -Val(TextBox5.Text)
    If OptionButton1.Value Then t = 1 Else t = 0
v2 = FV(r, n, p, v1, t)
s = Str(Round(v2, 2))
TextBox4.Text = s
          For i = 3 To 8
             Range("A" & i) = "Исходные данные:"
          Next
        Range("A" & 6) = "Результат:"
        Range("C" & 3) = TextBox1.Text & " %"
        Range("C" & 4) = TextBox2.Text
        Range("C" & 5) = TextBox3.Text & " $"
        Range("C" & 6) = TextBox4.Text & " $"
        If t = 1 Then Range("C" & 7) = "начало" Else _
                       Range("C" & 7) = "конец"
        Range("C" & 8) = TextBox5.Text & " $"
End Sub
```

Вариант 5. Процедура для щелчка кнопки «Выплата по кредиту»

```
Private Sub CommandButton5_Click()
r = Val(TextBox1.Text) / (12 * 100)
n = Val(TextBox2.Text)
v1 = Val(TextBox3.Text)
v2 = Val(TextBox4.Text)
    If OptionButton1.Value Then t = 1 Else t = 0
p = Pmt(r, n, v1, v2, t)
s = Str(-Round(p, 2))
TextBox5.Text = s
          For i = 3 To 8
               Range("A" & i) = "Исходные данные:"
          Next
        Range("A" & 8) = "Результат:"
        Range("C" & 3) = TextBox1.Text & " %"
        Range("C" & 4) = TextBox2.Text
        Range("C" & 5) = TextBox3.Text & " $"
        Range("C" & 6) = TextBox4.Text & " $"
        If t = 1 Then Range("C" & 7) = "начало" Else _
                      Range("C" & 7) = "конец"
        Range("C" & 8) = TextBox5.Text & " $"
End Sub
```

Зелёным цветом (и полужирным шрифтом) выделены строки, которые относятся к заданию 6. (В задании 5 их можно проигнорировать ☺.)

Задание 6. Каждый студент, выполнив свой вариант задания 5, должен дополнить программный код процедуры для одной из кнопок, относящейся к его варианту, строками, выделенными зелёным цветом. Это позволит при повторном запуске программы получить результат, показанный на рис. 6.9.

В зависимости от варианта автоматически, в результате работы программы, будут заполнены ячейки столбцов «А» и «С». Но студент должен самостоятельно вручную заполнить столбец «В» листа – в соответствии со своим вариантом. Это содержимое столбца «В» одинаково для всех вариантов и меняться не будет.

Рис. 6.9. Результат выполнения ***Задания 6*** на листе книги Excel

6.5. Пример программирования процедур с использованием функций, определяемых пользователем (расширение «задачи о треугольнике», «палиндром»)

Задание 7. Задание 7 является продолжением задания 1 данной лабораторной работы. В процедуру макроса «Треугольник» следует внести дополнение: вычисление площади треугольника по формуле Герона

$$S = \sqrt{p(p-a)(p-b)(p-c)},$$

где p – «полупериметр» треугольника: $p = (a + b + c)/2$.

Дополнение должно изменить вид листа книги Excel, показанного на рис. 6.5, на показанный ниже (рис. 6.10).

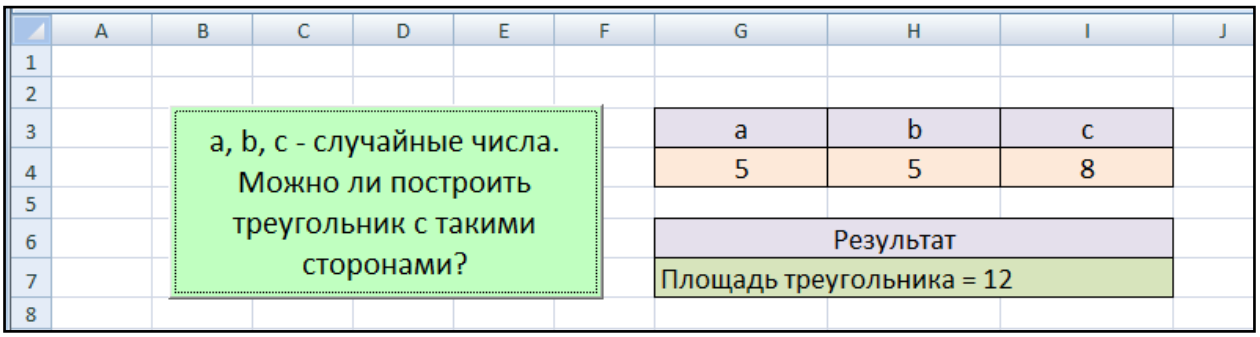


Рис. 6.10. Изменившийся Лист1 книги Excel с результатами работы макроса «Треугольник»

Изменённый код макроса «Треугольник» показан на рис. 6.11.

Отметим, что определение функции «ПлТреуг» (вычисление площади треугольника) в данном случае вставлено непосредственно в модуль макроса «Треугольник».

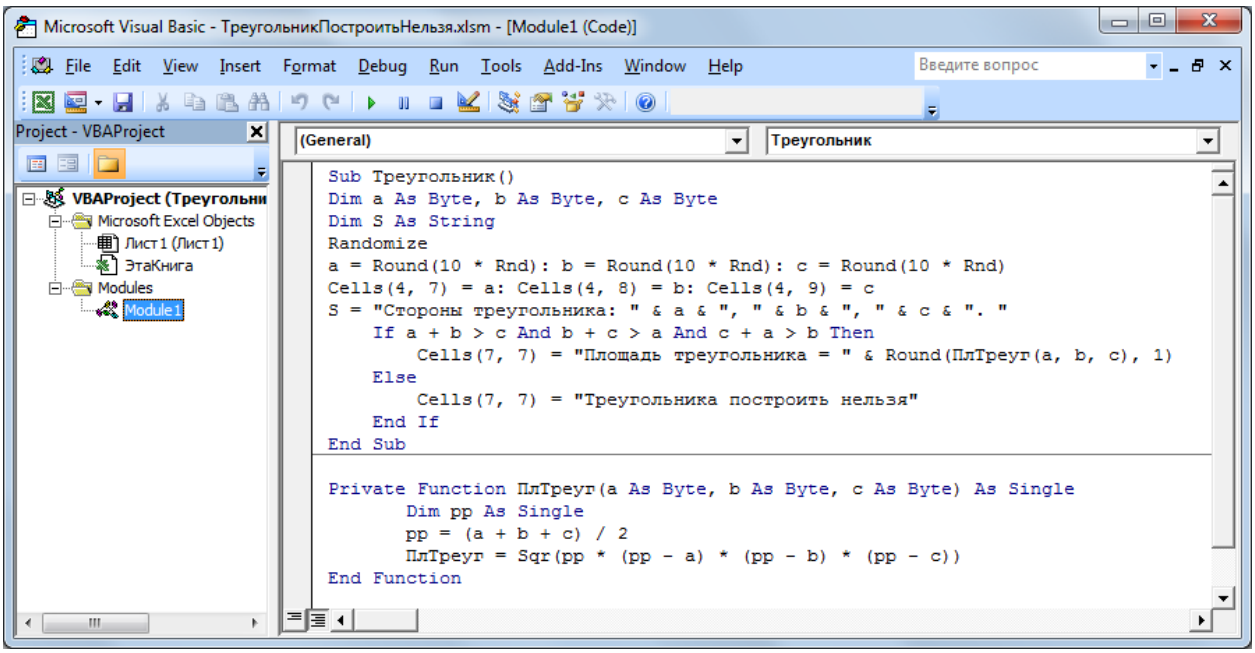


Рис. 6.11. Изменённый код макроса «Треугольник» с добавленным к нему определением функции «ПлТреуг»

Замечание. Программа может дать сообщение об ошибке, если сумма значений переменных *a, b* и *c* превысит максимально допустимое значение для типа **Byte**. Избежать этого можно двумя способами:

1. В определении функции заменить выражение $pp = (a + b + c)/2$ на выражение $a/2 + b/2 + c/2$ или
2. Присвоить этим переменным другой числовой тип, например, **Integer**.

Задание 8. Пусть в ячейке «A1» находится произвольная строка символов. Например: «`Клара у Карла украла кораллы а Карл у Клары украл кларнет`». Создать макрос, который все символы этой строки преобразует в символы нижнего регистра: «`клара у карла украла кораллы а карл у клары украл кларнет`» и удаляет из строки все пробелы: «кларауукарлаукралакораллыакарлуклырыукралкларнет».

Отметим, что ***задание 8*** является подготовительным для следующего ***задания 9***, в котором рассматривается определение рекурсивной функции «Реверс» и её использование для проверки, является ли строка палиндромом, т.е. одинаково читается как слева направо, так и справа налево.

Результат поместить в ячейки «A2» и «A3» листа Excel (рис. 6.12).

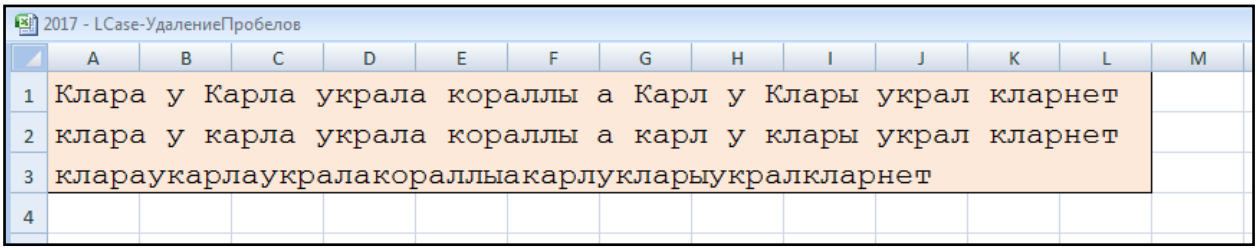


Рис. 6.12. Результат работы макроса, построенного в ходе выполнения задания 8

Решение (код макроса):

```
Sub НижнийРегистр()
    Dim S1 As String, S2 As String
    S1 = Range("A1")
    S2 = LCase(S1)
    Range("A2") = S2
    Range("A3") = УдалениеПробелов(S2)
End Sub

Function УдалениеПробелов(S As String) As String
   If S = "" Then GoTo M
   If Mid(S, 1, 1) = " " Then
      S = УдалениеПробелов(Mid(S, 2))
               Очередной символ (пробел) удалён
   Else
      S = Mid(S, 1, 1) & УдалениеПробелов(Mid(S, 2))
               Очередной символ (не пробел) оставлен
   End If
M:      УдалениеПробелов = S
End Function
```

Задание 9. Создайте макрос с определением рекурсивной функции «Реверс», а также с проверкой, является ли строка палиндромом, т.е. одинаково читается как слева направо, так и справа налево.

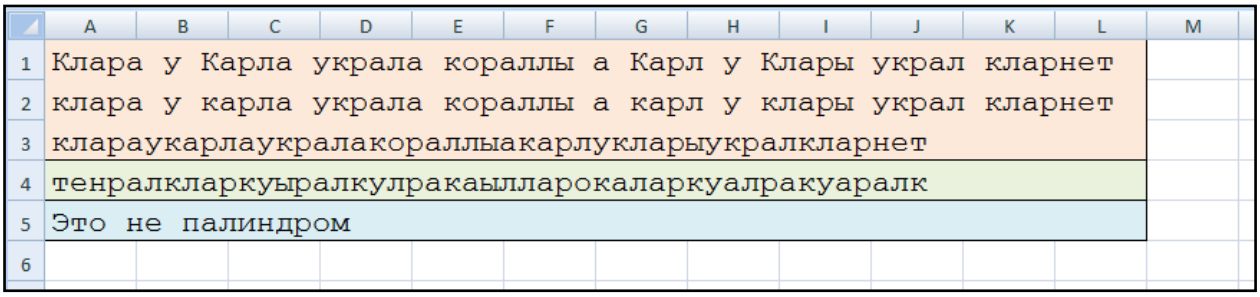
Для демонстрации работы этого макроса целесообразно воспользоваться результатом, полученным в предыдущем задании (расширить построенный в этом задании проект приложения Excel).

Решение (код макроса):

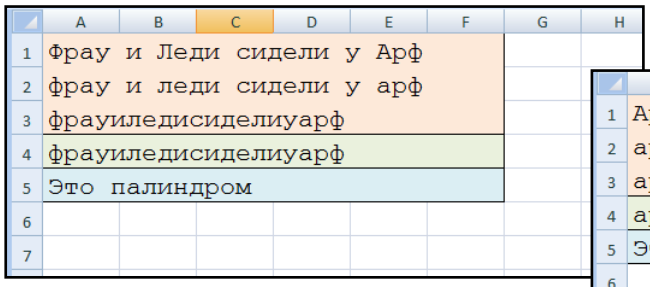
```
Sub РеверсПалиндром()
    Dim S1 As String, S2 As String
    S1 = Range("A3")
    S2 = Reverse(S1)
    Range("A4") = S2
```

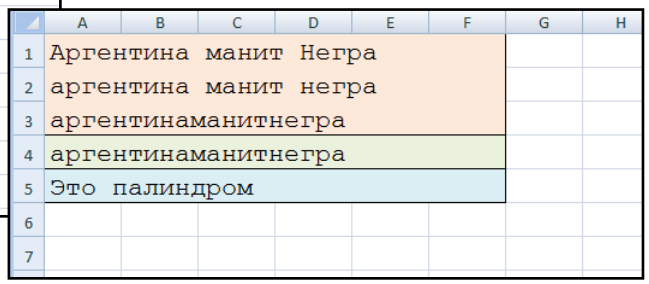
```
        If S1 = S2 Then
            Range("A5") = "Это палиндром"
        Else
            Range("A5") = "Это не палиндром"
        End If
End Sub

Private Function Reverse(S As String) As String
    If S = "" Then
        Reverse = S
    Else
        Reverse = Mid(S, Len(S), 1) & _
        Reverse(Mid(S, 1, Len(S) - 1))
    End If
End Function
```

Результат работы – на рис. 6.13.



а)





б) *в*)

Рис. 6.13. Результат работы макроса, построенного в ходе выполнения задания 9: *а*) строка не является палиндромом; *б*) и *в*) строки являются палиндромами

Глава 7

Данная глава – это текст пятой лекции по курсу программирования на языке Visual Basic для офисных приложений. Тема лекции «Операторы для организации повторений (циклов) языка VBA». Содержание лекции можно структурировать следующим образом:

- Оператор **For … Next** для организации повторений с использованием счётчика. Синтаксис, семантика и примеры применения оператора **For … Next.**
- Оператор **Do … Loop** для организации повторений с использованием проверки логических условий. Синтаксис, семантика и примеры применения оператора **Do … Loop.**
- Пример использования оператора **For … Next** для построения графика спирали – функции, заданной в полярных координатах.
- Использование многоуровневых операторов **For … Next** («цикл в цикле»). Пример использования «цикла в цикле» для построения изображения шахматной доски с разметкой клеток.
- Пример использования оператора **Do … Loop** для вычисления значения числа **π** методами Лейбница и Эйлера.

7.1. Оператор для организации повторений For … Next – «цикл со счётчиком»

Использование специальных операторов для организации циклов позволяет программировать многократно повторяющиеся действия более эффективно и наглядно, чем с помощью лишь условных и безусловных переходов, как это демонстрировалось в предыдущей лекции.

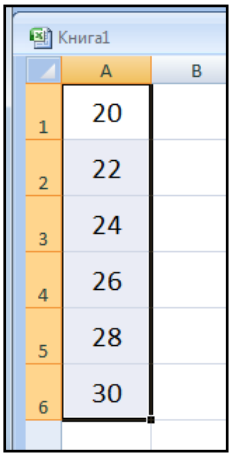
В данном разделе рассмотрим один из операторов цикла – «цикл со счётчиком». Его ***синтаксис*** следующий:

```
For <имя счётчика> = <нач. значение> To <кон. значение> [Step <шаг>]
    <операторы, которые должны повторяться>
Next [<имя счётчика>]
```

Напомним, что квадратные скобки [и] указывают на то, что значение между ними может отсутствовать.

Семантика (смысл) оператора **For … Next** такова:

<имя счётчика> – это имя переменной, которая в начале работы цикла принимает <нач. значение>. Это значение сохраняется неизменным при выполнении «тела цикла» (<операторов, которые должны повторяться>), т.е. всех операторов, записанных до ключевого слова **Next**. После чего значение счётчика изменяется на <шаг>, и, если оно ещё не достигло конечного значения (<кон. значения>), «тело цикла» опять выполняется.



Значение <шаг> – это число (оно может быть как положительным, так и отрицательным). Запись **Step** <шаг> может отсутствовать, тогда, по умолчанию, <шаг> считается равным единице.

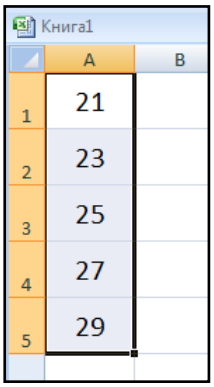
Пример 7.1. Сгенерировать с помощью цикла **For … Next** и поместить в столбец «A» листа Excel все чётные числа, лежащие на интервале [20, 30]. Сколько их?

Рис. 7.1. Чётные числа N ϵ [20, 30]

Всего этих чётных чисел 6.

Решение – на рис. 7.1 и в коде процедуры макроса:

```
Sub Четные()
   For N = 20 To 30 Step 2
      Cells(1 + (N - 20) / 2, 1) = N
   Next
End Sub
```



Пример 7.2. Сгенерировать с помощью цикла **For … Next** и поместить в столбец «A» листа Excel все нечётные числа на интервале [20, 30]. Сколько их?

Рис. 7.2. Нечётные числа N ϵ [20, 30]

Всего этих нечётных чисел 5.

Решение – на рис. 7.2 и в коде процедуры макроса:

```
Sub Нечетные()
   For N = 21 To 30 Step 2
      Cells(1 + (N - 21) / 2, 1) = N
   Next
End Sub
```

По сравнению с примером 7.1 изменено только начальное значение счётчика: 20 на 21. Отметим, что конечное значение счётчика, равное 30, можно не менять, хотя счётчик в ходе работы цикла это значение и не примет. (Оператор **For … Next** устроен так, что если значение счётчика превысит значение, заявленное как конечное, цикл завершается автоматически.)

Пример 7.3. Используем пример 4.4 из лекции № 4, в котором рассматривалось построение ряда Фибоначчи 1, 1, 2, 3, 5, 8, 13, … с помощью операторов условного и безусловного переходов. Теперь сделаем это с помощью оператора цикла со счётчиком **For … Next**.

Сначала программа должна запросить у пользователя число **imax** – максимальное значение счётчика – номера числа Фибоначчи в строящемся ряду (рис. 7.3), после чего вычислить и выдать ответ (рис. 7.4):

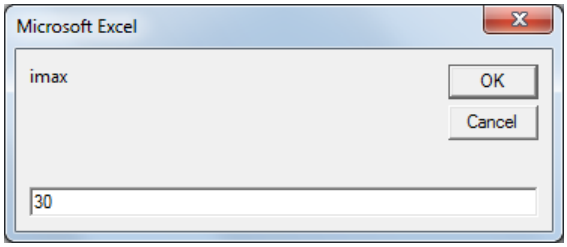


Рис. 7.3. Ввод пользователем значения imax - номера максимального числа в строящемся ряду Фибоначчи

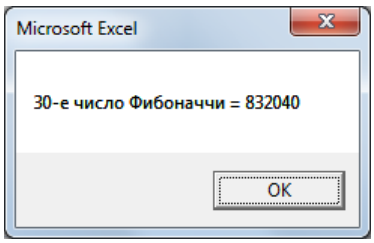


Рис. 7.4. Выдача искомого значения числа Фибоначчи по рассмотренному в примере 4.4 алгоритму

	A
1	1
2	1
3	2
4	3
5	5
6	8
7	13
8	21
9	34
10	55
11	89
12	144
13	233
14	377
15	610
16	987
17	1597
18	2584
19	4181
20	6765
21	10946
22	17711
23	28657
24	46368
25	75025
26	121393
27	196418
28	317811
29	514229
30	832040

Рис. 7.5. Первые 30 чисел ряда Фибоначчи, занесённые программой в столбец «А» листа книги Excel

Решение (код макроса):

```
Sub Фибоначчи()
    Dim imax As Long, F1 As Long, F2 As Long
    imax = InputBox("imax")
    F1 = 1
    F2 = 1
    Range("A" & 1) = F1
    Range("A" & 2) = F2
   For i = 3 To imax Step 2
      F1 = F1 + F2
      F2 = F2 + F1
      Range("A" & i) = F1
      Range("A" & i + 1) = F2
   Next
   MsgBox imax & "-е число Фибоначчи = " & F2
End Sub
```

7.2. Оператор для организации повторений Do … Loop – «цикл с условием»

Рассмотрим ***синтаксис*** оператора **Do … Loop –** оператора «цикла с условием».

Применяются следующие формы записи. Такая:

Do { **While, Until** } <условное выражение>

<операторы, которые должны повторяться>

Loop

Либо такая:

Do

<операторы, которые должны повторяться>

Loop { **While, Until** } <условное выражение>

Семантика. Фигурные скобки в описании синтаксиса означают, что возможен любой вариант из перечисленных, в данном случае:

или **While**, или **Until**. Если:

(1) **While**, тогда цикл *будет повторяться*, пока условное выражение имеет значение **True (Истина)**;

(2) **Until**, тогда цикл *закончится*, если условное выражение примет значение **True (Истина)**.

Пример 7.4. Для демонстрации цикла с условием рассмотрим ещё одну задачу на «ряд Фибоначчи». Но теперь надо найти не заданное количество этих чисел, а все числа в пределах тысячи: сколько их, заранее неизвестно. Решение этой задачи может быть представлено следующей процедурой:

```
Sub ДемонстрацияЦиклаDoLoop()
    Dim i As Integer
    Dim X As Long, Y As Long
    i = 1: X = 1: Y = 1
        Do While Y < 1000
            Range("C" & i) = X
            Range("C" & i + 1) = Y
            X = X + Y
            Y = X + Y
            i = i + 2
        Loop
End Sub
```

Выделенные полужирным шрифтом символы (ключевое слово **While** и стоящее за ним условное выражение – неравенство) могут стоять как после ключевого слова **Do**, так и после ключевого слова **Loop**. Вместо выражения **While** $Y < 1000$ можно записать выражение **Until** $Y > 1000$. Результат работы программы в данном случае от этого не изменится: в столбце «C» активного листа Excel появятся 16 первых чисел ряда Фибоначчи. Последнее из них – число 987.

7.3. Пример использования оператора For … Next для построения графика спирали – функции, заданной в полярных координатах

Графики функций принято строить либо в традиционных декартовых координатах: $y = F(x)$, либо в полярных координатах $\rho = F(\varphi)$, или: $r = F(f)$, где φ (или f) – это аргумент функции – угол, а ρ (или r) – это длина радиус-вектора.

Пример 7.5. Покажем, как можно использовать встроенные возможности приложения MS Excel («Вставка диаграмм») и язык VBA (оператор цикла со счётчиком **For … Next**), чтобы построить диаграмму в полярных координатах.

Известно, что такие диаграммы как «спираль», очень удобно представлять именно в полярных, а не в декартовых координатах, например:

$r = a * f + b$, где, например: $a = 3$, $b = 100$,

а угол меняется в пределах от 0 до 10π.

Диаграмма имеет вид, показанный на рис. 7.6.

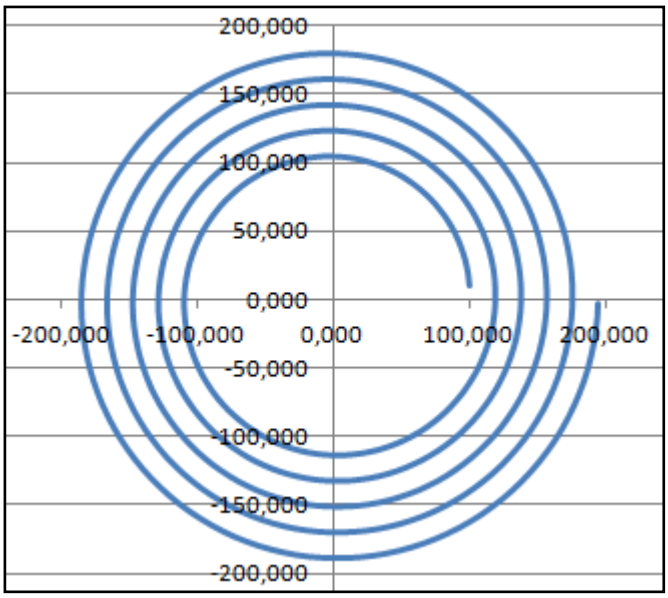


Рис. 7.6. Диаграмма спирали $r = a * f + b$, $a = 3$, $b = 100$, $0 < f < 10\pi$

Далее, представлен код макроса, который:

1) с помощью цикла со счётчиком строит координаты спирали в полярной системе координат (угол меняется от 0 до 10π);
2) заполняет столбцы «A» и «B» листа книги Excel соответственно координатами x и y графика спирали в декартовых координатах, которые легко пересчитываются по координатам f и r полярной системы:

$x = r * \mathrm{Cos}(f)$; $y = r * \mathrm{Sin}(f)$.

Отметим, что за основу процедуры данного макроса взята программа из учебного пособия НИЯУ МИФИ по изучению языка Visual Basic .NET[4].

