Министерство образования и науки Российской Федерации

Томский государственный университет систем управления и радиоэлектроники

**И.Г. Афанасьева**

**А.В. Дубровин**

# ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ

Методические указания по выполнению практических и самостоятельных работ

Томск

ТУСУР

2014

УДК 004.43 (076)

Афанасьева И.Г., Дубровин А.В.
Методические указания по выполнению практических и самостоятельных работ по дисциплине «Информационные технологии». 2014. –90с.

Предлагаемые методические указания по выполнению практических и самостоятельных работ выполняются студентами в компьютерном классе с использованием пакета Microsoft Office и встроенного редактора Visual Basic for Application.

Содержание

# Практическая работа №1

# Знакомство с системой VBA

## Тема и цель работы

Знакомство с понятием «макрос». Создание пользовательской панели инструментов.

## Теоретический материал, для освоения темы

Для работы с повторяющимися событиями наиболее удобно использовать заданный алгоритм действий. Например: «научить» компьютер создавать необходимую таблицу, а потом по мере необходимости лишь отдавать команду к подготовке таблицы, чтобы осталось внести в нее данные.

Для этого используется MacroRecoder – транслятор, позволяющий преобразовать все действия пользователя в макрос на языке Visual Basic for Application (VBA).

Макрос – последовательность команд на языке VBA.

Для сохранения последовательности действий в макрос необходимо перейти: **ВИД – Макросы –Запись макроса** (рис. 1).

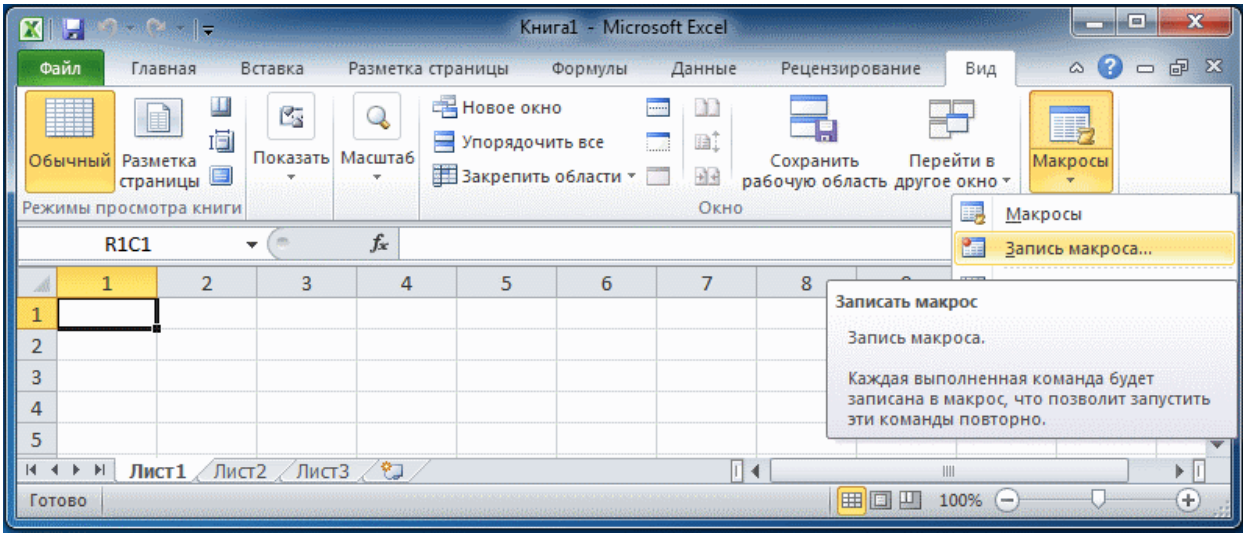



Рис. 1 Вкладка **Вид**на ленте в Excel

В появившемся диалоговом окне устанавливаем параметры макроса. Задаем имя макроса (по умолчанию присваиваются имена Макрос1, Макрос2 и т.д.), а в параметре **Сохранить в** указываем в какой книге Excelсохраняем макрос (рис. 2). Если выбрать **Личная книга макросов,** то макрос будет сохранен в специальной книге, макросы которой будут доступны для других рабочих книг.

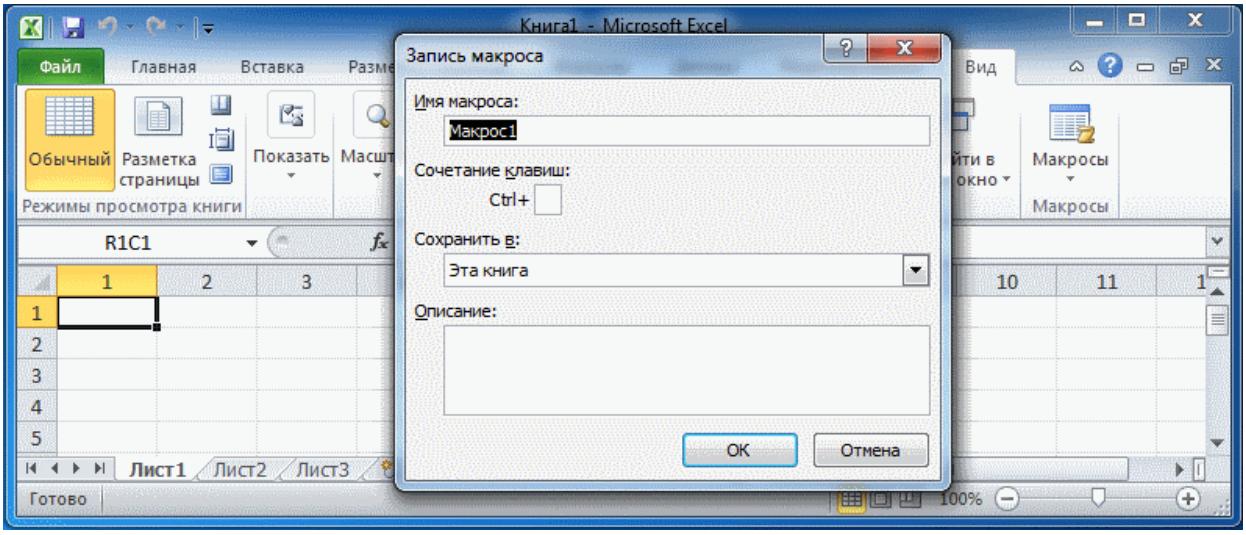



Рис. 2 Запись макроса в Excel

После указанных параметров и нажатия кнопки ОК, появляется плавающая панель записи макроса.

После ввода всей таблицы (или выполнения заданных действий: расчет функции, создание диаграммы и т.д.) необходимо нажать кнопку **Остановить** на плавающей панели.

После того как макрос был записан, его можно активизировать на другой рабочей области (Листе): **Вид – Макросы– Макросы**. В появившимся диалоговом окне выбираем нужный макрос и нажимаем кнопку **Выполнить**.

Для редактирования записанного макроса необходимо:

- Выбрать команду **Вид – Макросы – Макросы**. Выводится диалоговое окно **Макрос**.
- Выбрать имя требуемого макроса.

- Нажать кнопку **Изменить**. Макрос выводится в окне редактора Visual Basic for Applications.

**Создание кнопки для макроса на панели быстрого доступа**

Перейти на вкладку **Файл**. Нажать кнопку **Параметры**, для открытия диалогового окна **Параметры Excel**, а затем щелкнуть **Панель быстрого доступа**.

В списке под надписью **Выбрать команды из**: выбрать **Макрос**. В появившемся списке найти макрос, который нужно добавить, и выбрать его.

Нажать кнопку **Добавить >>**, чтобы добавить макрос в список с правой стороны, а затем нажать кнопку **Изменить…**, чтобы выбрать изображение кнопки, связываемой с макросом.

**Задания на лабораторную работу**

**Задание 1**. Microsoft Word. С помощью макрорекордера написать макрос, который бы запускался с помощью кнопки быстрого доступа. Макрос должен установить для выделенного текста следующие характеристики шрифтов: шрифт – Arial, начертание – курсив, высота символов – 14 пт., цвет символов – синий. Вокруг выделенных слов должна быть рамка – снизу одинарная линия толщиной 1 пт., цвет линии зеленый, слева двойная линия черного цвета, толщиной 0,5 пт. Выделенные слова должны быть залиты желтым цветом.

**Задание 2**. Microsoft Word. С помощью макрорекордера написать макрос, который бы при нажатии комбинации клавиш <Ctrl + 1> для выделенного текста устанавливал шрифты, имеющие следующие характеристики: Monotype Corsiva, начертание полужирный курсив, высота – 18 пт., цвет символов – синий. Эффекты – с тенью. Масштаб символов 200%.

**Задание 3**. Microsoft Word. Написать макрос, который бы для выделенного абзаца устанавливал следующие параметры: Величина выступа

– 15 мм, междустрочное расстояние 1,6 интервала, отступы слева и справа по 10 мм, сверху и снизу по 6 пт., выравнивание по ширине.

**Задание 4**. Microsoft Word. С помощью макрорекордера написать макрос, который бы при нажатии кнопки быстрого доступа для данного абзаца расставлял границы. Слева и справа граница должна быть синего цвета двойная линия толщиной 0,75 пт., вверху границы быть не должно, а внизу границей должна быть волнистая линий красного цвета толщиной 1,5 пт.

**Задания на самостоятельную работу**

**Задание 1**. Microsoft Excel. С помощью макрорекордера написать макрос, который устанавливал «Все границы» у выделенных ячеек. Цвет заливки в первой строке тех же ячеек установить оливковым 40 %, у остальных – 80 %.

**Задание 2**. Microsoft Word. С помощью макрорекордера написать макрос, который вставлял таблицу умножения для чисел до 5 и заполнял её.

| | | | | |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 |
| 2 | 4 | 6 | 8 | 10 |
| 3 | 6 | 9 | 12 | 15 |
| 4 | 8 | 12 | 16 | 20 |
| 5 | 10 | 15 | 20 | 25 |

Рис. 3 Пример таблицы

# Практическая работа №2

# Знакомство с системой VBA.Структура редактора VBA

## Тема и цель работы

Знакомство с интерфейсом редактора VBA. Рассмотрение структуры кода на примере кода простого макроса MS Excel.

## Запуск редактора VBA

Все приложения Office 2010 используют ленту. Одной из вкладок на ленте является вкладка **Разработчик** (рис. 4), на которой можно вызвать редактор Visual Basic и другие инструменты разработчика. Поскольку в Office 2010 вкладка **Разработчик** не показана по умолчанию, необходимо вывести ее на экран, выполнив следующую процедуру.

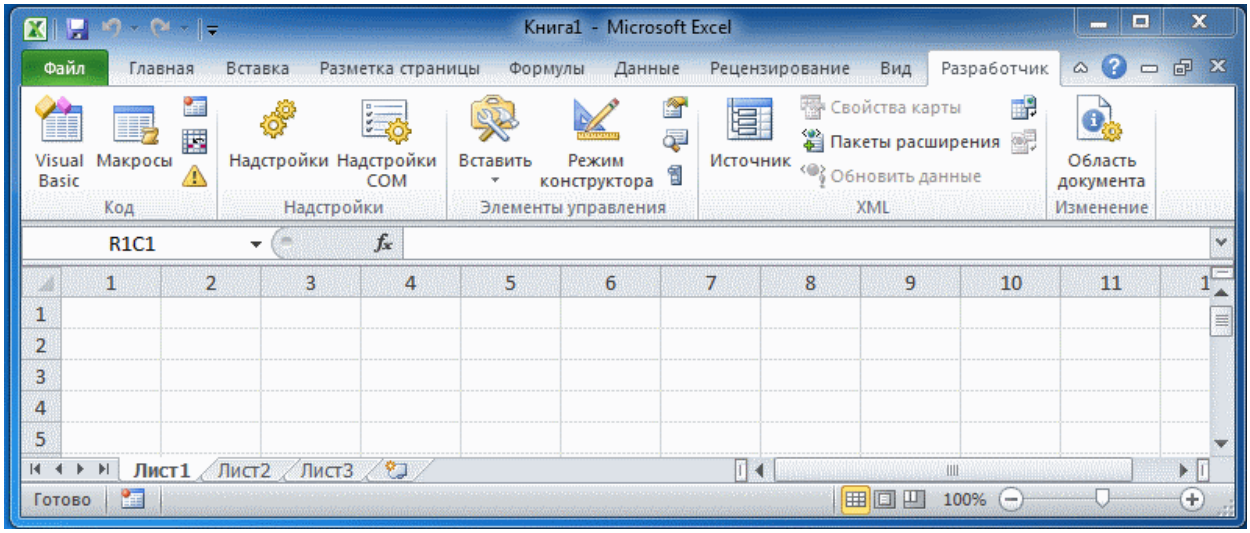



Рис. 4 Вкладка **Разработчик** на ленте в Excel

На вкладке **Файл** выбрать **Параметры**, чтобы открыть диалоговое окно **Параметры Excel**. Щелкнуть **Настройка ленты** в левой части диалогового окна. В разделе **Выбрать команды**, расположенном слева в окне, выбрать **Часто используемые команды**. В разделе **Настройка ленты**, который находится справа в диалоговом окне, выбрать **Основные вкладки**, а затем установить флажок **Разработчик**.

Также Редактор VBA можно открыть при помощи сочетания клавиш **<ALT+F11>**. Возвратиться из редактора VBA в рабочую книгу можно нажатием кнопки **Вид–Microsoft Excel (View–Microsoft Excel)**. Интерфейс VBA состоит из следующих основных компонентов: окно проекта, окно свойств, окно редактирования кода, окна форм, меню и панели инструментов (рис. 5).

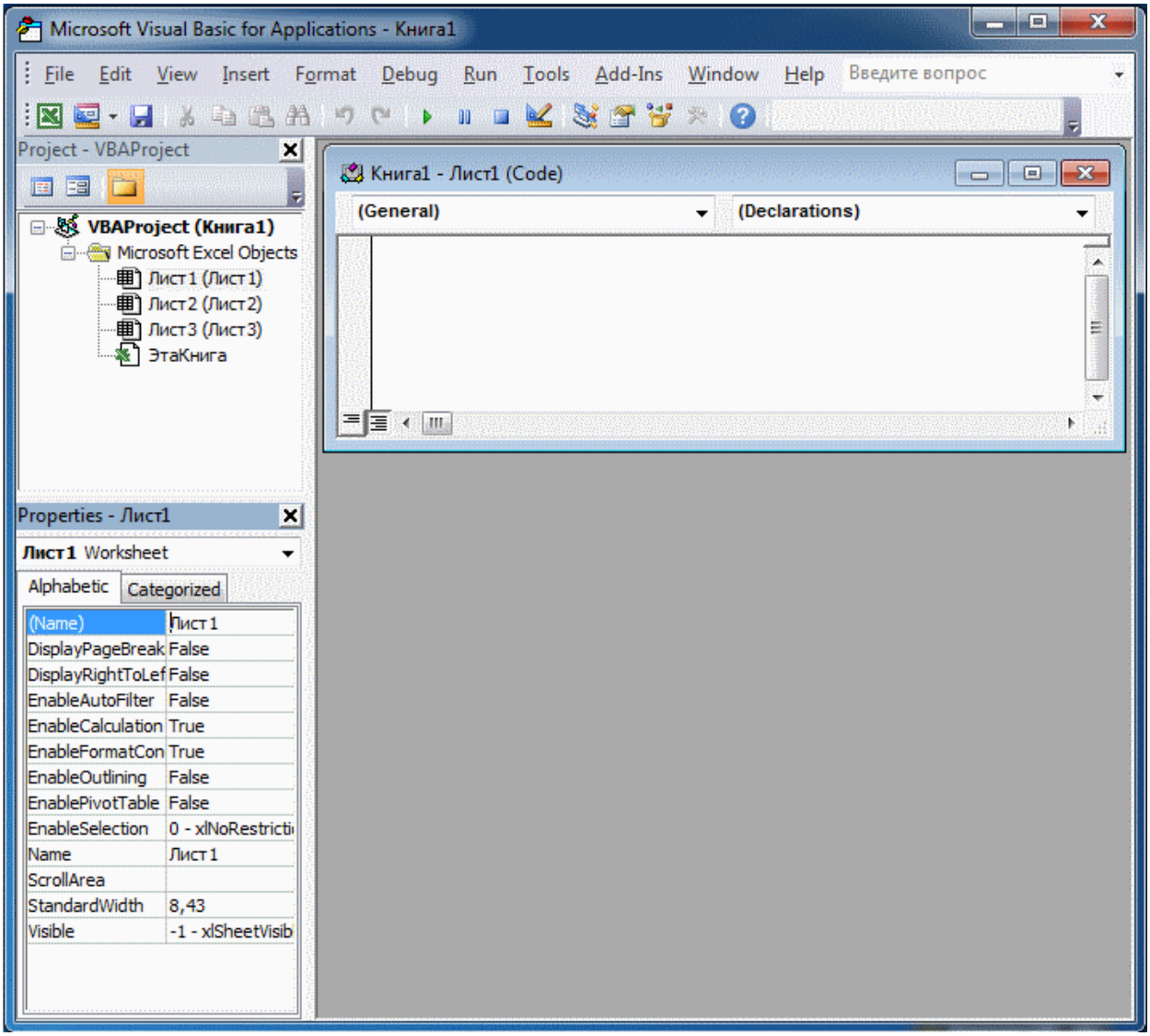



Рис. 5 Редактор VBA

**Проблемы безопасности**

Нажмите кнопку **Безопасность макросов**, чтобы определить, какие макросы могут выполняться и при каких условиях. Хотя неконтролируемый код макроса может серьезно повредить компьютер, условия безопасности,

запрещающие выполнять полезные макросы, могут серьезно ухудшить производительность работы. Безопасность макросов – это сложный и важный вопрос, в котором следует разобраться при работе с макросами Excel.

