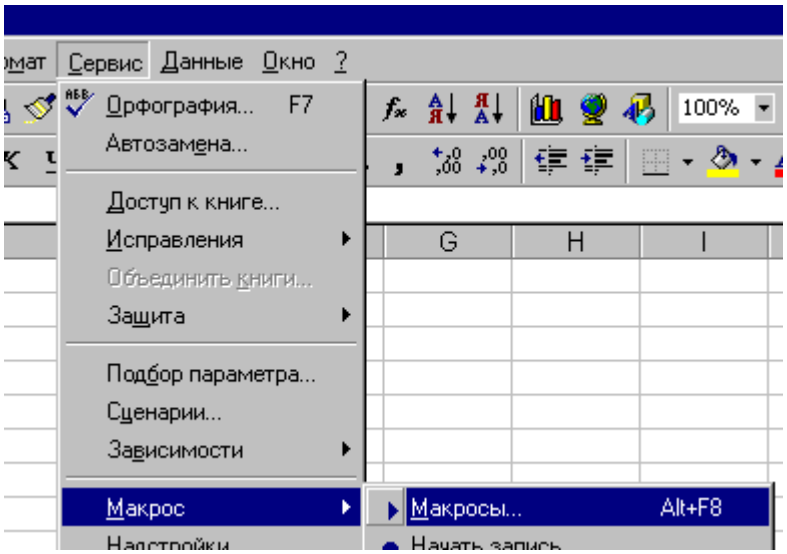
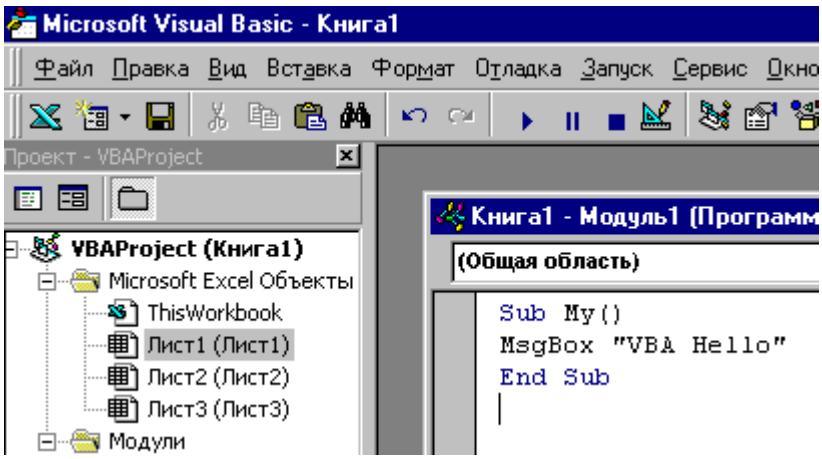
Шаг 1 - Первый макрос

Создаются макросы в меню **Сервис - Макрос - Макросы (Alt-F8)**:

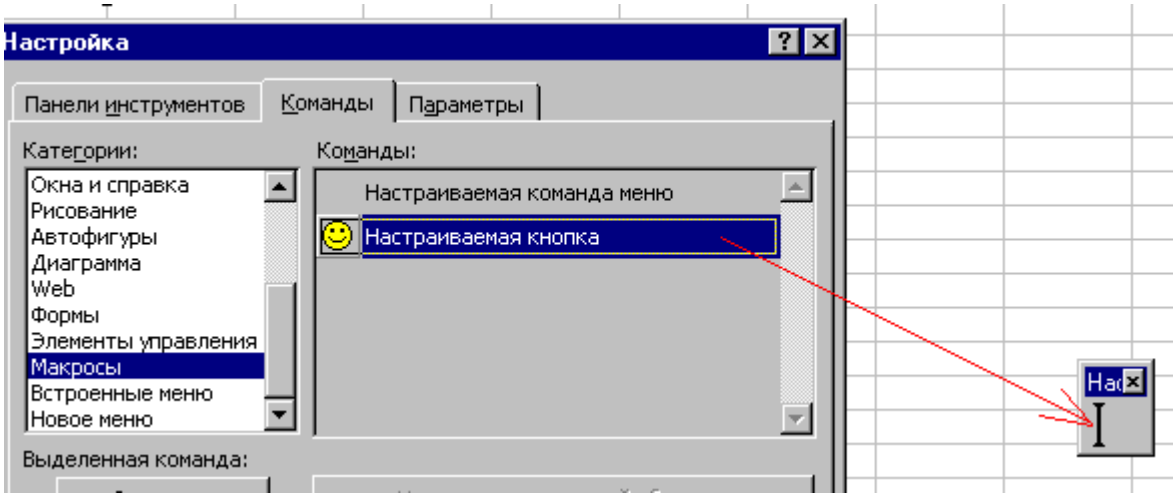


Появится табличка "макросы". В поле имя введите "**MY**", а место нахождения в поле "Находится" выберите - "Это книга". Кнопка "Создать" станет активной - нажмите её. Появится редактор **VBA,** введите код, как на рисунке ниже.



Здесь используется функция **MsgBox**, которая выводит на экран окно сообщения. Закройте редактор **VBA** файл - закрыть.

Дальше мы привяжем макрос к кнопке. Для этого создадим свою панель Инструментов. **Вид - панели инструментов - Настройка**. Нажмите создать и у Вас появится панель инструментов настраиваемая 1. Теперь перейдем к вкладке команды в категории выбираем макросы. Хватаем веселую желтую рожицу и тащим на панель.



Теперь на рожице нажимаем правой кнопкой мыши и выбираем пункт меню "назначить макрос". Выбираем наш макрос. Нажимаем **Ok** и закрываем окно настройки. Теперь можно испытать. Нажмите кнопку, макрос выполняется и появляется надпись.

Классно, работает.

Шаг 2 - Объектная модель Excel

Мы будем изменять наш макрос, зайдите в пункт меню "макросы", выберите наш и скажите "изменить":

```
Sub Test()
        Dim book As String
        Dim sheet As String
        Dim addr As String
        addr = "C"
        book = Application.ActiveWorkbook.Name
        sheet = Application.ActiveSheet.Name

        Workbooks(book).Activate
        Worksheets(sheet).Activate

        Range("A1") = book
        Range("B1") = sheet

        Dim xList As Integer
        xList = Application.Sheets.Count
        For x = 1 To xList
                Dim s As String
                s = addr + LTrim(Str(x))
                Range(s) = x
        Next x
End Sub
```

Программирование на **VBA** можно рассматривать, как управление объектами приложения. Вот именно объектами и управляет наше приложение. В нашем случае, если упростить иерархическую архитектуру, то это выглядит так.

```
Application
        Workbook
        ........
                Worksheets
                ......
                        Cell
                        .....
```

То есть главный объект - приложение. В приложении могут быть несколько книг (**Workbook**), внутри которых находятся листы (**Worksheets**) и листы разбиты на ячейки (**Cell**). При работе активными могут быть только одна книга и один лист. Вот я своим макросом и пытаюсь это выяснить. А заодно сколько листов в текущей книге.

DIM - объявляет переменную с типом **string**. Используя объект **Application**, мы получаем имена текущих книг и листа. С помощью **Range("...")** можно выделить ячейку и поместить значения в неё или считать. Вообщем объекты имеют огромное количество свойств. Задача программиста на **VBA** знать эти свойства и методы. Ну, я думаю мысль этого шага понятна :-))

Шаг 3 - Коллекции в VBA

В любом языке программирования массивы (коллекции) занимают большое место. При этом именно понятие коллекция принимает широкое распространение. В **Visual C++** понятие коллекция аналогична шаблону. При программировании на **VBA** понятие коллекции приобретает большой смысл. Коллекции встречаются на каждом шагу.

Обычно коллекции имеют 4 метода:

```
Add
Remove
Count
```

Item

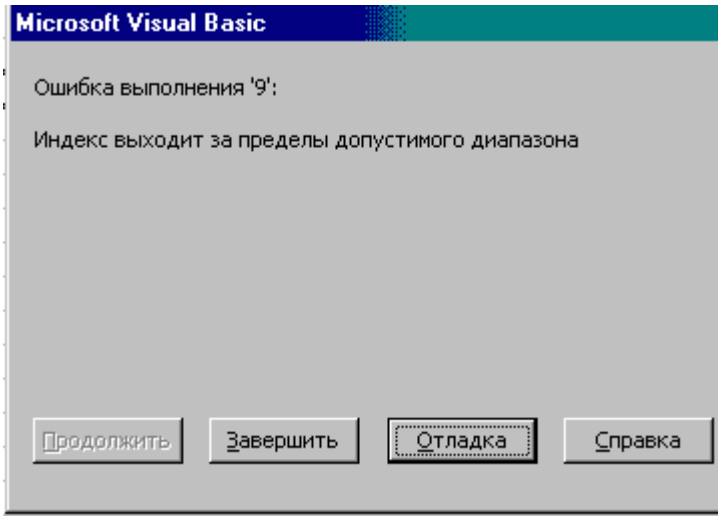
Было бы наверно логично подумать, что в **MS OFFICE** есть специализированный класс коллекций, на подобии шаблонов в **C++**. Но это не так. Для каждого типа объектов объявляется своя коллекция. Так же есть некоторые коллекции, которые отличаются названием методов. Всё это немного странно, но что сделаешь :-)

Для понимания работы с коллекциями создадим имитирующий коллекцию книг в **Excel**:

```
Sub Test()
        Dim MyCollection As New Collection
        With MyCollection
                .Add ("Книга 1")
                .Add ("Книга 2")
                .Add ("Книга 3")
                MsgBox (Str(.Count))
                MsgBox (.Item(1))
                .Remove (1)
                MsgBox (.Item(1))
        End With
End Sub
```

Первой строкой мы создаем переменную типа коллекция. Дальше мы используем оператор **With**, чтобы не использовать многократно **MyCollection** и тем самым сократить код. Методом **Add** мы добавляем данные в коллекцию. **Count** возвращает Вам количество элементов в коллекции. **Item** возвращает элемент коллекции, а **Remove** удаляет по индексу.

Метод **Remove** опасен тем, что он может выйти за пределы массива и вы получите подобную ошибку.

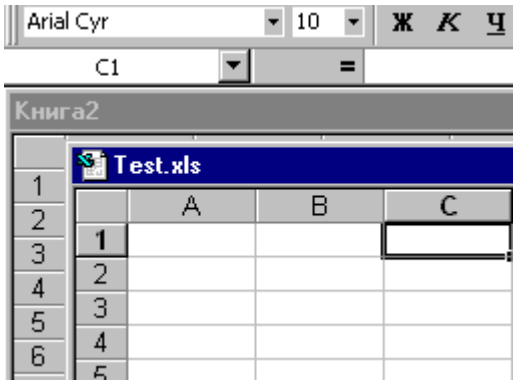


Обработку ошибок мы рассмотрим позже.

Заканчивая краткое описание коллекций следует отметить, что они могут содержать и элементы разных типов, что отличает их от массивов. Размер коллекции динамически изменяется. При удалении дырок не обнаруживается. Лафа вообще после **C++** :-))

Шаг 4 - Коллекция Workbooks в Excel

Итак, в **Excel** самую верхушку составляет объект **Application**. Это объект приложение. И этот объект содержит ряд коллекций. Первая коллекция это коллекция рабочих книг **Workbook**. Вот как это выглядит на экране (визуализуется). То что есть внизу на экране и есть отображение коллекции книг.



Естественно вы как программист должны уметь со всем этим работать программным путем. Определять какие книги загружены, добавлять и удалять, и делать еще много вещей. При этом без меню и мышки программно.

Первая коллекция это коллекция книг. Первое, что нужно узнать это сколько книг отрыто. Вот как это сделать.

```
Sub Test()
        MsgBox (Str(Application.Workbooks.Count))
End Sub
```

Функция **Str** переводит число в строку. Метод **Count** Вам известен. Он возвращает количество элементов коллекций. Как и на рисунке сверху у меня две книги, о чем и сообщит мне окно сообщения.

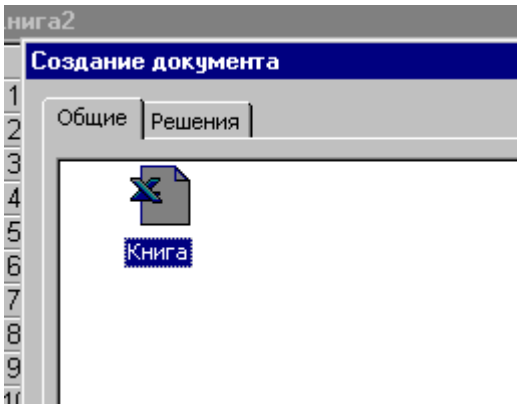
Данная коллекция обладает некоторой спецификой. Это связанно с тем, что книги могут храниться в файлах. Поэтому при работе с этой коллекцией предусмотрено две функции добавления **Add** и **Open**:

```
Add(template)
```

Добавляется книга на основе некоторого шаблона. Шаблоном может выступать, как настоящий шаблон с расширением **xlt**, так и просто файл **xls**. Смотрим пример:

```
Sub Test()
        Application.Workbooks.Add ("Книга")
End Sub
```

Откуда я взял имя шаблона Книга? Вот смотрите сами.