```
Sub Спираль()
    Const pi = 3.1415
    Dim a As Double, b As Double, s As Double
    a = 3
    b = 100
    s = 0.05
    i = 0
    For f# = 0.1 To 10 * pi Step s
        i = i + 1
        r# = a * f + b
        Cells(i, 1) = r * Cos(f)
        Cells(i, 2) = r * Sin(f)
    Next f
End Sub
```

В результате выполнения данного макроса и применения вставки диаграммы с помощью выполнения цепочки меню: **Вставка → Диаграммы → Точечная → Точечная с гладкими кривыми** (рис. 7.7).

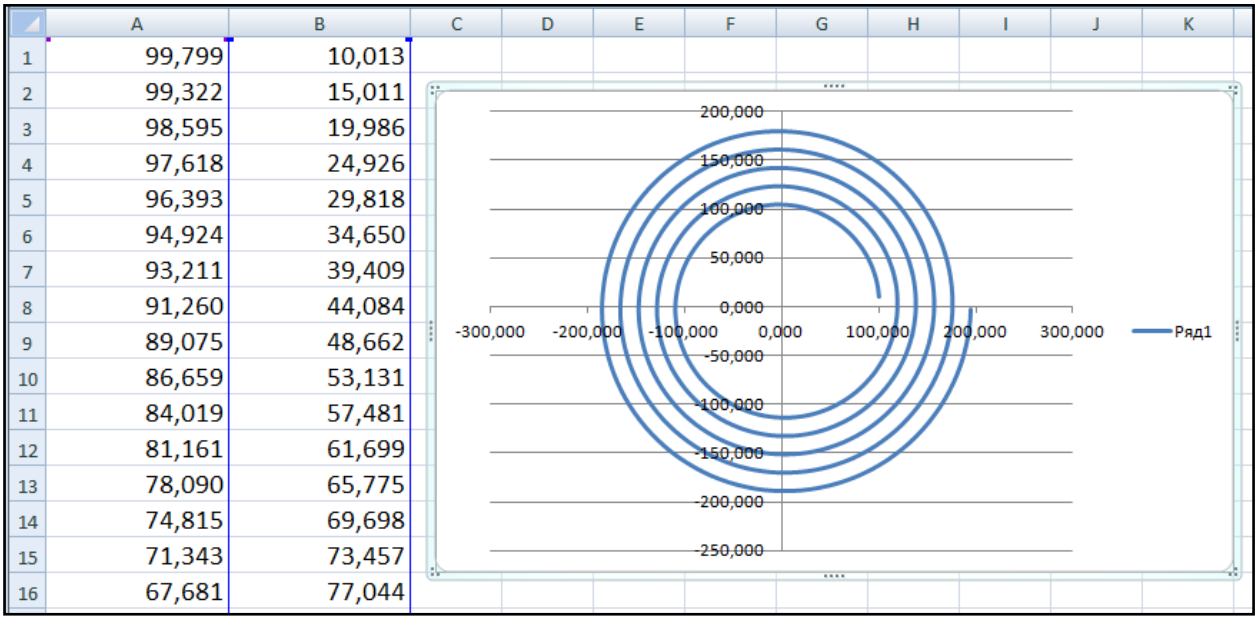


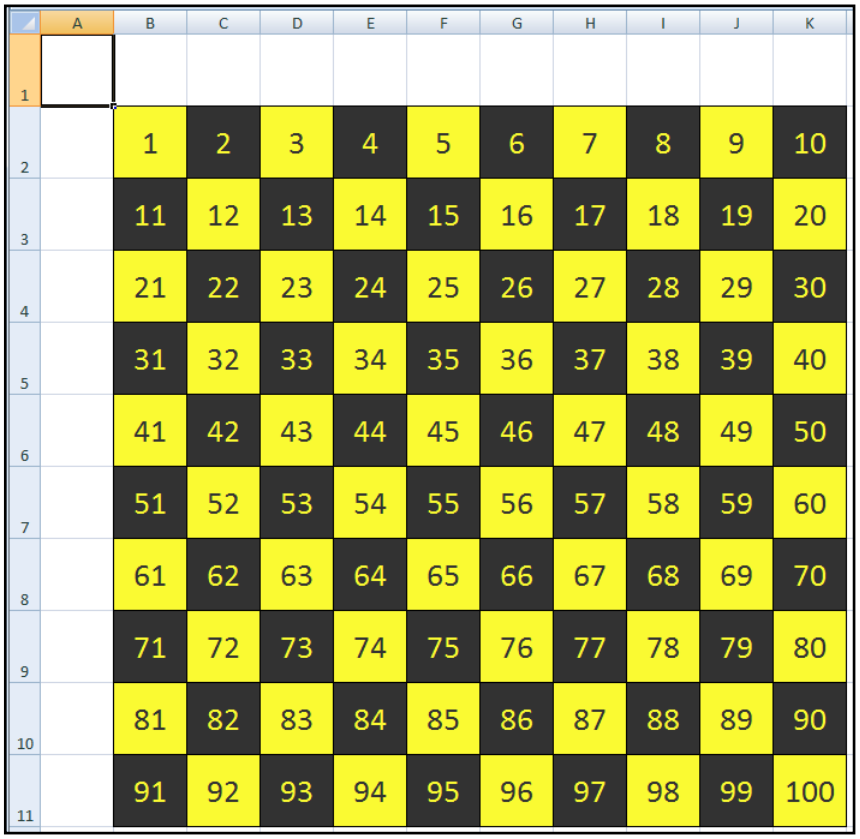
Рис. 7.7. Диаграмма спирали $r = a * f + b$, $a = 3$, $b = 100$, $0 < f < 10\pi$ на листе Excel

[4] Волченков Н.Г., Троицкий А.К. Проектирование Windows-приложений на языке Visual Basic .NET (2005, 2010). Учебно-методическое пособие. – М.: НИЯУ МИФИ, 2015.

7.4. Использование многоуровневых операторов For … Next («цикл в цикле»). Пример использования «цикла в цикле» для построения изображения шахматной доски с разметкой клеток

В данном разделе демонстрируется возможность применения так называемого двойного цикла **For … Next** (цикла в цикле). Здесь двойной цикл удобен тем, что позволяет осуществлять сканирование некой матрицы с помощью её просмотра, по каждой строке слева направо и сверху вниз – при переходе к новой строке.

Пример 7.6. Построить изображение шахматной доски произвольной размерности (её задает пользователь) и пронумеровать клетки – слева направо, сверху вниз (рис. 7.8).



	A	B	C	D	E	F	G	H	I	J	K
1											
2		1	2	3	4	5	6	7	8	9	10
3		11	12	13	14	15	16	17	18	19	20
4		21	22	23	24	25	26	27	28	29	30
5		31	32	33	34	35	36	37	38	39	40
6		41	42	43	44	45	46	47	48	49	50
7		51	52	53	54	55	56	57	58	59	60
8		61	62	63	64	65	66	67	68	69	70
9		71	72	73	74	75	76	77	78	79	80
10		81	82	83	84	85	86	87	88	89	90
11		91	92	93	94	95	96	97	98	99	100

Рис. 7.8. Сквозная нумерация клеток шахматной доски

```
Sub ШахДоска1()
  Dim N As Integer
  N = InputBox("Введите размерность шахматной доски", _
                "Ввод N", "10")
 For i = 1 To N
   For j = 1 To N
       k = N * (i - 1) + j
```

```
        Cells(i + 1, j + 1) = k
        Cells(i + 1, j + 1).Borders.LineStyle = 1
      If (i + j) Mod 2 = 0 Then
        Cells(i + 1, j + 1).Interior.Color = _
              RGB(250, 250, 50)
        Cells(i + 1, j + 1).Font.Color = _
              RGB(50, 50, 50)
      Else
        Cells(i + 1, j + 1).Interior.Color = _
              RGB(50, 50, 50)
        Cells(i + 1, j + 1).Font.Color = _
              RGB(250, 250, 50)
      End If
    Next
  Next
End Sub
```

В этой программе используются средства языка VBA for Excel:

1) для обрамления клеток рамкой используется свойство клетки **Borders.LineStyle**;
2) для закрашивания клеток определёнными цветами с помощью функции **RGB**, значение которой присваивается свойству клетки **Interior.Color**;
3) для выбора цвета шрифта надписи на клетке с помощью функции **RGB**, значение которой присваивается свойству клетки **Font.Color**.

Функция **RGB**, как нетрудно догадаться, возвращает цвет в соответствии со значениями трёх своих аргументов (соответственно, *красного, зелёного* и *синего*). Например, функция **RGB(250, 250, 50)** возвращает цвет, близкий к жёлтому (равные доли красного и зелёного дают жёлтый цвет), а функция **RGB(50, 50, 50)** возвращает тёмно-серый цвет.

Пример 7.7. Построить изображение шахматной доски размерностью 8x8 и пронумеровать клетки в соответствии с шахматной нотацией: первая снизу строка a1, b1, c1, …; вторая снизу строка a2, b2, c2, … и т.д. (рис. 7.9).

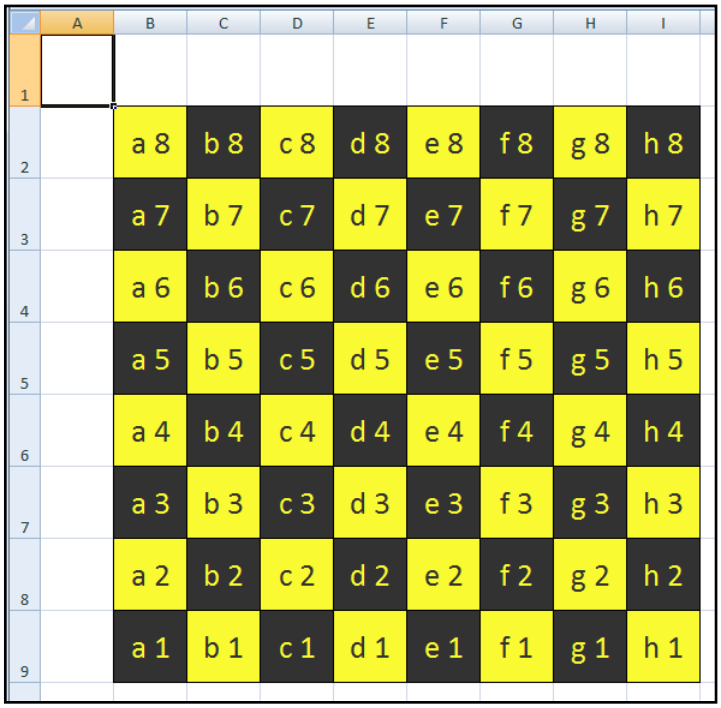


Рис. 7.9. Традиционная «шахматная» нумерация клеток шахматной доски

```
Sub ШахДоска2()
 Sa = "" : Sc = "" : N = 8
 For i = 1 To N
   For j = 1 To N
     Переименовать N, i, j, Sb, Sc
     Cells(i + 1, j + 1) = Sb & Sc
     Cells(i + 1, j + 1).Borders.LineStyle = 1
     If (i + j) Mod 2 = 0 Then
       Cells(i + 1, j + 1).Interior.Color = _
                           RGB(250, 250, 50)
       Cells(i + 1, j + 1).Font.Color = _
                           RGB(50, 50, 50)
     Else
       Cells(i + 1, j + 1).Interior.Color = _
                           RGB(50, 50, 50)
       Cells(i + 1, j + 1).Font.Color = _
                           RGB(250, 250, 50)
     End If
   Next
 Next
End Sub
```

```
Sub Переименовать(N As Integer, i As Integer, _
    j As Integer, Sb As String, Sc As String)
  Select Case j
    Case 1: Sb = "a" : Case 2: Sb = "b"
```

```
    Case 3: Sb = "c" : Case 4: Sb = "d"
    Case 5: Sb = "e" : Case 6: Sb = "f"
    Case 7: Sb = "g" : Case 8: Sb = "h"
  End Select
 Sc = Str(N + 1 - i)
End Sub
```

7.5. Пример использования оператора Do … Loop для вычисления значения числа π методами Лейбница и Эйлера

Пример 7.8. Задача вычисления площади круга в инженерных целях и связанная с ней задача приближённого вычисления числа π («ПИ») волновали человечество с древних времён. В новой истории (после 16 – 17-го веков) в связи с научно-технической революцией интерес к этой задаче возрос.

Великий немецкий математик Готфрид Вильгельм Лейбниц (1646 – 1716) предложил следующую формулу для вычисления приближенного значения числа π – сходящийся ряд:

`Pi = 4 * (1 – 1/3 + 1/5 – 1/7 + 1/9 …).`

Вычисление по этой формуле легко запрограммировать с использованием цикла с условием – оператора **Do While** <условие> … **Loop.** В качестве <условия> в этом цикле используется неравенство:

Abs(pi(k) – pi(k – 1)) < d.

Здесь pi(k) – значение, полученное по указанной формуле на k-м шаге; d – малое число, например 0,00000001, позволяющее получить заданную точность полученного результата (здесь значение числа π с точностью до заданного знака после десятичной точки).

Можно предложить и более простую задачу исследования вычисления по приведённой формуле: сравнивать значение, полученное на k-м шаге, не с предыдущим значением, полученным на (k-1)-м шаге, а с «квазиточным» значением числа π, например с числом, заданным 15-ю знаками после десятичной точки. Именно такой пример рассмотрен в литературе[5].

[5] Волчёнков Н.Г. Программирование на Visual Basic 6: В 3-х ч. Часть 3. – М.: ИНФРА-М, 2012, с.78-79, с.188-189 – Задача 6.10.

На полстолетия позже Лейбница, в 1736 г. российский математик Леонард Эйлер (1707 – 1783) предложил другой ряд для приближённого вычисления числа π:

```
Pi = Sqr(6 * (1/(1^2) + 1/(2^2) + 1/(3^2) + …)).
```

Вычисление по формуле Эйлера программируется аналогично тому, как это только что было продемонстрировано (точнее, представлено по ссылке на литературу) для формулы Лейбница[6].

Ниже приведены коды двух макросов, написанных на языке **VBA for Excel** – по «образу и подобию» программ на языке **Visual Basic 6**, приведённых в указанной литературе, с помощью которых можно получить и проанализировать результаты, показанные далее – на рис. 7.11 – 7.12.

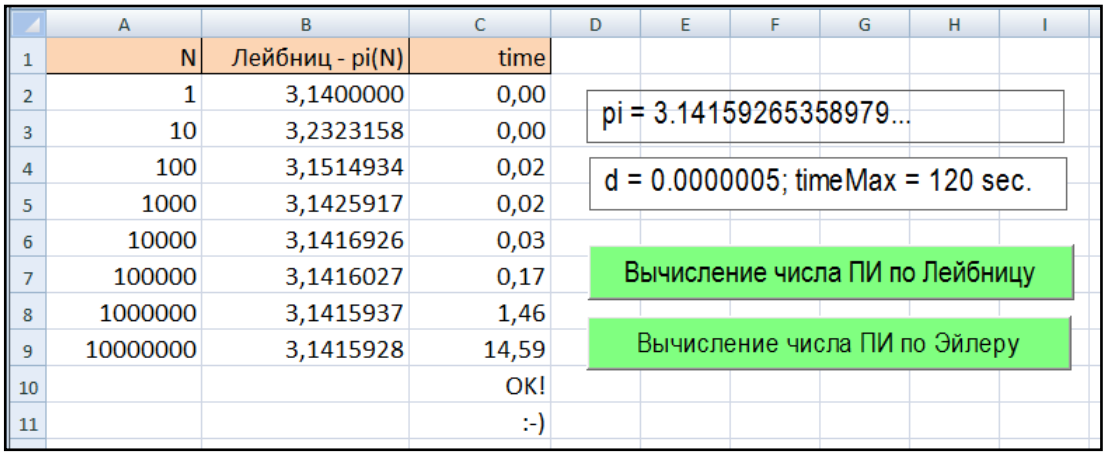
```
Sub ЛейбницNew()
    Const pi As Double = 3.14159265358979
    Dim N As Double, b As Double, d As Double, _
        piN As Double
    Dim z As Integer, i As Integer, k As Long
    Dim t0 As Single, t As Single
        k = 1
        d = 0.0000005
        i = 2
        Cells(1, 2) = "pi(N)"
        Cells(1, 2) = "Лейбниц - " & Cells(1, 2)
        Cells(i, 1) = 1: Cells(i, 2) = 3.14
        Cells(i, 3) = 0
        t0 = Timer
    Do While Abs(pi - piN) > d
        k = k * 10: z = -4: piN = 0
        For N = 0 To k
            z = -z
            b = z / (2 * N + 1)
            piN = piN + b
            If Timer - t0 > 10 Then
               Cells(i + 1, 1) = k
               Cells(i + 1, 2) = piN
               Cells(i + 1, 3) = "Time!"
               Exit Do
            End If
```

[6] Волчёнков Н.Г. Программирование на Visual Basic 6: В 3-х ч. Часть 3. – М.: ИНФРА-М, 2012, с.79, с.189-190 – Задача 6.12.

```
        Next
        i = i + 1
        Cells(i, 1) = k
        Cells(i, 2) = piN
        Cells(i, 3) = Round(Timer - t0, 2)
    Loop
        If Cells(i + 1, 3) = "Time!" Then
           Cells(i + 2, 3) = ";-("
        Else
           Cells(i + 1, 3) = "OK!"
           Cells(i + 2, 3) = ":-)"
           End If
End Sub
```

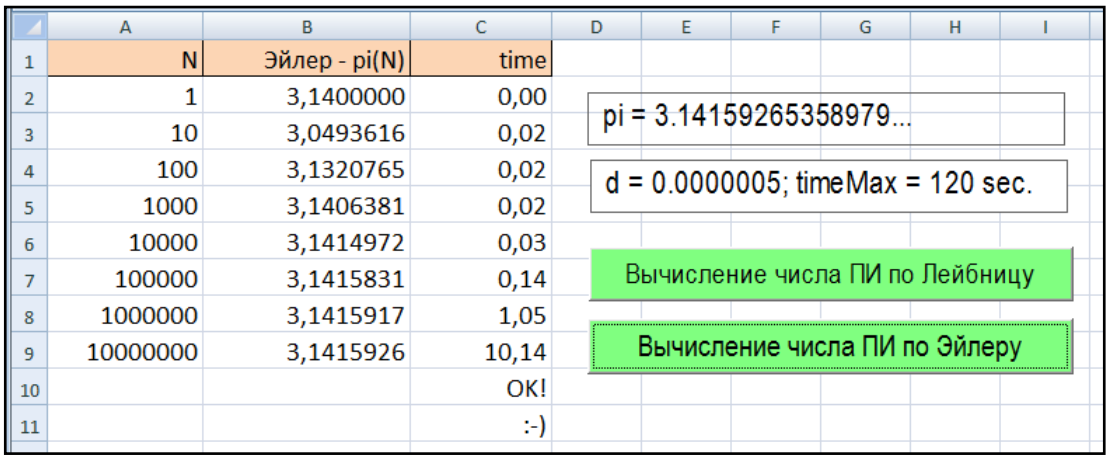
```
Sub ЭйлерNew()
    Const pi As Double = 3.141592653589793
    Dim N As Double, b As Double, d As Double, _
    piN As Double, z As Double
    Dim i As Integer, k As Long
    Dim t0 As Double, t As Double, ti As Double
        k = 1: piN = 3.14
        d = 0.0000005
        i = 2
        Cells(1, 2) = "pi(N)"
        Cells(1, 2) = "Эйлер - " & Cells(1, 2)
        Cells(i, 1) = 1: Cells(i, 2) = 3.14
        Cells(i, 3) = 0
        t0 = Timer
   Do While Abs(pi - piN) > d
            ti = Timer
            If Abs((ti - t0)) > 120 Then
               Cells(i + 1, 1) = ""
               Cells(i + 1, 2) = ""
               Cells(i + 1, 3) = "Time!"
               Exit Do
            End If
        k = k * 10: z = 0: piN = 0
        t = Timer
            For N = 1 To k
                z = z + 6 / N ^ 2
            Next
        piN = Sqr(z)
        i = i + 1
        Cells(i, 1) = k
        Cells(i, 2) = piN
```