Необходимо помнить, что если при открытии книги, содержащей макрос, между лентой и листом появляется строка **Предупреждение системы безопасности: запуск макросов отключен**, можно нажать кнопку **Включить содержимое**, чтобы включить макрос.

Кроме того, в качестве мер безопасности, нельзя сохранить макрос в формате файлов Excel, используемом по умолчанию (XLSX-файлы), вместо этого макрос должен быть сохранен в файл со специальным расширением, XLSM-файл.

**Окно проекта (Project)**

Окно проекта в редакторе VBA активизируется выбором команды **Вид, Окно проекта** (View, Projectexplorer). В окне проекта (Project – VBAProject) представлена иерархическая структура файлов форм и модулей текущего проекта (рис. 6).

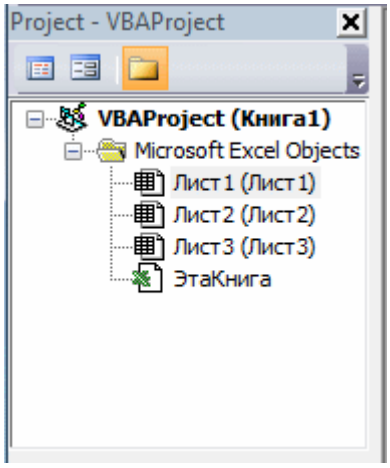



Рис. 6 Окно проекта

В проекте автоматически создается модуль для каждого рабочего листа и для всей книги. Кроме того, модули создаются для каждой пользовательской формы, макросов и классов. По своему предназначению модули делятся на два типа: модули объектов и стандартные. К стандартным модулям относятся те, которые содержат макросы. Такие модули

добавляются в проект командой **Вставка –Модуль(Insert–Module).** К модулям объектов относятся модули, связанные с рабочей книгой, рабочими листами, формами, и модули класса.

Формы создаются командой **Вставка –UserForm (Insert–UserForm),** а модули класса – командой **Вставка –Модуль класса(Insert–Module).** По мере создания, добавления и удаления файлов из проекта эти изменения отображаются в окне проекта. Отметим, что удаление файла из окна проекта производится выбором значка файла с последующим выполнением команды **Файл –Удалить(File–Delete).**

В окне проекта выводится проект всех открытых рабочих книг. Это позволяет легко копировать формы и коды из одного проекта в другой, что убыстряет процесс создания новых приложений.

**Окно свойств (Properties)**

В окне свойств отображаются основные параметры свойств выбранной формы или элемента управления (рис. 7). Для отображения этого окна надо либо щелкнуть кнопку **Properties Window**, либо выбрать команду **View – Properties Window**, либо нажать клавишу <**F4**>.

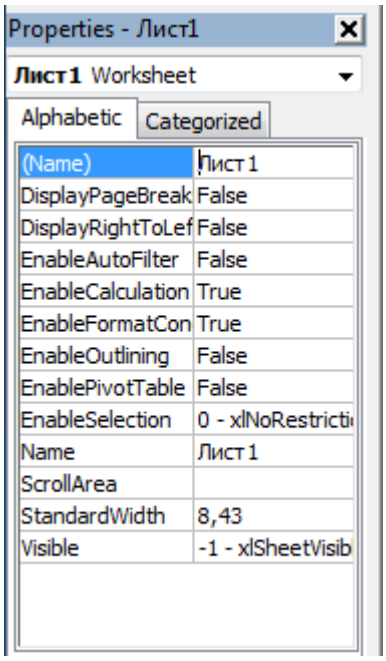



Рис. 7 Окно свойств

В верхней части окна свойств можно из раскрывающегося списка выбрать как форму, так и любой элемент на ней. Основная часть окна свойств состоит из списка свойств выбранного элемента, каждое из которых можно изменить, при необходимости, или с клавиатуры, или выбрать одно из значений выбранного свойства из раскрывающегося списка.

**Окно редактирования кода**

Окно редактирования кода (рис. 8) служит в качестве редактора для ввода и изменения кода процедур приложения. Код внутри модуля организован в виде отдельных разделов для каждого объекта, программируемого в модуле.

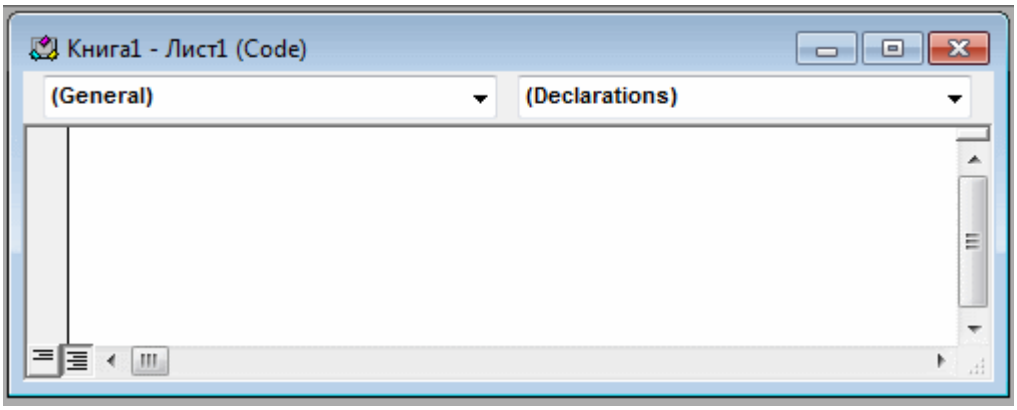



Рис. 8 Окно редактирования кода

В окне редактирования доступны два режима представления кода: просмотр отдельной процедуры и всего модуля. Переключение режимов работы окна редактирования кода осуществляется выбором одной из двух кнопок в нижнем левом углу окна редактирования кода, либо установкой или снятием флажка **Просмотр всего модуля (Default to Full Module View)** вкладки **Редактор (Editor)** диалогового окна **Параметры (Options),** отображаемого на экране командой **Инструменты–Параметры (Tools–Options)**.

Два раскрывающихся списка в верхней части окна редактора кода облегчают ориентацию в процедурах. Левый раскрывающийся список позволяет выбрать управляющий элемент или форму, а правый – содержит

список событий, допустимых для выбранного в левом списке объекта. Отметим, что при выборе элемента управления в форме посредством двойного щелчка или перемещении указателя на элемент управления и нажатии кнопки **Программа (View Code)** открывается окно редактирования кода как раз в том месте, где располагается процедура, связанная с этим элементом управления. Обратный переход от процедуры к объекту управления быстрее всего осуществить нажатием кнопки **Объект (View Object).**

### Интеллектуальные возможности редактора кода

Написание программ существенно облегчается за счет способности редактора кода автоматически завершать написание операторов, свойств и параметров. При написании кода редактор сам предлагает пользователю список компонентов, логически завершающих вводимую пользователем инструкцию. Например, набирая код `Range("A1")` после ввода точки на экране отобразится список компонентов (рис. 9), которые логически завершают данную инструкцию. Двойной щелчок на выбранном элементе из этого списка или нажатие клавиши <**Tab**> вставляет выбранное имя в код программы.

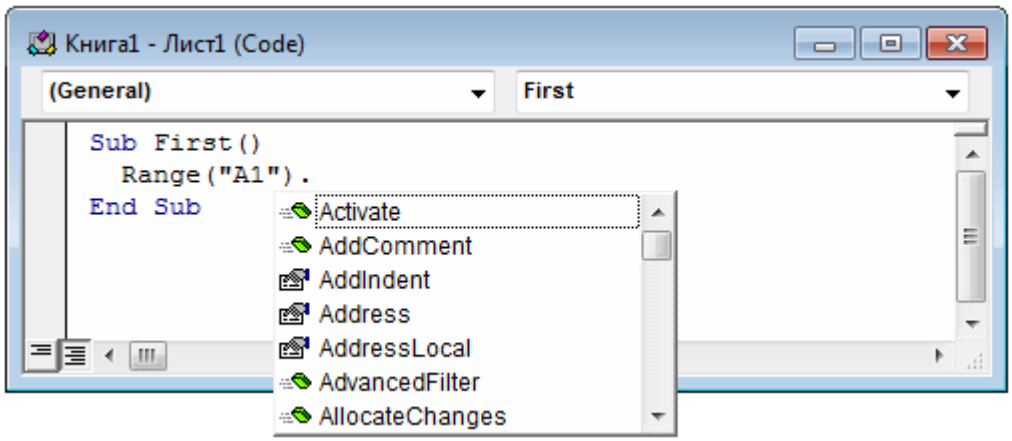



Рис. 9 Список компонентов

Автоматическое отображение списка компонентов происходит только при установленном флажке **Список компонентов (Auto List Members)** вкладки **Редактор (Editor)** диалогового окна **Параметры (Options),**

отображаемого на экране после выбора команды **Инструменты – Параметры (Tools–Options).**

Список компонентов можно выводить на экран нажатием комбинации клавиш <**Ctrl+J**>, при этом список отображается как при установленном, так и при снятом флажке **Список компонентов (Auto List Members)** вкладки **Редактор (Editor)** диалогового окна **Параметры (Options).**

Отображение списка компонентов, логически завершающих вводимую инструкцию, является одним из интеллектуальных качеств редактора кода. Этим качеством интеллектуальные ресурсы редактора кода не исчерпываются. Другим его такого рода качеством является автоматическое отображение на экране сведений о процедурах, функциях, свойствах и методах после набора их имени (рис. 10).

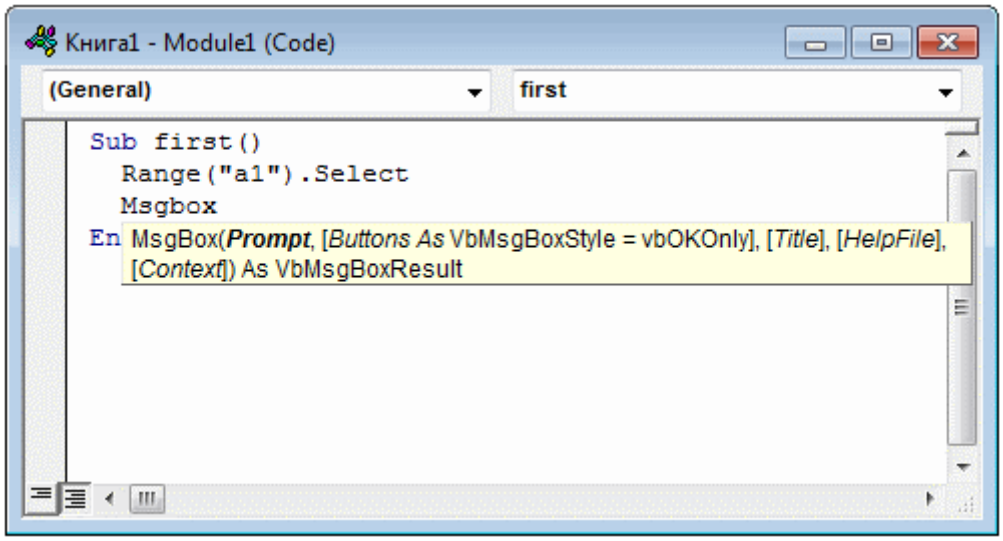



Рис. 10 Отображаемые сведения о вводимой процедуре

Автоматическое отображение на экране сведений о процедурах, функциях, свойствах и методах после ввода их имени происходит только при установленном флажке **Краткие сведения (Auto Quick Info)** вкладки **Редактор (Editor)** диалогового окна **Параметры (Options),** отображаемого на экране после выбора команды **Инструменты–Параметры (Tools–Options)**.

Описанную выше всплывающую подсказку можно также выводить на экран нажатием комбинации клавиш <**Ctrl+I**>. При этом всплывающая

подсказка отображается как при установленном, так и при снятом флажке **Краткие сведения** вкладки **Редактор** диалогового окна **Параметры (Options).**

Редактор кода также производит автоматическую проверку синтаксиса набранной строки кода сразу после нажатия клавиши <**Enter**>. Если после набора строки и нажатия клавиши <**Enter**> строка выделяется красным цветом, то это как раз и указывает на наличие синтаксической ошибки в набранной строке. Эту ошибку необходимо найти и исправить. Кроме того, если установлен флажок **Проверка синтаксиса (Auto Syntax Check)** вкладки **Редактор (Editor)** диалогового окна **Параметры (Options),** отображаемого на экране посредством выбора команды **Инструменты–Параметры (Tools–Options)**, помимо выделения красным цветом фрагмента кода с синтаксической ошибкой, на экране отображается диалоговое окно, поясняющее, какая возможная ошибка произошла.

Редактор кода обладает еще одной интеллектуальной возможностью, увеличивающей эффективность работы пользователя. Если курсор расположить на ключевом слове языка VBA, имени процедуры, функции, свойства или метода и нажать клавишу <**F1**>, то на экране появится окно со справочной информацией об этой функции.

**Окно Просмотр объектов (Object Browser)**

Окно **Просмотр объектов (Object Browser)** вызывается командой **Вид–Просмотр объектов (View–Object Browser)** или нажатием кнопки **Просмотр объектов (Object Browser)** (рис. 11). В этом окне приведен список всех объектов, которые имеются в системе и которые можно использовать при создании проекта.

Окно **Просмотр объектов (Object Browser)** состоит из трех основных частей:

1. Раскрывающегося списка **Проект/Библиотека (Project/Library)** в левом верхнем углу окна. В этом раскрывающемся списке можно выбрать

различные проекты и библиотеки объектов. В частности, библиотеки объектов Excel, VBA, Office и VBAProject (объекты пользовательского проекта).

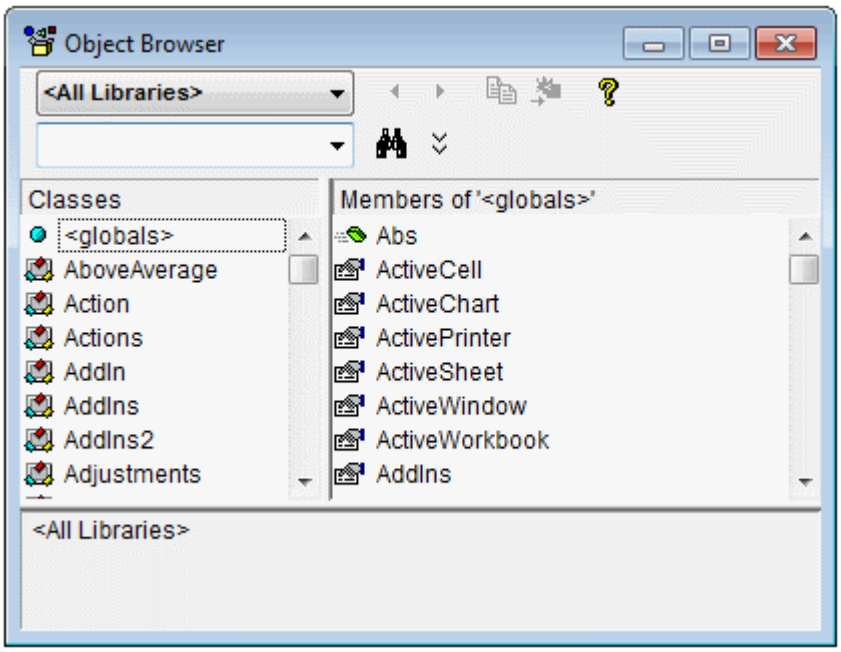



Рис. 11 Окно Просмотр объектов

Выбор в списке строки **Все библиотеки (All Libraries)** отображает список объектов всех библиотек.

2. Списка Классы (Classes). После выбора из раскрывающегося списка Проект/Библиотека (Project/Library) просматриваемой библиотеки, например, VBA, все классы объектов выбранной библиотеки выводятся в списке Классы (Classes).

3. Списка Компоненты (Members). После выбора класса из списка Классы (Classes) просматриваемой библиотеки, например, FileSystem, все компоненты выбранного класса выводятся в списке Компоненты (Members). При выделении строки в этом списке в нижней части окна Просмотр объектов (Object Browser) приводится дополнительная информация о выбранном компоненте. Кроме того, если нажать на кнопку Справка (Help), расположенную на панели инструментов в правой верхней часта окна Просмотр объектов (Object Browser), то на экране отобразится окно

Справочник Visual Basic (Microsoft Visual Basic Help) с подробной информацией о выделенном компоненте.

**Структура программ на VBA**

А теперь подробнее рассмотрим следующие понятия. Программы на VBA хранятся в проектах. Проект содержит модули различных типов, а модули включают различные процедуры.

Проект может содержать несколько модулей. Имеется три типа модулей:

- *стандартные модули* – это модули, в которых можно описать доступные во всем проекте процедуры;
- *модули класса* содержат описание объекта, который является членом класса. Процедуры, написанные в модуле класса, используются только в этом модуле. Среди модулей класса выделяют модули форм и отчетов, которые связаны с конкретной формой или отчетом, и Модули форм и отчетов часто содержит процедуры обработки событий, которые срабатывают в ответ на событие в форме или отчете. Процедуры обработки событий используются для управления поведением форм и отчетов и их реакцией на действия пользователя типа щелчка мыши на кнопке.

*Модули* содержат *описания* и *процедуры* – наборы описаний и инструкций, сгруппированных для выполнения. Существует три типа процедур:

- процедура*Sub* – набор команд, с помощью которого можно решить определенную задачу. При ее запуске выполняются команды процедуры, а затем управление передается в приложение пакета MSOffice или процедуру, которая вызвала данную процедуру.
- процедура*Function* (функция) также представляет собой набор команд, который решает определенную задачу. Различие заключается в том, что такие процедуры обязательно возвращают значение, тип которого можно описать при создании функции.

- процедура*Property*используется для ссылки на свойство объекта. Данный тип процедур применяется для установки или получения значения пользовательских свойств форм и модулей.