А здесь как он оказался. Молодцы !!! Честное слово. Это важный вопрос. Сначала, что такое шаблон. Шаблон это специальная рабочая книга, образец для создания документов. Тип файлов шаблонов **xlt**, то есть такое расширения носят шаблоны. Что задается в шаблоне ?

- Форматы ячеек
- Пользовательские меню, макросы и панели инструментов
- номера и типы листов
- стили строк и колонок
- текст, даты формулы . можно некоторые константы и графика.

Когда на основе шаблона создается документ , это просто копия шаблона, полная копия. Имея созданную книгу её можно сохранить как шаблон. Для этого используйте пункт меню "Сохранить как тип шаблона". Конечно шаблоны хранятся в некоторой папке, путь к ней указан в реестре:

```
HKEY_CURRENT_USER\Software\Microsoft\Office\8.0\Common\FileNew\Local Templates
```

Вы можете поменять этот тип на сетевой для одновременного изменения шаблона на всех компьютерах в случае, например, изменения телефона вашей фирмы.

Следующий метод **Open**, у которого куча параметров. Но единственный важный это имя файла, остальные можно опустить.

```
Sub Test()
        Application.Workbooks.Open ("c:\1\My.xls")
End Sub
```

Приведу я параметры на всякий случай:

```
expression.Open(FileName, UpdateLinks, ReadOnly, Format,
        Password, WriteResPassword, IgnoreReadOnlyRecommended,
        Origin, Delimiter, Editable, Notify, Converter, AddToMRU)
```

Шаг 5 - Далее про Workbooks в Excel

В прошлый раз мы научились добавлять книги в коллекцию. Теперь научимся удалять и получать информацию о книгах. Так как книга ассоциируется с файлом, то и метод удаления книги из коллекции называется **Close**. Но для удаления необходимо получить доступ к элементу.

```
Sub Test()
        Application.Workbooks.Item(1).Close
End Sub
```

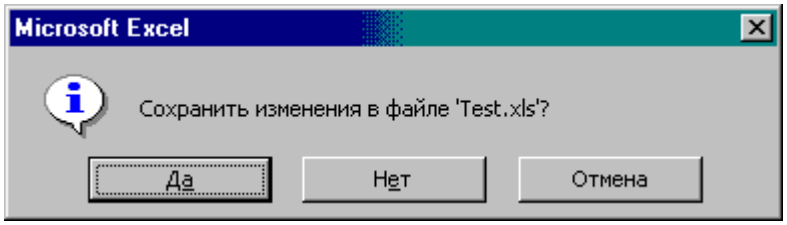
В таком варианте закроются все книги:

```
Sub Test()
        Application.Workbooks.Close
End Sub
```

Функция **Close** имеет ряд необязательных параметров. Вот они.

```
expression.Close(SaveChanges, FileName, RouteWorkbook)
```

Первый параметр **SaveChanges** типа **BOOL**, если установить **TRUE** сделанные изменения сохранятся, в противном случае нет. Если параметр упускается, то при закрытии появляется диалоговое окно с вопросом о необходимости сохранения.



Следующий параметр **FileName** необходим, когда идет вопрос о закрытии книги не связанной еще с именем файла. Последний параметр связан с одновременной работой над книгой. Он типа **BOOL**.

Получить доступ к книгам в коллекции можно используя метод **Item()**:

```
Sub Test()
        MsgBox (Application.Workbooks.Item(1).Name)
        MsgBox (Application.Workbooks.Item("Test.xls").FullName)
End Sub
```

Как видите, доступ можно получить по индексации и по имени книги, следует знать, что имя книги это имя файла, в котором он хранится. Получив доступ по индексу в первой строке я узнал имя. А во второй по имени получил доступ к свойствам объекта **WorkBook** и получил полное имя, название файла и путь, который и вывел на экран в виде сообщения.

Еще одно свойство - это создатель книги, и называется он **Creator**

```
Sub Test()
        If (Application.Workbooks.Creator = &H5843454C) Then
                MsgBox "Excel Creator"
        Else
                MsgBox "Not Excel Creator"
        End If
End Sub
```

С помощью **Parent** можно получить доступ к старшему объекту коллекции при выполнении кода изложенного ниже, появится **MS Excel**, ну а Вы что подумали ? :-)

```
Sub Test()
```

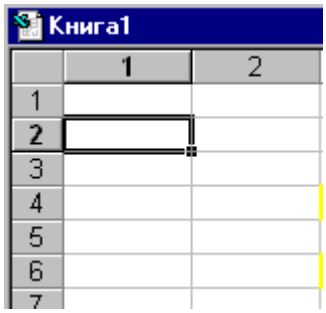
```
        MsgBox (Application.Workbooks.Parent.Name)
End Sub
```

Шаг 6 - Имена ячеек и адресация в Excel

Раз мы че-то задумали программировать нужно разобраться с тем, как **Excel** производит адресацию ячеек и как можно им давать имена. По умолчанию используется стиль **A1**. Это когда по строкам используется алфавит, а по горизонтали цифры. Например **D10** это десятая строка в колонке **D**. Есть и стиль называемый **R1C1**, который наиболее полезен при вычислении позиции строки и столбца в макросах, а также при отображении относительных ссылок. В стиле **R1C1**, после буквы "**R**" указывается номер строки ячейки, после буквы "**C**" номер столбца.

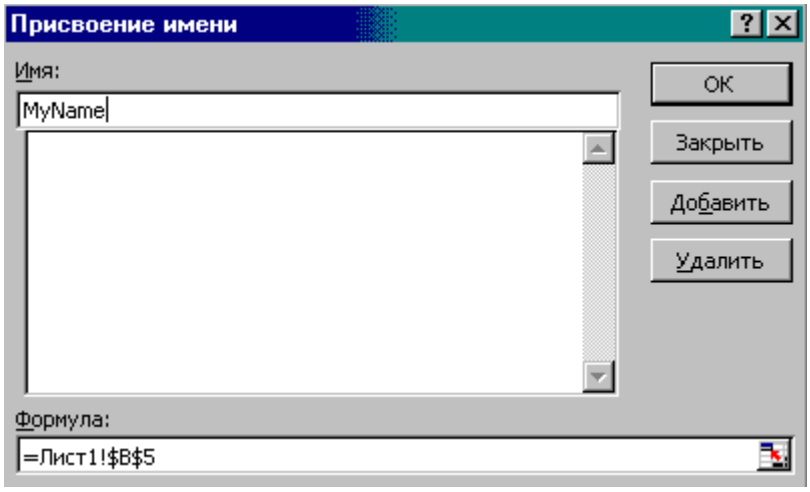
Стиль **A1**:

Стиль **R1C1**:



При работе стили переключаются в меню **Сервис -> Параметры -> Общие -> Стиль ссылок**, при реальном программировании наиболее удобно пользоваться не этими стилями, а именами ячеек. Тогда работа с вашей ячейкой похожа на роботу с обычной переменной. Что многим более привычно и удобно. Например для констант или полей форм.

Для того, чтобы дать имя ячейке наведите на неё курсор. Выберите меню **Вставка -> Имя -> Присвоить**. Появится диалоговое окно, куда надо ввести имя и нажать кнопочку **ОК**.

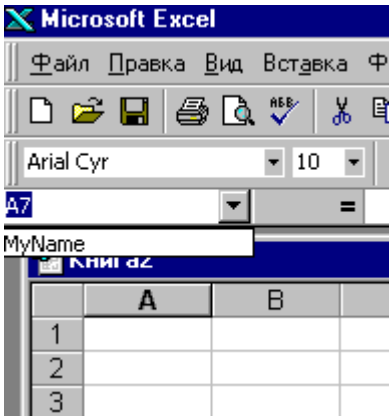


После присваивания имени вы введите число в эту ячейку, а в другой создайте формулу:

```
=MyName+10
```

Данная запись намного информативнее, кроме того вы можете не заботиться о местоположении имени в таблице , можете менять его местоположение не заботясь о том, что Ваши формулы будут изменены. А особенно это важно при программировании. Эта мелочь позволит избежать Вам сложной адресации и отслеживания данных.

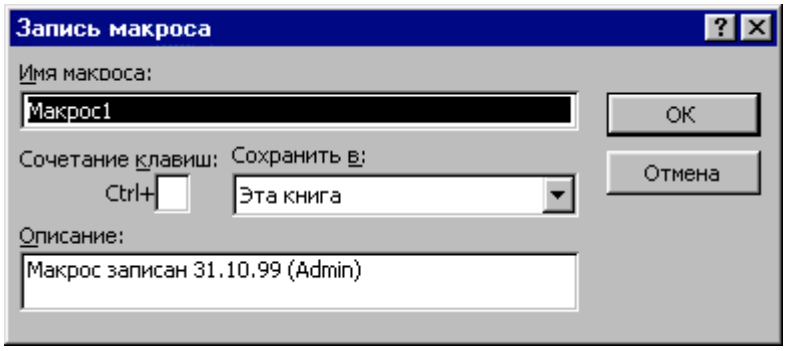
Узнать все имена можно здесь:



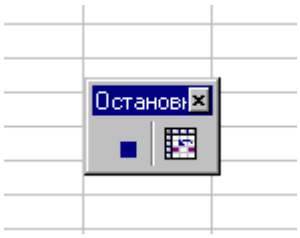
И здесь так же можно быстро переместиться к ячейке с заданным именем. Выберите её из списка и где бы она не находилась Вы окажетесь там :-)

Шаг 7 - Запись макросов и что это дает

Попробуем записать макрос. Для этого выбираем пункт меню **Сервис -> Макрос -> Начать запись**, в ответ на это Вы получите следующее диалоговое окно.



Здесь вы можете указать название макроса, быструю клавишу, где хранить макрос. Оставьте все как есть и нажмите кнопку **OK**. В результате у вас появится значок, который говорит о том, что идет запись. Вообще при записи макросов рекомендуется пользоваться клавишами, но я, например, и мышкой пользуюсь и записывается. Итак, появится значок.



А теперь выполните следующие действия. Создайте новую книгу, введите два числа в колонку, примените автосуммирование , сохраните книгу. После чего остановите запись макроса нажав на эту квадратную синюю кнопку. Зайдите в меню **Сервис -> Макрос -> Макросы**, у вас в диалоговом окне появится название вашего макроса. Выделите его мышкой и нажмите **Изменить**. Должен появиться такой код:

```
Sub Макрос1()
'
' Макрос1 Макрос
' Макрос записан 31.10.99 (Admin)
'
'
        Application.WindowState = xlMinimized
        Application.WindowState = xlNormal
        Workbooks.Add
        ActiveCell.FormulaR1C1 = "12"
        Range("A2").Select
        ActiveCell.FormulaR1C1 = "23"
        Range("A3").Select
```

```
        ActiveCell.FormulaR1C1 = "=SUM(R[-2]C:R[-1]C)"
        Range("A4").Select
        ChDir "C:\WINDOWS\Рабочий стол"
        ActiveWorkbook.SaveAs FileName:="C:\WINDOWS\Рабочий стол\Книга2.xls", _
                FileFormat:=xlNormal, Password:="", WriteResPassword:="", _
                ReadOnlyRecommended:=False, CreateBackup:=False
End Sub
```

Да Вы не ошиблись это код **VBA**. Этот код ваших операций. Конечно здесь нет циклов и массивов. Но здесь есть решение задачи. Если вы знаете как сделать в ручную, но не знаете как запрограммировать, запишите макрос, добавьте функциональность за счет выбора и циклов, продумайте адресацию. Но общая стратегия у Вас есть. Кроме того, если вы хотите запрограммировать, например, открытие файла **DBF** в **Excel**, то чего гадать с параметрами. Запишите макрос и посмотрите.