```
        Cells(i, 3) = Round(Timer - t0, 2)
    Loop
        If Cells(i + 1, 3) = "Time!" Then
            Cells(i + 2, 3) = ";-("
        Else
            Cells(i + 1, 3) = "OK!"
            Cells(i + 2, 3) = ":-)"
        End If
End Sub
```



	A	B	C
1	N	Лейбниц - pi(N)	time
2	1	3,1400000	0,00
3	10	3,2323158	0,00
4	100	3,1514934	0,02
5	1000	3,1425917	0,02
6	10000	3,1416926	0,03
7	100000	3,1416027	0,17
8	1000000	3,1415937	1,46
9	10000000	3,1415928	14,59
10			OK!
11			:-)

Рис. 7.11. Результат вычисления приближённого значения числа π методом Лейбница

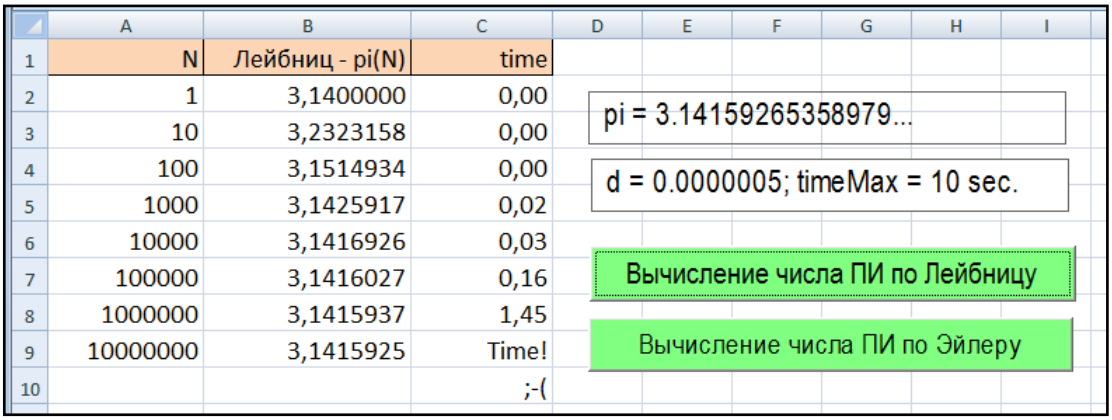


	A	B	C
1	N	Эйлер - pi(N)	time
2	1	3,1400000	0,00
3	10	3,0493616	0,02
4	100	3,1320765	0,02
5	1000	3,1406381	0,02
6	10000	3,1414972	0,03
7	100000	3,1415831	0,14
8	1000000	3,1415917	1,05
9	10000000	3,1415926	10,14
10			OK!
11			:-)

Рис. 7.12. Результат вычисления приближённого значения числа π методом Эйлера

Вывод. Вычисление приближённого значения числа π по методу Эйлера немного точнее и немного быстрее, чем по методу Лейбница ☺…

Покажем также результат работы программы (для определённости, по методу Лейбница) при более жестком ограничении, наложенном на время её работы (10 с вместо 120 с). Результат показан на рис. 7.13.



	A	B	C
1	N	Лейбниц - pi(N)	time
2	1	3,1400000	0,00
3	10	3,2323158	0,00
4	100	3,1514934	0,00
5	1000	3,1425917	0,02
6	10000	3,1416926	0,03
7	100000	3,1416027	0,16
8	1000000	3,1415937	1,45
9	10000000	3,1415925	Time!
10			;-(

Рис. 7.13. Процесс вычисления приближённого значения числа π, прерванный из-за недостатка времени

Глава 8

Данная глава – это текст следующей, шестой лекции по курсу программирования на языке Visual Basic для офисных приложений. Содержание лекции можно структурировать следующим образом:

- Понятие «массив» как обобщение понятия «переменная».
- Одномерный массив. Объявление одномерного массива. Пример заполнения массива случайными значениями и помещения этих значений на лист Excel.
- Статические и динамические массивы.
- Примеры использования одномерного массива: поиск максимального и минимального элемента, «пузырьковая» сортировка.
- Пример использования динамического массива: циклический ввод в массив заранее неизвестного числа элементов.
- Запись массива в файл и чтение из файла в массив.
- Бинарный поиск номера заданного элемента в отсортированном массиве (поиск методом «дихотомии») на двух примерах: поиск номера фамилии и угадывание задуманного числа.
- Многомерные массивы. Пример использования двумерного массива: поиск «минимакса» («седловой точки» на поверхности гиперболического параболоида).

8.1. Понятие «массив» как обобщение понятия «переменная»

Вспомним определённое в первой лекции понятие «переменная».

Переменная – это поименованная область компьютерной памяти, в которой хранятся данные определенного типа. Они могут меняться в ходе выполнения программы. (Можно сказать короче: «переменная – это изменяемая часть программы».) Переменная должна иметь *имя, тип, значение*.

Массив – это обобщение понятия переменной.

По определению, ***массив*** – это набор элементов одинакового типа, имеющих одинаковое имя. Как и отдельная переменная, массив является областью компьютерной памяти. Данные, хранящиеся в массиве (как и значения переменных), могут меняться в ходе выполнения программы.

Массивы бывают как одномерными, так и многомерными (двумерными, трёхмерными и т.д.). Далее мы ограничимся рассмотрением только одномерных и двумерных массивов.

Одномерный массив представляет *вектор* значений одного определенного типа. Двумерный массив – это не вектор, а *матрица* значений одного определённого типа.

8.2. Одномерный массив. Объявление одномерного массива. Пример заполнения массива случайными значениями и помещения этих значений на лист Excel

Так как в массиве не один, а несколько элементов с одинаковым именем, для выявления конкретного элемента надо указывать не только имя, но и номер (*индекс*) этого элемента. Например:

MMM(5) – это пятый элемент в массиве с именем MMM, если нумерация в этом массиве начинается с единицы (или шестой элемент, если нумерация начинается с нуля).

Как и переменные, массивы объявляются с помощью ключевых слов: **Dim, Private, Public**. В чём различие этих определений, было объяснено на первой лекции. Объявляются массивы двумя способами, которые демонстрирует пример.

Способ 1: **Dim** Фамилия(24) **As String**.

Так объявлен массив фамилий, состоящий из 25 элементов. По умолчанию, нумерация начинается с нуля. Если вы хотите, чтобы нумерация начиналась с единицы, надо объявить массив так:

Способ 2: **Dim** Фамилия(1 **To** 25) **As String**.

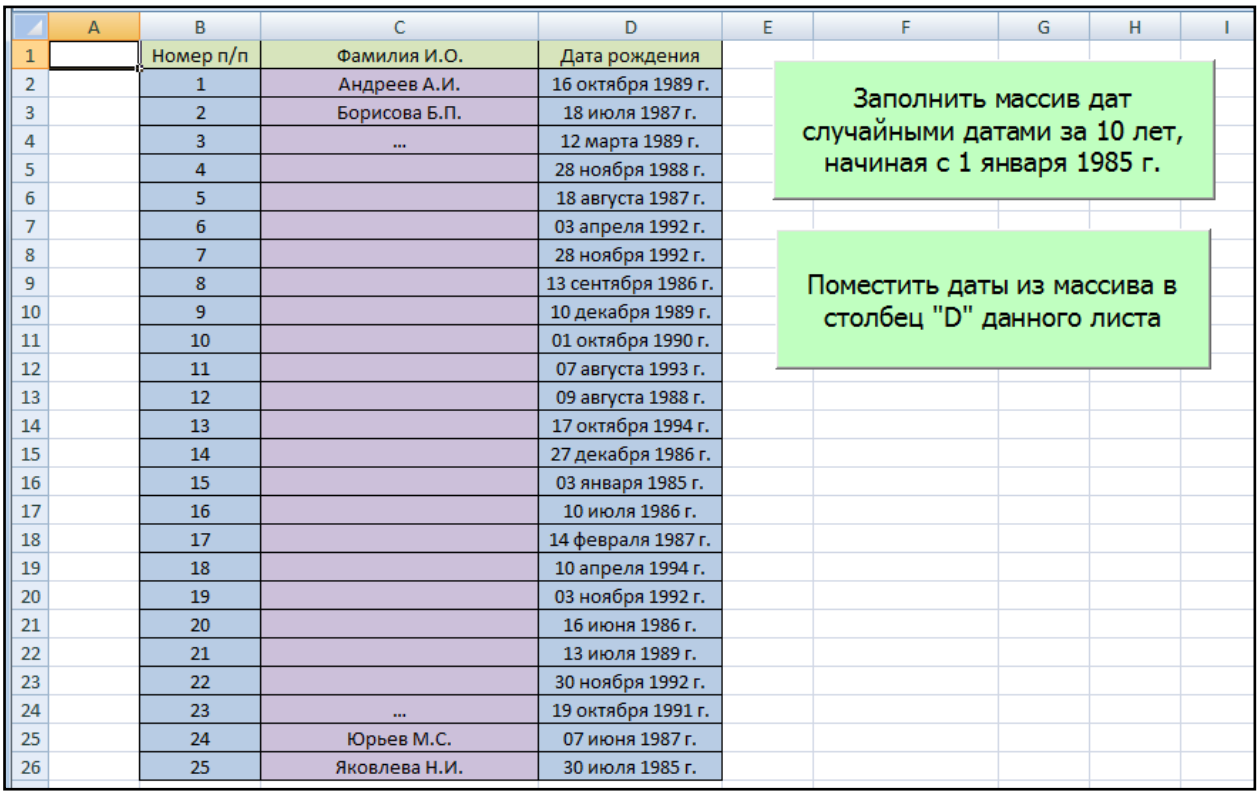
Выражение в скобках после имени массива называется ***диапазоном*** изменения индекса массива. Обращаем внимание на то, что в приведённых примерах, хотя диапазоны различны, количество элементов одинаково.

Пример 8.1. Рассмотрим одномерный массив с именем **BDate** данных типа **Date** – массив дат рождения сотрудников некоей фирмы. Допустим, сотрудников 25 человек. Предположим, «в шутку», что дни рождения этих сотрудников равномерно распределены в 10-летнем интервале – от 1/1/1985 до 31/12/1994.

Задача данного демонстрационного примера:

- поместить в столбец «C» листа книги Excel фамилии и инициалы сотрудников вручную: от Андреева А.И. до Яковлевой Н.И.;
- с помощью 1-й командной кнопки заполнить массив случайными датами;
- с помощью 2-й командной кнопки поместить в столбец «D» сгенерированные случайно данные из массива **BDate** – даты рождения сотрудников.

Результат – на рис. 8.1.



	A	B	C	D
1		Номер п/п	Фамилия И.О.	Дата рождения
2		1	Андреев А.И.	16 октября 1989 г.
3		2	Борисова Б.П.	18 июля 1987 г.
4		3	...	12 марта 1989 г.
5		4		28 ноября 1988 г.
6		5		18 августа 1987 г.
7		6		03 апреля 1992 г.
8		7		28 ноября 1992 г.
9		8		13 сентября 1986 г.
10		9		10 декабря 1989 г.
11		10		01 октября 1990 г.
12		11		07 августа 1993 г.
13		12		09 августа 1988 г.
14		13		17 октября 1994 г.
15		14		27 декабря 1986 г.
16		15		03 января 1985 г.
17		16		10 июля 1986 г.
18		17		14 февраля 1987 г.
19		18		10 апреля 1994 г.
20		19		03 ноября 1992 г.
21		20		16 июня 1986 г.
22		21		13 июля 1989 г.
23		22		30 ноября 1992 г.
24		23	...	19 октября 1991 г.
25		24	Юрьев М.С.	07 июня 1987 г.
26		25	Яковлева Н.И.	30 июля 1985 г.

Рис. 8.1. Лист книги Excel с датами рождения сотрудников фирмы и кнопками для вызова двух процедур: (1) генерации 25 дат и (2) размещения их в столбце «D»

Коды единого программного модуля для обеих процедур:

```
Dim BDate(24) As Date

Private Sub CommandButton1_Click()
    Randomize
    For i = 0 To 24
        N = Round(Rnd * 3652)
        BDate(i) = DateAdd("D", N, "01/01/1985")
    Next
    MsgBox ("Массив дат заполнен")
End Sub
```

```
Private Sub CommandButton2_Click()
    For i = 0 To 24
        Range("D" & i + 2) = BDate(i)
    Next
End Sub
```

8.3. Статические и динамические массивы

Массивы могут быть (и, чаще всего, бывают) *статическими*, когда заранее известен диапазон изменения индекса массива, но бывают и *динамическими*, когда заранее указанный диапазон неизвестен. В этом случае массив объявляется по-другому – с использованием «пустых» скобок:

Dim <имя>() **As** <тип>

В программе, если массив объявлен как динамический, перед первым его использованием, когда, например, станет известно число его элементов N, его необходимо переопределить с помощью ключевого слова **ReDim**:

ReDim <имя>(N – 1) **As** <тип> или

ReDim <имя>(1 **To** N) **As** <тип>

Далее, если массив опять должен изменить диапазон (при изменении значения N), его опять следует переопределить указанным выше способом.

При переопределении часто бывает необходимо сохранить в массиве уже помещённые в него значения. С этой целью используется ключевое слово **Preserve**.

ReDim Preserve <имя>(N – 1) **As** <тип> или

ReDim Preserve <имя>(1 **To** N) **As** <тип>

Часто, например, в программе на каждом новом шаге цикла приходится увеличивать верхнюю границу диапазона массива на единицу. Это можно реализовать с использованием специальной функции **UBound**, которая возвращает верхнюю границу диапазона массива:

ReDim Preserve <имя>(**UBound**(<имя>) + 1).

В разделе 8.5 будет рассмотрен пример программы, с помощью которой в динамический массив можно вводить произвольное число фамилий.

8.4. Примеры использования одномерного массива: поиск максимального и минимального элемента, «пузырьковая» сортировка

Массив – виртуальный объект: пользователь его не видит. Но результаты обработки массива в конкретной задаче, как правило, необходимо сделать видимыми для пользователя.

Рассмотрим пример такой задачи.

Пример 8.2. Пусть массив **DoubleRandom**(1 **To** 100) заполняется случайными десятичными числами типа **Double** с помощью функции **Rnd,** которая возвращает случайное число из интервала [0, 1]. Необходимо среди этих ста чисел найти максимальное и минимальное. Остальные числа нас (пользователей), допустим, не интересуют. Результат необходимо выдать с помощью окна сообщения **MsgBox**.

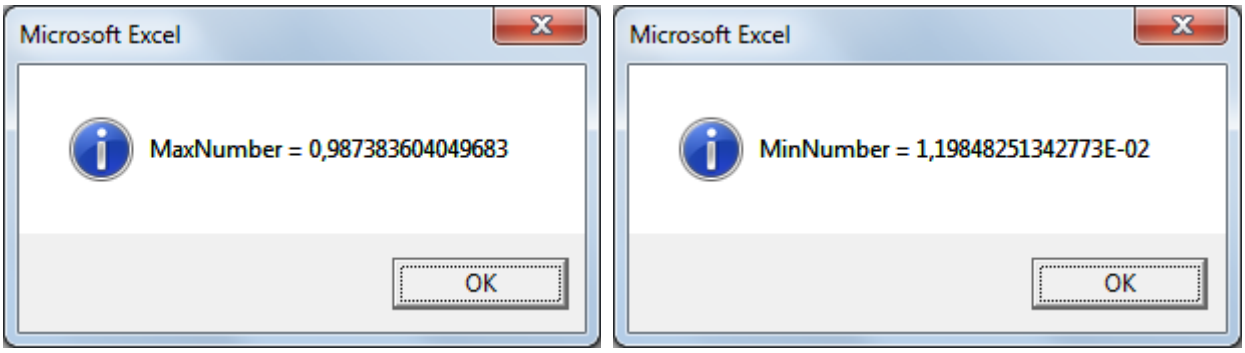
Решение задачи можно оформить в виде макроса `DoubleRandom`:

```
Sub MaxAndMin()
    Dim DoubleRandom(1 To 100) As Double
    Dim Max As Double, Min As Double
        Randomize
        For i = 1 To 100
            DoubleRandom(i) = Rnd
        Next
        Min = 1: Max = 0    ' Исходная "гипотеза"
        For i = 1 To 100
            If DoubleRandom(i) > Max Then _
                      Max = DoubleRandom(i)
            If DoubleRandom(i) < Min Then _
                      Min = DoubleRandom(i)
        Next
        MsgBox "MaxNumber = " & Max, vbInformation
        MsgBox "MinNumber = " & Min, vbInformation
End Sub
```

Здесь реализован популярный алгоритм поиска максимального и минимального элементов числового массива: поиск чисел начинается с принятия исходной, очевидно, ложной «гипотезы»: минимальное число – это пра-

вая граница интервала [0, 1], а максимальное число – это его левая граница. На каждом шаге цикла **For … Next** происходит корректировка «гипотезы».

Работа макроса завершается появлением двух сообщений (рис. 8.2).



а) *б*)

Рис. 8.2. Окна сообщений с результатами: *а*) максимальное число; *б*) минимальное число из сгенерированного списка случайных чисел

Ещё один пример: поиск самого старого и самого молодого сотрудника.

Пример 8.3. Это продолжение ***Примера 8.1.***

```
Sub СамыйМолодой()
    N = 24
    Dm = Bdate(0)
    For i = 1 To N
            If Bdate(i) > Dm Then
                Dm = Bdate(i)
            End If
    Next
    MsgBox "Дата рождения самого молодого " & Dm
End Sub
```

```
Sub СамыйСтарый()
    N = 24
    Dm = Bdate(0)
    For i = 1 To N
            If Bdate(i) < Dm Then
                Dm = Bdate(i)
            End If
    Next
    MsgBox "Дата рождения самого старого " & Dm
End Sub
```

В отличие от ***Примера 8.2***, здесь немного иная «гипотеза»: датой рождения самого «старого» и, одновременно, самого «молодого» сотрудника

вначале считается 0-й элемент массива **Bdate**(24). Как и в предыдущем примере, на каждом шаге цикла эта гипотеза корректируется.

Следующий пример – на тему алгоритма ***сортировки*** одномерного массива методом «пузырька».

Идею алгоритма демонстрирует пример на рис. 8.3.

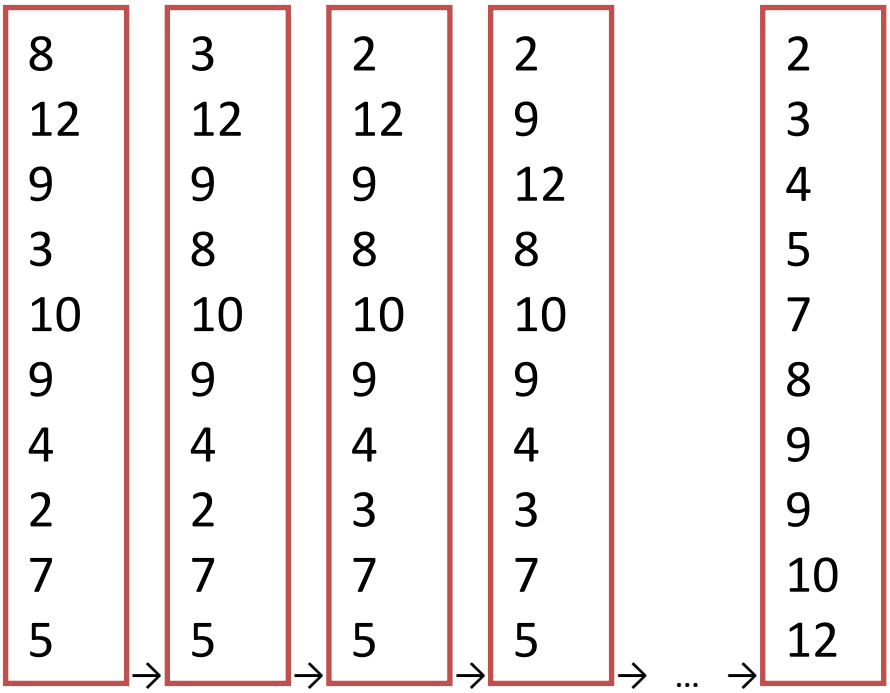


Рис. 8.3. Шаги сортировки методом «пузырька»

Массив изображается вертикально. Необходимо по шагам попарно менять местами его элементы так, чтобы в результате этих шагов элементы (числа) расположились сверху вниз по возрастанию. Такая сортировка называется «пузырьковой» в силу того, что самый маленький элемент считается самым «лёгким». Он должен «всплыть» наверх, как «пузырёк».

В данном примере первый (верхний) элемент (8) последовательно сравнивается с остальными (12, 9, 3, …). Если очередной элемент оказывается меньше, чем 8, они меняются местами (здесь 8 и 3). Затем, все остальные элементы сравниваются уже с числом 3, пока не найдётся ещё меньший элемент – это 2. Меняются местами 3 и 2. Достигли «дна». Элемент 2 закрепился на самом верху. Забыли о нём, начинаем процесс со второго элемента сверху (12). Меняем местами 12 и 9. И так далее…

Процесс реализуется с помощью «двойного цикла»:

```
For i = 0 To N - 1
   For j = i + 1 To N
     If  ai > aj Then ОбменМестами(ai, aj)
   Next j
Next i
```

Пример 8.4. В первых двух колонках листа Excel (рис. 8.4) номера фамилий и сами фамилии сотрудников некоего коллектива, отсортированные по алфавиту. Необходимо запрограммировать сортировку этого списка по длине фамилии методом «пузырька».

	A	B	C	D	E
1	1	Андреев	23	Чех	3
2	2	Борисов	27	Югов	4
3	3	Владимиров	7	Жаков	5
4	4	Григорьева	15	Петров	6
5	5	Дмитриевская	18	Титова	6
6	6	Егоров	6	Егоров	6
7	7	Жаков	14	Олегов	6
8	8	Звенигородская	2	Борисов	7
9	9	Иванова	9	Иванова	7
10	10	Константинопольский	19	Ульянов	7
11	11	Леонова	11	Леонова	7
12	12	Михайлова	24	Шагинян	7
13	13	Николаев	25	Щенкова	7
14	14	Олегов	1	Андреев	7
15	15	Петров	13	Николаев	8
16	16	Раскольников	26	Эдуардов	8
17	17	Свиридова	20	Федорова	8
18	18	Титова	12	Михайлова	9
19	19	Ульянов	17	Свиридова	9
20	20	Федорова	4	Григорьева	10
21	21	Харлампиев	21	Харлампиев	10
22	22	Цискаридзе	22	Цискаридзе	10
23	23	Чех	3	Владимиров	10
24	24	Шагинян	28	Ямпольский	10
25	25	Щенкова	5	Дмитриевская	12
26	26	Эдуардов	16	Раскольников	12
27	27	Югов	8	Звенигородская	14
28	28	Ямпольский	10	Константинопольский	19

Рис. 8.4. Лист Excel со списком фамилий и с этим же списком, отсортированным по длине фамилии.

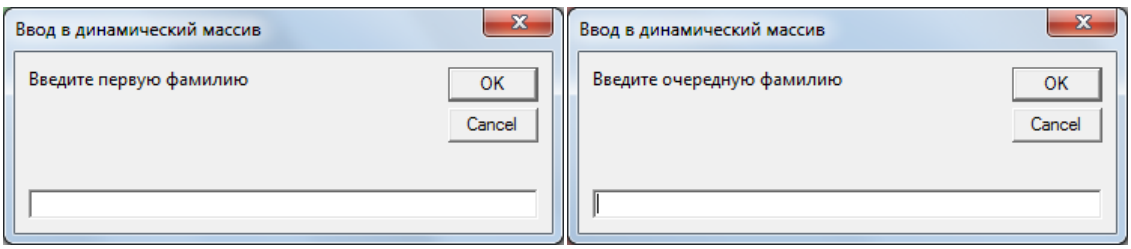
Далее – код макроса, решающего данную задачу. Между отдельными строками кода – комментарии, выделенные особым шрифтом:

```
Sub СортировкаПоДлинеФамилии()
    Const N As Integer = 28
```

```
Заполнение двух массивов:
порядковых номеров M1(1 To N)
и самих фамилий M2(1 To N)
    Dim M1(1 To N) As Integer
    Dim M2(1 To N) As String
      For i = 1 To N
        M1(i) = Cells(i, 1)
        M2(i) = Cells(i, 2)
      Next
Сортировка массива M2
по длине его элементов
с учётом привязки к ним
порядковых номеров - элементов массива M1:
    Dim A1 As Integer, A2 As String
      For i = 1 To N - 1
        For j = i + 1 To N
            If Len(M2(i)) > Len(M2(j)) Then
                A1 = M1(i): A2 = M2(i)
                M1(i) = M1(j): M2(i) = M2(j)
                M1(j) = A1: M2(j) = A2
            End If
        Next
      Next
      For i = 1 To N
        Cells(i, 3) = M1(i)
        Cells(i, 4) = M2(i)
        Cells(i, 5) = Len(M2(i))
      Next
End Sub
```

8.5. Пример использования динамического массива: циклический ввод в массив заранее неизвестного числа элементов

Пример 8.5. Продемонстрируем ввод в одномерный динамический массив **Fam**() любого числа фамилий с помощью окна ввода (рис. 8.5).



а) *б*)

Рис. 8.5. Ввод фамилий в динамический массив **Fam**() с помощью окна ввода

Окно ввода должно появляться до тех пор, пока не будет нажата кнопка **Cancel** вместо кнопки **OK**. После этого все введённые в массив элементы следует визуализировать – поместить в колонку «B» листа Excel, а также визуализировать результат сортировки этого массива в колонке «D» (рис. 8.6).

Рис. 8.6. Визуализация в колонке «B» листа Excel содержимого динамического массива **Fam**() и результата сортировки этого массива в колонке «D»

Щелчок командной кнопки на листе Excel (рис. 8.6) вызывает макрос, код которого приведён после кода щелчка этой кнопки:

```
Private Sub CommandButton1_Click()
    ДинамическийМассив
End Sub
```

```
Dim Fam() As String, S As String, N As Integer
Sub ДинамическийМассив()
    S = InputBox("Введите первую фамилию", _
                 "Ввод в динамический массив", "")
    ReDim Fam(0)
    Fam(0) = S
    Do Until S = ""
       S = InputBox("Введите очередную фамилию", _
                    "Ввод в динамический массив", "")
       ReDim Preserve Fam(UBound(Fam) + 1)
       Fam(UBound(Fam)) = S
    Loop
    MsgBox UBound(Fam)       ' число элементов массива
    N = UBound(Fam) - 1      ' макс. значение индекса
                             ' массива
```

```
      For i = 0 To N
        Range("B" & i + 2) = Fam(i)
      Next
    Ans = MsgBox("Сортировать по возрастанию (Yes)" & _
         "или по убыванию (No)?", vbQuestion + vbYesNo)
    If Ans = vbYes Then
       СортировкаПоВозрастанию
    Else
       СортировкаПоУбыванию
    End If
End Sub
```

```
Sub СортировкаПоВозрастанию()
  N = UBound(Fam) - 1
  For i = 0 To N - 1
    For j = i + 1 To N
      If Fam(i) > Fam(j) Then
         S = Fam(i): Fam(i) = Fam(j): Fam(j) = S
      End If
    Next
  Next
  For i = 0 To N
      Range("D" & i + 2) = Fam(i)
  Next
End Sub
```

Код процедуры **СортировкаПоУбыванию** отличается от кода процедуры **СортировкаПоВозрастанию** только одной строкой (даже только одним знаком в этой строке).

А именно, строка

```
    If Fam(i) > Fam(j) Then
```

заменяется на строку

```
    If Fam(i) < Fam(j) Then
```

8.6. Запись массива в файл и чтение из файла в массив

Продемонстрируем несколько операторов языка VBA, с помощью которых данные из одномерного массива легко перебрасываются в текстовый файл (который часто называют файлом последовательного доступа), а также совершается обратное действие – данные из файла записываются в массив.

Открытие файла «для записи» и *запись* в этот файл всех элементов массива по очереди:

```
Open <путь к файлу> For Output  As # <дескриптор файла>
    For i = 0 To N
      Write #<дескриптор файла>, <i-й элемент массива>
    Next
Close # <дескриптор файла>
```

Открытие файла «для чтения», *чтение* из этого файла всех записей и помещение их в массив по очереди:

```
Open <путь к файлу> For Input  As # <дескриптор файла>
   i = 0
   Do Until EOF(<дескриптор файла>)
      Input # <дескриптор файла>, <запись файла>
      <i-й элемент массива> = <запись файла>
      i = i + 1
   Loop
Close # <дескриптор файла>
```

Рассмотрим запись в файл и чтение из файла на двух примерах.

Пример 8.6. Запись в файл с именем **СписокДатРождения.txt**, который автоматически появится в директории **D:\TextFiles**, содержимого массива **BDate**:

```
Open "D:\TextFiles\" & "СписокДатРождения.txt" _
                       For Output  As #1
     For i = 0 To N
         record = BDate(i)
         Write #1, record
     Next
Close #1
```

Отметим, что число N+1 элементов массива должно быть известно.

Пример 8.7. Чтение содержимого файла с именем **СписокДатРождения.txt**, который находится в директории **D:\TextFiles**, и занесение записей (records) этого файла в массив **BDate**:

```
Open "D:\TextFiles\" & "СписокДатРождения.txt" _
                       For Input As #1
     i = 0
```

```
     Do Until EOF(1)
        Input #1, record
        BDate(i) = record
        i = i + 1
     Loop
Close #1
```

При циклическом чтении записей (records) из файла **BDate** их число заранее неизвестно, поэтому при чтении следует применить не цикл со счётчиком, как в предыдущем примере, а цикл с условием. Условие – это булевская функция **EOF** (end of file), которая принимает значение **Истина**, если мы пытаемся прочесть что-то после последней записи файла.

В языке VBA есть средства и для работы с *файлами прямого доступа* – **Random Access File** (RAF). Такие файлы удобны для хранения массивов, элементы которых – данные *пользовательского типа*. Они удобны тем, что имеют постоянную длину. Это позволяет непосредственно по номеру записи (напрямую!) обращаться к искомым данным. Естественно, это делается гораздо быстрее, чем в текстовых файлах последовательного доступа.

Аналогия:

файл ***последовательного доступа*** – магнитная лента;

файл ***прямого доступа*** – оптический компакт-диск.

Очень важно, что отдельные компоненты этих данных могут иметь *разные* типы.

Пример 8.8. Напомним, что структура данных пользовательского типа декларируется программно. Например, так:

```
Private Type StudentData
    ФИО As String * 20
    Группа As String * 10
    ДатаРождения As Date
    Телефон As String * 12
    ЭлПочта As String * 20
End Type
```

Отдельные компоненты переменной типа **StudentData** пользовательского типа имеют различный тип: **String** фиксированной длины, кроме компонента ДатаРождения типа **Date**. Т.е. запись файла будет иметь суммарную длину 20+10+8+12+20 = 70.

Далее – запись данных о студентах в файл прямого доступа (RAF) и чтение из этого файла. Демонстрация – на рис. 8.7 (процедуры для трёх командных кнопок).

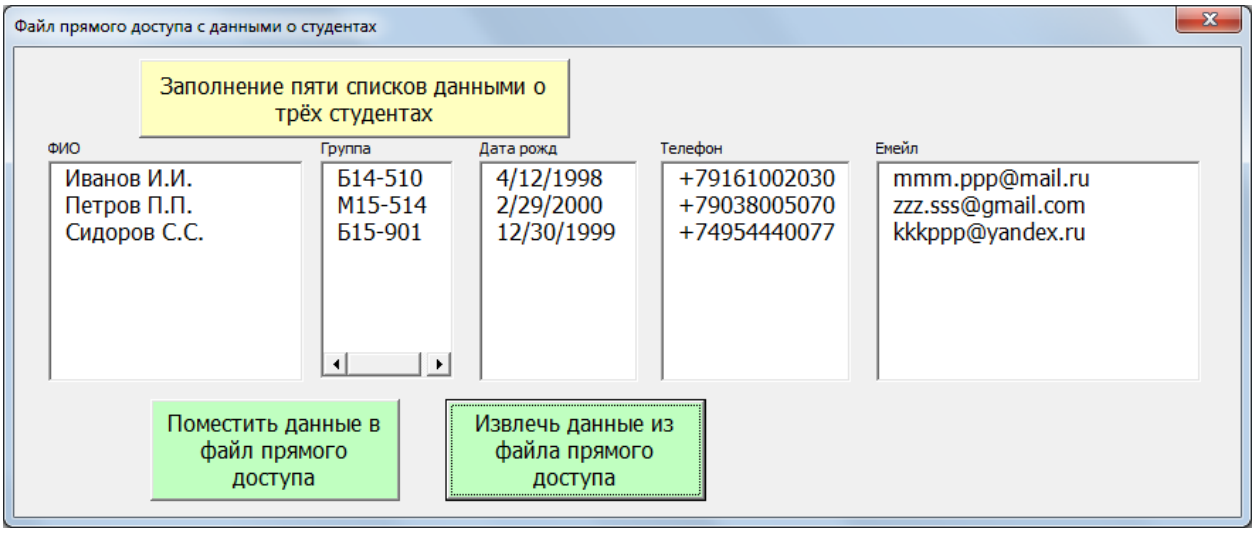


Рис. 8.7. Запись в файл и чтение из файла прямого доступа данных о студентах

Процедуры для командных кнопок:

```
Private Sub CommandButton1_Click()
    For i = 1 To 3
        ListBox1.AddItem InputBox("FIO-" & i)
        ListBox2.AddItem InputBox("Gr-" & i)
        ListBox3.AddItem InputBox("BDate-" & i)
        ListBox4.AddItem InputBox("Tel-" & i)
        ListBox5.AddItem InputBox("Email-" & i)
    Next
End Sub
```

```
Private Sub CommandButton2_Click()
      Dim X As Integer, Student As StudentData
      X = Len(Student)
      MsgBox X & " - длина переменной Student" & _
      " пользовательского типа данных StudentData"
  Open "D:\MyFiles\" & "Students.raf" For Random _
        As #1 Len = X
    For i = 1 To 3
        Student.FIO = ListBox1.List(i - 1)
```

```
        Student.Gr = ListBox2.List(i - 1)
        Student.BDate = ListBox3.List(i - 1)
        Student.Tel = ListBox4.List(i - 1)
        Student.EMail = ListBox5.List(i - 1)
        Put #1, i, Student
    Next
  Close 1
End Sub
```

```
Private Sub CommandButton3_Click()
    Dim X As Integer, n As Integer, _
        Student As StudentData
    X = Len(Student)
    n = 3
    Open "D:\MyFiles\" & "Students.raf" For Random _
             As #1 Len = X
    For i = 1 To n
    Get #1, i, Student
    ListBox1.AddItem Student.FIO
    ListBox2.AddItem Student.Gr
    ListBox3.AddItem Student.BDate
    ListBox4.AddItem Student.Tel
    ListBox5.AddItem Student.EMail
    Next
    Close #1
    ' Kill "Students.raf"
End Sub
```

8.7. Бинарный поиск номера заданного элемента в отсортированном массиве (поиск методом «дихотомии») на двух примерах: поиск номера фамилии и угадывание задуманного числа

Идея метода «дихотомии» («деления пополам» – *греч.*) заключается в следующем. На первом шаге массив длиной **N** делится пополам – на две равные части. Элемент, находящийся в середине массива, сравнивается с искомым элементом. Если искомый элемент меньше элемента, находящегося посредине массива, дальнейший поиск следует перенести на первую половину массива. Если больше – на вторую половину. Затем, оставшуюся половину

массива опять следует разделить пополам. И т.д., пока искомый элемент не станет равным элементу, находящемуся посредине очередной части массива.

Сколько разбиений надо сделать? Очевидно, не более $\boldsymbol{K = \log_2 N + 1}$.

Если, например, в массиве 1024 элемента ($N = 2^{10}$), потребуется не более 10 + 1 = 11 разбиений.

Пример 8.9. Поиск номера данной фамилии в отсортированном списке фамилий при условии, что все фамилии в данном списке *уникальны*, т.е. в списке нет двух одинаковых фамилий.

Данную задачу удобно решать с помощью VBA с привлечением экранной формы, показанной на рис. 8.8.

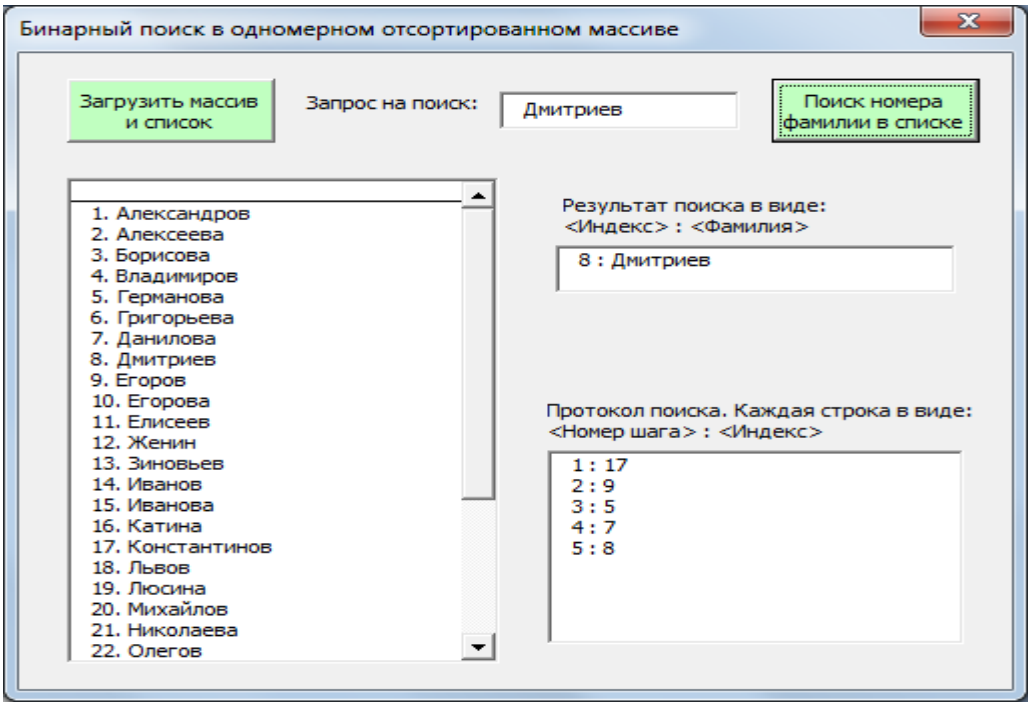


Рис. 8.8. Экранная форма для решения задачи бинарного поиска номера (индекса) заданной фамилии

На этой форме одно текстовое поле **TextBox1** для ручного ввода искомой фамилии (в данном случае, в это поле введена фамилия «Дмитриев») и три поля списка: **ListBox1** – для визуализации списка фамилий, **ListBox2** – для визуализации результата, **ListBox3** – для построения «протокола поиска».

Процедуры для двух командных кнопок на форме следующие:

```
Dim a(31) As String
Private Sub CommandButton1_Click()
   For i = 1 To 32
```

```
      a(i - 1) = Cells(i, 2)
      ListBox1.AddItem i & ". " & a(i - 1)
   Next
End Sub
```

Здесь реализован ввод в массив **a(31)** и в поле списка **ListBox1** экранной формы 32-х разных фамилий из 1-го столбца листа книги Excel, предварительно отсортированных в лексикографическом порядке (по алфавиту).

```
Private Sub CommandButton2_Click()
   N = 32
   ListBox2.Clear
   ListBox3.Clear
   imin = 0: imax = N + 1
   i = N - 1: K = 0
   K = 0
   limit = Log(N) / Log(2) + 1
     Do Until K > limit
        i = (imax + imin) \ 2
        If i > N - 1 Then ListBox2.AddItem "Нет": _
               Exit Sub
        ListBox3.AddItem K + 1 & " : " & i + 1
        K = K + 1
        If a(i) > TextBox1.Text Then
           imax = i
        ElseIf a(i) < TextBox1.Text Then
           imin = i
        Else
           Exit Do
        End If
     Loop
        If K <= limit Then
            ListBox2.AddItem i + 1 & " : " & a(i)
        Else
            ListBox2.AddItem "Нет"
        End If
End Sub
```

Пример 8.10. Рассмотрим следующую игру: компьютер «задумывает» целое число от 0 до 1000. Пользователь должен его угадать, затратив на угадывание не более 10 попыток. Эта игра красиво демонстрирует метод «дихотомии».

Для реализации игры можно предложить экранную форму, представленную на рис. 8.9.

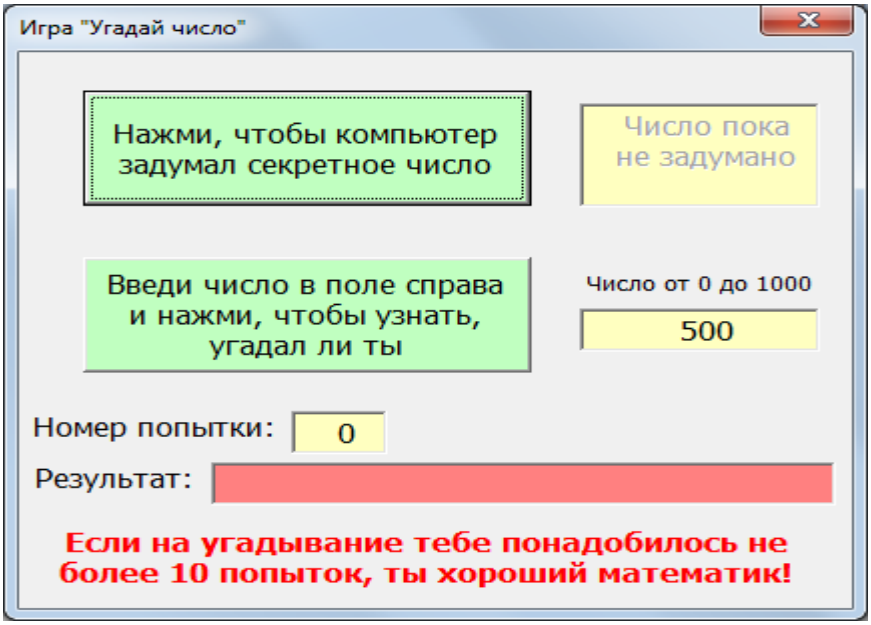


Рис. 8.9. Форма для игры «Угадай число»

На каждом новом шаге игры пользователь вводит во второе текстовое поле новое число и получает один из трёх ответов: «Больше», «Меньше» или «Вы угадали!».

Код модуля с процедурами для обеих командных кнопок:

```
Dim SN, N As Integer
Private Sub CommandButton1_Click()
    Randomize
    SN = Round(Rnd * 1000)
    TextBox1.Text = "Число задумано!"
    TextBox2.Text = 0
    TextBox3.Text = 0
    TextBox4.BackColor = RGB(250, 100, 100)
    TextBox4.Text = "Результата пока нет"
    CommandButton1.Enabled = False
    CommandButton2.Enabled = True
End Sub
```

```
Private Sub CommandButton2_Click()
    N = Val(TextBox2.Text)
    TextBox3.Text = Val(TextBox3.Text) + 1
    If N > SN Then
        TextBox4.Text = "Твоё число слишком большое"
    ElseIf N < SN Then
        TextBox4.Text = "Твоё число слишком маленькое"
```

```
    Else
        TextBox4.Text = "Твоё число равно задуманному"
        TextBox4.BackColor = RGB(100, 250, 100)
        TextBox1.Text = "Секретное число устарело"
        CommandButton1.Enabled = True
        CommandButton2.Enabled = False
    End If
End Sub
```

8.8. Многомерные массивы. Пример использования двумерного массива: поиск «минимакса» («седловой точки» на поверхности гиперболического параболоида)

Многомерные массивы (двумерные, трёхмерные и т.д.) используются для хранения матриц (двумерных, трёхмерных и т.д.) значений одного типа. Мы ограничимся рассмотрением только двумерных массивов.

Объявляются они, например, так:

Dim AAA(N1-1, N2-1) **As Double**

Dim AAA(1 To N1, 1 To N2) **As Double**.

Здесь объявлен двумерный массив AAA для одной и той же матрицы чисел типа **Double** размерностью N1 x N2.

Пример 8.11. Демонстрация использования двумерного массива – построение поверхности типа «седло» (*гиперболического параболоида*) и вычисление значения координат «седловой точки». Гиперболический параболоид – это поверхность функции:

$$z = \frac{(x-a)(x-b)}{p} - \frac{(y-c)(y-d)}{q}.$$

«Седловая точка» – это значение $z_0 = \min_y(\max_x z(x,y))$.

Вычисление этого значения иногда используется в прикладных задачах при нахождении оптимальных стратегий в так называемых «экономических играх». Рис. 8.10 демонстрирует понятие «седловой точки» на поверхности гиперболического параболоида.

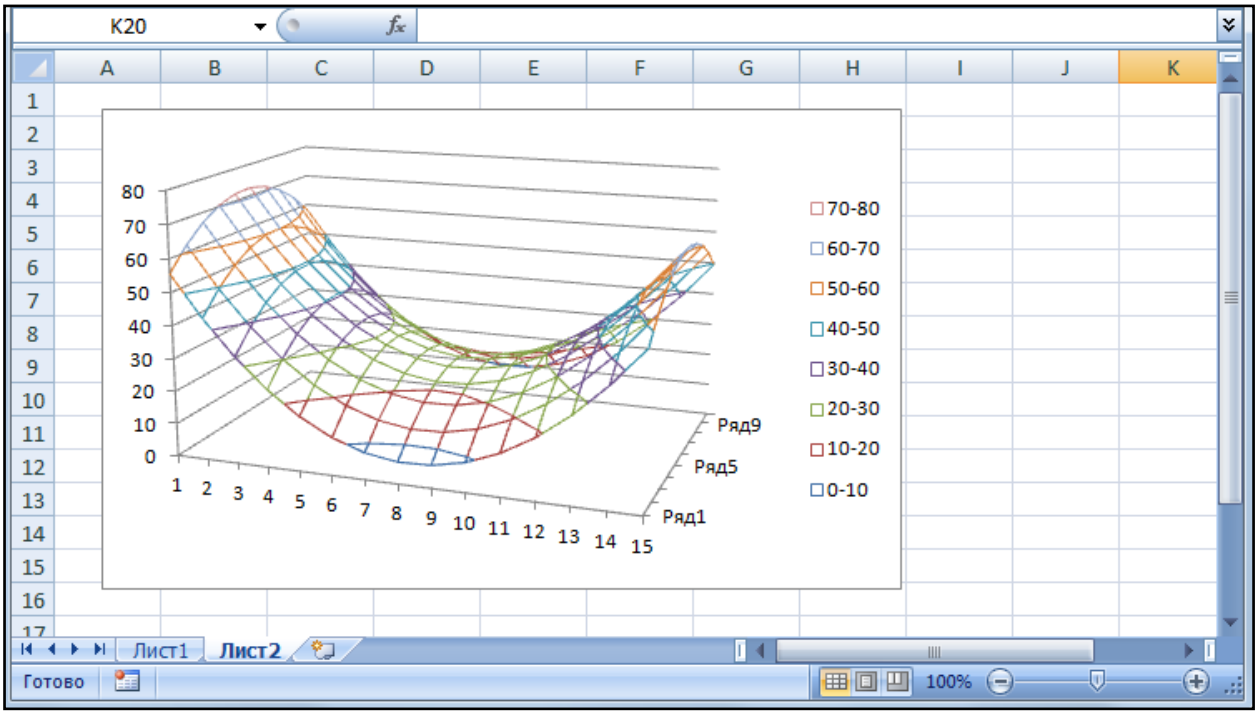


Рис. 8.10. Поверхность гиперболического параболоида

Код макроса «Седловая точка»:

```
Sub СедловаяТочка()
    Dim x As Integer, y As Integer
    Dim z(14, 9) As Double
    Const a As Double = 5.2,  b As Double = 9.5
    Const c As Double = -1.3, d As Double = 10.7
    Const p As Double = 1.1,  q As Double = 1.3
    For x = 0 To 14
        For y = 0 To 9
            z(x, y) = Round((x - a) * (x - b) / p - _
                      (y - c) * (y - d) / q, 2)
            Cells(x + 2, y + 2) = z(x, y)
        Next
    Next
```

Эта часть программы заполняет матрицу (двумерный массив) значениями функции z(x, y) и записывает эти значения в ячейки листа книги Excel. Значения параметров a, b, c, d, p и q подобраны экспериментально.