**Окно редактирования форм (UserForm)**

Для создания диалоговых окон, разрабатываемых приложений в VBA, используются формы. Редактор форм является одним из основных инструментов визуального программирования. Форма в проект добавляется с помощью команды **Вставка–Форма (Insert–UserForm)**. В результате на экран выводится незаполненная форма с панелью инструментов **Панель элементов (Toolbox)** (рис. 12).

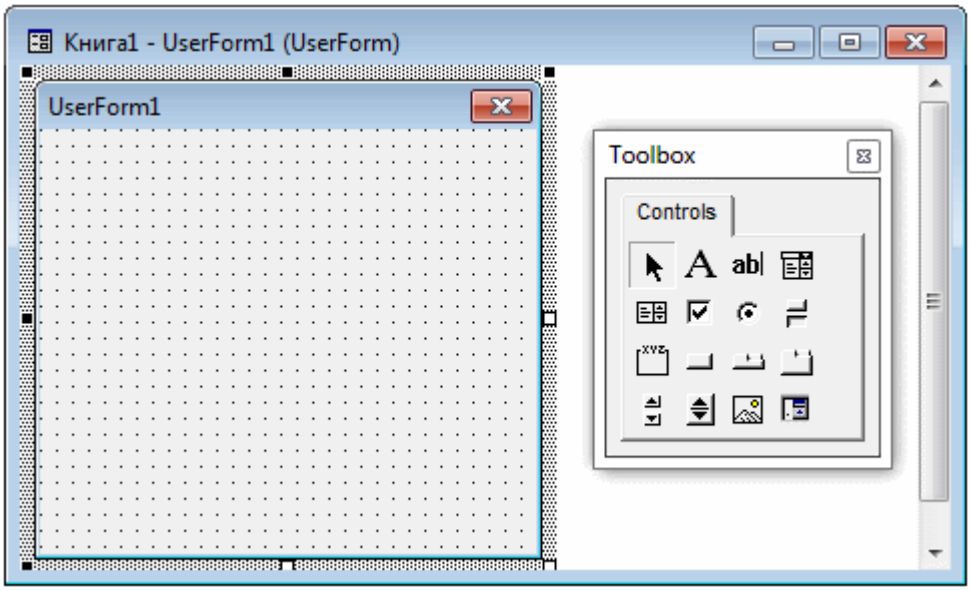



Рис. 12 Окно редактирования форм и панель инструментов

Используя панель инструментов **Панель элементов** из незаполненной формы, можно сконструировать любое требуемое для приложения диалоговое окно. Размещение нового управляющего элемента в форме осуществляется следующей последовательностью действий:

1. Щелкнуть значок того элемента, который необходимо разместить в форме.

2. Поместить указатель мыши на то место, где будет располагаться управляющий элемент.

3. Нажать левую кнопку мыши и, не отпуская ее, растянуть появившийся прямоугольник до требуемых размеров.

4. Отпустить кнопку мыши. Элемент управления на нужном месте создан.

Размеры формы и расположенных на ней элементов управления можно изменять. Технология изменения размеров стандартная для Windows: выделить изменяемый элемент, поместить указатель мыши на одном из размерных маркеров и протащить его при нажатой левой кнопки мыши так, чтобы объект принял требуемые размеры. Окно редактирования форм поддерживает операции буфера обмена.

Таким образом, можно копировать, вырезать и вставлять элементы управления, расположенные на поверхности формы. Для облегчения размещения и выравнивания элементов управления используется сетка. Активизировать ее можно с помощью вкладки **Общие (General)** диалогового окна **Параметры (Options),** вызываемого командой **Сервис–Параметры (Tools–Options),** там же устанавливается шаг сетки. Кроме того, команды меню **Формат (Format)** автоматизируют и облегчают процесс выравнивания элементов управления как по их взаимному местоположению, так и по размерам (рис. 13).

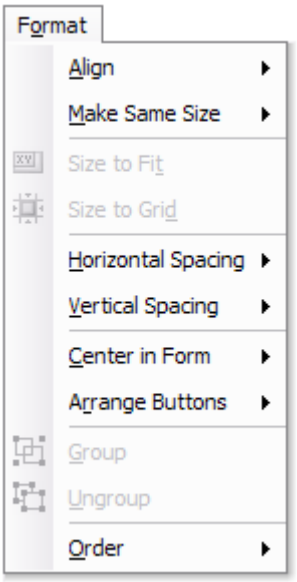



Рис. 13 Команды меню Формат

**Задания на лабораторную работу**

**Задание 1**. Открыть и проанализировать код VBAмакросов, созданных в предыдущей лабораторной работе.

**Задания на самостоятельную работу**

**Задание 1**. Microsoft Excel. Составить таблицу значений синуса и косинуса для чисел от –1,0 до 1,0 с шагом 0,1. Записать макрос, заполняющий формулу только для первой строки. Открыть созданный макрос и добавить строку (заменив соответствующие адреса ячеек)

```
Selection.AutoFill Destination:=Range("B1:C21"), Type:=xlFillDefault
```

Выполнить полученный макрос и проанализировать результат.

# Практическая работа №3

# Типы данных и переменные в Visual Basic

## Тема и цель работы

Типы данных и объявление переменных в Visual Basic. Операции, выражения, операторы. Приобрести навыки программирования с использованием встроенных функций ввода/вывода.

## Теоретический материал, для освоения темы

### Представление данных в памяти

**Данные**– величины, обрабатываемые программой. Имеется три основных вида данных: **константы, переменные и массивы**.

**Константы**–это данные, которые зафиксированы в тексте программы и не изменяются в процессе ее выполнения. Константы представляются в виде лексем, изображающих фиксированные числовые, логические, символьные или строковые значения.

**Числовые** константы могут быть целыми, вещественными (с фиксированной или плавающей точкой) и перечислимыми.

Целые константы могут быть десятичными, восьмеричными и шестнадцатеричными. Десятичная целая константа определена как последовательность десятичных чисел, начинающаяся не с нуля, если это не число нуль. Восьмеричные константы в Visual Basic for Application начинаются с префикса &O и содержат числа от 0 до 7. Шестнадцатеричные числа начинаются с префикса &H и содержат числа от 0 до 9 и латинские буквы от A до F.

**Примеры** констант: 123, &O247, &H1F.

Вещественные константы записываются в десятичной системе счисления и в общем случае содержат целую часть (десятичная целая

константа), десятичную точку, дробную часть (десятичная целая константа), признак (символ) экспоненты E и показатель десятичной степени (десятичная целая константа, возможно со знаком).

**Примеры** констант: 123.456, 3.402823E38.

Перечислимые константы – это набор обычных целочисленных констант. Перечисляемый набор может содержать конечный набор уникальных целых значений, каждое из которых имеет особый смысл в текущем контексте. Перечисляемые наборы являются удобным инструментом, обеспечивающим выбор из ограниченного набора параметров. Например, если пользователь должен выбрать цвет из списка, то можно установить соответствие: черный = 0, белый = 1 и т.д.

Логические (булевы) константы могут иметь лишь одно из двух значений: **да** (истина, **TRUE**), **нет** (ложь, **FALSE**).

Символьные и строковые константы. В отличие от большинства языков программирования, где существуют отдельно символьные (содержащие один символ алфавита) и строковые (массив символов) константы, в VBA существуют только строковые, имеющие два типа значений:

- Строки переменной длины, которые могут содержать до приблизительно 2 миллиардов (2^31) символов.
- Строки постоянной длины, которые могут содержать от 1 до приблизительно 64K (2^16) символов.

**Примеры** строковых констант: "abcde", "информатика", "" (пустая строка).

**Типы данных**

Тип данных определяет, каким образом биты данных, представляющие конкретное значение, хранятся в памяти ПК. В каждом языке программирования имеется свой фиксированный набор базовых типов данных. В VBA имеются следующие типы данных (таблица 1).

Таблица 1 – Типы данных

| Тип данных | Размер | Диапазон значений |
|---|---|---|
| **Byte** (байт) | 1 байт | От 0 до 255. |
| **Boolean** (логический) | 2 байт | True или False. |
| **Integer** (целое) | 2 байт | От –32 768 до 32 767 |
| **Long** (длинное целое) | 4 байт | От –2 147 483 648 до 2 147 483 647 |
| **Single** (с плавающей точкой обычной точности) | 4 байт | От –3,402823E38 до –1,401298E-45 для отрицательных значений; от 1,401298E-45 до 3,402823E38 для положительных значений |
| **Double** (с плавающей точкой двойной точности) | 8 байт | От –1,79769313486232E308 до –4,94065645841247E-324 для отрицательных значений; от 4,94065645841247E-324 до 1,79769313486232E308 для положительных значений |
| **Currency** (денежный) | 8 байт | От –922 337 203 685 477,5808 до 922 337 203 685 477,5807 |
| **Decimal** (масштабируемое целое) | 14 байт | +/-79 228 162 514 264 337 593 543 950 335 без дробной части; +/-7,9228162514264337593543950335 с 28 знаками справа от запятой; минимальное ненулевое значение имеет вид +/-0,0000000000000000000000000001 |
| **Date** (даты и время) | 8 байт | От 1 января 100 г. до 31 декабря 9999 г. |
| **Object** (объект) | 4 байт | Любой указатель объекта |
| **String** (строка переменной длины) | 10 байт + длина строки | От 0 до приблизительно 2 миллиардов |
| **String** (строка постоянной длины) | Длина строки | От 1 до приблизительно 65 400 |
| **Variant** (числовые подтипы) | 16 байт | Любое числовое значение вплоть до границ диапазона для типа Double |
| **Variant** (строковые подтипы) | 22 байт + длина строки | Как для строки (String) переменной длины |
| Тип данных, определяемый пользователем (с помощью слова **Type**) | Объем определяется элементами | Диапазон каждого элемента определяется его типом данных |

**Некоторые характерные для VBA типы данных**

**Byte –** Массивы данного типа служат для хранения двоичных данных, например, изображений. Использование данного типа предохраняет двоичные данные во время преобразования формата.

**Boolean–** для хранения логических (булевых) значений. По умолчанию значением булевской переменной является **False**– ложь.

**Currency–** для хранения чисел с дробной частью до четырех цифр и целой частью до 15 цифр, то есть данных с фиксированной десятичной точкой, удобных для денежных вычислений. Числа с плавающей десятичной точкой (Single, Double) имеют больший диапазон значений, но могут приводить к ошибкам округления.

**Date** – используется для хранения как даты, так и времени в виде чисел с плавающей точкой. Дата может находиться в диапазоне от 1 января 100 года до 31 декабря 9999 года, а время в интервале от 0:00:00 до 23:59:59. Значения даты могут быть представлены в любом распознаваемом формате и должны ограничиваться знаками "#". Например: `#01/01/2011#`, `#01-01-2011#`.

**Decimal–** в версии 5.0 поддерживается использование типа данных Decimalтолько в пределах типа Variant, т.е. невозможно описать переменную с типом Decimal. Пользователь, однако, имеет возможность создать переменную типа Variantс подтипом Decimalс помощью функции CDec.

**Object–** поскольку VBA является объектно-ориентированным языком, в нем можно манипулировать различными объектами, адрес расположения которых в памяти (указатели) имеют этот тип.

**String–** по умолчанию данные строкового типа имеют переменную длину и могут удлиняться или укорачиваться. Однако такие строки занимают на 10байт памяти больше, поэтому можно объявить строки фиксированной длины, явно указав количество символов. Если количество символов будет

меньше объявленного, то свободные места заполняются пробелами, при попытке занесения большего количества символов лишние отбрасываются.

**Variant–** может быть использован для хранения данных всех базовых типов без выполнения преобразования (приведения) типов. Применение данного типа позволяет выполнять операции, не обращая внимания на тип данных, которые они содержат. Удобен для объявления переменных, тип которых заранее неизвестен. Переменные этого типа могут содержать специальные значения: Empty, Null, Error.

**Идентификаторы, переменные, массивы**

**Имена** (идентификаторы) –употребляются для обозначения объектов программы (переменных, массивов, процедур и др.). В VBA имена констант, переменных и процедур должны удовлетворять следующим требованиям:

- должны начинаться с буквы;
- не могут содержать точки и символов объявления типа;
- не могут быть длиннее 255 символов. Длина имен объектов не должна превышать 40 символов.
- не могут быть ключевыми словами (именами операций, операторов, встроенных функций).

**Переменные** представляют собой зарезервированное место в памяти ПК для хранения значения. Переменные обозначаются именами – словами, используемыми для ссылки на значение, которое содержит переменная, и характеризуются типом, определяющим вид данных, которые можно хранить в переменной. Переменные могут изменять свои значения в ходе выполнения программы. Объявить переменную – значит заранее сообщить программе о ее существовании. Объявление переменной производится специальным оператором. Одновременно с объявлением переменной после ее имени можно записать ключевое слово **As**, после которого задается тип переменной.

**Операции, выражения, операторы**

**Операции**. В VBA существуют следующие типы операций:

- **арифметические** операции, используемые для выполнения математических вычислений: **^, *, /, \, Mod, +, -.** Здесь \ - Возвращает результат целого деления двух чисел, **Mod–**возвращает остаток при целом делении двух чисел (значение по модулю).
- операции **сравнения**, используемые для выполнения операций сравнения
- **< , > , <=, >= , = , <> ;**
- **логические** операции, используемые для выполнения логических операций

**And –** Возвращает результат конъюнкции (логического И) для двух выражений с операциями сравнения, либо выполняет поразрядное сравнение двух числовых выражений (таблица 2).

Таблица 2 – Результат операции And

| | | |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

**Eqv –** Используется для проверки логической эквивалентности двух выражений с операциями сравнения, либо выполняет поразрядное сравнение двух числовых выражений (таблица 3).

Таблица 3 – Результат операции Eqv

| | | |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

**Imp –** Выполняет операцию логической импликации для двух выражений с операциями сравнения, либо выполняет поразрядное сравнение двух числовых выражений (таблица 4).

Таблица 4 – Результат операции Imp

| | | |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

**Not –** Выполняет над выражением операцию логического отрицания, а также поразрядное изменение значений каждого разряда переменной (таблица 5).

Таблица 5 – Результат операции Not

| | |
|---|---|
| 0 | 1 |
| 1 | 0 |

**Or –** Выполняет операцию логического ИЛИ (сложения) для двух выражений (таблица 6).

Таблица 6 – Результат операции Or

| | | |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

**Xor –** Выполняет операцию исключающего ИЛИ для двух выражений (таблица 7).

- операция **конкатенации** символьных значений друг с другом с образованием одной длинной строки:

**&–**используется для слияния двух строковых выражений.

Таблица 7 – Результат операции Xor

| | | |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

**Выражения– предназначаются для выполнения необходимых вычислений**, состоят из констант, переменных, функций (например, exp(*x*)), объединенных знаками операций.

Выражения записываются в виде **линейных последовательностей символов** (без подстрочных и надстрочных символов и т.д.), что позволяет вводить их в компьютер, последовательно нажимая на соответствующие клавиши клавиатуры.

Различают выражения **арифметические, логические и строковые.**

- **Арифметические выражения служат для определения одного числового значения**. Например, (1+sin(*x*))/2. Значение этого выражения при *x*=0 равно 0.5, а при *x*=π/2 – единице.

- **Логические выражения описывают некоторые условия, которые могут удовлетворяться или не удовлетворяться**. Таким образом, логическое выражение может принимать только два значения –**"истина"** или "**ложь"** (**да** или **нет**). Рассмотрим в качестве примера логическое выражение *x*x + y*y<r*r*, определяющее принадлежность точки с координатами (*x,y*) внутренней области круга радиусом r с центром в начале координат. При *x*=1, *y*=1, *r*=2 значение этого выражения –**"истина"**, а при *x*=2, *y*=2, *r*=1 –**"ложь"**.

- **Значения строковых выражений – тексты**. В них могут входить литерные константы, литерные переменные и литерные функции, разделенные знаком операции сцепки. Например, A & B означает присоединение строки B к концу строки A. Если A = *"куст"*, а B = *"зеленый"*, то значение выражения A&B есть "*куст зеленый"*.

**Операторы** (команды). Оператор – это наиболее крупное и содержательное понятие языка: **каждый оператор представляет собой**

**законченную фразу языка и определяет некоторый вполне законченный этап обработки данных.** В состав операторов входят:

- ключевые слова;
- данные;
- выражения и т.д.

Операторы подразделяются на исполняемые и неисполняемые. **Неисполняемые** операторы предназначены для описания данных и структуры программы, а **исполняемые** – для выполнения различных действий (например, оператор присваивания, операторы ввода и вывода, условный оператор, операторы цикла, оператор процедуры и др.).

**Операторы описания**

Объявление переменной производится одним из операторов **Dim**, **Static, Private, Public,** за которым следует имя переменной и необязательная часть с ключевым словом **As**, после которого задается тип переменной, например **Dim *name* [As type]**. Оператор **Public**используется только вне модуля, в его общей части и делает описываемую переменную доступной из всех процедур всех модулей проекта. Оператор **Private** служит для объявления переменной уровня модуля, доступной только процедурам данного модуля. Можно использовать также оператор **Dim**, но применение **Private** предпочтительнее как противоположное **Public**.

Переменные могут быть объявлены внутри процедуры операторами **Dim** или **Static**. Такие переменные называют также локальными, поскольку доступны только в той процедуре, в которой они объявлены. Данное свойство (область видимости) позволяет использовать одинаковые имена переменных в разных процедурах, не опасаясь конфликтов или случайных изменений значений переменных. Время жизни локальных переменных, объявленных с помощью оператора **Dim** равно времени работы процедуры и по ее окончании значения таких переменных теряются.

Переменные, объявленные с помощью оператора **Static**, сохраняют свои значения в течение всего времени выполнения приложения. При повторном входе в процедуру, где описана такая переменная, ее значение сохраняется.