Шаг 8 - Коллекция Sheets

Данная коллекция представляет собой коллекцию листов (**Sheets**) в книге (**WorkBook**). Первое, что мы с Вами сделаем это получим количество листов в книге.

```
Sub Test()
        MsgBox (Str(Application.Workbooks.Item("Test.xls").Sheets.Count))
End Sub
```

Но под листом понимается не только клетки, но и диаграмма. Так же как и лист для расчетов диаграмма будет включена в подсчет листов. Как посмотреть имена листов. Просто. Есть свойство **Name**:

```
Sub Test()
        With Application.Workbooks.Item("Test.xls")
                For x = 1 To .Sheets.Count
                        MsgBox (Sheets.Item(x).Name)
                Next x
        End With
End Sub
```

А как же лист с формулами отличить от диаграммы ? Попробуйте так. **Type** вернет Вам тип. Только не знаю документированный способ это или нет.

```
Sub Test()
        With Application.Workbooks.Item("Test.xls")
                For x = 1 To .Sheets.Count
                        MsgBox (Sheets.Item(x).Type)
                        If Sheets.Item(x).Type = 3 Then
                                MsgBox Sheets.Item(x).Name
                Next x
        End With
End Sub
```

К коллекции листов есть возможность добавлять свои листы, для этого существует метод **Add**. Этот метод требует **4** параметра **Add(Before, After, Count, Type)**. Все эти параметры необязательные. Первые два отвечают за место вставки листа. Дальше количество вставляемых листов **Count** и тип листа. Типы могут быть, например, такие. **xlWorkSheet** для расчетного листа, **xlChart** для диаграммы. Если местоположение не указывать, то лист будет вставляться относительно текущего листа.

```
Sub Test()
        With Application.Workbooks.Item("Test.xls")
                Sheets.Add
        End With
End Sub
```

Метод **Parent** позволяет узнать какой книге принадлежит текущий лист.

```
Sub Test()
        With Application.Workbooks.Item("Test.xls")
                MsgBox (Sheets.Parent.Name)
        End With
End Sub
```

Если у Вас есть желание, то некоторые листы можно убрать из обзора. Это бывает полезно, если у Вас есть константы или расчеты, которые Вы не хотите чтобы видели на экране в виде листов. Для этого можно использовать метод **Visible**. Устанавливая это свойство в **TRUE** или **FALSE** вы сможете убирать и показывать лист.

```
Sub Test()
        With Application.Workbooks.Item("Test.xls")
        .Sheets.Item("Лист5").Visible = False
        End With
End Sub
```

Шаг 9 - Еще о Sheets

Один из полезный методов это метод **Copy**. Он позволяет создавать новый лист на основе существующего, то есть использовать лист как шаблон для других листов. Переименуйте любой лист в имя **Test**. Это можно сделать нажав правую кнопку мыши на названии листа и выбрав пункт меню **Переименовать**. Создайте на листе любое форматирование. **after** это лист, после которого произойдет вставка.

```
Sub Test()
        With Application.Workbooks.Item("Test.xls")
                Sheets("Test").Copy , after:=Sheets("Лист3")
        End With
End Sub
```

У метода **Copy** есть особенность. Если не указывать параметры, то будет создана новая книга с копируемым листом.

```
Sub Test()
        With Application.Workbooks.Item("Test.xls")
                Sheets("Test").Copy
        End With
End Sub
```

При необходимости передвинуть лист есть метод **Move**:

```
Sub Test()
        With Application.Workbooks.Item("Test.xls")
                Sheets("Test").Move , after:=Sheets("Лист3")
        End With
End Sub
```

Так как коллекция эта содержит объекты листа у неё есть несколько полезных методов. Один из них **PrintPreview** позволяющий вызывать предварительный просмотр.

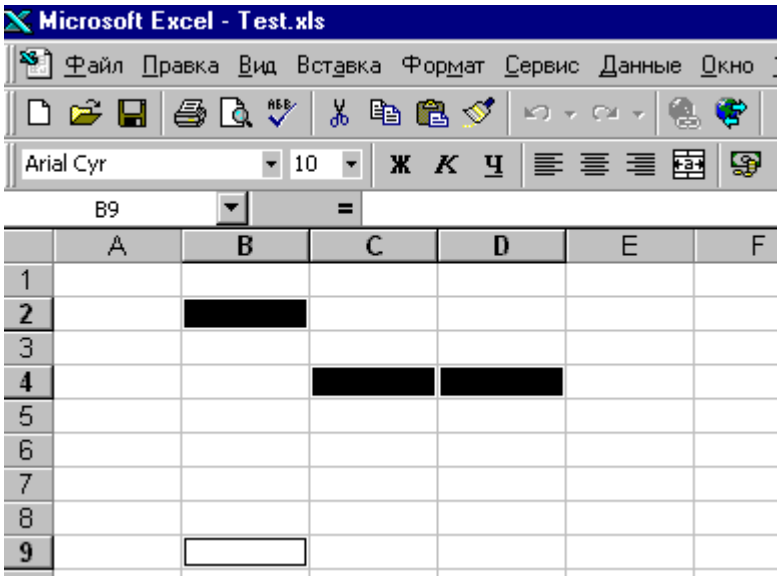
```
Sub Test()
        With Application.Workbooks.Item("Test.xls")
                Sheets("Test").PrintPreview
        End With
End Sub
```

Программным путем можно и выделять листы. Это метод **Select**. У него один параметр типа **BOOL**, если он установлен в **TRUE**, то происходит выделения листа, а если **FALSE**, то выделение объединяющее. Выделите другой лист и запустите следующий макрос.

```
Sub Test()
        With Application.Workbooks.Item("Test.xls")
                Sheets("Test").Select (False)
        End With
End Sub
```

Шаг 10 - Использование Range

Мы с вами будем использовать это свойство для выделения ячеек. Но прежде давайте просто посмотрим, как вообще можно их выделять. Конечно, если вы установите курсор на любую ячейку это то же выделение. Вы можете вводить в неё формулы или числа, менять формат. Выделить можно и несколько ячеек. Если установить курсор в одну ячейку и не отпуская левую кнопку мыши тащить, то выделится целый диапазон. Так же можно выделять отдельные ячейки, как на рисунке ниже.



Реализуется это довольно просто. Выделяете первую ячейку, держите клавишу **Ctrl** и не отпуская её другие ячейки. С выделенными диапазонами можно проводить большое количество операций. Вот для программной реализации этих возможностей и служит свойство **Range**. Оно есть у очень многих объектов **Excel**.

Этот пример показывает как выделить ячейку и поместить туда число.

```
Sub Test()
        With Application.Workbooks.Item("Test.xls")
                Worksheets("Лист2").Activate
                Range("A2") = 2
                Range("A3") = 3
        End With
End Sub
```

Следует обратить внимание, что для нормальной работы этого метода рабочая книга и лист должны быть активными, иначе возникнет ошибка. Кроме того, наверно, надо перед каждым его использованием указывать книгу и лист, с которым работаем, дабы не внести изменения в другой, чему конечно пользователь или Вы будете рады...

С помощью этого метода можно помещать и формулы:

```
.....
Range("A4") = "=A2+A3"
.....
```

Можно указать и целый диапазон:

```
.....
Range("A2:A5") = 2
.....
```

Если вам захочется наоборот получить из ячейки формулу или значение, то Вам ни кто не мешает сделать вот так:

```
Sub Test()
        With Application.Workbooks.Item("Test.xls")
                Worksheets("Лист2").Activate
                Range("A2") = 2
                Range("A3") = "=A2+2"
                MsgBox Range("A3").Formula + "  - " + Str(Range("A3").Value)
        End With
End Sub
```

Следующий пример очень важен, он показывает возможности абсолютной и относительной адресации. Мы создаем объект типа **Range**, а на основе его производим адресацию. В следующем примере число поместится не в **A1**, а в **D3**. То есть у нас есть возможность выделять диапазон по абсолютному адресу, а внутри его использовать относительную адресацию.

```
Sub Test()
        With Application.Workbooks.Item("Test.xls")
                Worksheets("Лист2").Activate
                Dim HelloRange As Range
```

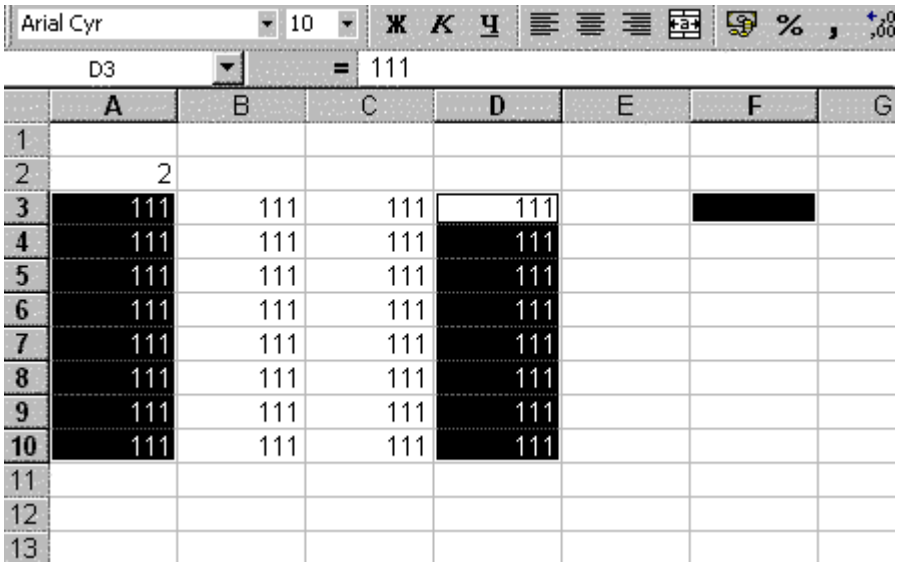
```
                Set HelloRange = Range("D3:D10")
                HelloRange.Range("A1") = 3
        End With
End Sub
```

Шаг 11 - Дальше о Range

Разговаривая о выделении ячеек с помощью **Range** мы с Вами должны знать, что возвращает этот метод множество. Это множество может состоять из одной или нескольких ячеек. А если множество, то информатика говорит о необходимости иметь возможность их объединения.

```
Sub Test()
        With Application.Workbooks.Item("Test.xls")
                Worksheets("Лист2").Activate
                Dim HelloRange As Range
                Set HelloRange = Range("D3:D10, A3:A10 , F3")
                HelloRange.Select
        End With
End Sub
```

А вот результат работы этого кода.



Раз есть объединения должно быть и пересечение, раз уж идет разговор о теории множеств. Получить его можно вот так.

```
Sub Test()
        With Application.Workbooks.Item("Test.xls")
                Worksheets("Лист2").Activate
                Dim HelloRange As Range
                Set HelloRange = Range("A1:A20  A8:D8")
                HelloRange.Value = "Hello"
        End With
End Sub
```

Будет выделена всего одна ячейка. Как Вы видите из предыдущих примеров отличаются типом обьявления - "**D3:D10, A3:A10**" это объединения, а "**A1:A20 A8:D8**" это пересечение. Используя пересечения и объединения можно строить область любого уровня сложности.

Получив объединение можно узнать количество ячеек.

```
Sub Test()
        With Application.Workbooks.Item("Test.xls")
                Worksheets("Лист2").Activate
                Dim HelloRange As Range
                Set HelloRange = Range("A1:A20,  D1:D20")
                MsgBox (Str(HelloRange.Count))
        End With
End Sub
```

Да пора бы уже рассказать о этой **Str**. Эта функция переводит число в строку. Вот пример:

```
Sub Test()
        Dim x As Integer
        x = 10
        Dim s As String
        s = Str(x)
        MsgBox (s)
End Sub
```

Или так с типом **Double** она сама определит как переводить, простая и умная, как женщина моей мечты :-)

```
Sub Test()
        Dim x As Double
        x = 10.333333
        Dim s As String
        s = Str(x)
        MsgBox (s)
End Sub
```

Шаг 12 - Обработка ошибок VBA

Программирование это как хождение по минному полю. Неизвестно где взорвешься. Наверно так. Вы наслышаны о том, что **Windows** напичкан ошибками, о том что среда разработки любая при том - тоже. Мне попадались исследования на эту тему. Типа, что на каждые **1000** строк кода одна ошибка, у хорошего программиста естественно :-). В общем это закон такой. Все равно ошибешься где-нибудь. Проводя аналогию между женщиной и компьютером :-))) вообщем понятно.

Для обработки ошибок в **VBA** и **VB** есть специальный оператор **On Error**. Его задача при возникновении ошибки передать управление в то место(процедура или кусок кода), в котором это ждут. Посмотрим пример:

```
Sub Test()
        On Error GoTo Errors1
        Dim x As Integer
        Dim a As Integer
        Dim c As Double
        x = 20
        a = 0
        c = x / a
        MsgBox (" Этого не должно быть")
        GoTo Ends:
Errors1:
        MsgBox ("Ну ты блин Тикурила Даещь")
Ends:
End Sub
```

В данном примере при возникновении ошибки управление передается по метке **Errors1** и дальше выполняется код. Я понимаю, что прерывать функцию из-за ошибки не всегда надо. И не только я так думаю, создатели **VBA** тоже так считали, и поэтому есть оператор **Resume Next**. Этот оператор реализует небезызвестный принцип - Ни шагу назад. Выполнение пойдет дальше, несмотря на ошибку.

```
Sub Test()
        On Error GoTo Errors1
        Dim x As Integer
        Dim a As Integer
        Dim c As Double
        x = 20
        a = 0
        c = x / a
        MsgBox ("Опаньки !!!")
        GoTo Ends:
Errors1:
        MsgBox ("Ну ты блин Тикурила Даещь")
        Resume Next
Ends:
End Sub
```

А вот, если Вы вообще не хотите ничего говорить по поводу ошибки, то можете поступить очень сурово. Вот так. Я рекомендую применять это для бухгалтерских расчетов. Ни кто и не догадается :-)))

```
Sub Test()
        On Error Resume Next
        Dim x As Integer
        Dim a As Integer
        Dim c As Double
        x = 20
        a = 0
        c = x / a
        x = 10
        a = 3
        c = 10 / 3
        MsgBox ("Опаньки !!!")
End Sub
```

Над резюме можно немного поэкспериментировать, вот возможные описания:

```
Resume Next
Resune строка
Resume метка
Resume 0
```

Пример ниже будет упорно требовать, чтобы ввели число отличное от 0:

```
Sub Test()
        On Error GoTo Error1
        Dim x As Integer
        Dim a As Integer
        Dim c As Double
        x = 20
        a = Str(InputBox("введите число"))
        c = x / a
        x = 10
        MsgBox ("Опаньки !!!")
        GoTo Ends:
Error1:
        MsgBox ("думай о программировании, а не о женщинах")
        a = Str(InputBox("введите число"))
        Resume 0
Ends:
End Sub
```

Шаг 13 - Объект Err

Да, странное совпадение, 13 шаг и зловещий объект **Err**, от которого одни неприятности. Этот объект хранит информацию о последней ошибке в результате выполнения того, что вы запрограммировали. Ну давайте попробуем.

```
Sub Test()
        On Error GoTo Error1
        Sheets.Item(1000).Delete
        GoTo Ends
Error1:
        MsgBox "Error detected"
        MsgBox (Str(Err.Number))
        MsgBox (Err.Source)
        MsgBox (Err.Description)
Ends:
End Sub
```

Итак, **Number** - это номер ошибки **Source**, где она появилась, а **Description** описание. В данном случае Вам скажут о выходе за границу массива. Вот это здорово. Особенно при создании программ. Получить такое сообщение пользователю не очень приятно, а вот программисту :-)) даже думать не надо.