```
    MinMax = z(0, 0): ix = 0: iy = 0
    For x = 0 To 14
        Max# = z(x, 0)
            For y = 0 To 9
              If z(x, y) > Max Then
                Max = z(x, y): iy = y
              End If
            Next y
```

```
                If Max < MinMax Then
                   MinMax = Max: ix = x
                End If
    Next x
    MsgBox "z (" & ix + 1 & ", " & iy + 1 & ") = " _
          & MinMax,  vbInformation, "Седл. точка"
    Лист1.Label1.Caption = "Седловая точка: P(" & _
          ix + 1 & "," & iy + 1 & ")=" &  MinMax
End Sub
```

Эта часть программы
ищет для каждого из 10 значений «y»
максимум (внутренний цикл),
а затем для каждого из 15 значений «x»
минимум (внешний цикл).
Это и будет «седловая точка».

Результат работы макроса «Седловая точка» показан на рис. 8.11.

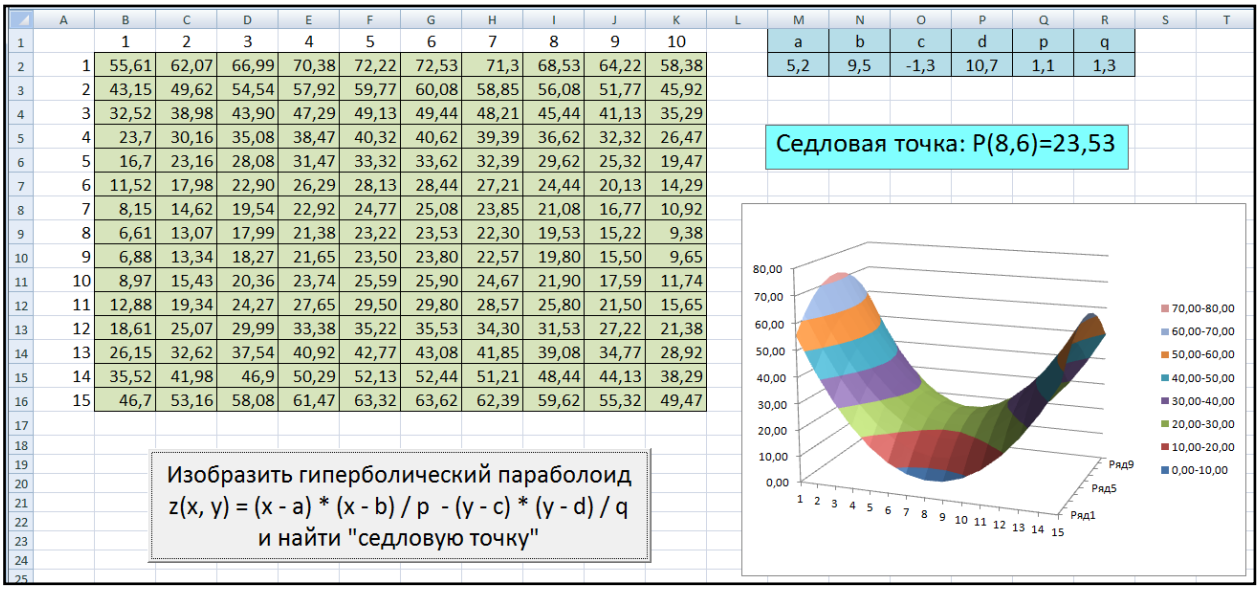


Рис. 8.11. Результат работы макроса «Седловая точка»

Глава 9

Данная глава – это изложение материалов третьей лабораторной работы, в которой закрепляется материал двух предыдущих лекций по курсу программирования на языке Visual Basic для офисных приложений. Содержание этих материалов можно структурировать следующим образом:

- Контроль знаний по теме лабораторной работы.
- Пример программирования построения графика функции в полярных координатах с использованием оператора **For … Next** («Лепесток Декарта»).
- Пример программирования процедуры с использованием оператора цикла с условием **Do … Loop** («задача о росте народонаселения»).
- Примеры программирования процедур для последовательности действий: (1) записи нескольких дат в массив; (2) записи содержимого массива в файл; (3) чтения данных из файла в массив; (4) пузырьковой сортировки этого массива; (5) помещения содержимого массива в ячейки листа Excel.
- Пример программирования процедуры с использованием понятия двумерного массива («построение имитации двумерного распределения Гаусса с помощью суммы нескольких случайных величин»).
- Программирование процедуры «дихотомии» (бинарного поиска) с использованием типа данных, определяемого пользователем, и файла прямого доступа на примере бинарного поиска данных в списке налогоплательщиков с помощью индексного файла ИНН.

9.1. Контрольные вопросы по теме лабораторной работы

1. Что такое циклический оператор (оператор цикла)? Какие два типа оператора цикла используются в языке Basic?
2. Как формально представляется синтаксис оператора цикла со счётчиком **For … Next**? Какие элементы синтаксиса этого оператора являются «необязательными»?
3. Как формально представляется синтаксис оператора цикла с условием **Do … Loop**? Какие элементы синтаксиса этого оператора являются «необязательными»?
4. В чём различие между смыслом проверки условий после ключевых слов **While** и **Until** в циклах с условием?

5. Как с помощью *цикла в цикле* **For i = 1 To m** … **For j = 1 To n** можно пронумеровать клетки листа Excel (**m** строк и **n** столбцов), двигаясь по методу «слева направо, сверху вниз» от левой верхней клетки до правой нижней клетки? ***Ответ:*** «тело» внутреннего цикла – это единственный оператор: Cells(i, j) = $j + n * (i - 1)$.
6. Как можно, при необходимости, обеспечить досрочный выход из цикла любого вида (как **For … Next**, так и **Do … Loop**)?
7. Как отличается объявление одномерного массива с нумерацией от **1** до **N-1** от объявления того же массива с нумерацией от **0** до **N** (по умолчанию)?
8. Как можно объявить двумерный массив размерностью **M** x **N**?
9. Как определяется *перед процедурой* и переопределяется *внутри процедуры* динамический массив? Как сохранить его содержимое при переопределении?
10. Каково минимальное число разбиений одномерного отсортированного массива по методу «дихотомии» обеспечивает успех поиска номера элемента в массиве?
11. Как обеспечить *чтение* (*read*) всех *записей* (*records*) из текстового файла и заполнение ими одномерного массива, если число записей файла не известно?
12. Как *записать* (*write*) в текстовый файл в качестве его *записей* (*records*) все элементы одномерного массива, число элементов которого **N** известно?
13. Чем принципиально отличается файл прямого доступа (**RAF – Random Access File**) от текстового файла?
14. Как открыть файл прямого доступа **RAF**? Чем отличается открытие такого файла для чтения от открытия для записи?

9.2. Пример программирования построения графика функции «Лепесток Декарта» в полярных координатах с использованием оператора For … Next

Задание 1. Взяв за основу технологию, рассмотренную в лекции 7 для построения графика спирали в полярных координатах, требуется построить график функции «Лепесток Декарта».

Лепесток Декарта – функция, график которой в полярных координатах имеет вид:

$$r = \frac{3\,b\,\mathrm{Sin}(f)\,\mathrm{Cos}(f)}{\mathrm{Sin}^3(f)\,\mathrm{Cos}^3(f)}.$$

Эту функцию интересно наблюдать (и строить!) в интервале

$$[\mathrm{arctg}(-1) + \tfrac{10}{b}\,,\ \tfrac{\pi}{2} - \mathrm{arctg}(-1) - \tfrac{10}{b}].$$

Следует отметить, что эта функция имеет асимптоту – прямую, имеющую следующий график в декартовых координатах: $y = -x - 100$. При построении диаграммы «Лепесток Декарта» для наглядности желательно изобразить и эту асимптоту.

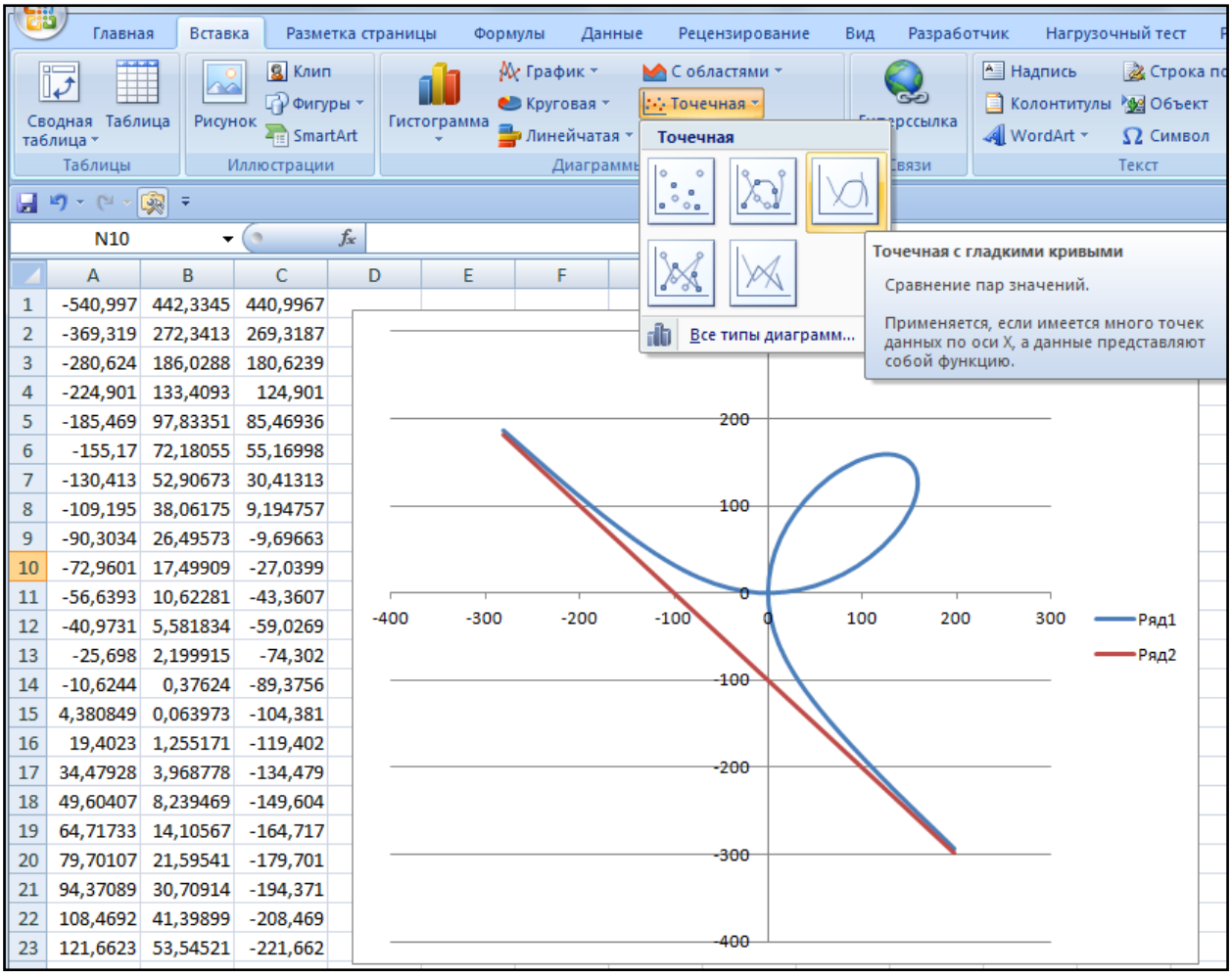
Далее представлен текст макроса, с помощью которого заполняются три колонки листа Excel:

- колонка «A» – координатами ***x*** декартовой системы координат функции «Лепесток Декарта»: `Cells(i, 1) = r * Cos(f);`
- колонка «B» – координатами ***y*** декартовой системы координат функции «Лепесток Декарта»: `Cells(i, 2) = r * Sin(f);`
- колонка «C» – координатами ***y*** функции асимптоты: `Cells(i, 3) = -Cells(i, 1) - 100.`

```
Sub ЛепестокДекарта()
    Const pi = 3.1415
    Dim b As Double, s As Double
    b = 100: s = 0.05
    fp1# = Atn(-1) + 10 / b
    fp2# = pi / 2 - Atn(-1) - 10 / b
    i = 0
```

```
    For f# = fp1 To fp2 Step s
        i = i + 1
        r# = 3 * b * Sin(f) * Cos(f) / _
             (Sin(f) ^ 3 + Cos(f) ^ 3)
        Cells(i, 1) = r * Cos(f)
        Cells(i, 2) = r * Sin(f)
        Cells(i, 3) = -Cells(i, 1) - 100
    Next f
End Sub
```

После заполнения программой колонок «A», «B» и «C» с помощью стандартной технологии вставки диаграммы, принятой в приложении MS Excel, строится диаграмма: **Вставка → Диаграммы → Точечная → Точечная с гладкими кривыми** (рис. 9.1).



	A	B	C
1	-540,997	442,3345	440,9967
2	-369,319	272,3413	269,3187
3	-280,624	186,0288	180,6239
4	-224,901	133,4093	124,901
5	-185,469	97,83351	85,46936
6	-155,17	72,18055	55,16998
7	-130,413	52,90673	30,41313
8	-109,195	38,06175	9,194757
9	-90,3034	26,49573	-9,69663
10	-72,9601	17,49909	-27,0399
11	-56,6393	10,62281	-43,3607
12	-40,9731	5,581834	-59,0269
13	-25,698	2,199915	-74,302
14	-10,6244	0,37624	-89,3756
15	4,380849	0,063973	-104,381
16	19,4023	1,255171	-119,402
17	34,47928	3,968778	-134,479
18	49,60407	8,239469	-149,604
19	64,71733	14,10567	-164,717
20	79,70107	21,59541	-179,701
21	94,37089	30,70914	-194,371
22	108,4692	41,39899	-208,469
23	121,6623	53,54521	-221,662

Рис. 9.1. Ручная вставка на лист Excel точечной диаграммы после выполнения макроса

Замечание. Для наглядности, чтобы избавиться от слишком длинных «хвостов» графика, имеет смысл не выделять по две-три строки в начале и в

конце выделяемого диапазона для изображения. Именно так сделано в примере, показанном на рис. 9.1.

9.3. Пример программирования процедуры с использованием оператора цикла с условием Do … Loop («задача о росте численности народонаселения»[7])

Задание 2. В некоей стране ежегодный прирост численности народонаселения составляет **P %**. Используя оператор цикла с условием **Do … Loop**, написать процедуру на языке VBA for Excel, которая определяет число лет, за которое число жителей страны увеличивается в **k** раз, например, удваивается, утраивается и т.д. Проверить её работу для конкретного значения **P**, например: 5, 6, 7, 8, 9, 10 % в год (в зависимости от варианта задания). Выберите подходящий вид диаграммы для изображения графика роста численности народонаселения в стране. Вставьте на лист Excel эту диаграмму.

В качестве решения можно предложить следующий макрос:

```
Sub РостЧисленностиНародонаселения()
  Const S0 = 15000000     ' Исходная численность
                          '  народонаселения
  Const k As Integer = 2  ' Во сколько раз должна
                          '  изменится численность
                          '  народонаселения
  Const P As Single = 9   ' Годовой прирост численности
                          '  народонаселения в процентах
  Dim R As Integer        ' Год очередной переписи
                          '  народонаселения
  Dim S As Double         ' Численность в год переписи
        R = 1: S = S0
        Cells(R + 1, 1) = R
        Cells(R + 1, 2) = S
    Do
        R = R + 1: S = S * (1 + P / 100)
        Cells(R + 1, 1) = R
        Cells(R + 1, 2) = S
    Loop While S < k * S0
End Sub
```

[7] Волченков Н.Г. Программирование на Visual Basic 6: В 3-х ч. – Ч. 3. Задачник. – М.: ИНФРА-М, 2002. Задача 6.5: с.72 – 74, 183 – 184.

Результат работы макроса – на рис. 9.2.

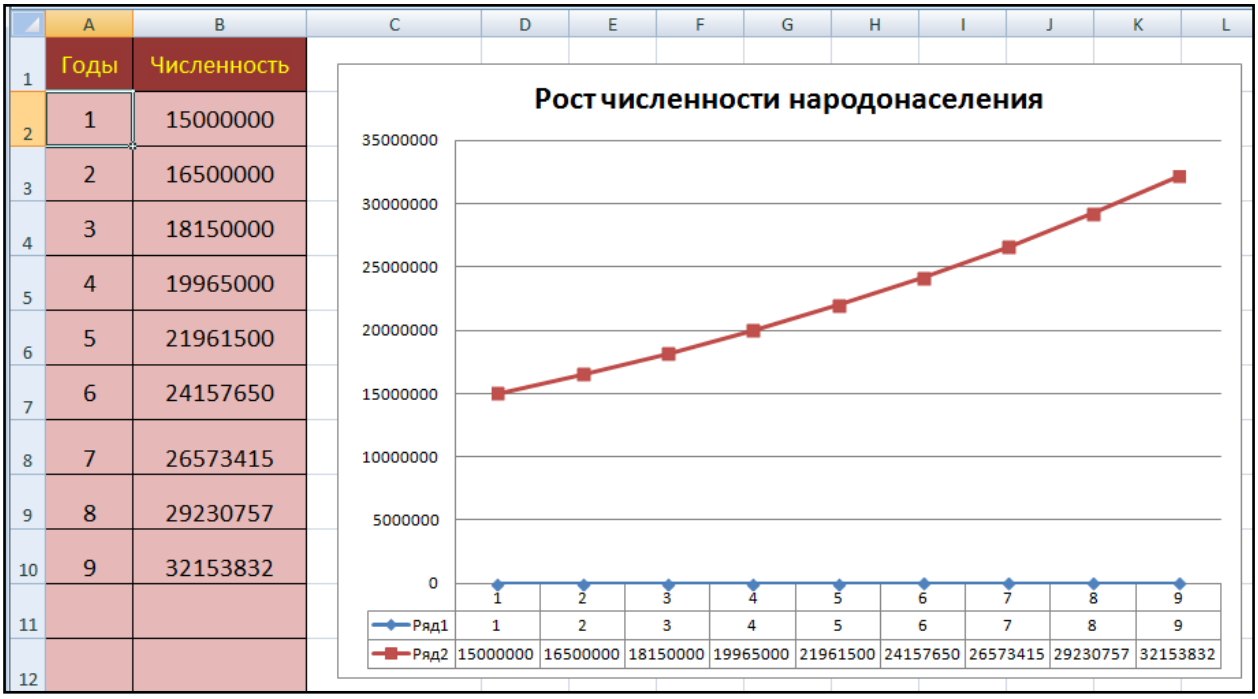


Рис. 9.2. Результат работы макроса **РостЧисленностиНародонаселения** и ручной вставки точечной диаграммы на лист Excel после выполнения этого макроса

Задачу, решаемую данным макросом, интерпретируйте как задачу о накоплении при вкладе денег в банк с заданной годовой процентной ставкой. Рассмотрите эту интерпретацию, как *вариант* выполнения ***Задания 2***.

9.4. Примеры программирования процедур для последовательности действий: (1) записи нескольких дат в массив; (2) записи содержимого массива в файл; (3) чтения данных из файла в массив; (4) пузырьковой сортировки этого массива; (5) помещения содержимого массива в ячейки листа Excel

Задание 3. Записать несколько дат из ячеек листа Excel сначала в массив, а затем в файл (1-я кнопка на рис. 9.3); после чего, прочитать данные из файла, записать их в массив, отсортировать и занести данные в ячейки (2-я кнопка).

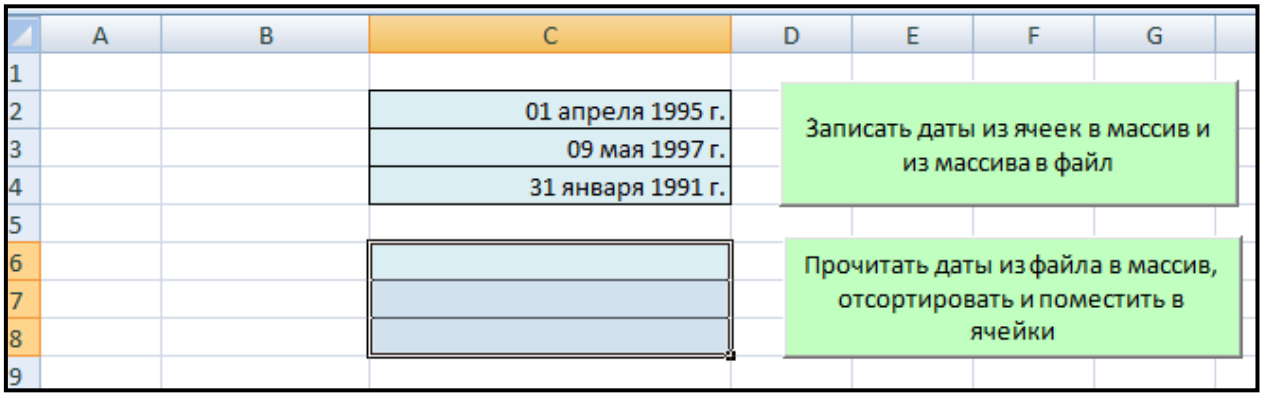


Рис. 9.3. Лист Excel с двумя установленными на нём кнопками

Процедура для 1-й из двух кнопок:

```
Private Sub CommandButton1_Click()
Const N As Integer = 2
Dim bdate(N)
          ' Запись в массив. N = 2 - это известно:
    For i = 0 To N
        bdate(i) = Cells(i + 2, 3)
    Next
          ' Запись в файл:
    Open "D:\MyFiles\" & "aaaa.txt" For Output As #1
        For i = 0 To N
            Record = bdate(i)
            Write #1, Record
        Next
    Close #1
MsgBox "OK!"
End Sub
```

Процедура для 2-й из двух кнопок:

```
Private Sub CommandButton2_Click()
Dim N As Integer      ' Значение N пока неизвестно,
Dim bdate() As Date   ' поэтому массив динамический.
                      ' Чтение из файла:
    Open "D:\MyFiles\" & "aaaa.txt" For Input As #1
        i = 0
        Do Until EOF(1)
            Input #1, Record
            ReDim Preserve bdate(i)
            bdate(i) = Record
            i = i + 1
        Loop
    Close #1
    N = i - 1     ' Теперь значение N стало известно:
    Сортировать bdate(), N
    For i = 0 To N
        Cells(i + 6, 3) = bdate(i)
    Next
End Sub
```

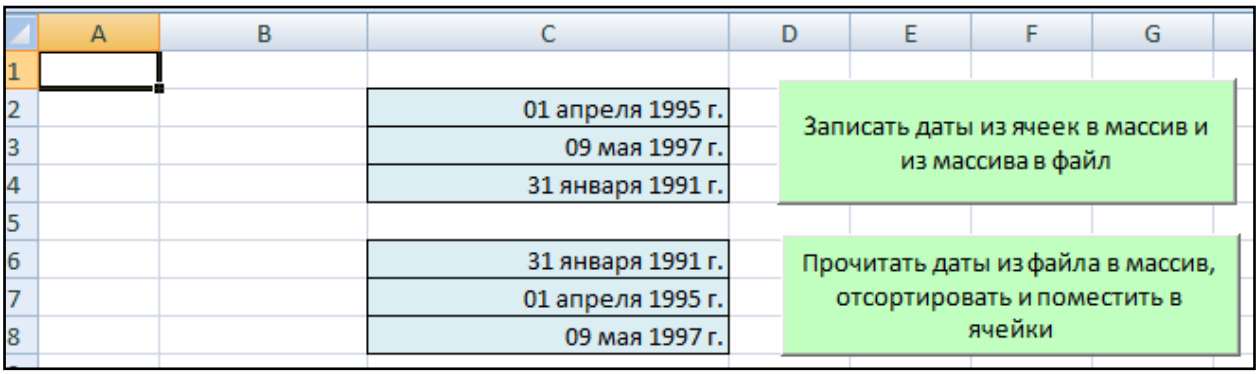


Рис. 9.4. Лист Excel после щелчка 2-й кнопки

Вызываемая процедура сортировки дат:

```
Private Sub Сортировать(bdate() As Date, N As Integer)
Dim d As Date
  For i = 0 To N - 1
    For j = i + 1 To N
      If bdate(j) < bdate(i) Then
         d = bdate(i)
         bdate(i) = bdate(j)
         bdate(j) = d
      End If
    Next
  Next
End Sub
```

9.5. Пример программирования процедуры с использованием понятия двумерного массива («построение имитации двумерного распределения Гаусса с помощью суммы нескольких случайных величин»)

Задание 4. Предлагается смоделировать результаты «стрельбы по мишени» с помощью имитации нормального закона распределения двух случайных величин: координаты ***x*** и координаты ***y*** точки «попадания пули» в мишень[8].

Ниже представлен вариант выполнения данного задания – макрос **СтрельбаПоМишени**, предназначенный для демонстрации *приблизительно* нормального закона распределения случайных величин – координат точек на мишени. Картину «нормального» распределения можно наблюдать при достаточно большом числе выстрелов.

На рис. 9.5 показан лист книги Excel, на котором появляется заполненная матрица размером 20 x 20 в результате выполнения первого этапа работы макроса **СтрельбаПоМишени**.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
2	0	0	0	3	3	4	11	11	17	20	20	11	9	7	7	2	0	0	0	0
3	0	1	3	3	15	27	54	83	100	157	114	113	84	47	33	11	8	3	0	0
4	0	1	5	15	55	117	210	294	420	469	467	411	320	207	120	51	17	5	1	0
5	0	3	15	53	147	303	549	822	1053	1208	1257	1069	852	492	334	147	50	25	2	0
6	0	8	34	122	294	631	1166	1742	2279	2603	2704	2293	1749	1269	689	331	96	33	8	0
7	0	7	59	196	559	1219	2172	3189	4133	4732	4804	4096	3216	2021	1246	529	205	49	6	0
8	1	13	76	318	823	1811	3300	4846	6284	7273	7203	6224	4676	3172	1783	809	315	71	10	1
9	2	17	125	436	1077	2339	4202	6345	8353	9377	9476	8294	6343	4218	2355	1128	427	126	19	2
10	0	14	132	457	1296	2726	4688	7210	9251	10645,5	10645,5	9375	7139	4801	2747	1257	450	120	17	0
11	1	12	146	487	1215	2668	4867	7241	9397	10645,5	10645,5	9475	7165	4752	2629	1266	462	150	15	0
12	1	21	114	423	1156	2439	4229	6232	8269	9237	9488	8343	6206	4163	2345	1082	400	93	14	0
13	1	12	95	288	863	1764	3059	4809	6281	7275	7274	6356	4861	3215	1835	825	296	67	13	0
14	2	9	68	217	585	1196	2073	3165	4145	4836	4710	4230	3122	2189	1166	534	191	68	7	0
15	0	9	21	126	291	670	1218	1813	2385	2651	2661	2462	1838	1189	685	291	117	35	4	0
16	0	5	17	55	140	294	544	858	1112	1216	1257	1092	860	564	334	153	69	17	1	0
17	0	0	9	18	51	105	205	274	401	482	444	415	309	210	117	44	19	5	2	0
18	0	0	1	5	15	31	46	94	117	114	126	113	62	56	40	8	6	1	0	0
19	0	0	0	0	2	4	4	13	16	16	18	18	19	4	9	2	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0
21																				
22																				
23																				
24																				
25																				
26																				
27																				
28																				

Вставить диаграмму Wireframe на Лист2

Вставить диаграмму Surface на Лист3

Рис. 9.5. Матрица, заполняемая макросом **СтрельбаПоМишени**

[8] Нормальный закон распределения хорошо имитируется распределением суммы нескольких равномерно распределённых на интервале [0, 1] случайных величин. Чем больше слагаемых в этой сумме, тем лучше имитация. В данном примере их пять. Автором этой имитации для данного примера является Волчёнков Н.Г.

В клетках матрицы Cells(*X*, *Y*) – моделируемое число попаданий в квадратную область поля с координатами от (*X*, *Y*) до (*X*+1, *Y*+1). Общее число выстрелов 500000 (полмиллиона!).

В четырёх центральных клетках – число попаданий в центр мишени с координатами центра (10.5, 10,5). Данная программа сообщает об этом пользователю (рис. 9.6).

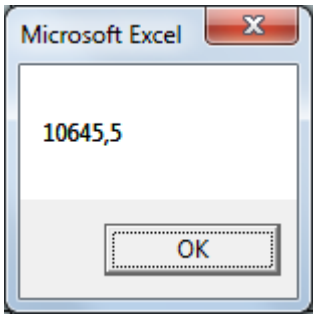


Рис. 9.6. Среднее число попаданий в четыре центральные области мишени. Это число одновременно присваивается программой четырём элементам массива *M*(*i*, *j*) и четырём клеткам матрицы: *M*(10, 10), *M*(10, 11), *M*(11, 10), *M*(11, 11)

Вставка диаграмм двух видов: «проволочной» (**Wireframe**) и «поверхностной» (**Surface**) – реализуется с помощью двух процедур, запускаемых с помощью двух командных кнопок на Листе 1, показанных на рис. 9.5.

Процедуры щелчков этих кнопок вызывают запуск двух вспомогательных макросов, расположенных в том же программном модуле, что и процедура макроса **СтрельбаПоМишени**:

```
Private Sub CommandButton1_Click()
    Макрос1
End Sub
```

```
Private Sub CommandButton2_Click()
    Макрос2
End Sub
```

Эти вспомогательные макросы построены автоматически после запуска режима «Запись макроса» на вкладке «Разработчик» основного меню приложения MS Excel. Коды процедур этих макросов следующие:

```
Sub Макрос1()
' Макрос Макрос
    Range("A1:T20").Select
    Selection.Copy
    Sheets("Лист2").Select
    Range("A1").Select
    ActiveSheet.Paste
    ActiveSheet.Shapes.AddChart.Select
    ActiveChart.SetSourceData _
                Source:=Range("'Лист2'!$A$1:$T$20")
    ActiveChart.ChartType = xlSurfaceWireframe
    Sheets("Лист2").Select
End Sub
```

```
Sub Макрос2()
' Макрос2 Макрос
    Range("A1:T20").Select
    Selection.Copy
    Sheets("Лист3").Select
    Range("A1").Select
    ActiveSheet.Paste
    ActiveSheet.Shapes.AddChart.Select
    ActiveChart.SetSourceData _
                Source:=Range("'Лист3'!$A$1:$T$20")
    ActiveChart.ChartType = xlSurface
    Sheets("Лист3").Select
End Sub
```

В текстах кодов полужирным шрифтом и синим цветом выделены названия типов диаграмм: «проволочная» и «поверхностная».

После щелчка двух кнопок, размещённых на Листе1, соответственно, на Листе2 и на Листе3 автоматически возникают диаграммы, показанные на рис. 9.7 и 9.8.

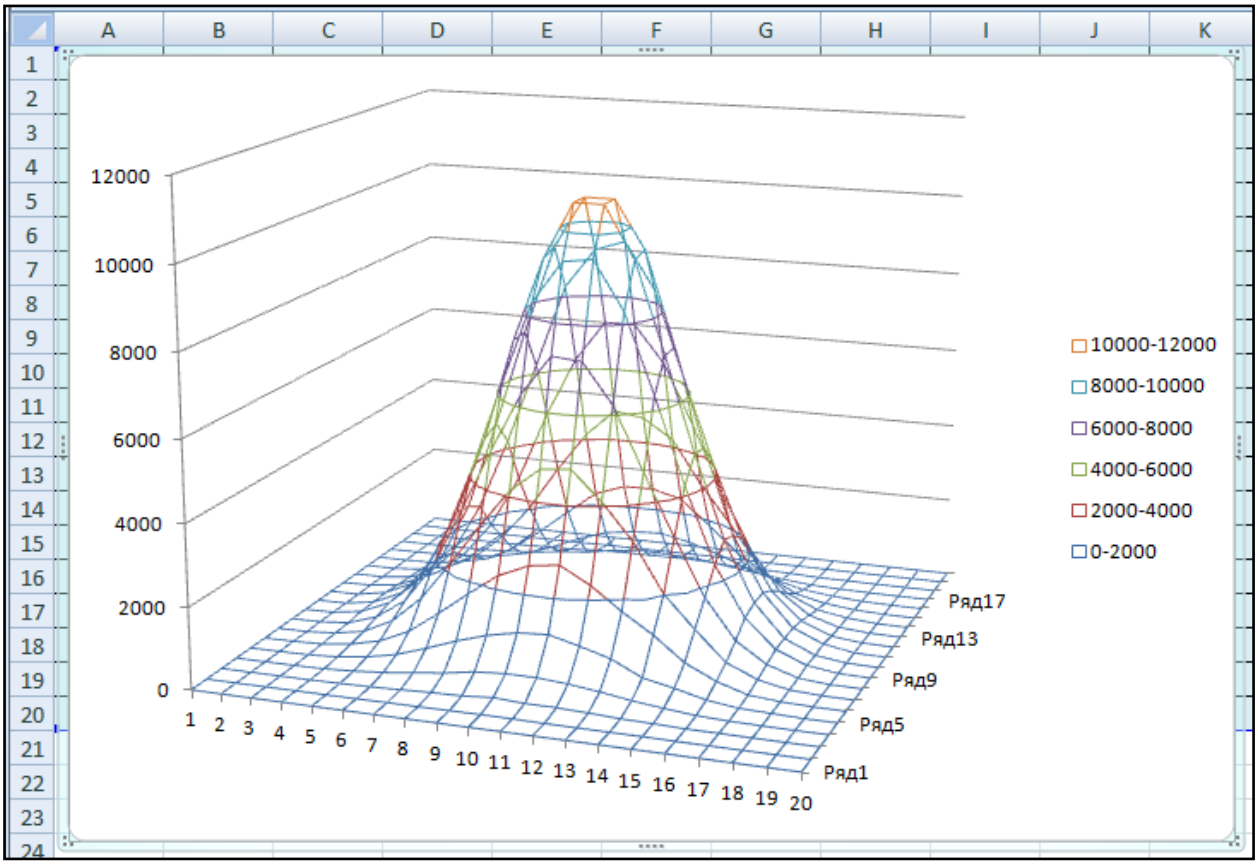


Рис. 9.7. «Проволочная» диаграмма, появляющаяся на Листе2

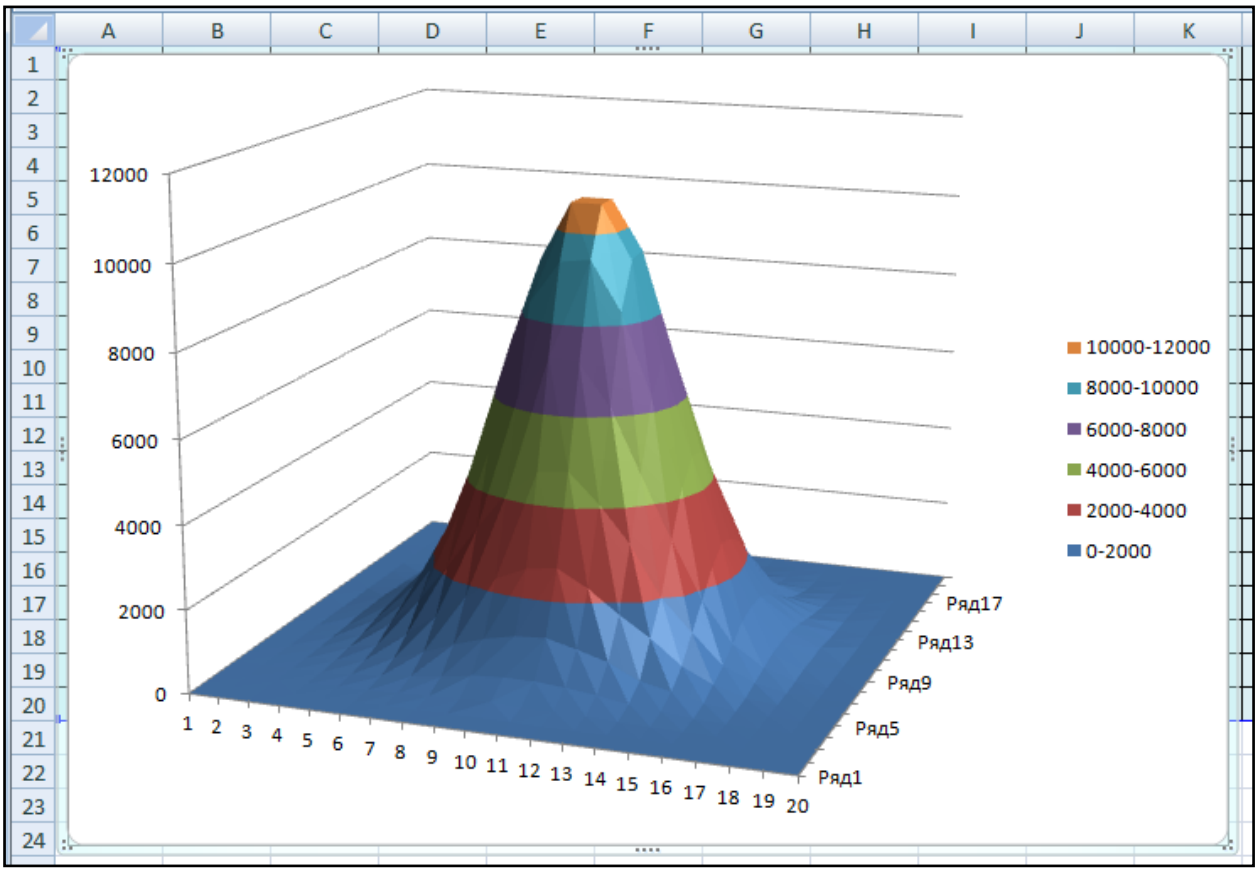


Рис. 9.8. «Поверхностная» диаграмма, появляющаяся на Листе3

Код основной процедуры макроса следующий:

```
Sub СтрельбаПоМишени()
    Dim x As Double, y As Double, c As Double
    Dim M(1 To 20, 1 To 20) As Double
    Dim i As Integer, j As Integer, _
        iw As Integer, jw As Integer
    Dim k As Long
    For i = 1 To 20       ' Обнуление матрицы M и _
                          ' и ячеек Листа Excel
        For j = 1 To 20
            M(i, j) = 0: Cells(i, j) = 0
        Next
    Next
    Randomize
    For k = 1 To 500000
            x = 4 * (Rnd + Rnd + Rnd + Rnd + Rnd)
                If x = 20 Then x = 19.9
                ' чтобы неравенство x < j _
                ' срабатывало и при j = 20
            y = 4 * (Rnd + Rnd + Rnd + Rnd + Rnd)
                If y = 20 Then y = 19.9
                ' чтобы неравенство y < i _
                ' срабатывало и при i = 20
      For i = 1 To 20
        For j = 1 To 20
            If x >= j - 1 And x < j And _
               y >= i - 1 And y < i Then _
            M(i, j) = M(i, j) + 1: iw = i: jw = j
        Next
      Next
    Next
    c = M(10, 10) + M(10, 11) + M(11, 10) + M(11, 11)
    c = c / 4
    MsgBox c
    M(10, 10) = c: M(10, 11) = c
    M(11, 10) = c: M(11, 11) = c
    For i = 1 To 20
        For j = 1 To 20
            M(i, j) = M(i, j)
            Cells(i, j) = M(i, j)
        Next
    Next
End Sub
```

9.6. Программирование процедуры «дихотомии» (бинарного поиска) с использованием типа данных, определяемого пользователем, и файла прямого доступа на примере бинарного поиска данных в списке налогоплательщиков с помощью индексного файла ИНН

В данной работе студентам предлагается воспользоваться знаниями, полученными на 6-й, предыдущей лекции, о бинарном поиске (с помощью «дихотомии») в одномерном отсортированном массиве.

Задание 5

Предлагается реализовать бинарный поиск в «реальных» условиях, когда имеются в виду большие объёмы информации, содержащейся в так называемых «длинных» файлах. Речь идёт о файлах прямого доступа (RAF), в записях которых содержатся многочисленные данные об очень большом количестве людей, например данные о налогоплательщиках. О разнообразии и объёме этих данных свидетельствует следующий код, в котором объявляется соответствующий пользовательский тип данных:

```
Private Type PersonLongData
    INN As Long
    fam As String * 30
    im As String * 30
    otch As String * 30
    bdate As Date
    bloc As String * 30
    tel As String * 40
    email As String * 40
End Type
```

Здесь **INN** – идентификационный номер налогоплательщика; **fam** – его фамилия; **im** – имя; **otch** – отчество; **bdate** – дата рождения; **bloc** – место рождения; **tel** – телефон (телефоны); **email** – адрес (адреса) электронной почты.

Для быстрого поиска по методу «дихотомии» интересующей нас записи в «длинном» файле создаётся так называемый «короткий» индексный файл. Запись этого «короткого» файла содержит всего два значения: (1) номер **INN** и (2) номер записи в «длинном» файле, содержащей этот номер

INN. Записи «короткого» файла отсортированы по индексу – номеру **INN**. Поэтому, даже если число этих записей **N** очень велико, поиск происходит очень быстро, за $k = \log_2 N$ шагов. И так как «длинный» файл является файлом прямого доступа, интересующая нас запись находится тоже быстро.

Тип данных для записи «короткого» файла также объявляется как пользовательский тип:

```
Private Type PersonShortData
    INN As Long
    IRecord As Long
End Type
```

Ниже – остальные строки кода макроса, который в данном случае является кодом экранной формы (рис. 9.9).

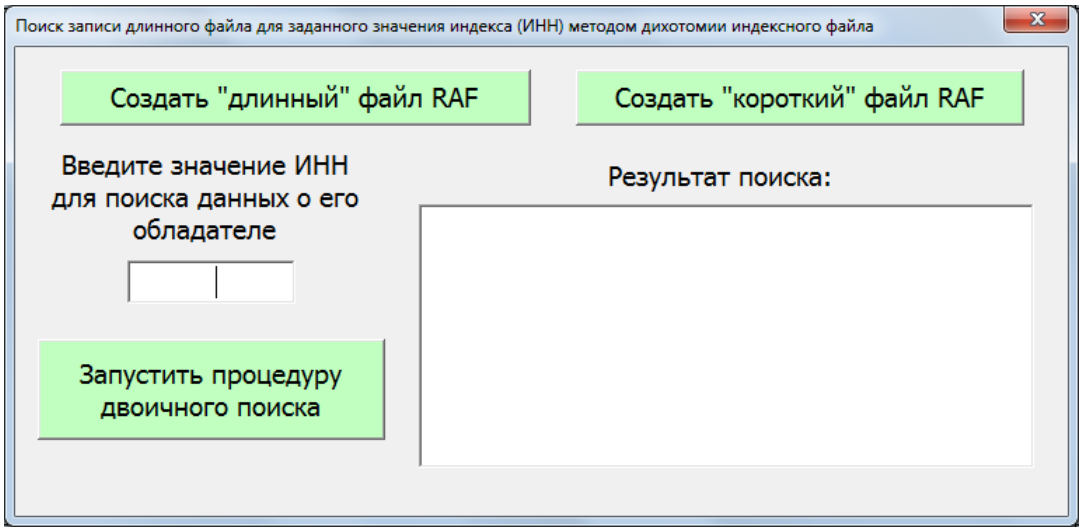


Рис. 9.9. Форма для двоичного поиска в «длинном» файле RAFс помощью «короткого» файла RAF

Здесь две верхние командные кнопки – с именами **CommandButton1** и **CommandButton2** – предназначены для создания «длинного» и «короткого» файлов. А третья кнопка с именем **CommandButton3** предназначена для запуска процедуры двоичного поиска.

Кроме кнопок, на форме есть текстовое поле **TextBox1**, в которое пользователь вводит значение **INN** искомого налогоплательщика, а также поле **ListBox1**, которое, в случае успешного завершения двоичного поиска, запол-

няется восемью значениями – данными о налогоплательщике с указанным значением **INN** из «длинного» файла[9].

Пример – на рис. 9.10.

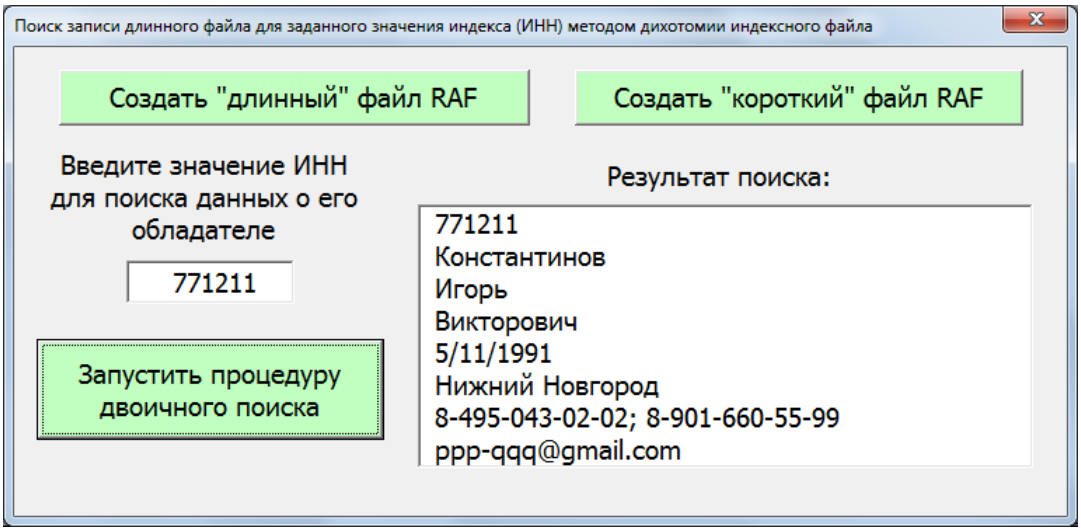


Рис. 9.10. Форма для двоичного поиска в «длинном» файле RAF с помощью «короткого» файла RAF с результатом, полученным после щелчка кнопки CommandButton3

Коды для нажатия каждой из трёх командных кнопок на показанной на экранной форме (рис. 9.9 и 9.10) следующие:

```
Dim L As Long, x As Long, y As Long
Dim N As Long, imin As Long, imax As Long
Dim k As Integer, limit As Integer, i As Integer
Dim Person As PersonLongData,
Dim IPers As PersonShortData
Dim INN As Long, fam As String, _
    im As String, otch As String
Dim bdate As Date, bloc As String, _
    tel As String, email As String
```

```
Private Sub CommandButton1_Click()
   x = Len(Person)
   MsgBox "Длина записи файла TaxPayers = " & x
   With Лист1
   Open "D:\MyFiles\" & "TaxPayers.raf" For Random _
                          As #1 Len = x
        For i = 2 To 33
            Person.INN = Val(.Cells(i, 1))
            Person.fam = .Cells(i, 2)
            Person.im = .Cells(i, 3)
```

[9] Для простоты, в данном примере количество знаков **INN** сокращено до шести.