Операторы **Public** и **Private** можно применять при описании констант и процедур, что позволяет указать их область видимости.

**Операторы присваивания**

Инструкция **Let**присваивает значение выражения переменной или свойству:

**[Let] имяПеременной = выражение**

Явное использование ключевого слова **Let**необязательно.

Значение выражения может быть присвоено переменной, только если оно имеет совместимый с этой переменной тип данных. Невозможно присвоить строковое выражение числовой переменной или числовое выражение строковой переменной. Такая попытка приведет к ошибке во время компиляции.

Для ввода значений переменных в программу применяют функцию **InputBox(сообщение[, заголовок] [, значение по умолчанию] [, координата x] [, координата y])**. Эта функция отображает диалоговое окно, содержащее окно ввода, кнопки ОК и Отмена, сообщение (подсказку для ввода) и (необязательно) заголовок окна, значение, вводимое по умолчанию, координаты окна по горизонтали и вертикали в твиках. Заметим, что функция **InputBox** всегда (даже при нажатии кнопки Отмена) возвращает значение строкового типа, поэтому вызов ее должен иметь вид:

**name = InputBox("Введи адрес ячейки", "Ввод", "a1", 100, 200)**

Для отображения значений переменных в режиме нормальной работы необходимо ввести в тело программы вызов функции **MsgBox (сообщение, [кнопки, заголовок]).** Эта функция отображает диалоговое окно, содержащее сообщение длиной до 1024 символов, в которое с помощью

операции конкатенации можно включить значение переменных, а также (необязательно) кнопки для реакции на отображения окна (по умолчанию только кнопка ОК) и заголовок окна (строковое выражение). Пример:

**MsgBox "Значение val=" &val**

Заметим, что VBA не имеет встроенных функций ввода/вывода в документ. Поэтому для вывода значений выражений и переменных в активный документ приходится создавать пользовательские процедуры.

**Функции обработки строк**

В VBAимеется несколько функций по работе со строковыми выражениями.

Функции **Asc(строка)** и **Chr(код)** позволяют получить ASCII-код начальной буквы строки и, наоборот, по ASCII-коду получить соответствующую букву. Для функции **Chr()** код может принимать значения от 0 до 255. Значения от 0 до 31 соответствуют управляющим кодам (например, **Chr(13)** вернёт символ перехода на новую строку). Для обозначения некоторых часто употребляемых клавиш в VBA имеются встроенные константы, например, для клавиши <**Enter**> – **vbCr**, для клавиши <**Tab**> – **vbTab**, для клавиши <**Backspace**> – **vbBack**.

Примеры:

```
debug.PrintAsc("F")
70
debug.PrintChr(97)
a
Msgbox "Этот текст расположен"&vbCr& "в две строки".
```

Функция **Len(строка)** определяет длину строки.

Пример:

```
debug.PrintLen("http://www.tusur.ru/")
20
```

Функции **Left(строка, количество)** и **Right(строка, количество)** возвращают подстроки, состоящие из заданного количества соответственно первых и последних символов данной строки. Функция **Mid(строка, позиция, количество)** возвращает подстроку, содержащую заданное количество символов, начиная с указанной позиции.

Пример:

```
debug.Print Left("http://www.tusur.ru",4)
http
debug.Print Right("http://www.tusur.ru",2)
ru
debug.Print Mid("http://www.tusur.ru",12,5)
tusur
```

Функции **InStr([старт], строка1, строка2, [сравнение])** и **InStrRev(строка1, сторка2, [старт], [сравнение])** возвращают позицию первого вхождения строки2 в строку1, начиная соответственно с ее начала и с конца. Если вхождения нет, то возвращается 0. Параметр сравнение указывает способ сравнения строк, значения: 0 или **vbBinaryCompare** для двоичного сравнения, 1 или **vbTextCompare**для посимвольного сравнения без учета регистра.

```
debug.PrintInStr(1, "Microsoft Office", "office", vbTextCompare)
 11
debug.PrintInStr(1, "Microsoft Office", "office", vbBinaryCompare)
0
```

**Функции обработки даты и времени**

В VBAимеется несколько функций даты и времени, которые позволяют производить самые разнообразные действия от определения текущей даты до сложения нескольких дат.

Функции **Date**, **Time** и **Now** возвращают значения типа **Variant (Date)**, содержащие соответственно текущие дату, время и одновременно дату и время.

Пример:

```
debug.Print Date
20.10.2011
debug.Print Time
13:18:02
debug.PrintNow
20.10.2011 13:18:02
```

Функции **Hour(время)**, **Minute(время)**, **Second(время)** и **Day(дата)**, **Month(дата)**, **Year(дата)** возвращают значения типа **Variant (Integer)**, являющиеся целыми числами, которые представляют собой соответственно час, минуту, секунду и день, месяц, год в значении даты.

Пример:

```
debug.Print Hour(Now), Minute(Now), Second(Now)
13            18            55
debug.Print Day(Now), Month(Now), Year(Now)
20            10            2011
```

Функция **DatePart(интервал, дата [, день_недели, неделя_года]** возвращает значение типа **Variant (Integer)**, содержащее указанную часть даты. Параметр интервал – строковое значение, обозначающее какой временной интервал должен быть найден, допустимые значения – "yyyy"(год), "q" (квартал), "m" (месяц), "y" (день года), "d" (день месяца), "w" (день недели), "ww" (неделя), "h" (час), "m" (минута), "s" (секунда); параметр дата – дата, часть которой необходимо найти; параметры день_недели и неделя_года – необязательные параметры, указывающие первый день недели и первую неделю года.

Пример:

```
debug.PrintDatePart("w", #01-01-2011#)
```

```
7
```

Функция **DateDiff(интервал, дата1, дата2 [, день_недели, неделя_года]** возвращает значение типа **Variant (Long)**, содержащее временной интервал между двумя датами. Параметр интервал – строковое значение, обозначающее какой временной интервал необходимо различать, допустимые значения – "yyyy" (год), "q" (квартал), "m" (месяц), "y" (день года), "d" (день месяца), "w" (день недели), "ww" (неделя), "h" (час), "m" (минута), "s" (секунда); параметры дата1 и дата2 – две даты, разность между которыми необходимо найти; параметры день_недели и неделя_года – необязательные параметры, указывающие первый день недели и первую неделю года.

Пример:

```
debug.PrintDateDiff("yyyy", #01-10-2000#, #01-09-2011#)
11
```

Функция **DateAdd(интервал, количество, дата)** возвращает значение типа **Variant (Date)**, содержащее дату, которой добавлено указанное количество времени. Параметр интервал – строковое значение, обозначающее какой временной интервал необходимо добавить, допустимые значения – "yyyy"(год), "q" (квартал), "m" (месяц), "y" (день года), "d" (день месяца), "w" (день недели), "ww" (неделя), "h" (час), "m" (минута), "s" (секунда); параметр количество – число, обозначающее количество времени, которое необходимо добавить к параметру дата.

Пример:

```
debug.PrintDateAdd("yyyy", 5, #01-10-2000#)
10.01.2005
```

**Некоторые функции проверки типов**

Функции проверки типов необходимы, когда нужно определить, является ли значение переменной значением необходимого типа (таблица 8).

Таблица 8 – Функции проверки типов

| Функция | Проверяемый тип |
|---|---|
| IsArray | если переменная является массивом – возвращает True, иначе – False |
| IsDate | если переменная содержит дату – возвращает True, иначе – False |
| IsEmpty | была ли переменная инициализирована – возвращает True, иначе – False |
| IsNull | если переменная содержит какое-либо значение – возвращает True, иначе – False |
| IsNumeric | если переменная является числом – возвращает True, иначе – False |

**Функции преобразования типов**

Таблица 9 – Функции проверки типов

| Функция | Возвращаемый тип |
|---|---|
| `CBool` | `Boolean` |
| `CByte` | `Byte` |
| `CCur` | `Currency` |
| `CDate` | `Date` |
| `CDbl` | `Double` |
| `CDec` | `Decimal` |
| `CInt` | `Integer` |
| `CLng` | `Long` |
| `CSng` | `Single` |
| `CVar` | `Variant` |
| `CStr` | `String` |

**Форматирование значений разных типов**

Функция **Format(выражение [, формат, день_недели, неделя_года])** возвращает значение типа **Variant (String)**, содержащее выражение, отформатированное согласно заданному формату; параметры **день_недели** и **неделя_года** – необязательные параметры, указывающие первый день недели и первую неделю года.

В таблице 9 перечислены некоторые форматы для представления значений.

Таблица 10 – Форматы для представления значений

| Формат | Описание |
|---|---|
| 1 | 2 |
| General Number | Число без разделителя тысяч |
| Currency | Число в денежном формате, используя настройки операционной системы |
| Fixed | Число, у которого отображается хотя бы одна цифра слева и две справа от десятичного символа |
| Standard | Число, у которого отображается хотя бы одна цифра слева, две справа от десятичного символа и разделитель тысяч |
| Percent | Число в процентном формате с двумя цифрами справа от десятичного символа |
| Scientific | Число в формате с плавающей точкой |
| Yes / No | No, если число равно 0, Yes в противном случае |
| True / False | False, если число равно 0, True в противном случае |
| On / Off | Off, если число равно 0, On в противном случае |
| General Date | Отображает дату или время |
| Long Date | Дата в полном формате |
| Medium Date | Дата в обычном формате |

Продолжение таблицы 10

| 1 | 2 |
|---|---|
| Short Date | Дата в сокращенном формате |
| Long Time | Время с часами, минутами и секундами |
| Medium Time | Время с часами и минутами в 12-часовом формате |
| Short Time | Время с часами и минутами в 24-часовом формате |

Примеры:

```
debug.Print Format(456789.0123, "General Number")
456789,0123
debug.Print Format(456789.0123, "Currency")
456 789,01р.
debug.Print Format(456789.0123, "Fixed")
456789,01
debug.Print Format(456789.0123, "Standard")
456 789,01
debug.Print Format(456789.0123, "Scientific")
4,57E+05
debug.Print Format(#01-01-2011#, "General Date")
01.01.2011
debug.Print Format(#01-01-2011#, "Long Date")
1 Январь2011 г.
debug.Print Format(#01-01-2011#, "Medium Date")
01-янв-11
debug.Print Format(#01-01-2011#, "Short Date")
01.01.2011
debug.Print Format(#13:50:50#, "Long Time")
13:50:50
debug.Print Format(#13:50:50#, "Medium Time")
01:50
debug.Print Format(#13:50:50#, "Short Time")
```

```
13:50
```

Если же нужного формата нет, можно настроить вид отображения выводимого значения при помощи пользовательских форматов с использованием специальных символов "0", "#", "%", ",", ":", "/", ("E+", "E–", "e+", "e–"), ("d", "m", "s"), ("h", "m", "s").

Примеры:

```
debug.Print Format(125 / 2, "###.###")
62,5
debug.Print Format(125 / 2, "000.000")
062,500
debug.Print Format(125 / 2, "#.##e+##")
6,25e+1
debug.Print Format(#01-01-2011#, "dd/mm/yy")
01.01.11
debug.Print Format(#13:50:50#, "hh:mm:ss")
13:50:50
```

Для форматирования чисел в VBA имеется отдельная функция **FormatNumber(число [, число_знаков, ведущий_ноль, отрицательные, группировать])**, где **число** – то число, которое нужно отформатировать; **число_знаков** – параметр, задающий число знаков после десятичного символа (значения – vbTrue, vbFalse, vbUseDefault); **ведущий_ноль** – параметр, указывающий надо ли отображать нулевую целую часть (значения – vbTrue, vbFalse, vbUseDefault); **отрицательные** – параметр, указывающий, надо ли отображать отрицательные значения в скобках (значения – vbTrue, vbFalse, vbUseDefault); **группировать** – параметр, указывающий, надо ли группировать цифры (значения – vbTrue, vbFalse, vbUseDefault).

Примеры:

```
debug.PrintFormatNumber(sin(5), 4)
-0,9589
debug.PrintFormatNumber(sin(5), 4, vbFalse)
```

```
-,9589
debug.PrintFormatNumber(sin(5), 4, vbFalse, vbTrue)
(,9589)
```

Для форматирования процентов в VBA имеется отдельная функция **FormatPercent**, которая имеет такой же синтаксис, как и **FormatNumber**.

Для форматирования денежных значений в VBA имеется отдельная функция **FormatCurrency**, которая имеет такой же синтаксис, как и **FormatNumber**.

Для форматирования значений даты и времени в VBA имеется отдельная функция **FormatDateTime(дата[, формат])**, которая имеет такой же синтаксис, как и **FormatNumber**. где дата – параметр, задающий дату, которую необходимо отформатировать; формат – необязательный параметр, указывающий нужное форматирования (значения – vbGeneralDate, vbLongDate, vbShortDate, vbLongTime, vbShortTime).

**Отладка, использование среды для отладки программ**

Поскольку идеальных программистов не существует, в большинстве систем разработки программ имеются инструменты, с помощью которых можно решить проблемы, возникающие в процессе программирования. В VBA также есть средства, которые позволяют либо исключить ошибки при разработке, либо задать обработку ошибок при выполнении программ.

**Отладка программ**– это проверка и внесение исправлений в программу при её разработке. Отладка позволяет идентифицировать ошибки, допущенные при программировании (синтаксические – ошибки в выражениях и именах, и логические – в логике работы программы).

**Обработка ошибок** – это задание реакции на ошибки, которые возникают при выполнении программы. Их причиной могут быть как ошибки программиста, так и внешние факторы – отсутствие нужных файлов, отказы аппаратуры, неправильные действия пользователя.

**Типы ошибок**. Ошибки в программе делятся на три категории:

*Ошибки компиляции* – возникают, когда компилятор не может интерпретировать введённый текст. Некоторые ошибки компиляции обнаруживаются при вводе, а другие – перед выполнением программы. Такие ошибки легко определить и исправить, поскольку VBA выявляет их автоматически, а сами ошибки очевидны.

Примечание.VBA автоматически компилирует программу каждый раз при запуске на выполнение после внесения изменений. Можно также запустить компиляцию командой *Отладка –Компилировать*.

*Ошибки выполнения* – возникают при выполнении программы после успешной компиляции. Их причиной обычно является отсутствие данных или неправильная информация, введённая пользователем. Такие ошибки идентифицируются VBA с указанием инструкции, при выполнении которой произошла ошибка. Для исправления таких ошибок обычно приходится выводить значения переменных или другие данные, которые влияют на успешное выполнение программы.

*Логические ошибки*трудно заметить и устранить. Они не приводят к прекращению компиляции или выполнения, однако являются причиной того, что программа не выдаёт желаемых результатов. Выявление таких ошибок производят путём тщательной проверки с помощью средств отладки VBA.

**Средства отладки**. В VBA имеется большое количество средств, предназначенных для отладки программ. К ним относятся: использование оператора **OptionExplicit**, пошаговое выполнение программы, работа в режиме прерывания, использование точек останова, вывод значений переменных.

*Использование OptionExplicit*. Данный оператор описания требует явного задания переменных в программах. При его использовании возникает ошибка компиляции при неправильном написании имени переменной или использовании неописанной переменной. Кроме того, явное описание переменных позволяет обойтись без использования типа данных Variant и

связанных с его использованием ошибок при неявном приведении типов данных.

*Пошаговое выполнение программы*. Этот режим служит для локализации ошибок в теле программы. Для его запуска используются команды меню, клавиатура или панель инструментов *Отладка*, отображаемая командой *Вид* - *Панели инструментов* - *Отладка*. В меню и на панели инструментов имеются четыре команды (кнопки) для выполнения программы в пошаговом режиме.

- Команда *Отладка – Шаг с заходом* (клавиша <**F8**>) позволяет выполнить одну строку программы и перейти к следующей. Если следующая строка – вызов процедуры, то происходит переход к первому выполняемому оператору этой процедуры.

- Команда *Отладка – Шаг с обходом* (клавиши <**Shift+F8**>) также выполняет одну строку программы, но если строкой является вызов процедуры, то она выполняется как одна инструкция. Данная команда используется, если известно, что эта процедура работает правильно.

- Команда *Отладка – Шаг с выходом* (клавиши <**Ctrl+Shift+F8**>) заканчивает выполнение текущей процедуры и останавливается на следующей после вызова текущей процедуры инструкции в вызывающей подпрограмме.

- Команда *Отладка – Выполнить до текущей позиции* (клавиши <**Ctrl+F8**>) выполняет программу от текущей до выбранной инструкции. Перед выбором данной команды требуется установить курсор в окне модуля на требуемую позицию.

*Работа в режиме прерывания*. Переход в данный режим выполняется:

- При нажатии кнопки *Отладка* в окне сообщения об ошибке выполнения.

- При прерывании работы программы нажатием клавиш <**Ctrl+Break**>. Текущая строка программы выделяется в окне модуля.

- По достижении точки останова.

- По достижении оператора Stop.
- При пошаговом выполнении программы.

В режиме прерывания можно:

- Вывести значение переменной.
- Вычислить выражение в окне отладки.
- Сбросить программу
- Выполнить программу в пошаговом режиме.
- Продолжить выполнение программы.

Для выхода из режима прерывания используется команда *Запуск – Сброс*.

**Использование точек останова**. *Точка останова* – это строка в процедуре, на которой приостанавливается выполнение программы. Все команды, находящиеся выше точки останова, выполняются с обычной скоростью, а по достижении контрольной точки программа переходит в режим прерывания. Затем можно отлаживать процедуру в пошаговом режиме, либо использовать различные способы вывода значений переменных. Кроме того, имеется возможность остановить выполнение или сбросить процедуру командами меню *Запуск* или кнопками панели инструментов *Отладка*. В одном проекте можно задать несколько точек останова, причём в различных процедурах.