У объекта **Err** есть метод очистки **Clear**, он все очищает. Вот в этом случае Вы не получите никаких сообщений. После обработки ошибки неплохо применить этот метод. Так, ради профилактики.

```
Sub Test()
        On Error GoTo Error1
        Sheets.Item(1000).Delete
        GoTo Ends
Error1:
        Err.Clear
        MsgBox "Error detected"
        MsgBox (Str(Err.Number))
        MsgBox (Err.Source)
        MsgBox (Err.Description)
Ends:
End Sub
```

Нельзя не сказать, что этот объект автоматически очистится после ..

```
Resume
Exit Sub(Function)
On Error
```

При отладке или специально в программе вы и сами можете сгенерировать ошибку методом **Raise**, только надо знать, что ошибки до **1000** зарезервированы **VBA**, а максимальный код **65535**. Любое правило подвержено изменениям и поэтому есть специальная константа, от которой вы можете сложением получать коды ошибок. Она называется **vbObjectError**.

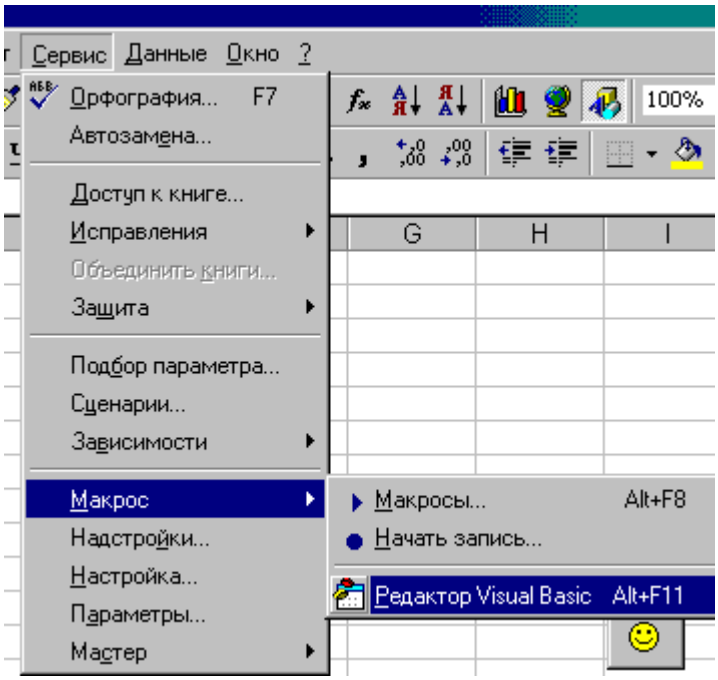
```
Sub Test()
        On Error GoTo Error1
        Err.Raise 1001, "Test()", "Это я сделал"
Error1:
        MsgBox "Error detected"
        MsgBox (Str(Err.Number))
        MsgBox (Err.Source)
        MsgBox (Err.Description)
Ends:
End Sub
```

Шаг 14 - События объектов

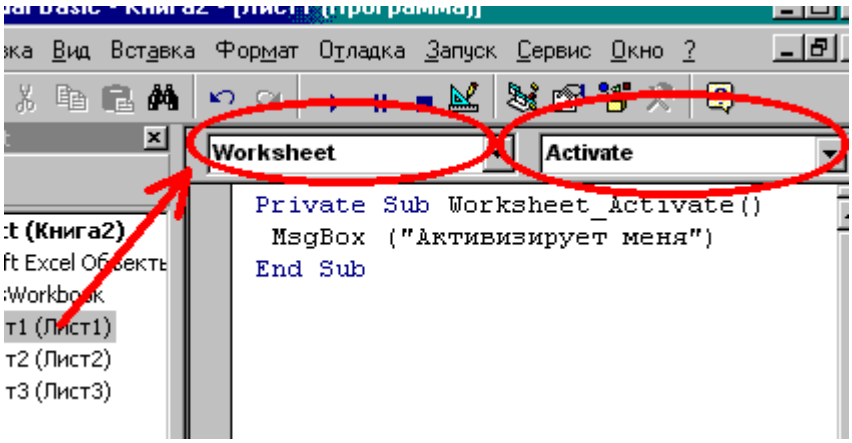
Обработать можно события следующих объектов **Excel**:

```
Application
WorkBoor
WorkSheet
Chart
```

Функции обработки создаются автоматически. Открывайте **Excel** и любую книгу. Запускайте редактор **VBA**.



Появится редактор **VBA**. Нас интересует список объектов в окне **VBAProject**. Выберите **Лист1** и два раза щелкните по нему. Появится белое окно. Вам нужно выбрать объект и событие, смотрите как на рисунке ниже, да я чуть не забыл, код **VBA** еще нужен.



Пришло время испытаний. Переключитесь на **Лист2**, потом назад на **Лист1** должно появиться диалоговое окно о том, что активизирован лист. Это очень полезно. Например у Вас есть скрытый лист, пользователь открывает его и пробует смотреть, а Вы ему **format.com** за это :-). Хотя не смешно, Вам и восстанавливать.

Многие события имеют параметры. Вот как это.

```
Private Sub Worksheet_BeforeRightClick(ByVal Target As Excel.Range, Cancel As Boolean)
        .....
End Sub
```

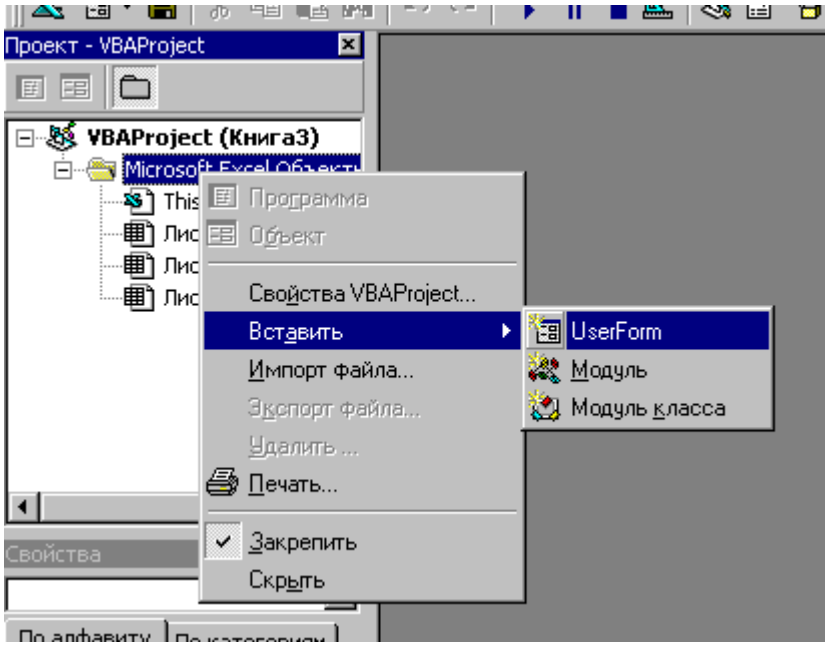
И еще события посылают не только по иерархии объектов вниз, но и вверх. Вот то же событие активации обрабатывается и на верхнем уровне.

```
Private Sub Workbook_SheetActivate(ByVal Sh As Object)
        MsgBox (Sh.Name)
End Sub
```

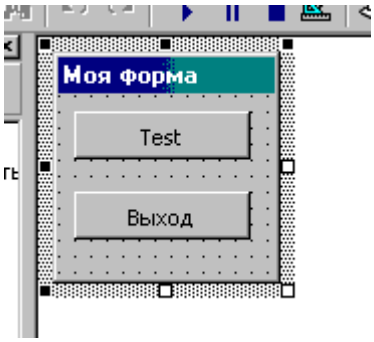
Попробуйте его создать и проверить.

Шаг 15 - Пользовательские формы

Создавайте новую книгу. Запускайте редактор **VBA**, как в прошлый раз. Наводим курсор на **Microsoft Excel Объекты**, правую кнопку мыши, вставить **UserForm**. Да смотрите сами ниже. Чего это Я.



У Вас появится диалоговое окно (форма), панель инструментов, и окно свойств. В окне свойств нас интересует свойство **Caption**, которое позволяет нам изменить заголовок окна. Поменяйте его на осмысленное имя, например, первая форма :-). Разместите на форме кнопки. Выбираете инструмент кнопка. Не знаете какой он ? Подводите к каждому и задержите мышку, Вам подскажут. Нам надо две кнопки. Одна с именем "тест", а вторая с именем "выход". Имя у кнопок так же меняется в **Caption**. Все должно быть вот так.



Пора добавить к кнопкам код. Это просто. Двойной щелчок на кнопке и вы попадете в код вызываемый при нажатии кнопки. Давайте для кнопки **Test**.

```
Private Sub CommandButton1_Click()
        MsgBox ("Test Button Press")
End Sub
```

И для второй:

```
Private Sub CommandButton2_Click()
        Unload Me
End Sub
```

Unload Me выгружает форму из памяти. А вот теперь нам нужно создать макрос для загрузки. Создавайте макрос с именем **FormsRun** или другим. Код ниже.

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

Вот теперь запустите макрос. Появится диалоговое окно. Нажмите кнопку **Test**, появится сообщение, нажмите кнопку **Выход**, окно диалога закроется.

Шаг 16 - Чтение и запись текстовых файлов

Наверно можно смело утверждать, что умение читать и записывать информацию в текстовый файл это основа импорта и экспорта :-) Практически любая серьезная программа хранящая информацию позволяет сохранить её в текстовом формате, какое бы он расширение не имел. Открываются файлы командой **Open**.

```
Sub Test()
        Open "c:\1.txt" For Input As #1
        Close #1
End Sub
```

Команда **Open** может открывать для чтения **Input** и для записи **Output**. Цифра после **as** это идентификатор файла. На основании его производится чтение и запись файла.

Следующий пример демонстрирует запись и чтение файла

```
Sub Test()
        Open "c:\1.txt" For Output As #1
        Print #1, "Hello File"
        Close #1

        Open "c:\1.txt" For Input As #1
        Dim s As String
        Input #1, s
        MsgBox s
        Close #1
End Sub
```

Как видите, для записи можно использовать **Print**, а для чтения **Input** воспользовавшись идентификатором открытого файла. Естественно здесь свои тонкости работы. Вот, если Вы запишите такую строку:

```
......
Print #1, "Hello , File"
......
```

То оператор **Input #1** прочитает только **Hello** и все. Запятая воспринимается как разделитель. И это правильно. Есть форматы текстовых файлов когда числа разделены запятой. В коде ниже:

```
......
Input #1, s
MsgBox s
Input #1, s
MsgBox s
.....
```

Последовательно выведутся надписи **Hello** и **File**, но с этим можно бороться оператором **Line Input**.

```
Sub Test()
        Open "c:\1.txt" For Output As #1
        Print #1, "Hello , File"
        Close #1

        Open "c:\1.txt" For Input As #1
        Dim s As String
        Line Input #1, s
        MsgBox s
        Close #1
End Sub
```

Этот код прочитает строку целиком. Следом возникает важный вопрос, а как узнать конец файла ? Для этого есть функция **EOF(идентификатор)**, которая позволяет вам определить конец файла.

```
Sub Test()
Open "c:\1.txt" For Output As #1
        Print #1, "Hello , File"
Close #1
Open "c:\1.txt" For Input As #1
        Dim s As String
        While Not EOF(1)
                Input #1, s
                MsgBox s
        Wend
Close #1
End Sub
```

Шаг 17 - Win32 API и VBA

На данный момент использование **Win32 API** является стандартом для любой среды или языка программирования, это и понятно, как иначе писать программы для **Windows** ? Вместе с тем пользоваться этим же **API** надо осторожно, реализации в версиях **Windows** отличаются вплоть до присутствия некоторых функций. Для того, чтобы использовать функции **Win 32 API** их необходимо объявить, используя **Declare**.