```
            Person.otch = .Cells(i, 4)
            Person.bdate = CDate(.Cells(i, 5))
            Person.bloc = .Cells(i, 6)
            Person.tel = .Cells(i, 7)
            Person.email = .Cells(i, 8)
            Put #1, i - 1, Person
        Next
      End With
   Close 1
   MsgBox "File TaxPayers.raf with long records" & _
          " created!"
End Sub
```

```
Private Sub CommandButton2_Click()
      y = Len(IPers): _
      MsgBox "Длина записи файла ITaxPayers = " & y
      With Лист2
   Open "D:\MyFiles\" & "ITaxPayers.raf" For Random _
            As #1 Len = y
        For i = 2 To 33
            IPers.INN = Val(.Cells(i, 1))
            IPers.IRecord = Val(.Cells(i, 2))
            Put #1, i - 1, IPers
        Next
      End With
   Close 1
   MsgBox "File ITaxPayers.raf with short records" & _
          " created!"
End Sub
```

```
Private Sub CommandButton3_Click()
        ListBox1.Clear
        x = Len(Person): y = Len(IPers)
        MsgBox "Len(Person) = " & x & "; _
                Len(IPers) = " & y
   Open "D:\MyFiles\" & "TaxPayers.raf" For Random _
            As #1 Len = x
   Open "D:\MyFiles\" & "ITaxPayers.raf" For Random _
            As #2 Len = y
   L = LOF(1) \ x
   N = Val(TextBox1.Text)
   imin = 0: imax = L + 1
   i = imax
   k = 0
   limit = Int(Log(L) / Log(2) + 1)
        Do Until k > limit
        i = (imin + imax) \ 2
        k = k + 1
        Get #2, i, IPers
            If IPers.INN > N Then
                imax = i
            ElseIf IPers.INN < N Then
                imin = i
            Else
                Exit Do
            End If
      Loop
      If k <= limit Then
        Get #1, IPers.IRecord, Person
        ListBox1.AddItem Person.INN
        ListBox1.AddItem Person.fam
        ListBox1.AddItem Person.im
        ListBox1.AddItem Person.otch
        ListBox1.AddItem Person.bdate
        ListBox1.AddItem Person.bloc
        ListBox1.AddItem Person.tel
        ListBox1.AddItem Person.email
      Else
        ListBox1.AddItem "Искомого ИНН нет в файлах"
      End If
      Close 1
      Close 2
End Sub
```

Ниже, на рис. 9.11 и 9.12, приведены два фрагмента одного листа книги Excel с данными о 32-х налогоплательщиках в текстовой форме. Эти данные расположены в восьми колонках этого листа.

Для наглядности 32 строки этого листа отсортированы по названию места рождения. Их можно было бы отсортировать и по-другому. Например, по фамилии или по дате рождения. На результат работы программы двоичного поиска это, разумеется, влияния не имеет.

	A	B	C	D	E	F
1	ИНН	Фамилия	Имя	Отчество	Дата рождения	Место рождения
2	771177	Дмитриев	Олег	Иннокентьевич	01.04.91	Владивосток
3	770973	Петров	Павел	Сергеевич	15.07.91	Волоколамск
4	771176	Терентьев	Иван	Иванович	24.09.90	Дубна
5	770222	Германова	Жанна	Константиновна	23.02.92	Киев
6	770818	Львов	Богдан	Степанович	22.08.91	Киев
7	770136	Олегов	Иван	Петрович	13.05.91	Курск
8	770812	Александров	Иван	Ильич	19.08.91	Минск
9	770510	Егоров	Игорь	Сергеевич	15.12.91	Минск
10	770630	Егорова	Анна	Сергеевна	15.12.91	Минск
11	770532	Алексеева	Мария	Викторовна	31.12.91	Москва
12	771078	Владимиров	Дмитрий	Алексеевич	01.05.89	Москва
13	770915	Данилова	Светлана	Дмитриевна	30.09.91	Москва
14	770340	Елисеев	Владимир	Владимирович	13.01.92	Москва
15	771171	Иванов	Сергей	Александрович	16.11.91	Москва
16	771172	Иванова	Дарья	Александровна	16.11.91	Москва
17	771244	Панин	Егор	Петрович	14.06.91	Москва
18	770519	Петров	Павел	Сергеевич	16.08.91	Москва
19	770316	Сидоров	Сергей	Павлович	23.08.91	Москва
20	770433	Харитонов	Владимир	Владимирович	17.01.91	Москва
21	770304	Яковлева	Юлия	Петровна	01.01.92	Москва
22	771211	Константинов	Игорь	Викторович	11.05.91	Нижний Новгород
23	771070	Михайлов	Семен	Семенович	02.01.92	Одесса
24	770441	Федорова	Зоя	Сергеевна	25.09.91	Одесса
25	771205	Юрьев	Николай	Иванович	27.06.92	Пенза
26	771245	Борисова	Елена	Петровна	08.03.91	Санкт-Петербург
27	770131	Григорьева	Анна	Ивановна	01.04.91	Санкт-Петербург
28	770438	Иванов	Юрий	Павлович	07.11.92	Санкт-Петербург
29	770437	Иванов	Михаил	Павлович	07.11.92	Санкт-Петербург
30	770690	Романов	Николай	Николаевич	19.08.91	Санкт-Петербург
31	770213	Николаева	Инга	Эдуардовна	12.04.91	Смоленск
32	771175	Женин	Иван	Петрович	27.06.92	Тамбов
33	770999	Люсина	Людмила	Геннадьевна	17.02.92	Ярославль
34						
35			Открыть форму "Двоичный поиск"			
36						

Рис. 9.11. Левая сторона (колонки «A» – «F») листа книги Excel с данными о налогоплательщиках

G	H
Телефоны	Электронная почта
8-495-456-55-66; 8-901-166-90-01	vvv-18-aaa.mmm@yandex.ru
8-903-032-01-03; 8-985-11-80	petrov.pasha@yandex.ru
8-495-701-14-99; 8-901-777-09-09	qqq121212sss@mail.ru
8-495-443-52-62; 8-901-220-20-10	gg1999-hh112233@gmal.com
8-903-932-61-63; 8-965-17-18	zzzxxxyyy@gmail.com
8-495-001-02-00; 8-913-555-66-00	qqq121212sss@mail.ru
8-499-123-22-33; 8-915-541-91-95	gghh44112233@gmal.com
8-913-332-00-55	aaa.bbb.1999@yandex.ru
8-495-423-22-44; 8-916-546-06-07	aaa.bbb-1991@yandex.ru
8-495-456-55-66; 8-901-110-00-11	www-qqq12@mail.ru
8-495-423-22-44; 8-916-546-46-47	ccc.15-1999.ddd@yandex.ru
8-499-123-22-33; 8-915-541-91-95	eeefff2001.bbb@yandex.ru
8-499-843-72-66; 8-905-223-00-88	ppp0011-222-qqq@gmail.com
8-495-120-72-73; 8-916-547-31-35	ffgghh112233@gmal.com
8-495-888-89-99; 8-916-112-51-52	aaa.bbb@yandex.ru
8-499-888-22-22; 8-915-220-20-28	aaa.bbb@yandex.ru
8-495-943-01-72; 8-901-960-11-22	petrov-pavel@gmail.com
8-499-127-17-19; 8-915-111-45-95	ffgghh112233@gmal.com
8-495-801-22-00; 8-914-505-77-77	nnn.bbb.1999@yandex.ru
8-903-032-04-04; 8-915-16-86	petrusha-11-22@gmail.com
8-495-043-02-02; 8-901-660-55-99	ppp-qqq@gmail.com
8-499-006-04-96; 8-901-766-99-99	qqq121212sss@mail.ru
8-916-912-50-30	zzz-18-aaa.mmm@yandex.ru
8-499-008-22-22; 8-915-220-20-00	mika-1991@yandex.ru
8-903-332-00-33	aaa.bbb@yandex.ru
8-903-332-81-33; 8-985-77-88	qqq121212sss@mail.ru
8-495-156-55-00; 8-901-880-77-55	qqq121212sss@mail.ru
8-985-333-00-44	aaa.bbb@yandex.ru
8-903-999-69-69; 8-965-10-10	zzzxxxyyy@gmail.com
8-916-002-80-50	ff-gg-hh123@mal.ru
8-903-332-81-33; 8-985-77-88	zzz-1998-xxx-yyy@gmail.com
8-499-123-11-89; 8-915-241-41-98	ffgghh112233@gmal.com

Рис. 9.12. Правая сторона (колонки «G» – «H») листа книги Excel с данными о налогоплательщиках

Глава 10

Данная глава – это текст седьмой, заключительной лекции по курсу программирования на языке Visual Basic для офисных приложений. Содержание лекции можно структурировать следующим образом:

- Понятие объекта как совокупности данных и действий. Примеры объектов.
- Объекты в ООП – объектно-ориентированном программировании на языке **VBA for MS Excel**.
- Свойства и методы объектов языка **VBA for MS Excel**.
- Инкапсуляция, наследование, полиморфизм.
- Коллекции в языке **VBA for MS Excel**.
- Объекты **Range** и **Cells** для обработки ячеек электронной таблицы (листа книги **Excel**). Примеры использования этих объектов в программах.
- Работа с несколькими книгами **Excel** и с несколькими листами одной книги.
- Пример программирования макросов в приложении **MS Word** с объектами, отличными от объектов электронных таблиц.

10.1. Понятие объекта как совокупности данных и действий. Примеры объектов

Объект – это синтез **данных** (в частности, переменных величин) и **действий**, которые эти данные обрабатывают (рис. 10.1).

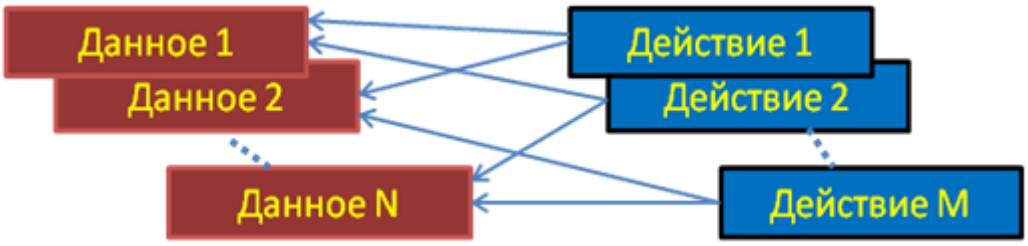


Рис. 10.1. Объект как совокупность данных и действий

Примеры объектов

Пример 10.1. *«Механические часы».* Детали часов (шестерёнки, анкерный механизм) – это **данные**. Процессы их взаимодействия – это **действия**.

Пример 10.2. *«Человек».* ***Анатомия*** (сердце, мозг, скелет, мышцы, язык, глаза, уши …) – это **данные.** ***Физиология*** (взаимодействие органов в процессе жизнедеятельности) – это **действия**.

Пример 10.3. *«Автомобиль».* Марка, номер, масса, цвет, размеры, характеристики двигателя и т.д. – это **данные.** Изменение направления движения автомобиля под действием руля, зажигание, включение фар и т.д. – это **действия**.

Пример 10.4. «*Экранная форма»* – в визуальном программировании, например, на языке Visual Basic. Элементы управления (*Controls*) на экранной форме (**ListBox1, CommandButton1, TextBox2, ScrollBar2** и т.д.) – это **данные**. Процедуры-события (**CommandButton1_Click, TextBox2_Change** и т.д.) – это **действия**.

10.2. Понятие объекта в ООП – объектно-ориентированном программировании на языке VBA for MS Excel

Программирование на языке VBA for MS Excel называют **ООП – «объектно-ориентированным программированием».** ***Объекты*** – это элементы («кирпичики»), из которых строится программное приложение.

Объектов в программе может быть много – сколько угодно. В то же время, ***видов*** этих объектов ограниченное число. Эти виды называются ***классами*** объектов.

Конкретный объект – это представитель (или ***экземпляр***) класса.

Например, на экранной форме можно установить произвольное число командных кнопок. Например, при моделировании экранной клавиатуры или экранного калькулятора: **CommandButton1, CommandButton2, CommandButton3, …** Число этих элементов управления может быть любым. Но все они принадлежат одному классу **CommandButton** и представляют собой экземпляры этого класса.

У класса должно быть уникальное ***имя*** и набор ***свойств***. Значения многих свойств определённого класса объектов передаются ***по наследству*** от класса к его экземплярам – конкретным объектам. Например, значения свойств «шрифта» (**Font**) класса объектов **Label** на экранной форме оказались равными:

Font.Name = “Tahoma”; Font.Size = 8,25.

Перед установкой на экранной форме десятка разных меток мы можем заменить заданные «по умолчанию» значения указанных свойств указанного класса на новые значения, например: **“Arial”** и **12**. При установке на форме этих десяти меток (***экземпляров класса***) замечаем, что у них всех будут именно эти новые, изменённые проектировщиком (пользователем) значения.

Отметим, что есть наследуемые свойства, а есть ненаследуемые.

Например, у класса животных под названием «Тигр» есть наследуемое свойство «Окрашенность шкуры». Но свойство «Вес» объекта «Тигр1» (как экземпляра класса «Тигр») является ненаследуемым свойством.

10.3. Свойства и методы объектов языка VBA for MS Excel

Детализация представленных выше понятий «данные» и «действия»:

- *данные*, которые можно видеть снаружи, называются ***свойствами***;
- *действия,* которыми можно управлять снаружи, называются ***методами***.

В этом есть глубокий смысл: зачем, например, широкому пользователю знать, как устроен телевизор внутри – ему достаточно уметь пользоваться предназначенным ему пультом управления с двумя десятками кнопок.

В объектно-ориентированном программировании (ООП) программисту-пользователю (какими являемся и мы с вами) видны таблицы свойств, значения которых мы можем менять, а также встроенные процедуры (методы), которыми мы можем пользоваться, реализуя алгоритм для решения нашей задачи. «Невидимых» действий и данных, которые эти действия обрабатывают, гораздо больше, но программисту нашего уровня их знать не нужно и даже вредно ☺! Это утверждение демонстрирует так называемый принцип инкапсуляции, о котором будет идти речь далее.

10.4. Инкапсуляция, наследование, полиморфизм

Инкапсуляция. Это слово означает«сокрытие» информации. Объекты должны скрывать свою внутреннюю структуру от пользователя (с целью сохранности этой структуры) и проявлять себя только своей «видимой» стороной – через свои свойства и методы.

Наследование. Новый объект можно определить на основе уже существующих, наследуя у них значения многих свойств.

Вернемся к примеру, приведённому в разделе 10.1.2.

Пусть «по умолчанию» значения свойств «шрифта» экранной формы установлены равными: **Font.Name = “Tahoma”; Font.Size = 8,25**. Если эти значения не изменить ***сразу***, то у всех элементов (новых объектов), которые будут устанавливаться на этой форме, они будут такими же.

Полиморфизм. Этот термин означает возможность использования ***одних и тех же*** методов ***разными*** классами объектов.

Например, метод «+» можно использовать и в арифметических выражениях, и при «склеивании» (конкатенации) строк, и при объединении констант в аргументе функции **MsgBox**, определяющем вид окна сообщения.

10.5. Коллекции в языке VBA for MS Excel

Коллекция – это свойство класса объектов, которое позволяет обращаться к любому из представителей этого класса. Свойство это весьма специфическое: оно представляет *семейство объектов* данного класса.

Понятие коллекции аналогично понятию массива объектов данного класса.

Рассмотрим наиболее важные коллекции приложения **MS Excel.**

Объектом самого высокого уровня в программировании на VBA for Excel является объект **Application**, в него входит ***коллекция* Workbooks.**

В эту коллекцию входят все ***рабочие книги***, которые являются объектами с именем **Workbook**. Каждый объект **Workbook** в свою очередь содержит ***коллекцию* Worksheets**. В эту коллекцию входят все ***рабочие листы,***

которые являются объектами с именем **Worksheet**. Каждый объект **Worksheet** в свою очередь содержит более «мелкие» коллекции, например **Cells** (***ячейки***), **Range** (***диапазоны***) и т.д.

Они объединяются в т.н. ***иерархию объектов*** приложения MS Excel.

В операторах программы после имени ***коллекции*** ставятся скобки, в которых может быть или номер объекта в ***коллекции***, или имя этого объекта.

(Может быть и выражение, значением которого являются указанные номер или имя…)

Например:

Worksheets(“Лист3”).Activate или: **Worksheets(3).Activate**

10.6. Объекты Range и Cells для обработки ячеек электронной таблицы. (листа Excel). Примеры использования этих объектов в программах

Обрабатывать ячейки можно с помощью объекта **Range**.

Другой способ – с помощью объекта **Cells**.

Пример 10.1. Заполнить матрицу 3x3 так, как показано на рис. 10.2.

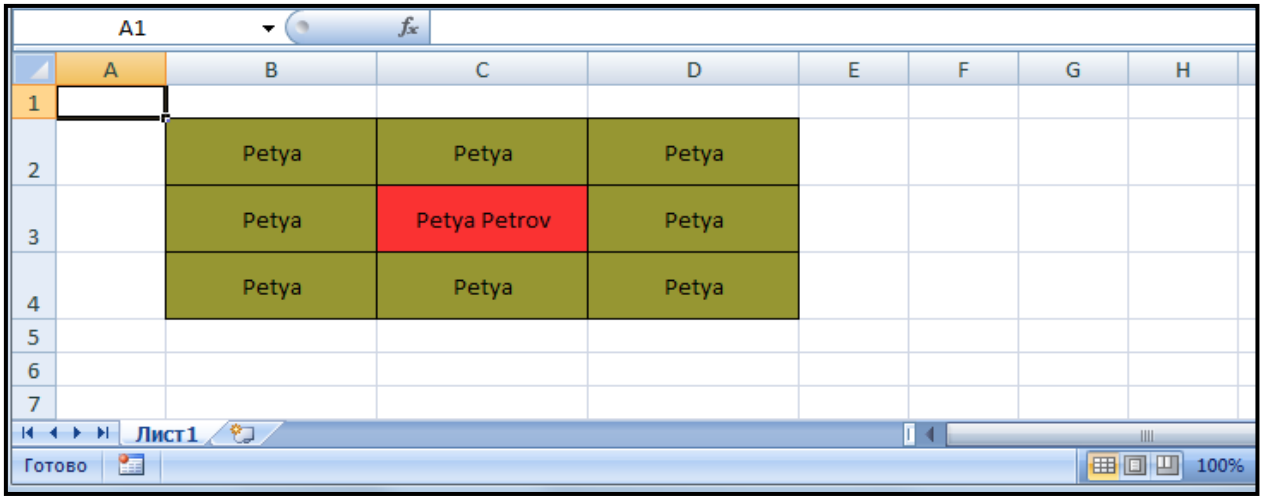


Рис. 10.2. Матрица 3x3, заполненная с использованием объектов **Cells** или **Range**

Код макроса:

```
Sub ОбработкаЯчеек()
    Range("2:4").RowHeight = 30
    Range("B:D").ColumnWidth = 16.86
    Range("B2:D4").Value = "Petya"
    Range("B2:D4").Interior.Color = RGB(150, 150, 50)
    Cells(3, 3).Value = Cells(3, 3).Value & " Petrov"
```