Чтобы установить или снять точку останова, используется команда *Отладка – Точка останова* или клавиша <**F9**>, либо кнопка *Точка останова* панели инструментов *Отладка*. Можно также установить или снять точку останова, щелкнув левой кнопкой мыши на полосе индикатора против требуемой строки. Точка останова отмечается коричневой жирной точкой на полосе индикатора, а сама строка выделяется коричневым цветом.

Чтобы быстро удалить все точки останова открытого проекта, используется команда *Отладка – Снять все точки останова* или комбинация клавиш <**Ctrl+Shift+F9**>. Для данной команды не предусмотрена кнопка на панели инструментов *Отладка*.

**Вывод значений переменных**. При наличии тестового примера вывод значений переменных позволяет сравнить ожидаемые и полученные значения переменных. Для отображения значений переменных в режиме прерывания необходимо:

- При установленном флажке *Подсказки значений переменных* в окне *Сервис –Параметры* достаточно переместить указатель мыши на требуемую переменную для отображения имени и значения переменной во всплывающей подсказке.
- Выбрать команду *Отладка –Контрольное значение* (нажать клавиши <**Shift+F9**>) для вывода диалогового окна *Контрольное значение*. При этом курсор должен находиться возле переменной, значение которой надо контролировать. В окне *Контрольное значение* отображается контекст (имя модуля и процедуры), выделенное выражение (переменная) и кнопки *Добавить* и *Отмена*. При нажатии кнопки *Добавить* откроется окно *Контрольные значения*, содержащее имена переменных (выражения), их значения, тип данных и контекст.
- Для добавления других контрольных значений используется команда *Отладка –Добавить контрольное значение*.
- Выбрать команду *Вид –Окно локальных переменных*. Откроется окно *Локальные переменные*, в котором в режиме прерывания отображаются имена, значения и типы всех переменных модуля.

Выбрать команду *Вид –Окно отладки*. В нем немедленно выполняется введённая в него инструкция, обычно операция отображения значения выражения вида Print имя, или операция присваивания значения переменной.

**Задания на лабораторную работу**

Составить программу, которая переводит одни единицы измерения в другие. Исходные данные вводятся с клавиатуры, результат выводится на экран с необходимым форматированием выводимых значений.

**Пример** Перевести Фаренгейты в градусы.

```
Sub Celsius()
Const Celsius2Fahrenheit = 1.8
  Dim c As Single, f As Single
f = CSng(InputBox("Введите количество градусов Фаренгейта"))
c = f / Celsius2Fahrenheit
MsgBox (FormatNumber(f, 1) & " градусовФаренгейтаравно " &FormatNumber(c, 1) & " градусовЦельсия")
End Sub
```

**Варианты заданий**

1. Метры в километры.
2. Дюймы в метры (1 дюйм = 2,54 см).
3. Километры в дециметры.
4. Килограммы в тонны.
5. Граммы в центнеры.
6. Квадратные метры в гектары.
7. Ары в квадратные километры (1 ар=100 $м^2$).
8. Кубические дециметры в кубические метры.
9. Кубические сантиметры в литры.
10. Фунты в килограммы (1 фунт = 0,45359 кг).

**Задания на самостоятельную работу**

Составить программу, присваивающую все вычисляемые значения строковой переменной через соответствующие функции. Переменную добавить в *Контрольное значение* и отслеживать её изменение. Необходимо вычислить количество следующих значений между датой своего рождения и текущей датой: количество дней, недель, месяцев, кварталов, лет. Выполнить программу в пошаговом режиме.

# Практическая работа №4

# Условные операторы и операторы циклов

**Цель работы**

Повторить основы программирования линейных и разветвляющихся вычислительных процессов.

**Теоретический материал, для освоения темы**

**Управляющие структуры VBA. If . . . Then, If . . . Then . . . Else, Select Case**

Управляющие структуры позволяют управлять последовательностью выполнения программы. Без операторов управления все операторы программы будут выполняться слева направо и сверху вниз. Однако иногда требуется многократно выполнять некоторый набор инструкций автоматически, либо решить задачу по-другому в зависимости от значения переменных или параметров, заданных пользователем во время выполнения. Для этого служат конструкции управления и циклы.

VBA поддерживает следующие конструкции принятия решений:

```
If . . . Then
If . . . Then . . . Else
Select Case
```

**Конструкция If . . . Then**

Конструкция**If . . . Then**применяется, когда необходимо выполнить один или группу операторов в зависимости от некоторого условия. Синтаксис этой конструкции позволяет задавать ее в одной строке или в нескольких строках программы:

```
IfусловиеThenвыражение
Ifусловие Then
выражение
```

```
EndIf
```

Обычно условие является простым сравнением, но оно может быть любым выражением с вычисляемым значением. Это значение интерпретируется как **False**(Ложь), если оно нулевое, а любое ненулевое рассматривается как **True** (Истина). Если условие истинно, то выполняются все выражения, стоящие после ключевого слова **Then**. Для условного выполнения одного оператора можно использовать как синтаксис для одной строки, так и синтаксис для нескольких строк (блоковую конструкцию).

Заметим, что синтаксис оператора **If . . . Then**для одной строки не использует оператор **EndIf**. Чтобы выполнить последовательность операторов, если условие истинно, следует использовать блоковую конструкцию **If . . . Then . . . EndIf**.

Если условие ложно, то операторы после ключевого слова **Then** не выполняется, а управление передается на следующую строку (или строку после оператора **EndIf** в блочной конструкции).

**Конструкция If . . . Then . . . Else** определяет несколько блоков операторов, один из которых будет выполняться в зависимости от условия:

```
If условие1 Then
выражение1
ElseIf условие2 Then
выражение2
. . .
Else
выражение-n
EndIf
```

При выполнении сначала проверяется **условие1**. Если оно ложно, VBAпроверяет следующее **условие2** и т. д., пока не найдет истинного условия. Найдя его, VBAвыполняет соответствующий блок операторов и затем передает управление инструкции, следующей за оператором **Endif.** В

данную конструкцию можно включить блок оператора **Else**, который VBAвыполняет, если не выполнено ни одно из условий.

Конструкция **If. . . Then. . . ElseIf**в действительности всего лишь специальный случай конструкции **If. . . Then. . . Else**. Заметим, что в данной конструкции может быть любое число блоков **ElseIf**, или даже ни одного. Блок **Else**можно включать независимо от присутствия или, наоборот, отсутствия блоков **ElseIf**.

Рассмотрим пример вычисления функции

```
PrivateSubUserForm_Initialize()
Dim q, w, e As String
If CheckBox2 = True Then
q = InputBox("Введитеназваниефирмы")
w = InputBox("Введитеназваниетура")
e = InputBox("Введите название страны")
End If
Range("A8") = q
Range("C8") = w
Range("E8") = e
EndSub
```

Заметим, что можно добавить любое число блоков **Elself**в конструкцию **If. . . Then**. Однако количество блоков **Elself**может стать настолько большим, что конструкция **If. . . Then**станет очень громоздкой и неудобной. В подобной ситуации следует применять другую конструкцию принятия решения – **SelectCase**.

**Конструкция SelectCase**

*Конструкция* **SelectCase**является альтернативой конструкции **If. . . Then. . . Else**в случае выполнения блока, состоящего из большого набора операторов. Конструкция **SelectCase**предоставляет возможность, похожую на возможность конструкции **If. . . Then. . . Else**, но в отличие от нее она делает код более читаемым при наличии нескольких вариантов выбора.

Конструкция **SelectCase** работает с единственным проверяемым выражением, которое вычисляется один раз при входе в эту конструкцию. Затем VBA сравнивает полученный результат со значениями, задаваемыми в операторах **Case** конструкции. Если найдено совпадение, выполняется блок операторов, ассоциированный с оператором **Case**:

```
SelectCaseпроверяемое_выражение
[Case список_выражений1
[блок_операторов1]]
[Case список_выражений2
[блок_операторов2]]
. . .
[Case Else
[блок_операторовn]]
End Select
```

Рассмотрим пример вычисления функции

```
Private Sub UserForm_Initialize()
Dim nR As Integer
Dim nC As Integer
Dim RC As String
nR = ActiveCell.Row
nC = ActiveCell.Column
RC = nR&nC
If Not Check(nR, nC) Then Exit Sub
ActiveCell = "0"
With WRK
  Select Case RC
    Case "23": .Cells(2, 2) = "X"
    Case "34": .Cells(2, 4) = "X"
    Case "32": .Cells(4, 2) = "X"
    Case "43": .Cells(4, 4) = "X"
```

```
        Case "22": .Cells(2, 4) = "X"
        Case "24": .Cells(4, 4) = "X"
        Case "42": .Cells(4, 4) = "X"
        Case "44": .Cells(4, 2) = "X"
      End Select
  End With
  End Sub
```

Обратим внимание на то, что здесь используется RCссылка, для удобства выбора диапазона ячеек, а также заметим, что конструкция **SelectCase**вычисляет выражение только один раз при входе в нее, а в конструкции **If. . . Then. . . Else**вычисляются различные выражения для каждого оператора **Elself**. Конструкцию **If. . . Then. . . Else**можно заменить конструкцией **SelectCase**, только если оператор **If**и каждый оператор **Elself**вычисляют одно и то же выражение.

**Операторы циклов. Вложенные циклы**

**Операторы циклов**

Циклы позволяют выполнить одну или несколько строк кода несколько раз. VBAподдерживает следующие циклы:

```
For...Next
ForEach...Next
Do... Loop
```

*Конструкция***For. . . Next.** Когда число повторений известно заранее, используют цикл **For. . . Next**. В цикле **For**используется переменная, называемая переменной цикла или счетчиком цикла, которая увеличивается или уменьшается на заданную величину при каждом повторении цикла. Синтаксис этой конструкции следующий:

```
For counter = start To end [Step increment]
операторы
Next [counter]
```

Параметрысounter (счетчик), start (начало цикла), end (конец цикла) иincrement (приращение) являются числовыми.

Примечание. Параметр increment может быть как положительным, так и отрицательным. Если он положителен, параметр startдолжен быть меньше или равен параметру end, иначе цикл не будет выполняться. Если параметр incrementотрицателен, то параметр startдолжен быть больше или равен значению параметра end, чтобы выполнялось тело цикла. Если параметр **Step** не задан, то значение параметра incrementпо умолчанию равно 1.

VBA выполняет цикл For в следующей последовательности:

1. Устанавливает значение переменной цикла counter в значение start.
2. Сравнивает значение переменной цикла counter и значение параметра end. Если переменная counterбольше, VBA завершает выполнение цикла. (Если значение параметра incrementотрицательно, то VBA прекращает выполнение цикла при условии, что значение переменной цикла countermеньше значения параметра end.)
3. Выполняет операторы тела цикла statements.
4. Увеличивает значение переменной цикла counterна 1 или на величину значения параметра increment, если он задан.
5. Повторяет шаги со 2 по 4.

Рассмотрим пример: Заполнение диапазона ячеек информацией по предметам и расчет стипендии, в зависимости от полученных оценок.

```
Private Sub UserForm_Initialize()
Dim A, F, Z, X As String
    Dim B As Single
    For i = 1 To 3
      A = "A" &i + 2
      Range([A]).Value = InputBox("Введите ФИО " & i)
    Next

    For i = 1 To 3
```

```
        A = "B" &i + 2
        Range([A]).Value = InputBox("Введите оценку по
Бухучету " & i)

    Next
    For i = 1 To 3
        A = "C" &i + 2
        Range([A]).Value = InputBox("Введите оценку по
ПЯВУ " & i)

    Next
    For i = 1 To 3
        A = "D" &i + 2
        Range([A]).Value = InputBox("Введите оценку по
информатике " & i)

Next

    For i = 3 To 5
        A = "B" &i
        Z = "C" &i
        X = "D" &i
        F = "E" &i

        If Range([A]).Value <= 3 Or Range([Z]).Value
<= 3 Or Range([X]).Value <= 3 Then Range([F]) = 0 Else
Range([F]) = 500
     Next

End Sub
```

**Конструкция Do...Loop**

Цикл **Do** применяется для выполнения блока операторов неограниченное число раз. Существует несколько разновидностей конструкции **Do. . . Loop**, но каждая из них вычисляет выражение-**условие**, чтобы определить момент выхода из цикла. Как и в случае конструкции **If. . . Then условие** должно быть величиной или выражением, принимающими значение **False**(нуль) или **True** (не нуль).

В следующей конструкции **Do. . . Loop**операторы выполняются до тех пор, пока значением условия является **True**(Истина):

```
DoWhile условие
операторы
Loop
```

Выполняя этот цикл, VBA сначала проверяет условие. Если условие ложно (**False**), он пропускает все операторы цикла. Если оно истинно (**True**), VBA выполняет операторы цикла, снова возвращается к оператору **DoWhile** и снова проверяет условие.

Следовательно, цикл, представленный данной конструкцией, может выполняться любое число раз, пока значением условия является не нуль или **True** (Истина). Отметим, что операторы тела цикла не выполняются ни разу, если при первой проверке условия оно оказывается ложным (**False**).

Другая разновидность конструкции **Do . . . Loop** сначала выполняет операторы тела цикла, а затем проверяет условие после каждого выполнения. Эта разновидность гарантирует, что операторы тела цикла выполнятся по крайней мере один раз:

```
Do
операторы
LoopWhile условие
```

Две другие разновидности конструкции цикла аналогичны предыдущим, за исключением того, что цикл выполняется, пока условие ложно (**False**):

Цикл не выполняется вообще или выполняется много раз:

```
DoUntil условие
операторы Loop
```

Цикл выполняется по крайней мере один раз:

```
Do
операторы
LoopUntil условие
```

**Вложенные циклы**

Можно помещать структуры управления внутрь других структур управления (например, блок **If . . . Then** внутрь цикла **For . . . Next**). Говорят, что структура управления, помещенная внутрь другой структуры управления, является вложенной.

При вводе/выводе элементов двумерного массива на рабочий лист Microsoft Excel удобно применять пользовательские процедуры ввода/вывода:

```
Sub readcell(i As Integer, j As Integer, val As Variant)
val = Лист1.Cells(i, j).Value
End Sub
Sub outcell(i As Integer, j As Integer, val As Variant)
Лист1.Cells(i, j).Value = val
EndSub
```

где i – номер строки, j – номер столбца рабочего листа.

**Задания на лабораторную работу**

**Задание 1**. Найти минимальный и максимальный элементы массива из 10 элементов, заполненного случайными значениями, и поменять их местами.

**Задание2**. Microsoft Excel. Составить таблицу начисления заработной платы работникам ООО «Воронья слободка».

| Ф.И.О. | Тарифный разряд | % выполнения плана | Тарифная ставка, руб. | Заработная плата с премией, руб. |
|---|---|---|---|---|
| Пряхин Н.П. | 3 | 102 | ? | ? |
| Суховейко А.Д. | 2 | 98 | ? | ? |
| Лоханкин В.А. | 1 | 114 | ? | ? |
| Пферд Л.Ф. | 1 | 100 | ? | ? |
| Севрюгов Л.А. | 3 | 100 | ? | ? |
| Гигиенишвили Г.С. | 2 | 94 | ? | ? |
| Птибурдуков А.И. | 3 | 100 | ? | ? |

Примечание 1. Тарифная ставка определяется в зависимости от разряда: 1-й разряд – 4000 руб.; 2-й разряд – 6500 руб.; 3-й разряд – 8000 руб. Тарифные ставки оформить отдельной таблицей.

Примечание 2. Размер премиальных определяется в зависимости от выполнения плана:

- ниже 100 % – премия не начисляется;
- 100 % – премия 20 % от тарифной ставки;
- 101…110 % – премия 30 %;
- 111…115 % – премия 40 %.

**Задание3**. Составить программу, переводящую числовое значение (до сотен включительно) в строковое. Например: 132 – "сто тридцать два".

**Задание на самостоятельную работу**

**Задание 1**. Изменить задание на лабораторную работу 2 следующим образом. Создаётся таблица, состоящая только из строки-заголовка. Необходимые исходные данные вводятся с клавиатуры до тех пор, пока Ф.И.О. будет не пустым.

**Задание 2**. Изменить программу, переводящую числовое значение в строковое, для обработки числовых значений до миллионов включительно.

# Практическая работа №5
# Использование управляющих элементов
# (панель элементов Visual Basic)

**Тема и цель работы**

Знакомство с управляющими элементами пользовательской формы.

**Теоретический материал, для освоения темы**

Панель элементов представляет собой окно, внутри которого находятся значки различных элементов, используемых в приложениях.

Чтобы работать с элементами в приложениях, программист должен:

- понимать, что такое свойства, события и методы соответствующего элемента;
- уметь использовать свойства, события и методы элемента.

*Свойства* – атрибуты объекта, которые изменяют внешний вид объекта и его поведение.

*События* – действие, распознаваемое объектом, для которого можно запрограммировать отклик.

*Метод* – команда, которую Вы отдаете объекту. При помощи методов можно приказать объекту выполнить те или иные действия, например, заставить выгрузиться из памяти форму.

Рассмотрим наиболее часто используемые элементы.

**Форма** – это визуальная основа приложений Visual Basic (рис. 14). Любое приложение, выводящее информацию на экран, построено на основе формы того или иного типа.

Чтобы создать новую форму, выполните команду **Insert–UserForm**. На экране появится форма, состоящая из нескольких компонентов.