В общей области (в описании) надо объявить функцию. Сделать это можно поднявшись на самую верхнюю строчку окна редактирования макроса и ввести описание.

```
Declare Function GetWindowsDirectory Lib "kernel32" Alias "GetWindowsDirectoryA" (ByVal buffer As String, ByVal nSize As Long) As Long
```

Вот тут-то Вы и должны быть поражены. Говорят **VBA** это для ..... Так вот. Использовать подобную функцию на **VC++** намного проще. Во-первых, Вам наверно всё равно, где она находится :-))) в **kernel**, **user** или **gdi**, и вам вообще-то и не надо знать её имя в виде **GetWindowsDirectoryA**, а если вы пользуетесь каркасной библиотекой типа **MFC**, то часто получаете упрощенный вид функции типа **AfxMessageBox**. Вот и думай теперь чего проще :-))

Давайте на **Declare** посмотрим повнимательнее. У него два синтаксиса для функции или процедуры.

```
REM то что в скобках необязательно
[Public или Private] Declare Sub имя_процедуры lib "имя_динамической_библиотеки" [Alias "псевдоним"] [(параметры)]
[Public или Private] Declare Function имя_процедуры lib "имя_динамической_библиотеки" [Alias "псевдоним"] [(параметры)] [as тип возврата]
```

Вот так надо знать где находится и псевдоним, если нужно и все параметры. Вот он язык для домохозяек :-)

А теперь применение. Вот тут все стало опять просто.

```
Sub Test()
        Dim buffer As String
        Dim lens As Long
        buffer = String(256, 0)
        lens = GetWindowsDirectory(buffer, Len(buffer))
        buffer = Left(buffer, lens)
        MsgBox (buffer)
End Sub
```

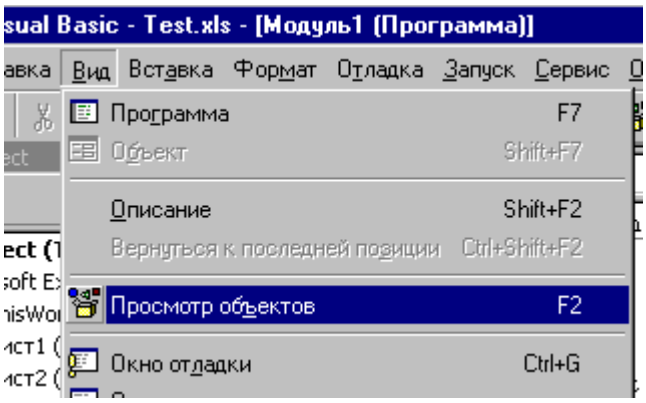
Объявляю переменные, **buffer = String(256, 0)** - заполняю строку нулями имитируя строку символов **char**. Зачем ? Ну есть подозрение, что **Windows** написан на **C** или **ASM**, даже без **++** и поэтому другого он не понимает :-), не на бейсике точно. Вызываем функцию, передавая параметры. Полученную строку обрезаем функцией **Left**.

Наверно лучше создавать описания в отдельном модуле и просто его экспортировать в проект, дабы не мучаться. И, наверно, есть уже готовые модули. Но этот метод позволяет Вам подключить любую динамическую библиотеку. Посмотрите в разделе **MFC** шаг за шагом : "Шаг 46 - Dll для Excel".

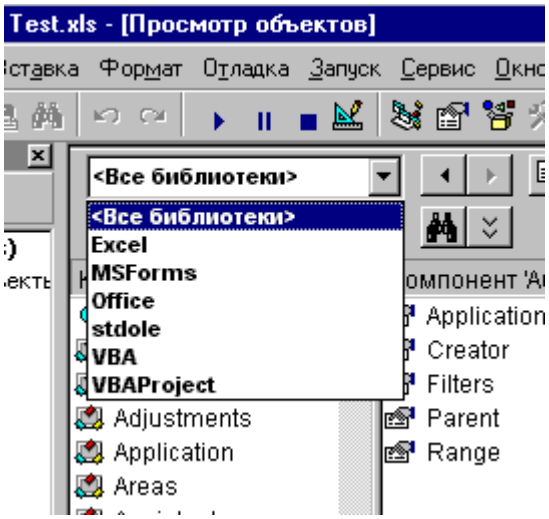
Шаг 18 - Просмотр объектов

Весь **Office 97** можно рассматривать как набор объектов. Кроме того операционная система предоставляет дополнительные объекты. Каждый объект имеет свои свойства. Объекты операционной системой предоставляются с использованием технологии **OLE** и интерфейсов следующих поколений на базе него.

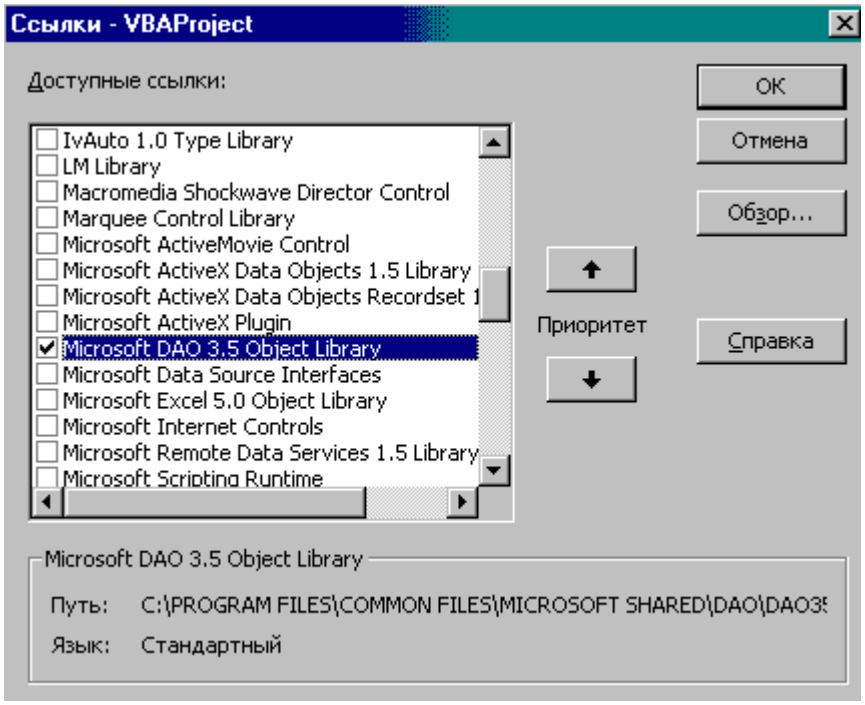
Редактор **VBA** позволяет вам просмотреть доступные Вам объекты и их свойства. Для этого Вам необходимо запустить **Просмотр объектов** из меню **Вид** (**F2**).



Появится окно. Оно разделено на 3 части. Вверху библиотеки, справа объекты библиотек, а слева свойства. По умолчанию у Вас подключаются библиотеки необходимые по мнению создателей **Office**. В верхнем окне у Вас должно быть написано **Все библиотеки**. Давайте посмотрим какие присутствуют.



Как вы видите количество библиотек ограничено. Вот тут Вы должны возмутиться, а где библиотеки **ACCESS** или **DAO**. И вообще мало .... Не мало. Просто их необходимо подключить. Точнее надо. Если вы будете работать с базами данных или хотите расширить возможности среды, то их нужно подключать. Делается это из меню **Сервис-Ссылки**. Выберите этот пункт меню.



Подключаем **Microsoft DAO 3.5 Object Library**. Без этой библиотеки Вы не сможете работать с базами данных на основе **DAO**. Только библиотека должна быть зарегистрирована в системе, иначе в

списке её не будет. Что делать тогда ? В окне подключения библиотек есть кнопка **Обзор**, которая позволяет Вам подключить их используя, например, **TLB** файлы.

Что же произойдет после подключения? В списке объектов появиться **DAO** и все связанные с ним свойства.

Шаг 19 - Информация о типе переменной

А зачем, вы спросите, иметь информацию о типе переменной в ходе работы программы ? Ведь это делает программист. Опаньки :-). В **VBA** есть тип переменной **Variant**, который может быть любого типа за исключением пользовательского. Не верите ? Смотрите код:

```
Sub Test()
        Dim string_var As String
        Dim int_var As Integer
        Dim test_variant As Variant
        string_var = "Hello Variant"
        int_var = 123
        test_variant = string_var
        test_variant = int_var
End Sub
```

Как видите, и **Variant** можно передавать в процедуры, поэтому определение типа нужно, конечно если подобными вещами вы будете пользоваться. Для определения кода есть функция **TypeName (...)**, которая вернет строку с именем переменной. Вот так, например, можно её использовать:

```
Sub Test()
        Dim string_var As String
        Dim int_var As Integer
        Dim test_variant As Variant
        string_var = "Hello Variant"
        int_var = 123
        test_variant = string_var
        MsgBox (TypeName(test_variant))
        test_variant = int_var
        MsgBox (TypeName(test_variant))
End Sub
```

Кроме этого есть ряд вспомогательных функций позволяющих получить информацию о переменных. **IsArray** позволяет проверить является ли переменная массивом.

```
Sub Test()
        Dim arr_var(10) As String
        If IsArray(arr_var) Then
                MsgBox ("Массив")
End Sub
```

IsEmpty проверка инициализации (наличия) переменной. Запустите код ниже, а потом раскомментируйте строку.

```
Sub Test()
        ' Dim arr_var As String
        If IsEmpty(arr_var) Then MsgBox ("NO")
End Sub
```

IsDate проверяет можно ли преобразовать переменную к типу даты. Ниже надпись **YES** появится один раз.

```
Sub Test()
        Dim arr_var As String
        arr_var = "01.01.1998"
        If IsDate(arr_var) Then
                MsgBox ("YES")
        arr_var = "41.01.1998"
        If IsDate(arr_var) Then
                MsgBox ("YES")
End Sub
```

Так же проверяется можно ли перевести в число **IsNumeric**:

```
Sub Test()
        Dim arr_var As String
```

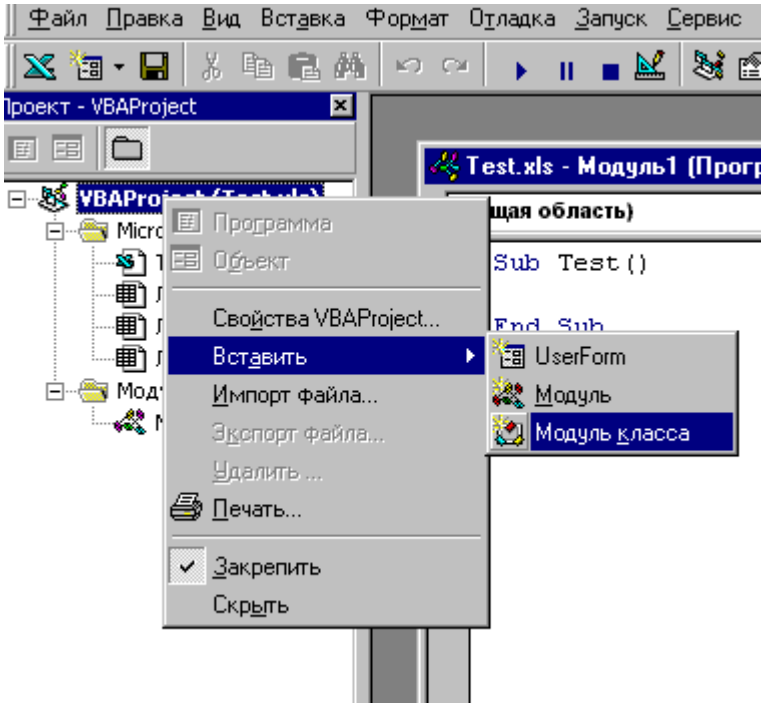
```
        arr_var = "not numeric"
        If IsNumeric(arr_var) Then MsgBox ("YES")
        arr_var = "1998"
        If IsNumeric(arr_var) Then MsgBox ("YES")
End Sub
```

Есть еще ряд подобных функций:

- **IsObject** - проверка, что переменная объект
- **IsNull(выражение)** - проверка на пустое значение
- **IsError(выражение)** - проверка выражения, представляет ли оно значение ошибки

Шаг 20 - Пользовательские классы

В **VBA** есть свои классы, но можно создавать и самим. Для этого в проект необходимо добавить модуль класса. Это можно сделать щелкнув правой кнопкой мыши на проекте и выбрав пункт меню **вставить -> модуль класса**.