```
    Cells(3, 3).Interior.Color = RGB(250, 50, 50)
End Sub
```

Пример 10.2. Заполнить матрицу 10x10 так, как показано на рис. 10.3, без обращения к макросу на этапе открытия книги (**Workbook Open**).

	A	B	C	D	E	F	G	H	I	J	K
1	111	110	192	416	643	996	557	324	415	517	
2	592	830	851	161	849	547	629	666	128	338	
3	673	21	481	316	159	464	208	846	514	483	
4	859	539	413	817	236	260	781	453	390	122	
5	796	631	621	487	881	838	480	970	714	812	
6	768	433	219	543	535	991	573	268	83	5	
7	202	475	961	90	781	903	972	108	236	540	
8	165	173	743	624	842	649	727	644	553	147	
9	865	338	99	533	82	694	531	919	554	950	
10	149	668	706	595	505	394	215	367	399	960	
11											

Рис. 10.3. Матрица 10x10, заполненная на этапе открытия книги Excel случайными числами и раскрашенная случайными цветами с использованием объекта **Cells**

Код процедуры открытия книги Excel на рис. 10.4.

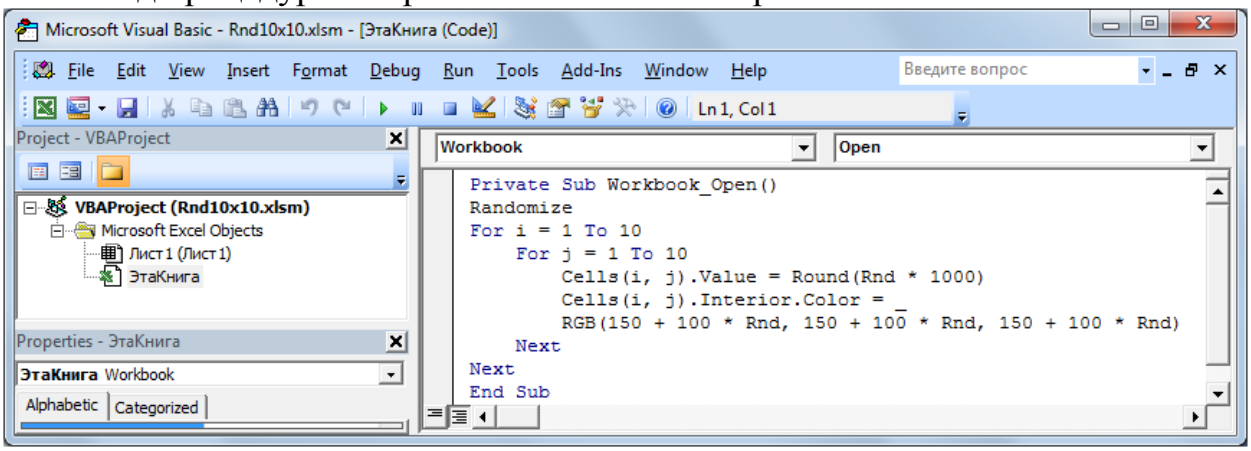


Рис. 10.4. Код процедуры открытия книги Excel с одновременным заполнением матрицы с использованием объекта Cells

Отметим, что для решения задачи нам не пришлось создавать никаких кнопок и макросов. Как показано на рис. 10.4, используется объект, о котором шла речь выше. Это объект **Workbook**. Окно программного кода для этого объекта открывается простым щелчком по значку **ЭтаКнига** в Окне обозревателя проекта – слева на рис. 10.4.

10.7. Работа с несколькими книгами Excel и с несколькими листами одной книги

Рассмотрим вопрос, как можно организовать работу с несколькими книгами **Excel** одновременно.

Пример 10.3. На рис. 10.5 показаны окна трёх книг Excel, между которыми необходимо организовать взаимодействие.

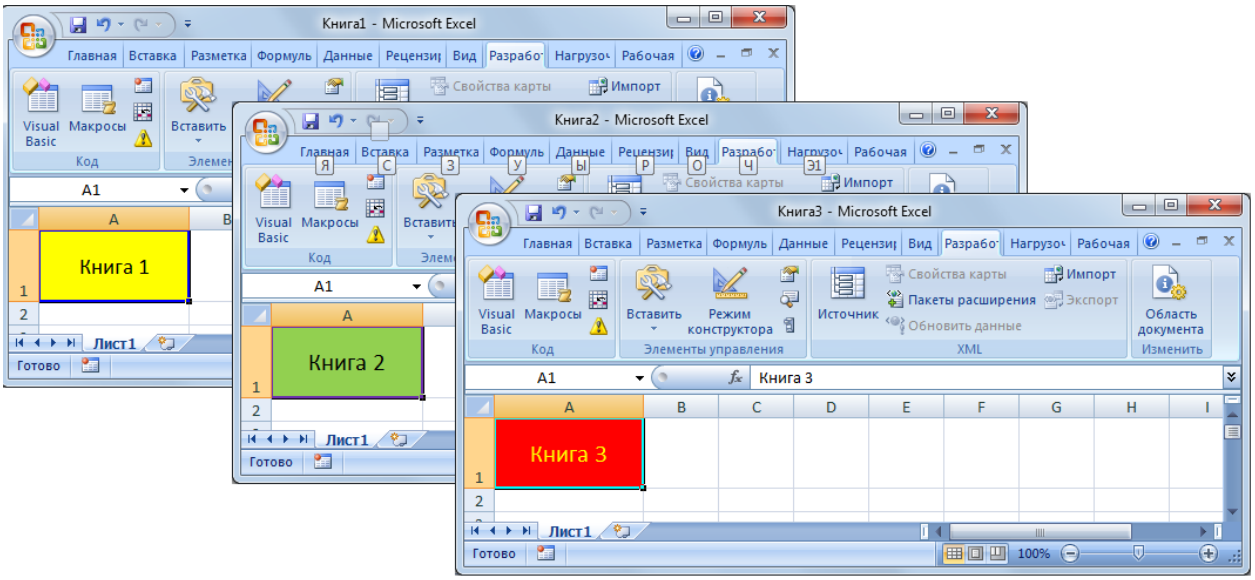


Рис. 10.5. Окна трёх книг, между которыми организовано взаимодействие

В каждой из трёх книг, показанных выше, необходимо создать один простой макрос: его выполнение активирует другую книгу.

На рис. 10.6 показана панель Visual Basic для решения этой задачи.

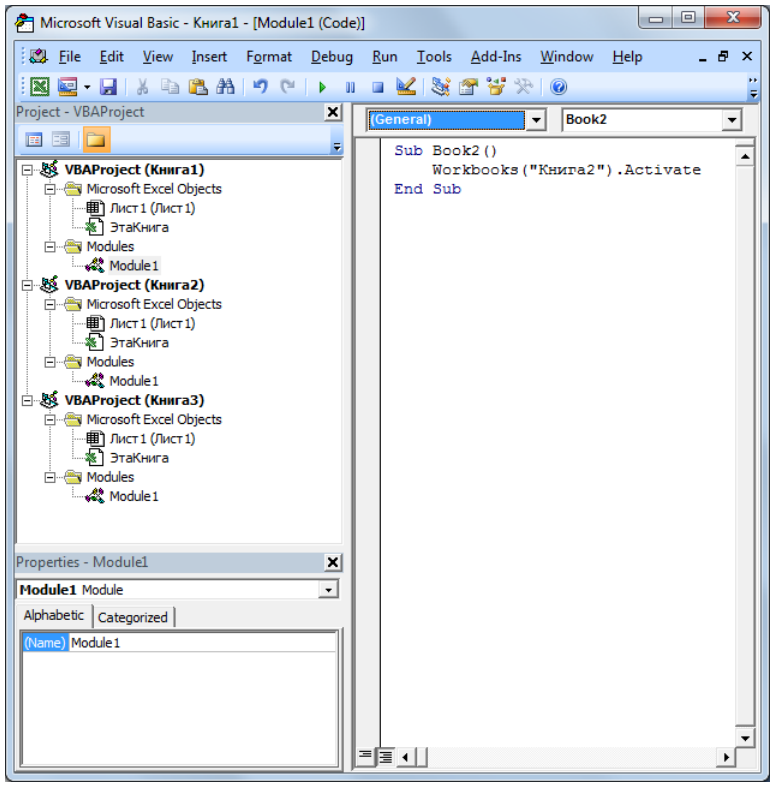


Рис. 10.6. Панель для трёх книг **Excel** в одном проекте

Создадим по одному макросу (процедуре) в каждой книге:

1) Макрос в Книге1 под именем Book2, который активирует Книгу2.
2) Макрос в Книге2 под именем Book3, который активирует Книгу3.
3) Макрос в Книге3 под именем Book1, который активирует Книгу1.

Все три процедуры в каждом из 3-х модулей (с одинаковым именем **Module1** для 1-й, 2-й и 3-й Книги соответственно) выглядят так:

```
Sub Book1()
    Workbooks("Книга1").Activate
End Sub
```

```
Sub Book2()
    Workbooks("Книга2").Activate
End Sub
```

```
Sub Book3()
    Workbooks("Книга3").Activate
End Sub
```

Пример 10.4. На рис. 10.7 показано окно одной книги Excel с четырьмя листами, которым присвоены имена **Север**, **Восток**, **Юг** и **Запад**. Необходимо организовать взаимодействие между этими листами.

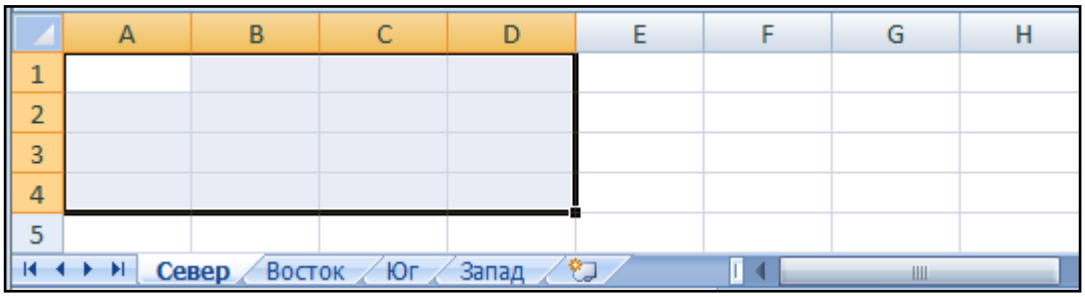


Рис. 10.7. Окно книги с четырьмя листами: Север, Восток, Юг, Запад

В частности, предлагается организовать открытие любого из этих листов щелчком опциональной кнопки на экранной форме на рис. 10.8:

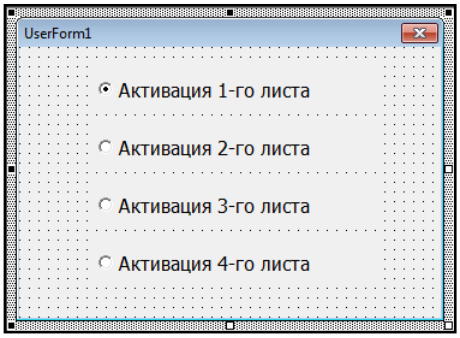


Рис. 10.8. Экранная форма с опциональными кнопками для активации любого из листов

Щелчок любой из «радио кнопок» на форме вызывает заполнение 16 клеток на одной из четырёх страниц лишь одним словом – именем выбранной страницы (рис. 10.9).

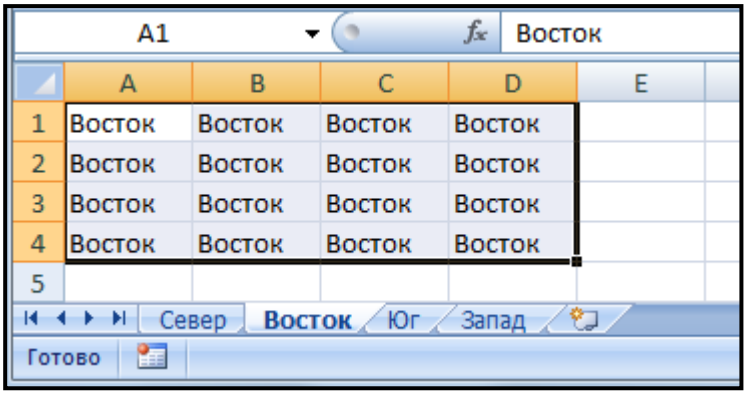


Рис. 10.9. Страница с заполненной её именем матрицей 4x4

Ниже приведены коды процедур для щелчков каждой из четырёх опциональных кнопок. В процедурах используется обращение к элементам коллекции **Worksheets** – к именам ***S*** соответствующих листов книги Excel – и последующим затем заполнением именами **s** диапазона **A1:D4**.

```
Dim i As Integer, S As String
Private Sub OptionButton1_Click()
    i = 1:  S = Worksheets.Item(1).Name
    Zapolnenie i, S
End Sub
```

```
Private Sub OptionButton2_Click()
    i = 2: S = Worksheets.Item(2).Name
    Zapolnenie i, S
End Sub
```

```
Private Sub OptionButton3_Click()
    i = 3: S = Worksheets.Item(3).Name
    Zapolnenie i, S
End Sub
```

```
Private Sub OptionButton4_Click()
    i = 4: S = Worksheets.Item(4).Name
    Zapolnenie i, S
End Sub
```

```
Sub Zapolnenie(j As Integer, S As String)
    Worksheets.Item(j).Range("A1:D4").Value = S
    Worksheets.Item(j).Activate
End Sub
```

10.8. Пример программирования макросов в приложении MS Word с объектами, отличными от объектов электронных таблиц

Далее следуют четыре страницы документа **MS Word** с поддержкой макросов. На них приведены примеры работы двух макросов: (1) «Выплата по кредиту» и (2) «Реверсирование строки символов». В данное учебное пособие эти страницы вставлены, разумеется, уже без поддержки макросов. Но на них сохранены элементы (командная кнопка и текстовые поля), предназначенные для работы с указанными макросами.

(1) Макрос «Выплата по кредиту»[10]

Пример 10.5. В меню приложения **MS Word** на ленте «Разработчик» выбирается «Режим конструктора». После этого, в произвольном месте открытого документа устанавливается командная кнопка для демонстрации работы макроса «Выплата по кредиту» (рис. 10.10).

Рис. 10.10. Кнопка на данной странице документа Word

На ленте «Разработчик» в разделе «Код» (как это демонстрировалось во всех предыдущих лекциях для приложения MS Excel) открывается окно **Visual Basic**. В этом окне выбирается команда меню **Insert** и в данный проект добавляется экранная форма **UserForm1** (рис. 10.11).

[10] Данный материал изложен в разделе 6.4 лекции 6 данного пособия.

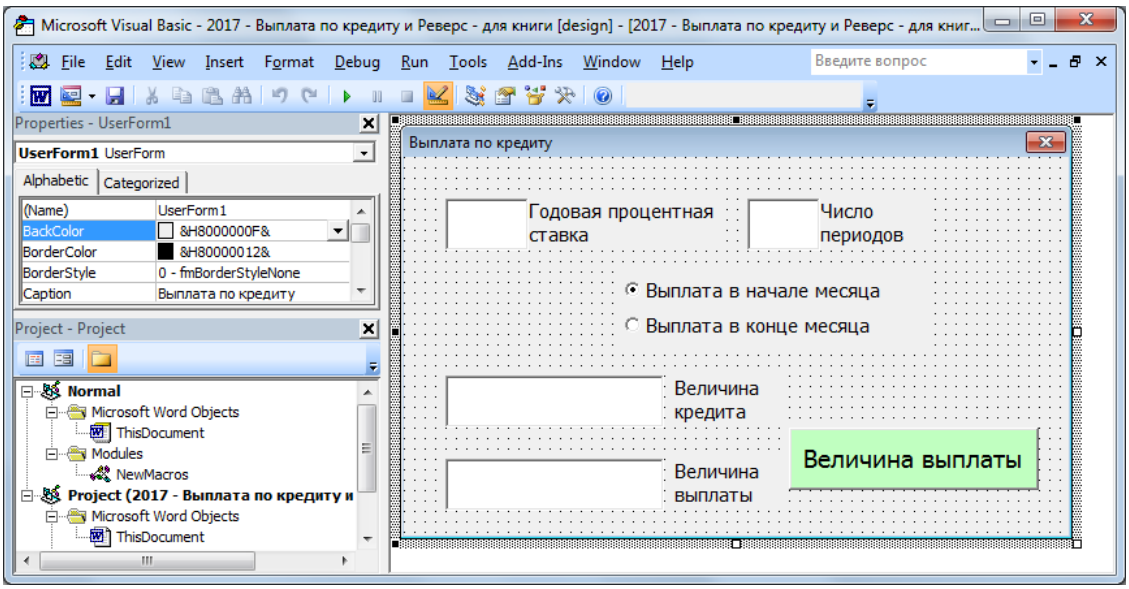


Рис. 10.11. Форма **UserForm1**, вызываемая кнопкой на данной странице документа Word

Для открытия этой формы щелчком командной кнопки (рис. 10.10) в объект **ThisDocument** записывается код следующей процедуры:

```
Private Sub CommandButton1_Click()
    UserForm1.Show
End Sub
```

Для работы с этой формой в объект **UserForm1** вставляется код процедуры щелчка кнопки на данной форме:

```
Dim r As Double, n As Integer, _
    v As Currency, t As Variant, _
    p As Currency

Private Sub CommandButton1_Click()
    r = Val(TextBox1.Text) / 1200
    n = Val(TextBox2.Text)
    v = Val(TextBox3.Text)
        '   t = InputBox("Выплата в начале (1)
        '   или в конце (0) периода?",
        '   "Учет задержки выплаты", "1")
    If OptionButton1.Value = True _
       Then t = 1 Else t = 0
    p = -Pmt(r, n, v, 0, t)
    TextBox4.Text = Str(Round(p, 2))
End Sub
```

После отключения «Режима конструктора» щелчок кнопки в документе вызывает появление формы. После заполнения её полей и щелчка кнопки «Величина выплаты» мы получаем результат, показанный на рис. 10.12.

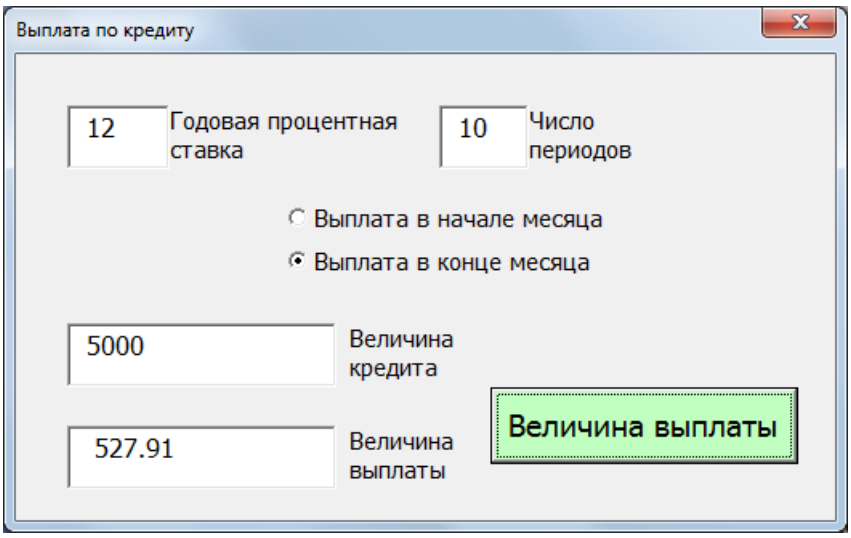


Рис. 10.12. Форма **UserForm1** с полученным результатом

(2) Макрос «Реверсирование строки символов»[11]

Пример 10.6. Как и в предыдущем примере 10.5, в меню приложения MS Word на ленте «Разработчик» выбирается «Режим конструктора», после чего в произвольном месте открытого документа устанавливаются два текстовых поля для демонстрации работы макроса «Реверсирование строки символов» (рис. 10.13).

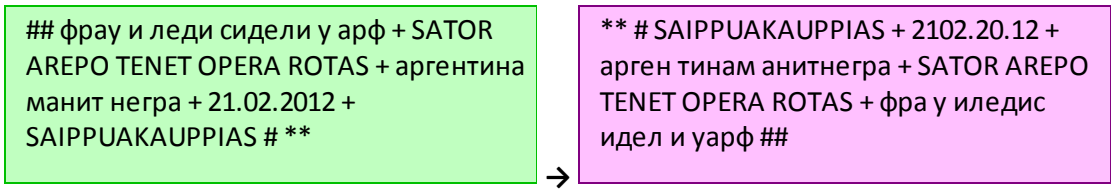


Рис. 10.13. Текстовые поля для демонстрации макроса «Реверсирование строки символов»

При вставке с помощью клавиатуры любого символа в любое место первого текстового поля автоматически меняется текст во втором поле.

Отметим, что в данном случае «для интереса» в качестве частей исходной строки (текста в первом текстовом поле) использовались так называемые палиндромы – строки, которые одинаково читаются как слева направо, так и справа налево.

[11] Данный материал изложен в разделе 6.5 лекции 6 данного пособия.

Примеры палиндромов

Палиндромы уже знали в Древнем Риме:

SUM SUMMUS MUS

(«Я — сильнейшая мышь»).

Или: **SATOR AREPO TENET OPERA ROTAS** («Сеятель Арепо с трудом удерживает колёса»). (рис. 10.14)

Рис. 10.14. Древнеримский палиндром

21.02.2012 – эта замечательная дата – тоже ***палиндром***!

Ещё пример из Википедии:

SAIPPUAKAUPPIAS – продавец мыла (финский язык).

Добавляем к уже имеющейся в модуле **ThisDocument** процедуре **CommandButton_Click** две другие процедуры: **TextBox1_Change** – для любого изменения содержимого первого текстового поля, и рекурсивную процедуру **Reverse** – для реверсирования этого содержимого (рис. 10.15).

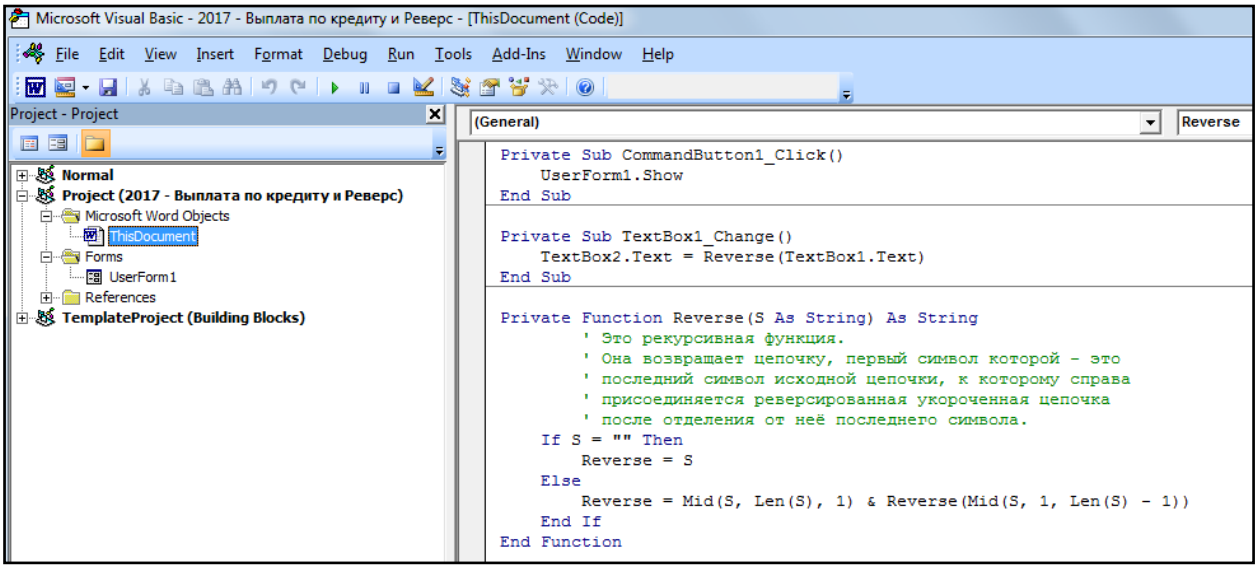


Рис. 10.15. Окно программного кода модуля **ThisDocument** с добавленными в него двумя процедурами: **TextBox1_Change** и **Reverse**

Коды указанных процедур:

```
Private Sub TextBox1_Change()
    TextBox2.Text = Reverse(TextBox1.Text)
End Sub
```

```
Private Function Reverse(S As String) As String
  ' Это рекурсивная функция.
  ' Она возвращает цепочку, первый символ _
  ' которой – это последний символ
  ' исходной цепочки, к которому справа _
  ' присоединяется реверсированная укороченная
  ' цепочка после отделения от неё
  ' последнего символа.

    If S = "" Then
        Reverse = S
    Else
        Reverse = Mid(S, Len(S), 1) & _
        Reverse(Mid(S, 1, Len(S) - 1))
    End If

End Function
```

Список литературы

1. Волчёнков Н.Г. Учимся программировать: Visual Basic 5. – М.: Диалог-МИФИ, 1998.
2. Волчёнков Н.Г. Программирование на Visual Basic 6. – В 3-х т. – М.: Изд-во ИНФРА-М, 2000, 2002.
3. Весь OFFICE 2007. Полное руководство / П.В. Колосков, А.Н. Тихомиров, А.К. Прокди, И.А. Клеандрова – СПб.: Наука и Техника, 2009.
4. Слепцова Л.Д. Программирование на VBA в Microsoft Office 2010. Самоучитель. – М.: ООО «И.Д. Вильямс», 2010.