**Граница** формы придает необходимую степень гибкости. Все эти возможности задаются при помощи свойств **BorderStyle**.

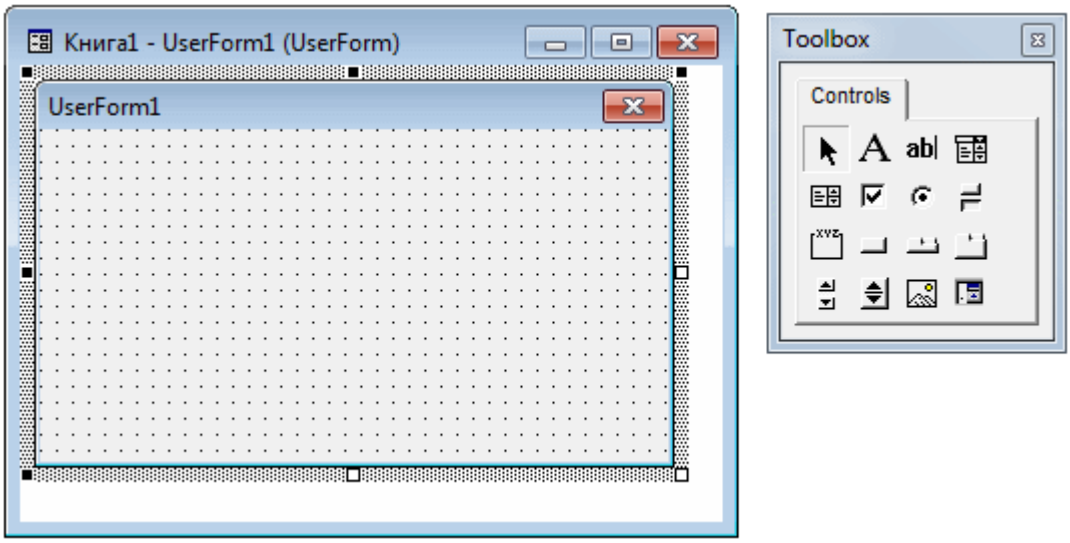



Рис. 14 ОбъектUserForm

**Название** – текст, выводимый в заголовке формы. В нем может содержаться имя приложения, краткое описание формы или информация о текущем состоянии.

Чтобы изменить название формы, следует присвоить нужный текст свойству **Caption** в окне свойств.

**Свойства формы**

Свойство **BackColor** определяет цвет фона для формы.

Свойство **BorderStyle** определяет особенности границы, окружающей форму (см. таблицу 11).

Таблица 11 – Свойство BorderStyle

| Значение | Описание |
|---|---|
| 0 – None | Позволяет оформлять вид без границ |
| 1 – Fixed Single | Позволяет оформлять вид одинарной границей |

Свойство**Caption** определяет текст, который выводится в заголовке формы.

Свойство**ForeColor**определяет цвет текста, выводимого на форме.

Свойство**Height**определяет высоту формы.

СвойствоWidthопределяет ширину формы.

СвойствоLeftопределяет расстояние формы от левого края экрана.

СвойствоTopопределяет расстояние формы от верхнего края экрана

Данные значения задаются в твипах (twips). Соотношение размеров твипа и пиксела изменяется в зависимости от разрешения экрана, и измеряется при помощи переменной Screen.TwipsPerPixelX и Screen.TwipsPerPixelY для горизонтальных и вертикальных размеров соответственно.

СвойствоNameопределяет имя объекта, используемое в дальнейшем процедурой.

**События форм**

СобытиеActivate активизирует форму. Активизация формы производится после ее инициализации. С событиемActivate тесно связаны события Initialize, Load, GotFocus.

Между этими событиями существуют достаточно тонкие отличия, причем главное из них – порядок, в котором события возникают в приложении. Этот порядок выглядит так:

- **Initialize**. Событие происходит во время конфигурации и до загрузки формы.
- **Load**. Событие происходит после инициализации формы, но до ее отображения на экране. Добавляя код в процедуру события Load, Вы можете настроить внешний вид или поведение формы.
- **Activate**. Событие происходит после загрузки формы в память, но до того, как форма станет активной.
- **GotFocus**. Это событие, если оно происходит, возникает при получении фокуса формой – когда форма загружается или когда пользователь обращается к ней, щелкая мышью.

После открытия формы из перечисленных событий могут произойти только GotFocus или Activate – хотя в отдельных, очень специфических случаях может произойти и событие Initialize.

Событие **Initialize** происходит, когда Visual Basic впервые узнает о существовании формы. В режиме выполнения это происходит сразу же после команды **Run–Start**. За ним следует событие **Load** – оно происходит, когда Visual Basic загружает форму. После загрузки формы и передачи ей фокуса (другими словами, при активизации формы) происходит событие **Activate**. Через считанные миллисекунды после него происходит событие **GotFocus**. Тем не менее, событие **GotFocus** может произойти лишь в том случае, если на форме нет ни одного элемента. Если же на форме есть видимый элемент, то фокус получает он, а событие **GotFocus** формы будет пропущено – вместо него будет вызвано событие **GotFocus** элемента.

Следовательно, в нормальной ситуации при запуске приложения будет вызвано событие **Load** первой отображаемой формы, за которым последует событие **Activate**. Разумеется, приложение может иметь и другие окна. Когда пользователь или программа повторно переключится в первое окно, снова будет вызвано событие **Activate**, но на этот раз без **Load**. Впрочем, событие **Load** может быть повторно вызвано, если форма была выгружена во время выполнения программы.

Событие **Deactivate**по смыслу противоположно **Activate**. Оно происходит в том случае, если форма перестает быть активной, т.е., когда фокус передается другой форме или приложению. В зависимости от выбранной цветовой схемы Windows цвет заголовка формы может измениться.

Событие**Unload**по смыслу противоположно **Load**. Чаще всего процедура этого события используется для того, чтобы спросить у пользователя, действительно ли он желает закрыть форму. Если Вы посмотрите на процедуру **Unload** в окне программы, то увидите, что она немного отличается от процедур других событий. За именем процедуры

следует аргумент (CancelAsInteger), с его помощью можно отменить выгрузку формы.

Событие**Resize** происходит, когда пользователь изменяет размеры формы. Чаще всего оно применяется в двух случаях:

- для масштабирования управляющих элементов, размещенных на форме;
- для восстановления исходных размеров формы.

В обоих случаях используются свойства Height и Width.

**Кнопки** (CommandButton) являются самым распространенным элементом любого приложения (таблица 12).

Таблица 12 – Свойства CommandButton

| Свойство | Описание |
|---|---|
| 1 | 2 |
| Cancel | Если данное свойство имеет значение True, нажатие клавиши Esc приводит к закрытию формы. Если на форме несколько кнопок, то лишь у одной кнопки свойство Cancel может иметь значение True |
| Caption | Определяет текст, который выводится на кнопке. |
| Default | Если данному свойству задать значение True, пользователь сможет нажать кнопку при помощи клавиши Enter. Если на форме несколько кнопок, то лишь у одной кнопки свойство Default может иметь значение True |
| Enable | Если установить для данного свойства значение False, кнопка станет недоступной. Кнопка остается на экране |
| Name | Имя кнопки, по которому в программе на VisualBasic отличается одна кнопка от других |
| Picture | Если вы хотите, чтобы в нормальном, не нажатом состоянии на кнопке присутствовал рисунок, задайте имя графического файла в данном свойстве |
| Style | Выбирая значения данного свойства можно оставить кнопку чисто текстовой или поместить на нее рисунок |
| TabIndex | Определяет порядок перебора элементов клавишей Tab. Если данное свойство определено значением 0, то соответствующий элемент получает фокус первым (при условии, что он видим и не заблокирован). При изменении значения данного свойства у одного элемента, изменяется порядок перебора и у других элементов |

Продолжение таблицы 12

| 1 | 2 |
|---|---|
| TabStop | Если определить данное свойство значением False, пользователь не сможет перейти к соответствующему элементу клавишей Tab. Но мышью элемент активизируется (при условии, что он видим и не заблокирован) |
| Visible | Если установить для данного свойства значение False, кнопка станет недоступной. Кнопка исчезает с экрана |

**События** кнопок рассмотрим на примере события Click – реакция кнопки на нажатие.

```
Private Sub CommandButton1_Click()
UserForm1.Hide
EndSub
```

Даная процедура позволяет при нажатии на кнопку скрывать пользовательскую форму.

**Методы** кнопок рассмотрим на примере метода SetFocus, который используется для передачи фокуса конкретной кнопке.

```
Private Sub UserForm_Initialize()
CommandButton1.SetFocus
EndSub
```

Данная процедура позволяет сделать активной кнопку CommandButton при инициализации пользовательской формы UserForm.

**Текстовые поля**(TextBox) обычно применяются для ввода данных или для получения информации от пользователя (таблица 13).

Таблица 13 – Свойства TextBox

| Свойство | Описание |
|---|---|
| 1 | 2 |
| Locked | Если данное свойство определить значением True, то текстовое поле будет доступно только для вывода информации |
| MaxLength | Ограничивает длину вводимого текста заданным количеством символов |
| MultiLine | Позволяет вывести текст в несколько строк |

Продолжение таблицы 13

| 1 | 2 |
|---|---|
| Name | Программное имя поля. Рекомендуется использование префикса имени txt |
| PasswordChar | Задает символ, вводимый в данное поле (наиболее распространен символ *) |
| ScrollBar | Позволяет выполнять прокрутку информации в поле |
| SelLength | Количество символов, которые будут выделены. Доступно только в программном коде |
| SelStart | Порядковый номер символа, после которого будет мигать курсор. Доступно только в программном коде |
| SelText | Автоматически заменяет выделенный текст. SelText=«NewText». Доступно только в программном коде |
| TabIndex | Определяет порядок перебора текстовых полей формы |
| Text | Содержимое поля |

**События** текстового поля рассмотрим на примере события Change. Оно происходит каждый раз, когда пользователь вставляет, заменяет или удаляет символы текстового поля.

```
Private Sub TextBox1_Change()
TextBox1.Text = "s"
EndSub
```

В данной процедуре при изменениях в текстовом поле TextBox1 будет отображаться символ S.

**Методы** текстовых полей рассмотрим на примере метода SetFocus– передать фокус текстовому полю.

```
Private Sub UserForm_Initialize()
TextBox1.SetFocus
EndSub
```

**Надписи** (Label) используется для вывода текста, но пользователь не может по своему усмотрению изменить текст надписи (таблица 14).

**Переключатели** (OptionButton) позволяют выбрать один вариант из группы. Обычно переключатели группируются в рамках (Frame).

В **событиях** необходимо обратить внимание на событие Click.

Таблица 14 – Свойства Label

| Свойство | Описание |
|---|---|
| BackColor | Цвет фона для поля надписи |
| BorderStyle | Используется совместно с BackColor. Если назначить данному свойству значение = 1, а BackColoropределить белым цветом, то надпись будет выглядеть как текстовое поле, но останется доступной только для чтения |
| Caption | Определяет текст, который содержится в надписи |
| Name | Определяет программное имя надписи. Рекомендуется префикс имени lbl |
| TabIndex | Определяет порядок перебора элементов |
| UseMnemonic | Если необходимо в тексте надписи прописать символ &, то данному свойству определяют значение False |
| WordWrap | Позволяет определить динамическое изменение размеров надписи при изменении ее содержимого |
| Свойство | Описание |
| Caption | Текст, выводимый на поле переключателя |
| Name | Программное имя переключателя |
| Value | Если установлено значение True в режиме конструирования, то данный переключатель активен после запуска программы |

**Список** (ListBox) применяется для хранения списка значений. Из списка пользователь может выбрать одно или несколько значений, которые впоследствии будут использоваться в тексте программы (таблицы 15, 16).

Таблица 15 – Основные свойства ListBox

| Свойство | Описание |
|---|---|
| 1 | 2 |
| Enabled | Допустимые значения: True (запрещен выбор значения из списка пользователем) и False (в противном случае) |
| Text | Возвращает выбранный в списке элемент |
| List | Возвращает элемент списка, стоящий на пересечении указанных строки и столбца. List (row, column) |
| MultiSelect | Устанавливает способ выбора элементов списка. Допустимые значения:<br>fmMultiSelectSingle (выбор только одного элемента)<br>fmMultiSelectMulti (разрешен выбор нескольких элементов посредством либо щелчка, либо нажатием клавиши <Пробел>)<br>fmMultiSelectExtended (разрешено использование клавиши <Shift> при выборе ряда элементов списка) |

Продолжение таблицы 15

| 1 | 2 |
|---|---|
| Selected | Используется для определения выделенного текста, когда свойство MultiSelect имеет значение fmMultiSelectMulti или fmMultiSelectExtended. Допустимые значения False, True |
| ColumnCount | Устанавливает число столбцов в списке |
| ListCount | Возвращает число элементов списка |
| ListIndex | Возвращает номер текущего элемента списка. Нумерация элементов начинается с 0 |

Таблица 16 – Методы ListBox

| Метод | Описание |
|---|---|
| Clear | Удаляет все элементы из списка |
| RemoveItem | Удаляет из списка элемент с указанным номером<br>RemoveItem(index) |
| AddItem | Добавляет элемент в список<br>AddItem([ item[, varIndex]])<br>Item– элемент (строковое выражение), добавляемое в список<br>varIndex– номер добавляемого элемента |

**Заполнение списка**

Заполнить список можно одним из следующих способов (таблица 17).

Таблица 17 – Способы заполнения списков

| Способ заполнения | Описание |
|---|---|
| 1 | 2 |
| Поэлементно (список состоит из одной колонки) | `ListBox1.AddItem "Hello"`<br>`ListBox1.AddItem "Bye"`<br>`ListBox1.ListIndex=0` |
| Массивом, если список состоит из одной колонки | `ListBox1.List=Array ("Hello", "Bye")`<br>`ListBox1.ListIndex=1` |
| Поэлементно, если список состоит из нескольких колонок | `ListBox1.ColumnCount =2`<br>`ListBox1.AddItem "Hello"`<br>`ListBox1.List(0,1)= "Gane"`<br>`ListBox1.AddItem "Hello"`<br>`ListBox1.List(1,1)= "Lisa"`<br>`ListBox1.AddItem "Hello"`<br>`ListBox1.List(2,1)= "Dim"` |

Продолжение таблицы 17

| 1 | 2 |
|---|---|
| Массивом, если список состоит из нескольких колонок | Dim A(2,1) As String<br>A(0,0)= "Hello"<br>A(0,1)= "Gane"<br>A(1,0)= "Hello"<br>A(1,1)= "Lisa"<br>A(2,0)= "Hello"<br>A(2,1)= "Dim"<br>ListBox1.ColumnCount=2<br>ListBox1.List=A |

**Выбор нескольких элементов из списка**

Вычисление среднего значения выбранных в списке элементов:

```
ListBox1.List= Array (1,2,3,4,5,6,7,8)
ListBox1.ListIndex=0
ListBox1.MultiSelect= fmMultiSelectMulti
Среднее=0
N=0
For i=0 to ListBox1.ListCount-1
If ListBox1.Selected(i) Then
N=N+1
Среднее=Среднее+ ListBox1.List(i)
Endif
Next
Среднее= Среднее/N
```

**Переключатель.Флажок** (CheckBox)

Элемент управления (OptionButton) позволяет выбрать из нескольких взаимоисключающих параметров или действий. Переключатели типа CheckBox обычно изображаются группами, обеспечивая возможность выбора альтернативного варианта (таблица 18).

Таблица 18 – Свойства ChekBox

| Свойство | Описание |
|---|---|
| Value | Возвращает True, если переключатель выбран и False в противном случае |
| Enabled | True – пользователь может выбрать переключатель, False – в противном случае |
| Visible | True – переключатель отображается во время выполнения программы, False – в противном случае |
| Capture | Надпись, отображаемая рядом с переключателем |

**Комбинированные поля** (Combobox) сочетают возможности текстового поля и списка. Комбинированное поле позволяет выбрать из списка заранее определенную строку или ввести значение, которого нет в списке. Существует три разновидности комбинированных полей, выбираемые в режиме конструирования: раскрывающиеся комбинированные поля, простые комбинированные поля и раскрывающиеся списки.

По свойствам, событиям и методам комбинированные поля очень похожи на списки. Тем не менее, свойство Text в комбинированных полях работает несколько иначе. Если для списков свойство Text при выполнении программы может лишь вернуть текст текущей выделенной строки, то для комбинированных полей значение этого свойства можно задавать и во время выполнения – текст задается даже в том случае, если строка отсутствует в списке.

Особую роль для комбинированного поля играет свойство Style. Оно может принимать три значения, определяющих поведение и внешний вид комбинированного поля.

0 - Раскрывающееся комбинированное поле похоже на стандартное текстовое поле, справа от которого имеется кнопка со стрелкой. Если нажать кнопку, под текстовым полем раскрывается список. Пользователь может либо выбрать строку из списка, либо внести в поле свой собственный вариант. Именно этот вариант обычно называется комбинированным полем.

1 - Простое комбинированное поле представляет собой разновидность описанного выше – единственное отличие состоит в том, что список постоянно остается открытым. Этот вариант выбирается в том случае, если на Вашей форме остается много свободного места.

**События** комбинированного поля представлены в таблице 19.

**Методы** комбинированных полей совпадают с методами списков.

Таблица 19 – Свойства ComboBox

| Событие | Описание |
|---|---|
| Click | Реакция на щелчок мыши |
| DblClick | Реакция на двойной щелчок мыши. Имеет значение лишь для простых полей |

**Задания на практическую работу**

**Задание 1.**

1. Вставьте новую форму. Высота формы – 150, ширина – 200.
2. Поместите на ней два объекта CommandButton. Одну кнопку назвать «Уменьшить», вторую – «Увеличить».
3. При нажатии на кнопку «Увеличить» размер формы должен увеличиваться на 10, не превышая максимальных значений: для высоты – 450; для ширины – 600.
4. При нажатии на кнопку «Уменьшить» размер формы должен уменьшаться на 10, не переходя за минимальный размер формы.

**Задание 2.**

1. Откройте форму предыдущего задания.
2. Дважды щелкните по форме, чтобы перейти к окну программы. В двух раскрывающихся списках, раскрывающихся в верхней части окна программы, должны быть выбраны строки Form и Load; это означает, что в настоящий момент Вы работаете с событием Load формы.
3. Раскройте второй список и прокручивайте его вниз то тех пор, пока не найдете в нем строку Resize. Тем самым Вы переходите к событию Resize данной формы.