В результате у Вас появится окно для кода класса, и в окне просмотра проекта появится значок класса. Вероятнее всего с именем **Класс1**. Объявим переменные для внутреннего использования. **Private** говорит о том, что использоваться они будут только внутри класса.

```
Private NamePiple As String
Private DatePiple As String
```

Теперь создадим функцию **GetPipleName**. Пишите ниже:

```
Public Sub GetPipleName()
        NamePiple = InputBox("Enter Name - ")
End Sub
```

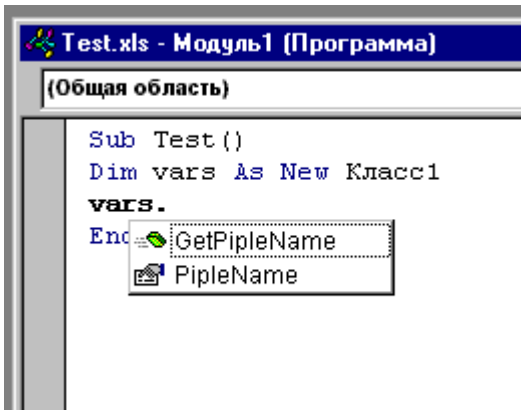
Теперь свойства для получения имени. Пишите ниже:

```
Property Get PipleName() As String
        PipleName = NamePiple
End Property
```

И для установки тоже. Пишите ниже:

```
Property Let PipleName(s As String)
        NamePiple = s
End Property
```

Закрывайте редактор и открывайте любой макрос для редактирования, если его нет создайте. Начинайте вводить код, как на рисунке ниже. И о чудо !!! Наш класс имеет те же возможности, что и встроенный класс **VBA**, он показывает свойства.



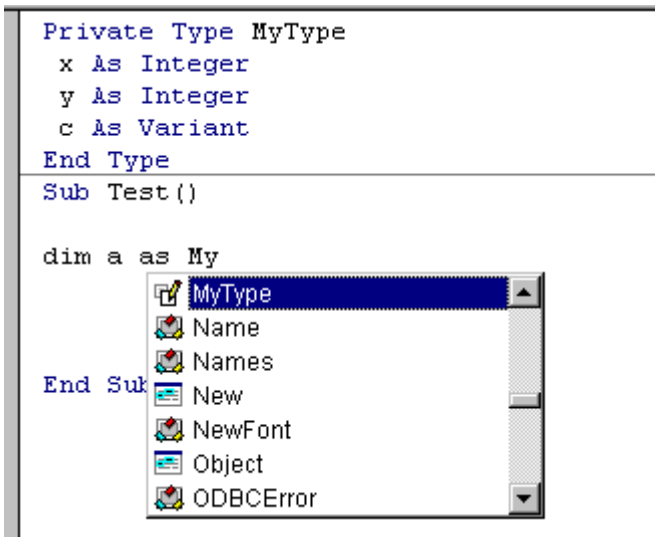
Настало время испытать его в действии:

```
Sub Test()
        Dim vars As New Класс1
        vars.GetPipleName
        MsgBox vars.PipleName
        vars.PipleName = "VBA"
        MsgBox vars.PipleName
End Sub
```

По удобству и простоте это круче **C++** и **MFC** и так далее. Кроме того класс легко сохранить для дальнейшего использования. Вообщем класс. Просто оцените эту возможность даже если вы читаете просто так.

Шаг 21 - Пользовательские типы

В общей части Вы должны описать структуру и объявить переменную типа этой структуры. Смотрите картинку.



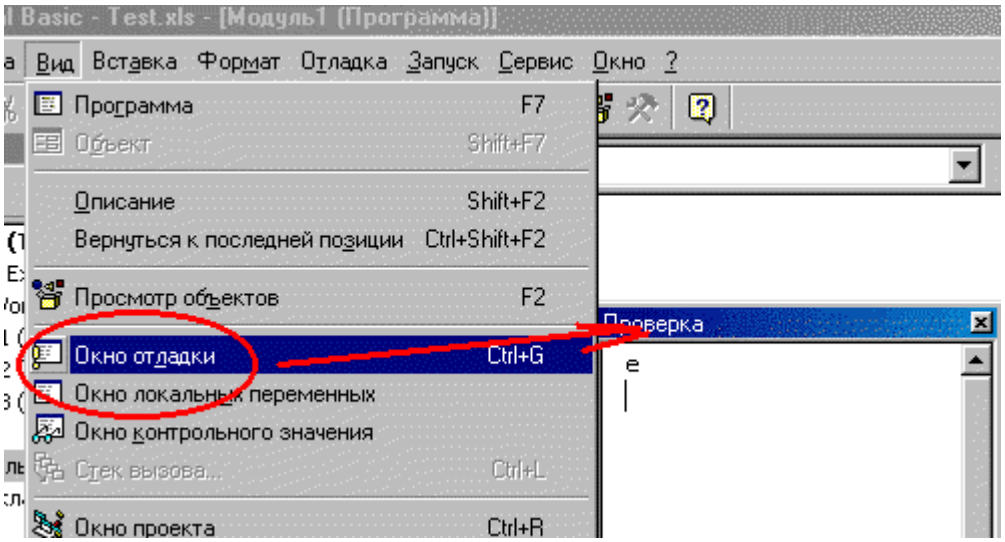
Итак мы создадим тип с элементами **x,y**, типа **Integer** и **c** с типом **Variant**. Ох уж этот **Variant**, он позволяет нам в структуру помещать все, что угодно. Хоть массив. Мы то сделаем. Но разве Вас не впечатляет простота и мощь. Ведь так можно создать структуры любой сложности очень просто. Нет **OLE**, нет указателей. После **C++** это просто сказка.

```
' Описание
Private Type MyType
        x As Integer
        y As Integer
        c As Variant
End Type

Sub Test()

'-------------------------
' Область функции
```

```
Sub Test()
        Dim a(2) As MyType
        Dim arrays(2) As String
        arrays(1) = "Hello"
        arrays(2) = "Cool"
        a(1).x = 1
        a(1).y = 2
        a(1).c = arrays
        a(2).x = 10
        a(2).y = 20
        arrays(1) = "Hello 2"
        arrays(2) = "Cool !!!"
        a(2).c = arrays
        Debug.Print "a1"
        Debug.Print a(1).c(1)
        Debug.Print a(1).c(2)
        Debug.Print "a2"
        Debug.Print a(2).c(1)
        Debug.Print a(2).c(2)
End Sub
```

Во всем этом есть и подводный камень. Создавать структуры можно только на уровне модуля. Не зря она у нас **Private**. Но можно создавать классы, которые умеют с ними работать :-) Это несколько радует.

Для просмотра результата Вам необходимо открыть окно отладки. Это можно сделать в меню **Вид - Окно отладки** (**CTRL-G**).



Вот такой результат вы должны увидеть:

```
a1
Hello
Cool
a2
Hello 2
Cool !!!
```

Шаг 22 - For Each

Этот цикл придуман для того, чтобы облегчить действия над массивами и наборами. Он позволяет произвести однотипные операции над всем массивом. Описание его такое:

```
For Each переменная In массив

        Действия

Next переменная
```

Переменная должна иметь тип **Variant** или **Object**, в котором можно хранить практически всё. Пример ? Пожалуйста.

```
Sub Test()
	Dim arrays(1) As String
	arrays(0) = "Hello"
	arrays(1) = "Each :-)"
	Dim vari As Variant
	For Each vari In arrays
		MsgBox (vari + " - Steps")
	Next vari
End Sub
```

For Each очень удобен для работы с коллекциями. Вот так можно пробежаться по открытым книгам.

```
Sub Test()
	Dim vars As Variant
	For Each vars In Workbooks
		MsgBox (vars.Name)
	Next vars
End Sub
```

Или по листам книги:

```
Sub Test()
	Dim vars As Variant
	For Each vars In Workbooks.Item("Test.xls").Sheets
		MsgBox (vars.Name)
	Next vars
End Sub
```

Шаг 23 - Работа с каталогами

Первым делом определим откуда запущенно приложение. У объекта **Application** есть свойства **path**, которое и позволит нам получить информацию.

```
Sub Test()
	MsgBox (Application.Path)
End Sub
```

Команды создания и удаления каталогов очень похожи на **DOS** аналоги. Это **MkDir** и **RmDir**. Ниже создаем каталог на диске **C**.

```
Sub Test()
	MkDir ("c:\test")
End Sub
```

И удаляем.

```
Sub Test()
	RmDir ("c:\test")
End Sub
```

А вот теперь важный вопрос. Помещается ли удаленный каталог в корзину. Нет не помещается. Это очень возмутительно. Почему ???? Ведь программа создана для **Windows** с использованием среды разработки от **Microsoft** и такое возмутительное безобразие. Ладно **Linux** им судья. Для того, чтобы убедиться в этом запустите следующий код и загляните в корзину.

```
Sub Test()
	MkDir ("c:\test")
	RmDir ("c:\test")
End Sub
```

Для получения текущего каталога есть функция **CurDir**.

```
Sub Test()
	MsgBox (CurDir)
End Sub
```

Для того, чтобы сменить каталог тоже есть функция - **chdir**:

```
Sub Test()
	ChDir ("c:\windows")
	MsgBox (CurDir)
End Sub
```

Команда **dir** позволяет просмотреть все файлы в каталоге. Только использование её несколько специфично. Сначала Вы вызываете **dir** с параметрами и получаете первое имя файла, в дальнейшем можно вызвать её без параметров и получить следующее имя и так до тех пор, пока не вернется пустое имя файла.

```
Sub Test()
        Dim s As String
        s = Dir("c:\windows\inf\*.*")
        Debug.Print s
        Do While s <> ""
                s = Dir
                Debug.Print s
        Loop
End Sub
```

Результат работы в Окне отладки (**Ctrl-G**).

Шаг 24 - Использование Automation

Использование элементов **ActiveX** на базе модели **COM** - компонентная модель объектов, позволяет создавать сложные составные документы, то есть там могут находиться материалы из разных программ - **Excel**, **Access**, **PowerPoint** и так далее. Кроме этого есть возможность пользоваться другим приложением для решения задач. Например **Excel** может использовать **Access** для хранения данных или наоборот **Access** может использовать **Excel** для расчетов. Вообщем это можно назвать построением пользовательских приложений на базе готовых программ.

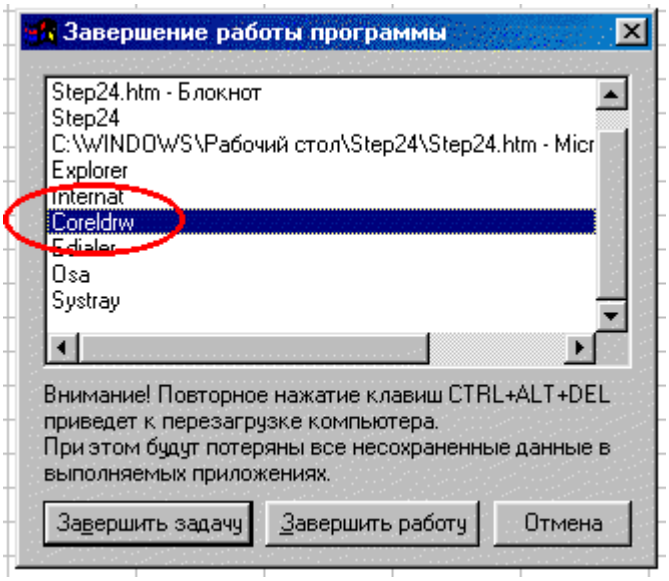
Понятие, которое используется в основе всех интегрированных систем является служба. **MS OFFICE** обеспечивает все необходимые службы для создания офисных приложений:

- **ACCESS** - База данных
- **EXCEL** - Расчеты
- **WORD** - Текстовый редактор
- **PowerPoint** - Презентационная графика
- **Office Binder** - Интеграция документов
- **Outlook** - Служба управления документами
- **Internet Exploler** - Работа с Интернет

Объект с вашим приложением можно связать используя позднее и ранее связывание. Позднее связывание происходит на этапе выполнения кода и для него используется понятие **Object**. Ниже будет приведен код для программы **Corel Draw** и использование её в качестве объекта для **Automation**.

```
Sub Test()
        Dim objCorel As Object
        Set objCorel = CreateObject("CorelDraw.Graphic.8")
        MsgBox ("press")
End Sub
```

В момент когда на экране появится сообщение **PRESS** нажмите **Ctrl-Alt-Delete** для просмотра активных объектов. Вот смотрите ниже.



Для позднего связывания используется меню **Сервис - Ссылки**, в предыдущих шагах мы об этом пункте меню упоминали. Вот пример для **Excel**.

```
Sub Test()
        Dim objExcel As Excel.Application
        Set objExcel = CreateObject("Excel.Application")
End Sub
```

Ну и напоследок как можно использовать объект **Word** из **Excel**:

```
Sub Test()
        Dim objWord As Word.Application
        Set objWord = CreateObject("Word.Application")
        MsgBox (objWord.Caption)
        MsgBox (objWord.UserName)
End Sub
```

Шаг 25 - О функции SendKeys

Эта функция позволяет имитировать ввод с клавиатуры в Окно вот её описание:

SebdKeys строка, [режим ожидания]

Этот макрос прокрутит таблицу на страницу вниз.

```
Sub Test()
        SendKeys ("{PGDN}")
End Sub
```

Режим ожидания это как будет произведен возврат. Если **TRUE** возврат в процедуру будет только после обработки кодов. Обработка может быть длительной, если у Вас есть обработчики событий. **FALSE** вернет сразу ничего не ожидая.