4. Введите в событие Resizeследующую строку:

```
Width = Height
```

5. Запустите проект.

6. Попробуйте изменить размеры формы различными способами. Как ведет себя форма?

7. Остановите программу.

**Задание 3.**

Создать форму, которая позволяет выбрать несколько чисел, выводимых в списке, заполняемом случайными значениями при инициализации окна, в диалоговом окне **Операции над элементами** (рис. 15).

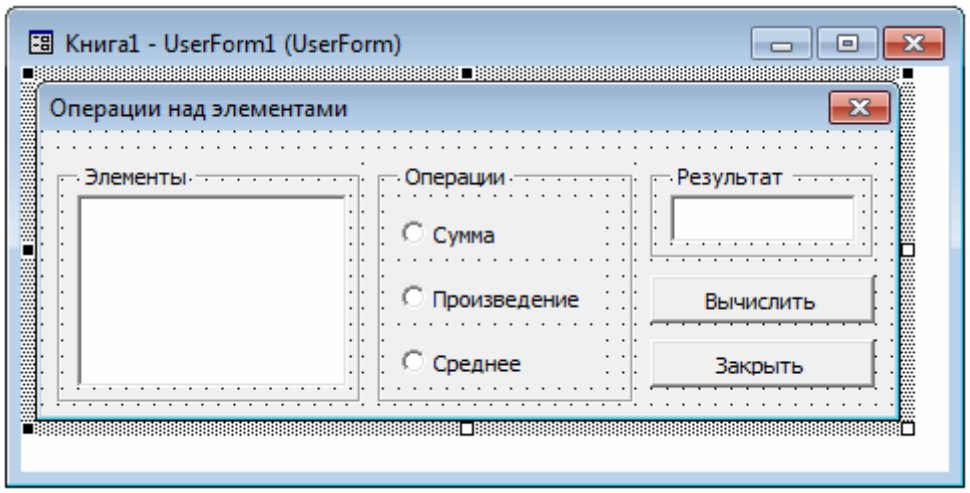



Рис. 15 ОбъектUserForm

В группе Операции следует установить один из переключателей: **Сумма, Произведение** или **Среднее**, чтобы указать какая из операций будет выполняться над выбранными числами. Нажатие кнопки **Вычислить** должно привести к выполнению операции и выводу результата в поле **Результат**. Кнопка **Закрыть** выполняет закрытие диалогового окна, а также должен быть запрет для ввода в поле **Результат** объекта UserForm.

# Практическая работа №6
# Классы и объекты в Visual Basic for Application

## Тема и цель работы

Знакомство с объектамиVBA. На практике рассмотреть основные операции и функции.

## Теоретический материал, для освоения темы

### Объектная модель

Объектная модель MS Excel представляет собой иерархию объектов, подчиненных одному объекту `Application`, который соответствует самому приложению Office. Многие из этих объектов собраны в библиотеке объектов конкретного приложения, но некоторые из них, например, объект `Assistant`, входят в библиотеку объектов Office, которая является общей для всех офисных приложений.

### Полная и неявная ссылка на объект

Полная ссылка на объект состоит из ряда имен вложенных последовательно друг в друга объектов. Разделителями имен объектов в этом ряду являются точки, ряд начинается с объекта `Application` и заканчивается именем самого объекта. Например, полная ссылка на ячейку **A1** рабочего листа **Продажи** рабочей книги с именем **Архив** имеет вид:

```
Application.Workbooks("Архив").Worksheets("Продажи").Ra
nge("A1")
```

Приводить каждый раз полную ссылку на объект совершенно не обязательно. Обычно достаточно ограничиться только неявной ссылкой на объект. В неявной ссылке, в отличие от полной, объекты, которые активны в данный момент, как правило, можно опускать. В рассмотренном случае, если

ссылка на ячейку **A1** дана в программе, выполняемой в среде Excel, то ссылка на объект `Application` может быть опущена, т.е. достаточно привести относительную ссылку:

```
Workbooks("Архив").Worksheets("Продажи").Range("Al")
```

Если в этом примере ссылки рабочая книга **Архив** является активной, то ссылку можно еще сократить:

```
Worksheets("Продажи").Range("Al")
```

Если и рабочий лист **Продажи** активен, то в относительной ссылке вполне достаточно ограничиться упоминанием только диапазона **A1**:

```
Range("Al")
```

**Основные объекты VBA**

Объект Application (приложение) является главным в иерархии объектов Excel и представляет само приложение Excel. Он имеет более 120 свойств и методов и методов. Эти свойства и методы предназначены для установки общих параметров приложения Excel. Кроме того, объект Application позволяет вызывать более 400 встроенных функций рабочего листа при помощи конструкции вида:

```
Application.ФункцияРабочегоЛиста (Аргументы)
```

Например, для определения постоянных периодических платежей при постоянной процентной ставке можно воспользоваться следующей функцией:

```
Application.Pmt (Аргументы)
```

**Свойства объекта Application**

Таблица 20 – Свойства Application

| | |
|---|---|
| ActiveWorkbook<br>ActiveSheet | Возвращает активный объект: рабочую книгу, лист, ячейку, диаграмму диалоговое окно. Свойство ActiveCell |

| | |
|---|---|
| ActiveCell<br>ActiveChart<br>ActiveDialog | содержится в ActiveSheet, а свойства ActiveChart и ActiveDialog в ActiveWorkbook.<br>Например:<br>ActiveCell.Value = “Привет!”<br>Здесь в активную ячейку вводится фраза **Привет!** |
| ThisWorkbook | Возвращает рабочую книгу, содержащую выполняющийся в данный момент макрос. |
| Calculation | Устанавливает режим вычислений |
| Caption | Возвращает текст в строке имени главного окна Excel. Установка свойства равным Empty возвращает заголовок, используемый по умолчанию |
| DisplayAlerts | Значения: True – отображаются встроенные предупреждения о работе программы, False – не отображаются предупреждения |
| DisplayFormulaBar | Значения: True – строка формул выводится в окне Excel, False – не выводится |
| DisplayScrollBars | Значения: True – полоса прокрутки выводится в окне Excel, False– полосы прокрутки не отображаются |
| Height | Высота приложения в пунктах |
| Width | Ширина приложения в пунктах |
| Right | Расстояние в пунктах от правой границы окна приложения до правого края окна |
| Left | Расстояние в пунктах от левой границы окна приложения до левого края окна |
| Top | Расстояние в пунктах от верхней границы приложения |
| WindowState | Устанавливает размер окна |

**Методы объекта Application**

Таблица 21 – Методы объекта Application

| | |
|---|---|
| Calculate | Вызывает принудительное вычисление во всех открытых рабочих книгах. |
| Run | Запускает на выполнение программу или макрос. |
| OnKey | Устанавливает сочетание клавиш для выполнение выбранной процедуры.<br>OnKey(Key, Procedure) |
| Quit | Закрывает приложение. |
| OnTime | Назначает выполнение процедуры на определенное время. |

**События объекта Application**

Таблица 22 – События объекта Application

| | |
|---|---|
| NewWorkbook | При создании новой рабочей книги |
| WorkbookActivate | При активизации рабочей книги |
| WorkbookBeforeClose | Перед закрытием рабочей книги |
| WorkbookBeforePrint | Перед печатью рабочей книги. |
| WorkbookBeforeSave | Перед сохранением рабочей книги. |
| WorkbookNewSheet | При добавлении нового листа |
| WorkbookOpen | При открытии рабочей книги |

**Объект Workbook (книга)**

**Свойства объекта Workbook**

Таблица 23 – События объекта Workbook

| | |
|---|---|
| ActiveSheet | Возвращает активный лист книги. |
| ActiveDialog | Возвращает активное диалоговое окно |
| ActiveChart | Возвращает активную диаграмму<br>Например: **MsgBox "Название диаграммы" &ActiveChart.Name**<br>выводит в диалоговом окне имя активной диаграммы |
| Saved | True– если не производились изменения в документе со времени его последнего сохранения, False – в противном случае. Например:<br>**If WorkBook.Saved=True Then**<br>**MsgBox " удачносохранились! "**<br>**End If** |
| WriteReserved | True– если книгу закрыть для записи, False – в противном случае |

**Методы объекта Workbook**

Таблица 24 – Методы объекта Workbook

| | |
|---|---|
| Activate | Активизирует рабочую книгу |
| Add | Создает новую книгу |
| Protect | Защищает рабочую книгу от внесения в неё изменений.<br>**Protect(Password, Structure, Windows)**<br>Например: **ActiveWorkbook.Protect Password:= "Невлезай!"** |

| | |
|---|---|
| UnProtect | Снятие защиты с листа **UnProtect(Password)** |
| Close | Закрытие рабочей книги |
| Open | Открытие существующей книги |
| Save | Сохранение рабочей книги |
| SaveAs | Сохранение книги с другими параметрами |
| PrintPreview | Предварительный просмотр |

**События объекта Workbook**

Таблица 25 – События объекта Workbook

| | |
|---|---|
| BeforeClose | При закрытии книги |
| BeforePrint | Перед печатью рабочей книги |
| BeforeSave | Перед сохранением рабочей книги |
| NewSheet | При добавлении нового листа. |
| Open | При открытии рабочей книги. |
| SheetActivate | При активизации любого рабочего листа. |
| SheetDeactivate | Когда рабочий лист теряет фокус. |

**Объект Worksheet (лист)**

**Свойства объекта Worksheet**

Таблица 26 – События объекта Worksheet

| | |
|---|---|
| Name | Возвращает имя рабочего листа. |
| Visible | Отображает «видимость» или «невидимость» листа (True, False – соответственно) |
| ActiveCell | Возвращает активную ячейку активного листа. |
| StandartHeight | Возвращает стандартную высоту всех строк рабочего листа |
| UsedRange | Возвращает диапазон (объект Range), который содержит данные. |
| Union | Возвращает диапазон, являющийся объединением нескольких диапазонов. |

**Методы Worksheet**

Таблица 27 – Методы объекта Worksheet

| | |
|---|---|
| Activate | Активизирует указанный рабочий лист. **Worksheets(1).Activate** |
| Add | Создает новый рабочий лист |

| | **Add(Before, After, Count, Type)** |
|---|---|
| Delete | Удаляет рабочий лист |
| Protect | Защищает рабочий лист от внесения в него изменений |
| Unprotect | Снятие защиты с рабочего листа |
| Copy | Копирование рабочего листа в другое место<br>**Copy (Before, After). Допустимо использование только одного аргумента** |
| Move | Перемещение рабочего листа в другое место |
| Evaluate | Преобразует выражение в объект или значение. Используется при вводе формул и ячеек из диалоговых окон |

**Объекты Range (диапазон)**

При работе с объектом Rangeнеобходимо помнить, как в Excel ссылаются на ячейку рабочего листа (см. относительные и абсолютные ссылки).

Так как ячейка является частным случаем диапазона, состоящим только из единственной ячейки, объект Range также позволяет работать с ней. Объект Cells(ячейки) – это альтернативный способ работы с ячейкой. Например, ячейка A2 как объект описывается Range(“A2”) или Cells(1,2).

**Свойства объекта Range**

Таблица 28 – Свойства объекта Range

| | |
|---|---|
| Value | Возвращает значение из ячейки или в ячейки диапазона. Например: **h=Range (“C1”).Value** |
| Name | Возвращает имя диапазона. |
| WrapText | Позволяет переносить текст при вводе в диапазон.<br>**With Range (“F3”)**<br>**.Value= “Приветвсем!”**<br>**.WrapText=True**<br>**EndWith** |
| Comment | Добавление комментария к данным в ячейки. |
| Font | Возвращает объект Шрифт с указанными параметрами. |
| Formula | Возвращает формулу в формате A1. |
| Text | Возвращает содержание диапазона в текстовом формате. |

**Наиболее часто используемые методы Range**

Таблица 29 – Методы объекта Range

| | |
|---|---|
| Clear | Очисткадиапазона.<br>**Range (“A1:B1).Clear** |
| Copy | Копирует диапазон в другой или буфер обмена. |
| Cut | Копирует диапазон в другой или буфер обмена с удалением. |
| Delete | Удаляет диапазон. |
| Insert | Вставка ячейки или диапазона ячеек.<br>**WorkSheets (“Лист1”).Rows(4).Insert** |
| Select | Выделение диапазона. |

Заполнение произвольного диапазона данных по столбцам можно осуществить следующим образом для перебора адреса ячейки с A3 по A5:

```
For i = 1 To 3
  A = "A" &i + 2
  Range([A]).Value = InputBox("Введиданные " &i)
Next
```

**Задания на практическую работу**

**Задание 1.**

Создать приложение в VBA, позволяющее определить и вывести в ячейки Excel количество лет, кварталов, месяцев, недель и дней, прошедших между двумя датами.

1. Создать приложение в VBA, позволяющее определить была ли сохранена открытая рабочая книга.

**Задание 2*.**

Создать приложение в VBA, позволяющее произвести расчет амортизации стандартным и методом k кратного учета и выводить данные на лист Excel.

# Практическая работа №7
# Автоматизация работы в MSWord с помощью Visual Basic for Application

## Тема и цель работы

Знакомство с основными объектами VBA Microsoft Word. На практике рассмотреть основные операции над объектами Application, Document, Selection, Range, Bookmark

## Теоретический материал, для освоения темы

Объектная модель Microsoft Word, объекты Application, Document, Selection, Range, Bookmark

На практике для решения большинства программных задач достаточно знать всего лишь пять объектов (с сопутствующими коллекциями):

- объект Application;
- объект Document (с коллекцией Documents);
- объект Selection;
- объект Range;
- объект Bookmark (с коллекцией Bookmarks).

Ниже все эти самые важные объекты будут подробно рассмотрены. Поскольку наиболее часто встречающаяся задача программирования в Word – это создание документа (на основе шаблона) и запись в нужное место документа необходимой информации, то акцент будет сделан использовании соответствующих объектов для решения этой задачи.

Кроме того, для каждого объекта будут перечислены самые важные свойства, методы и события с кратким их описанием.

**Объект Application, свойства, методы и события**

ActiveDocument – возвращает объект активного документа в данном экземпляре Word. Используется очень активно, обычно без упоминания объекта Application, например:

```
ActiveDocument.Save
```

Это свойство доступно только для чтения, поэтому, чтобы сделать какой-нибудь документ активным, придется вызывать для его объекта метод Activate().

ActivePrinter – позволяет получить или настроить активный принтер в ходе работы программы. Также используется очень активно, например, если результаты работы вашего приложения необходимо печатать на определенном сетевом принтере.

AutomationSecurity – определяет уровень безопасности при программном открытии файлов. По умолчанию установлено значение msoAutomationSecurityLow, что значит – открывать со включенными макросами. Можно также использовать значения msoAutomationSecurityForceDisable – отключить макросы и msoAutomationSecurityByUI – то, что настроено на графическом интерфейсе.

BackgroundPrintingStatus – сколько заданий Word стоит в очереди на печать.

Caption – позволяет заменить слово Microsoft Word в заголовке окна на другой текст, например, "Мое приложение".

CheckLanguage – определяет ли Wordв автоматическом режиме язык, на котором производится ввод текста. Если в системе установлено несколько языков ввода, то по умолчанию проверяет. При помощи этого свойства можно изменить режим работы Word.

CustomizationContext – свойство, которое позволяет указать шаблон или документ, на который будут распространяться внесенные вами изменения в меню, панели инструментов и клавиатурные комбинации. Например, код вида

CustomizationContext = NormalTemplate

говорит о том, что все изменения, которые вы будете вносить начиная с этого момента, будут сохраняться в шаблоне Normal.Dot (и, таким образом, будут применяться ко всем документам).

Для диалоговых окон, которые предназначены для работы с файлами, в объекте Applicationпредусмотрено отдельное свойство FileDialog, возвращающее одноименный объект.

DefaultSaveFormat – определяет формат сохранения файлов Wordпо умолчанию (тот, который будет предлагаться пользователь в диалоговых окнах **SaveAs**). Можно настроить на сохранение в формате обычного текста, текста Unicode, RTF и т.п.

DisplayAlerts – очень важное свойство. Оно позволяет подавить вывод ошибок и диалоговых окон при работе макросов и приложений VBA. Во многих ситуациях без него не обойтись. Особенно часто прибегать к этому свойству требуется, когда необходимо в ходе работы программы что-нибудь удалить или закрыть без сохранения.

DisplayAutoCompleteTips – включить/отключить подсказки для автозавершения текста. Чаще всего необходимо отключить.

FileDialog – возвращает объект FileDialog, то есть окно выбора файла, каталога, открытия файла или сохранения. Для открытия этого окна необходимо воспользоваться методом Show() этого объекта.

FileSearch – возвращает объект FileSearch, который может использоваться для поиска файлов по определенным параметрам.

IsObjectValid – очень удобное свойство для всевозможных проверок. Проверяет, существует ли еще объект, к которому хотим обратиться. Позволяет уберечься от ошибок, когда, к примеру, документ или объект в документе был удален пользователем.

KeyBindings – очень удобное во многих ситуациях свойство. Оно позволяет вернуть коллекцию KeyBindings – привязок клавиатурных комбинаций. Говоря проще, при помощи этого объекта и подобъектов вы

можете назначить любую команду Wordили любой макрос любому сочетанию клавиш (в том числе и сочетаниям, уже занятым служебными командами, например, <Alt>+<F4>).

MacroContainer – очень полезное свойство для программистов. Позволяет в ходе выполнения определить, откуда был запущен текущий программный код (обычно проверяются два варианта –normal.dotили обычный текущий документ).

NewDocument – одна из возможностей создать новый документ Word. Возвращает объект NewDocument. Для создания нового документа используется метод Application.NewDocument.Add().