Вы обратили внимание на фигурные скобки. В них указываются команды и символы:

```
+
^
%
~
(
)
DEL {DEL}
INS {INS}
```

и так далее :-) догадаетесь?

```
{BS} {BREAK} {CAPSLOCK} {ENTER} {DOWN} {PGUP}
```

Это не все, но направление понятно.

Функция ниже переведет указатель на страницу ниже, введет **123** и даже **ENTER** нажмет :-)

```
Sub Test()
        SendKeys ("{PGDN}")
        SendKeys ("123{ENTER}")
```

```
End Sub
```

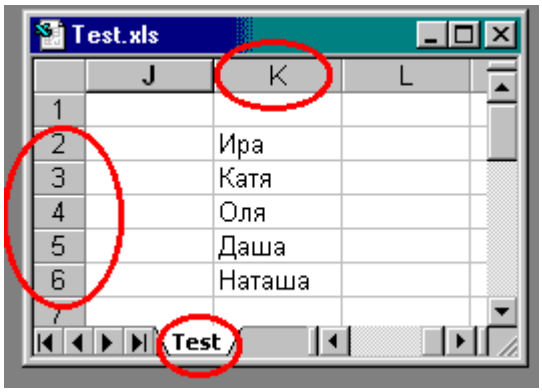
Вот так можно вызвать функциональную клавишу:

```
Sub Test()
        SendKeys ("{F1}")
End Sub
```

Когда экспериментируете запускайте макрос из активной рабочей книги.

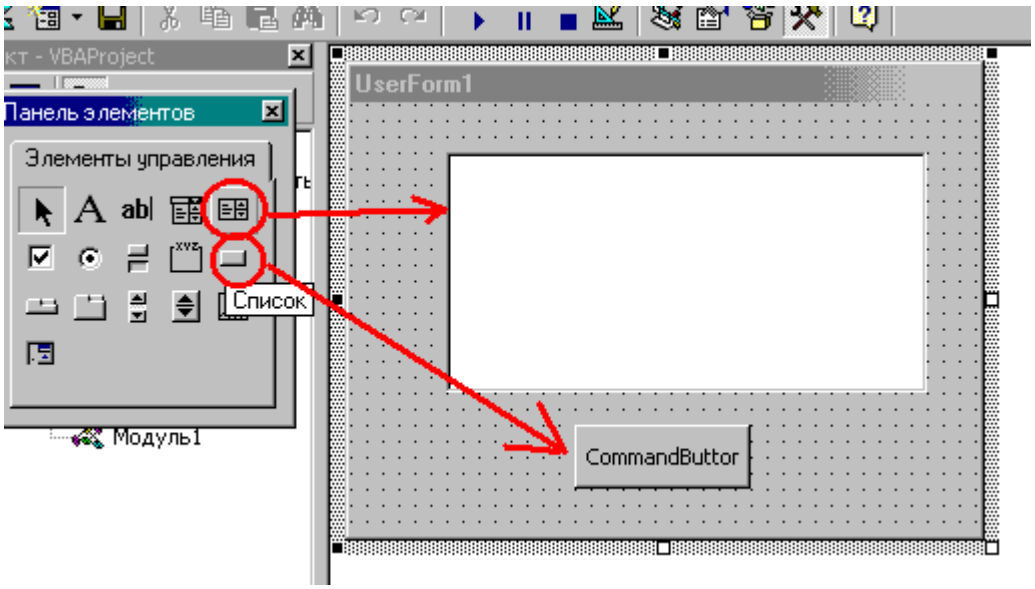
Шаг 26 - Заполнение списка на форме из таблицы

Итак, задача простая. У нас есть в таблице список девушек и мы ходим создать макрос, который будет зачитывать этот список и выводить в диалоговом окне. Вот под эту таблицу создавался макрос.



Создайте точно такую таблицу, если не трудно :-)

Теперь переходим в редактор **Visual Basic** и создаем форму. На эту форму надо поместить два элемента управления: кнопку и список. Вот она какая.



Щелкайте два раза по кнопке и вы попадете в редактирование события нажатия. Введите код:

```
Private Sub CommandButton1_Click()
        Unload Me
End Sub
```

Что означает "уничтожь меня" :-). То есть форму. Это **me** похожа на **this** в **C++** и идентифицирует объект, в котором производятся события. Даже спрашивать не надо, где я нахожусь. **Unload** и всё.

Заполнять список мы будем при активизации формы. Поэтому нам необходимо обработать событие инициализации. Щелкните по форме два раза и выберите из меню событий (справа) **Activate**. И код.

```
Private Sub UserForm_Activate()
        ' Активизируем нужный лист
        Worksheets.Item("Test").Activate
        ' Выбиделяем диапазон
        Dim девушки As Range
```

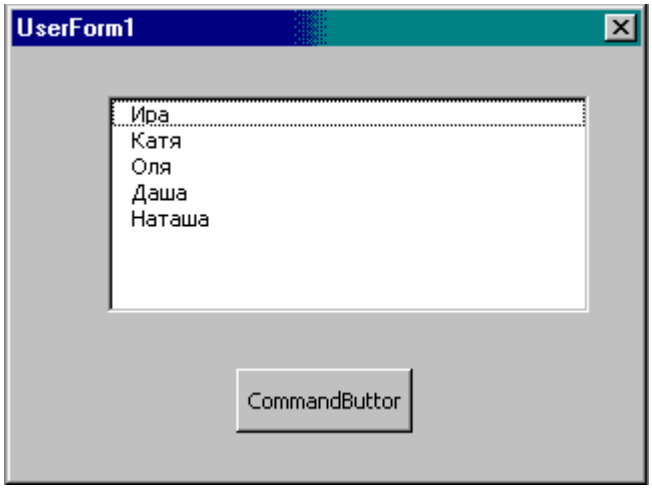
```
        ' Объект выделения
        Set девушки = Range("K2:K6")
        ' Выделяем
        Dim vars As Variant
        ' Пойдем по девчатам :-)
        For Each vars In девушки
                ' Добавляем в список
                UserForm1.ListBox1.AddItem (vars)
        Next vars
End Sub
```

Код прокомментирован и наверно понятен. Использование русских слов для переменных это не ошибка. Наконец это делать можно. Здесь в **VBA** можно использовать русские буквы для имен переменных.

И макрос для запуска формы:

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

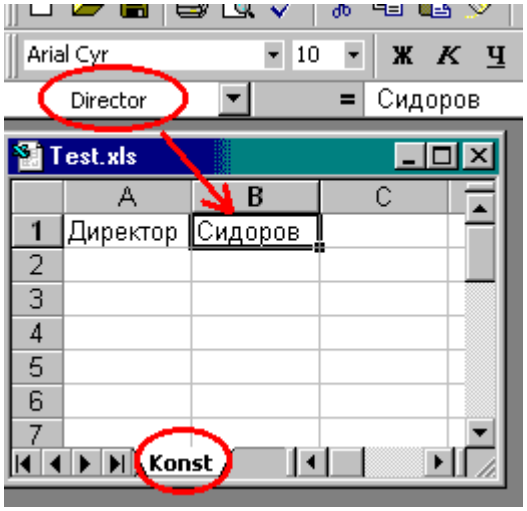
В макросе написано - "форма покажись". Ну вот. Запускайте. Вот как это получится:



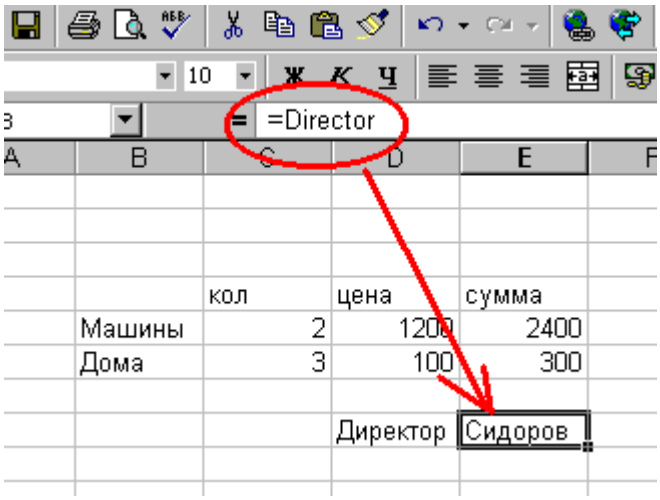
Шаг 27 - Обмен данными между формой и таблицей

Задача. Я хочу сделать ряд справок и страницу с константами. Одной из констант будет фамилия директора, которая используется в справках. При изменении этой константы фамилия в справке должна автоматически меняться. И я хочу менять фамилию из формы.

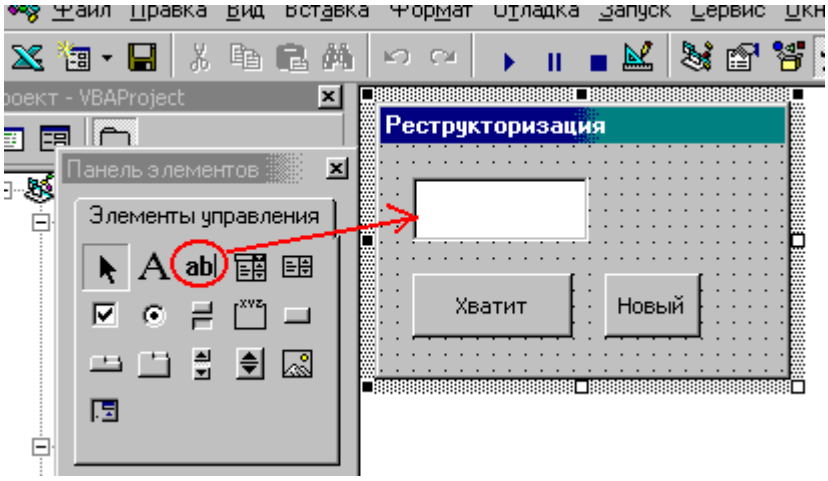
Создаем лист Константы и на нем ячейке даем имя.



И любое количество листов со справками. Ссылаясь на ячейку с фамилией.



Как вы понимаете, сколько я листов не создам, воспользовавшись ссылкой на ячейку **=director**, стоит мне изменить данные в ячейки с фамилией директора она везде поменяется. Это само нормально. Вот только менять я хочу из формы, например, чтобы с константами спрятать лист подальше от пользователя. Ну давайте создавать форму. Идем в редактор **VBA**:



При запуске формы мы должны прочитать данные с листа:

```
Private Sub UserForm_Activate()
        Worksheets.Item("Konst").Activate
        UserForm1.TextBox1.Text = Range("Director").Text
End Sub
```

При нажатии на кнопку **"Новый"** заменить данные на листе константы (автоматически поменяются на справках):

```
Private Sub CommandButton2_Click()
        Range("Director").Value = UserForm1.TextBox1.Text
End Sub
```

По нажатию на **"Хватит"** закрыть форму:

```
Private Sub CommandButton1_Click()
        Unload Me
End Sub
```

Как видите использование констант в тех случаях, где это разумно на отдельном листе позволяет просто создать форму. Потом форму можно скрыть и вызвать из меню для замены значений.

Шаг 28 - Работа с Датами

Для работы с датой в **VBA** предусмотрен специальный тип **Date**. Этот тип занимает 8 байт. Оно вам надо ? Это так для информации :-). Пробуем.

```
Sub Test()
        Dim MyDate As Date
        MsgBox (Str(Year(MyDate)))
End Sub
```

У меня выдает 1899 год. Это говорит, что при создании этой переменной она не инициализируется текущей датой. Это плохо. Поместить Дату и время можно из строки воспользовавшись функциями **DateValue** и **TimeValue**.

```
Sub Test()
        Dim MyDate As Date
        MyDate = DateValue("1/1/96")
        Debug.Print Year(MyDate)
End Sub
```

Так же и со временем:

```
Sub Test()
        Dim MyDate As Date
        MyDate = TimeValue("10:10:12")
        MsgBox Str(Minute(MyDate))
End Sub
```

Только одновременно хранить и время и дату так не удастся, вот этот код приведет к очень интересному результату.

```
Sub Test()
        Dim MyDate As Date
        MyDate = DateValue("6/1/72")
        MsgBox Str(Year(MyDate))
        MyDate = TimeValue("10:10:12")
        MsgBox Str(Minute(MyDate))
        MsgBox Str(Year(MyDate))
End Sub
```

Если вы хотите хранить вместе и дату и время, то поступите так:

```
Sub Test()
        Dim MyDate As Date
        MyDate = DateValue("6/1/72") + TimeValue("10:10:12")
        MsgBox Str(Minute(MyDate))
        MsgBox Str(Year(MyDate))
End Sub
```

Чтобы извлекать части даты и часов используйте такие функции:

```
Month(переменная типа Date)
Day(переменная типа Date)
Year(переменная типа Date)
Hour(переменная типа Date)
Minute(переменная типа Date)
Second(переменная типа Date)
WeekDay(переменная типа Date)
```

WeekDay - это день недели, если Вам это нужно, то вы можете написать что-то типа этого.

```
Sub Test()
        Dim MyDate As Date
        MyDate = DateValue("9/1/72")
        If (WeekDay(MyDate) = vbSunday) Then
                MsgBox ("Sunday")
End Sub
```

vbSunday это константа, есть еще **vbMonday**, ну дальше понятно.