NormalTemplate – позволяет получить ссылку на объект Template, представляющий normal.dot – для внесения в него изменений.

Option – возвращает объект Optionc огромным количеством свойств. Через этот объект программным способом можно настроить значения со всех вкладок, доступных на графическом экране через меню **Сервис** – **Параметры**.

Path – возвращает путь к программным файлам Wordна диске.

PrintPreview – перейти в режим предпросмотра текущего документа или проверить, находимся ли мы в этом режиме. Очень удобно для показа документа пользователю или для реализации своей процедуры печати.

Selection – еще одно важнейшее свойство. Возвращает объект Selection – упрощенно говоря, место, в котором находится указатель вставки.

StatusBar – еще одно очень полезное свойство. Позволяет вывести текст в StatusBar – строке состояния, то есть строке в нижней части окна приложения, в которой выводится информация о страницах, столбцах, языке, режимах работы и т.п.

System – возвращает одноименный объект System, предназначенный для получения информации из операционной системы (региональный настройки, тип курсора мыши, разрешение экрана, тип процессора и т.п.).

Позволяет также подключать сетевые диски и запускать приложение MicrosoftSystemInformation.

UserControl – очень важное свойство (оно есть и в Excel). Это свойство позволяет определить как именно был запущен Word – пользователем вручную или программным образом. На основе этого можно, например, сделать вывод, нужно ли его программным образом закрывать.

UserInitialsи UserName – возможность получить или определить информацию об инициалах или имени пользователя. Инициалы используются в исправлениях, а имя пользователя – в свойствах документа.

Version – свойство возвращает версию Word.

Visible – позволяет спрятать окно MicrosoftWord.

Windows – возвращает информацию об одноименной коллекции Windows, представляющей объекты окон документов Word. Эта коллекция также используется очень часто.

WindowsState – позволяет свернуть/развернуть/восстановить окно Word.

Самые важные методы объекта Application:

Activate() – просто активизировать окно Wordc текущим документом. Обычно нужно активизировать определенный документ, поэтому этот метод используется для объекта Document.

BuildKeyCode() – позволяет узнать уникальный номер для клавиатурной комбинации в Word. Пример применения этого метода приведен чуть выше при рассмотрении свойства Application.KeyBindings.

ChangeFileOpenDirectory() – этот метод позволяет изменить каталог, который по умолчанию открывает Wordдля работы с документами (по умолчанию, конечно, "Мои документы".

CleanString() – очень полезный метод. Позволяет "чистить" передаваемое символьное значение (полученное, например, от объектов Selectionили Range) от специальных символов Wordи превращать их в обычный текст – как будто он был набран в Блокноте.

GoBack() – этот метод обеспечивает переход на последнее место редактирования в документе. Wordсохраняет с документом три последние точки редактирования, так что открыть последний документ в Wordи перейти на точку, где вы остановились, можно очень просто:

```
RecentFiles(1).Open
Application.GoBack
```

Метод GoForward() обеспечивает переход вперед по точкам сохранения.

KeyBoard() – очень полезный метод. Позволяет программным способом переключать раскладку клавиатуры в Word, уберегая таким образом пользователей от ошибок. Переключение на русский выглядит как

```
Application.Keyboard 1049
```

а на английский

```
Application.Keyboard 1033
```

Если этому методу ничего не передавать, он вернет текущую раскладку клавиатуры.

OnTime() – очень интересный метод. Он позволяет выполнить макрос Word либо в указанное вами время, либо по прошествии какого-то времени. В Word одновременно может работать только один таймер.

OrganizerCopy() – еще один полезный метод. Позволяет скопировать макрос, панель инструментов, запись автотекста или стиль между документами. Для объекта Applicationпредусмотрены также методы с самообъясняющими OrganizerDelete() и OrganizerRename().

PrintOut() – метод, который принимает огромное количество параметров и позволяет вывести на печать весь документ или его часть.

Quit() – метод, который используется, видимо, чаще всех. Позволяет закрыть Wordс сохранением или без сохранения документов.

Repeat() – просто повторяет последнюю выполненную команду указанное вами количество раз.

Run() – позволяет запустить процедуру/макрос из открытого шаблона/документа и передать ей параметры.

ShowClipboard() – показать панель буфера обмена Word(если вы работаете с несколькими буферами).

У объекта Applicationмножество событий – открытие/закрытие/сохранение/печать документа, щелчки мышью, активизация, выход из приложения и т.п.

**Работа с объектом Selection**

**Объект Word.Selection, работа с выделенным участком текста, преимущества и недостатки**

После того, как запущено приложение, найден и активизирован нужный нам файл, следующее действие, которое выполняется чаще всего – ввод или редактирование текста в нужном месте. Для этого используются объекты **Selection, Range**и **Bookmark**. Каждое из них используется в своих ситуациях и для своих задач.

Первый объект, который рассмотрим – это объект **Selection.**

Обычно перед тем, как что-либо сделать в окне документа Word, пользователь либо выделяет нужный участок текста, либо переставляет указатель вставки текста в нужное место. Объект Selection представляет именно такой выделенный участок текста (а если ничего не выделено, то место, где находится указатель вставки). Именно этот объект обычно использует макрорекордер.

Создавать объект Selection и получать на него ссылку в переменную не обязательно (а обычно и просто невозможно). Дело в том, что объект Selection в документе может быть только один. Он создается автоматически при запуске Word и всегда доступен. Обращаться к нему можно так:

```
Application.Selection.Text = "Вставляемыйтекст"
```

или просто

```
Selection.Text = "Вставляемый текст"
```

Обычно нужно правильно определить то место, на которое указывает объект Selection, чтобы выделить нужный нам участок текста или точку для ввода.

**Как настроить выделение в документе Word**

Самый простой способ – просто положиться на выделение нужного текста пользователем. Обычно такой способ применяется для сложного редактирования/форматирования участков текста и для ввода информации в указанное пользователем место документа, когда в автоматическом режиме нужное место не найти;

- воспользоваться методом Select(), который предусмотрен для огромного числа объектов (Document, Range, Bookmark, Tableco всеми подобъектами типа столбцов и строк, PageNumber, Fieldи т.п.). Этот метод просто выделяет весь документ, закладку, таблицу и т.п.
- воспользоваться многочисленными методами объекта Selection, чтобы преобразовать уже существующее выделение;
- воспользоваться объектом Findдля поиска нужного участка текста.

Если нужно вводить информацию в самое начало документа, можно вообще ничего не делать. По умолчанию указатель вставки устанавливается на начало документа.

Несмотря на то, что применение объекта Selection– самый простой и наглядный метод редактирования текста, и чаще всего именно он используется макрорекордером, на практике программисты используют его редко. Объясняется это очень просто: при использовании этого объекта существует зависимость от действий пользователя. Если во время выполнения нашего кода пользователь проявит инициативу и начнет щелкать по документу мышью, результат может быть совершенно непредсказуемым. Защититься от вмешательства пользователя можно двумя способами: работать со скрытым (то есть невидимым) документом или, возможно, со

скрытым экземпляром Word. Для включения/отключения невидимости можно использовать свойство Visibleдля объектов Documentи Application; более удобный способ – вместо объекта Selectionиспользовать объекты Rangeи Bookmark.

### Объект Word.Bookmark, применение закладок в шаблоне, получение из объектов Bookmark объектов Selection и Range

Объект Bookmark – это просто закладка. На практике это – самый удобный способ навигации по документам, созданных при помощи шаблонов (например, отчетов). Принципиальное отличие его от объектов Selection и Range заключается в том, что все выделения и диапазоны теряются при закрытии документа (объекты Range вообще существуют только во время работы создавшей их процедуры, а закладки сохраняются вместе с документом. Если документ создан на основе шаблона, то все закладки, которые были определены в шаблоне, будут определены и в созданном на основе этого шаблона документе.

Создать закладку (меню **Вставка - Закладка**) намного проще, чем считать количество символов для объекта Range от начала документа/абзаца/предложения, или выполнять операции Move() (MoveDown(), MoveRight(), MoveNext()) для объекта Selection.

Функциональность объекта Bookmark невелика. Свойств и методов у этого объекта намного меньше, чем у объектов Selection и Range. Однако обычно никто и пытается использовать объект Bookmark для работы с текстом напрямую. Из объекта Bookmark очень просто получить объект Selection (при помощи метода Select()) или объект Range (при помощи свойства Range()) – и дальше можно пользоваться уже свойствами и методами этих объектов, например:

```
ThisDocument.Bookmarks("Bookmark1").Select
MsgBoxSelection.Text
```

Создавать объекты Bookmark программным способом необязательно, но если есть необходимость, то можно использовать метод Add() коллекции Bookmark:

```
ThisDocument.Bookmarks.Add Name:="temp", Range:=Selection.Range
```

У этого метода – всего лишь два параметра, оба которых используются в примере.

**Некоторые важные свойства объекта Bookmark**

Empty – если это свойство возвращает True, то это значит, что закладка указывает на точку вставки, а не та текст;

Name – имя закладки. Очень удобно, что найти нужную закладку в коллекции закладок можно не только при помощи индекса (номера) закладки, но и по ее имени.

Range – возвращает объект Range на месте этой закладки.

Start, End, StoryType – аналогично таким же свойствам у объекта Selection.

Методов у объекта Bookmark всего три –Copy(), Delete() и Select(). Copy() – создает закладку на основе существующей, Delete() – удаляет ее, а Select() – выделяет то, на что ссылается закладка.

## Объект Word.Range, программная работа с диапазоном в документе, свойства и методы объекта Range, преимущества по сравнению с объектом Selection

Как уже говорилось выше, чаще всего разработчиками для определения места ввода текста и навигации по документу используется объект Selection. Для этих же целей можно использовать и объект Range. Главное отличие между объектами Range и Selection заключается в том, что объект Selection может определить и пользователь (выделив текст мышью), а объект Range

можно определить только программно, и он не зависит от текущего положения указателя или действий пользователя.

Формальное определение объекта Range выглядит так: это программный объект, который представляет непрерывный участок текста в документе. Этот объект не зависит от объекта Selection – вы можете работать с объектом Range, не изменяя текущего выделения. Он может не включать в себя ни одного символа (представлять курсор ввода текста).

Объектов Range в каждый момент времени может быть сколько угодно, а объектов Selection – только один.

**Как создается объект Range**

Первый способ – воспользоваться методом Range() объекта Document. В этом случае вам потребуется передать номера начального и конечного символа диапазона, а также documentstory, в который будут отсчитываться эти символы. Например, создать диапазон, который будет включать в себя первые 10 символов документа, можно так:

```
Dim rngDoc As Range
Set rngDoc = ActiveDocument.Range(Start:=0, End:=10)
```

Второй способ – воспользоваться свойством Range, которое предусмотрено для огромного количества объектов (Bookmark, Selection, Table-Row-Cell, Paragraph и т.п.). В этом случае при помощи этого свойства получаем объект Range, представляющий данный объект;

Третий способ – воспользоваться большим количеством вспомогательных свойств (Characters, Words, Sentences и т.п.), которые делят текст на отрезки – объекты Range. Эти свойства возвращают коллекции объектов Range.

Четвертый способ – переопределить существующий объект Range. Обычно для этой цели используется метод Range.SetRange();

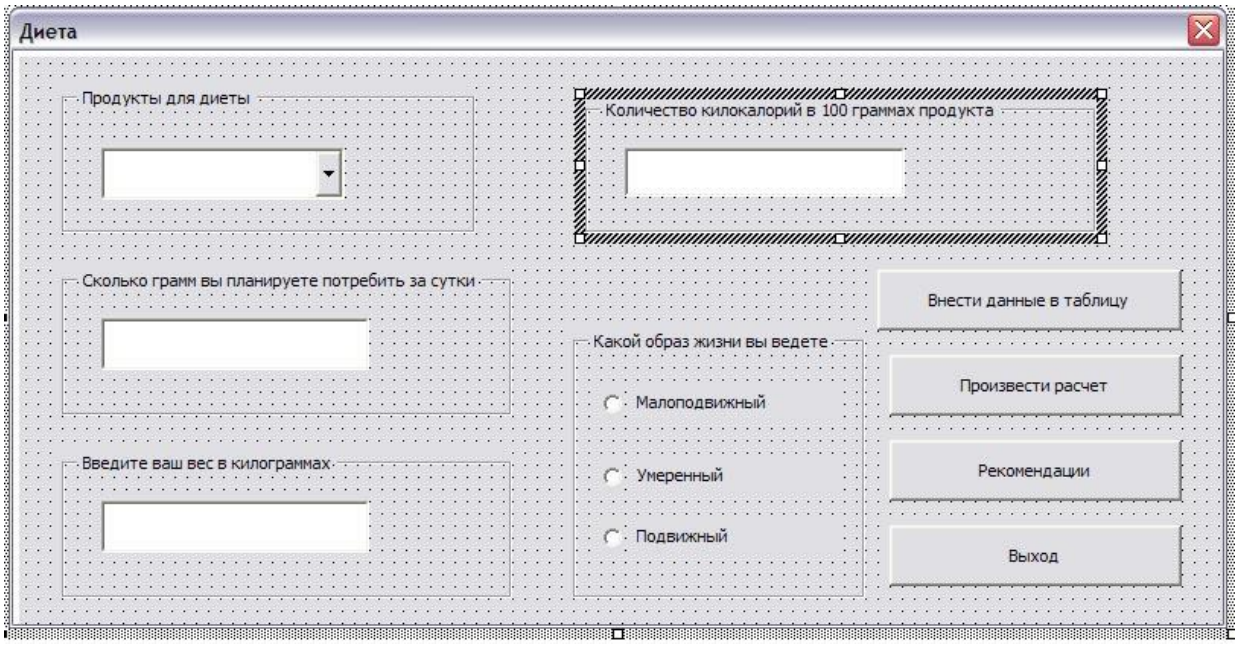
Пятый способ, самый удобный в реальных приложениях. Он заключается в том, что вначале создается шаблон нужного вам документа (договора, приходного ордера, отчета и т.п.), в который при создании помещаете **закладки** в тем места, в которые потом потребуется произвести вставку данных. Затем программным способом для каждой закладки создается объект Range, и уже с его помощью производится ввод информации (данные о заказчике, сумма в кассовом ордере и т.п.)

Для целей отладки (чтобы убедиться, что объект Range действительно включает в себя тот участок текста, который вы планировали) можно создавать на основе объекта Range объект Selection (то есть выделять диапазон). Для этого у объекта Range предусмотрен метод Select().

Большая часть свойств и методов объекта Range совпадает с аналогичными свойствами и методами объекта Selection. Точно так же чаще всего для объекта Range используется одно-единственное свойство Text, которое позволяет ввести в место в документе, представленное этим объектом, нужный текст.

**Задания на практическую работу**

Необходимо разработать приложение следующего вида:




Данное приложение должно заполнять таблицу MSWord и производить расчет калорий согласно выбранным продуктам, а также давать рекомендации по выбранному вами меню.

1. В разделе «Продукты для диеты» пользователь должен выбрать предложенный из списка продукт. После этого, в разделе «Количество килокалорий в 100 граммах продукта» должна появиться информация необходимая информация.

2. Затем пользователь вводит в текстовые поля информацию о планируемом потреблении данного продукта и о своем весе в килограммах.

3. Также пользователь должен указать информацию о своем образе жизни.

4. Для того, чтобы внести информацию о выбранном продукте в таблицу, пользователю нужно нажать кнопку с одноименным названием.

5. Для итогового расчета, пользователь должен нажать на кнопку «Произвести расчет».

6. Для получения рекомендаций по диете – нажать на кнопку «Рекомендации».

7. Для выхода из приложения – нажать кнопку выход.

**Ход работы:**

1. Вызвать редактор VBAи создать макет данного приложения (разместить объекты на форме). Для работы с вложенным меню воспользуйтесь объектом ComboBox (смотри документы «Практическая работа №3» и «Лабораторный практикум №3» на сервере).

2. Заполнить список объектов (данные смотри в приложении 1).

3. Описание действий для кнопки «Внести данные в таблицу»:

После того, как пользователь выбрал продукт и внес всю необходимую информацию (в том числе, указал какой он образ жизни ведет) данная информация должна быть внесена в таблицу, следующего вида:

| Продукты | Кол-во килокалорий в 100 граммах | Общая сумма килокалорий по продукту |
|---|---|---|
| | | |
| | | |
| Ваш вес=--- | | |
| Ваше меню составляет = ---калорий | | |

Вес и общее количество килокалорий, которое содержит меню пользователя должны будут заполняться, только тогда, когда пользователь нажмет кнопку «Произвести расчет» (воспользуйтесь объектом FormFields для вывода данных).

Продуктов может быть выбрано несколько, по мере заполнения таблицы должны добавляться строчки.

4. При нажатии кнопки «Рекомендации» пользователь должен получить информацию о том подходит ему данный рацион или нет. Это должно быть рассчитано следующим образом: если пользователь ведет малоподвижный образ жизни, то его вес должен быть умножен на 24, если умеренно активен, то вес необходимо умножить на 30, и на 44, если очень активен. Данные параметры будут считаться нормой, если меню в килокалориях будет меньше нормы или больше, то должны быть

соответствующие рекомендации в виде сообщений (MsgBox). Например: «Ваше меню составляет 2300 калорий, что превышает вашу норму (1500). Советуем выбрать менее калорийное меню».

## Список используемой литературы

1. Гарнаев А.Ю.Самоучитель VBA: 2-ое издание – СПб.: БХВ-Петербург, 2004. – 560 с.

2. Гарнаев А.Ю. Самоучитель VBA: Технология создания пользовательских приложений. – СПб.: БХВ-Петербург, 2002. – 512 с.

3. Кузьменко В.Г.Самоучитель VBA 2003: – М.: БИНОМ, 2004. – 430 с.