Шаг 29 - Использование With

Оператор **With** используется для явного указания объекта, к свойствам которого мы хотим получить доступ. Вот так это выглядит в глобальном плане.

```
With объект
        операции с объектом
End With
```

Давайте рассмотрим пример. Ниже реализованы два сообщения, которые выводят имя и статус видимости объектов:

```
Sub Test()
        MsgBox (Application.Worksheets.Item(1).Name)
```

```
        MsgBox (Str(Application.Worksheets.Item(1).Visible))
End Sub
```

Используя **With** это можно сделать так:

```
Sub Test()
        With Application
                With .Worksheets
                        MsgBox (.Item(1).Name)
                        MsgBox (Str(.Item(1).Visible))
                End With
        End With
End Sub
```

Используя **With** можно получить доступ и к пользовательским структурам.

```
'---------- Описание --------
Type Family
        Name_I As String
        Name_Cat1 As String
        Name_Cat2 As String
End Type

'---------- Код --------

Sub Test()
Dim fam As Family

With fam
        .Name_I = "Pety"
        .Name_Cat1 = "Vasi"
        .Name_Cat2 = "Fisa"
        MsgBox (.Name_I)
End With

End Sub
```

Шаг 30 - Рекурсия в VBA

При программировании многие задачи решаются на основе рекурсии. Т.е. есть ряд задач, которые вообще без рекурсии не решаются. Это задачи имитации человеческого интеллекта на основе перебора вариантов. Без рекурсии есть возможность решить подобные задачи только для частных случаев. Понятие рекурсии довольно молодое. Вот справка:

1958 год. В руководстве по программированию ЕРМЕНТ
появилось понятие рекурсивности. Рутисхаузер.

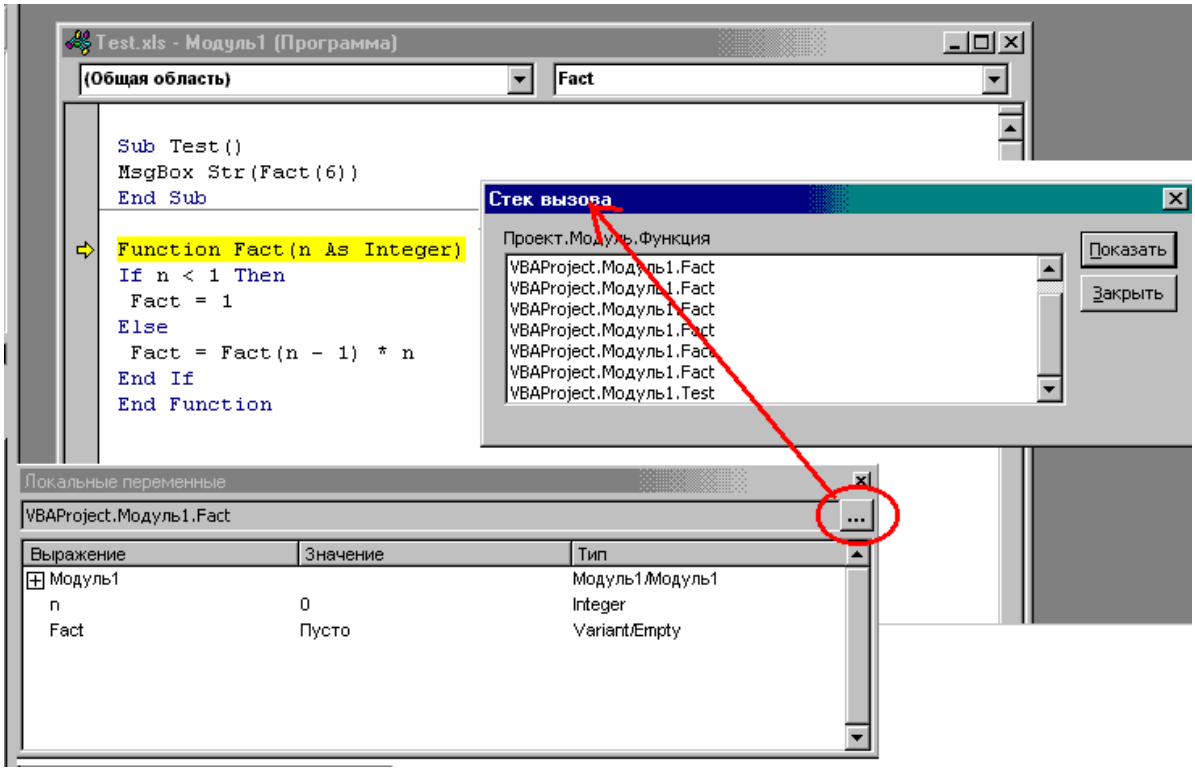
Рекурсивная процедура - это процедура вызывающая сама себя. Классический пример подсчет факториала. Мы то его и реализуем:

```
Sub Test()
        MsgBox Str(Fact(3))
End Sub

Function Fact(n As Integer)
        If n < 1 Then
                Fact = 1
        Else
                Fact = Fact(n - 1) * n
        End If
End Function
```

Всё это хорошо, только для рекурсивных функций используется стековая память, которая имеет предел :-(. В этой памяти размещаются и аргументы. Если их много или они большие по памяти хранения - финиш настанет еще быстрее. С рекурсивными функциями связанно еще и время выполнения. То же в плохую сторону.

В **Excel** нет рекурсивных объектов. Листы, книги, ячейки не рекурсивные. Но вот данные :-) им всё ни почем. Вы можете создавать используя **Type** структуры и создавать деревья. Для обработки их удобно использовать рекурсию.

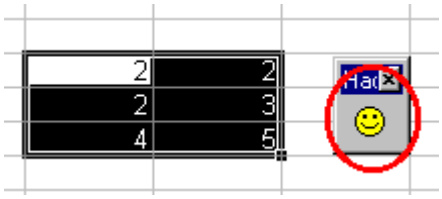
Что происходит при вызове рекурсивных процедур можно увидеть выполняя программу по шагам (**F8**) и просматривая окно локальных переменных или сразу стек вызовов из меню **Вид**.



Я тут ставил эксперимент и оказалось, что факториал числа больше 100 можно рассчитать, но вот 200 уже нельзя. Переполнение говорит. Вот так.

Шаг 31 - Работаем с выделенным диапазоном

Наша задача научиться обрабатывать выделенный диапазон. Я надеюсь, что кнопка до сих пор связанна у Вас с макросом. Как ниже. То есть пользователь выделяет диапазон, а по нажатию на кнопку над ним производится работа. Например умножения всех чисел на два.



Попробуем получить информацию о выделенном диапазоне:

```
Sub Test()
        ' объявим переменную типа Range
        Dim cur_range As Range
        ' активный расчетный лист
        With ActiveSheet
                ' объект Range включает выделенный диапазон
                Set cur_range = Selection
                ' активизируем Range
                cur_range.Activate
                ' Адрес и количество строк и колонок
                Debug.Print cur_range.Address
                Debug.Print cur_range.Columns.Count
                Debug.Print cur_range.Rows.Count
        End With
End Sub
```

А вот и код. Ниже написана функция, которая значения в ячейках умножит на 2. Будь то одна ячейка или диапазон ячеек.

```
Sub Test()
	Dim cur_range As Range
	With ActiveSheet
		Set cur_range = Selection
		cur_range.Activate
		For x = 1 To cur_range.Rows.Count
			For y = 1 To cur_range.Columns.Count
			' значению ячейки присвоить значение умноженное на 2
				cur_range(x, y) = cur_range(x, y).Value * 2
			Next y
		Next x
	End With
End Sub
```

Подводя короткий итог можно сказать, что выделенный диапазон можно получить используя объект **Selection**, перевести его в объект **Range**, от которого можно получить данные о местоположении выделенного диапазона, количества выделенных ячеек, а также иметь доступ к отдельным ячейкам используя объект **Range**.

Шаг 32 - Перемещение по ячейкам и информация

Вас может заинтересовать, а как можно сдвинуться влево или вправо назад или вперед от текущей ячейки. Для этого у объекта **Range** есть метод **Offset**, который и позволяет производить подобные действия.

```
Sub Test()
	Dim cur_range As Range
	Set cur_range = Range("A1")
	Set cur_range = cur_range.Offset(1, 0)
	Debug.Print cur_range.Address
End Sub
```

А вот результат работы. Мы от текущего объекта сдвинулись влево на 1 колонку.

```
$A$2
```

Если вы хотите узнать максимальные размеры листа, то у Вас есть возможность это сделать используя **UsedRange**. У вас будет объект типа **Range**, из которого вы сможете узнать максимальную колонку или строку.

```
Sub Test()
	With ActiveSheet
		Dim cur_range As Range
		Set cur_range = .UsedRange
		Debug.Print cur_range.Address
	End With
End Sub
```

Адресовать ячейки можно и двумя цифрами по колонки и строке. Это избавляет Вас от утомительного разбора адресов типа **$A10**. Так как адрес строка придется её резать и собирать. Использования **Cells(x,y)** очень гибко в использовании и позволяет строить легкие циклы. Пример ниже находит на листе левый верхний угол из всех ячеек с введенными данными и в эту ячейку записывает слово.

```
Sub Test()
	' объект Range
	Dim cur_range As Range
	' Весь лист
	With ActiveSheet
		Set cur_range = .UsedRange
		Debug.Print cur_range.Address
		' у меня печатает $C$5:$J$48
		Dim y_min As Integer
		' минимальная колонка
		y_min = cur_range.Columns.Column
		Dim x_min As Integer
```

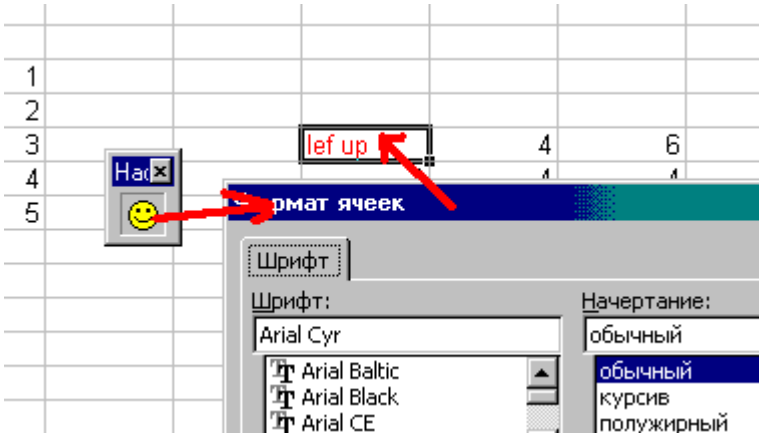
```
                ' минимальная строка
                x_min = cur_range.Rows.Row
                Set cur_range = Range(Cells(x_min, y_min), Cells(x_min, y_min))
                cur_range = "lef up"
        End With
End Sub
```

Шаг 33 - Встроенные диалоговые окна

Excell имеет несколько встроенных диалоговых окон. Несколько это слабо сказано, их более 200. Предназначены они для облегчения работы и программирования. Например, вашему приложению необходимо вызывать окно диалога для выбора цвета ячейки. Вот код:

```
Sub Test()
    Application.Dialogs(xlDialogActiveCellFont).Show
End Sub
```

А вот результат работы при запуске макроса. Это окно появится и будет изменять свойства выделенного диапазона.



Кроме типа окна далее можно передавать параметры. Вот, например, для открытия **DBF** файла.

```
Sub Test()
    Application.Dialogs(xlDialogOpen).Show "*.dbf"
End Sub
```

Появится диалоговое окно с предложением выбрать **DBF** файл. Как вы заметили для отображения окна используем метод **Show** и уникальную константу диалогового окна.

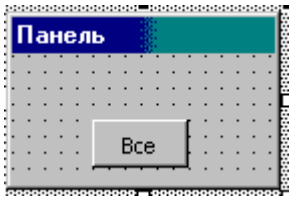
С аргументами особый разговор. Во-первых их может быть много. Если нужно оставить аргумент по умолчанию, то используйте вот такую конструкцию **"*.dbf", ,TRUE**. Две запятые позволяют пропустить аргумент (оставить по умолчанию). Так же следует знать, что аргументы нужно задавать строго в определенной последовательности при работе со встроенными диалоговыми окнами.

Аргументы могут быть именованными. Ниже пример аналогичен примеру с двумя запятыми.

```
Sub Test()
    Application.Dialogs(xlDialogOpen).Show arg1 = "*.dbf", arg3 = True
End Sub
```

Шаг 34 - Архитектура программ VBA

Одна из концептуальных идей **Windows** и программирования для **Windows** заключается в том, что объекты обмениваются сообщениями. Именно обмен, получение и обработка сообщения являются смыслом жизни любого объекта. Давайте посмотрим. У нас есть диалоговая панель и кнопка. Например такая.



Эту панель можно создать, показать методом **Show** и убрать методом **Unload**. Между вызовами этих процедур объект живет. То есть получает и обрабатывает сообщения. Например, при нажатии на кнопку.

Посылку сообщения можно рассматривать, как вызов метода соответствующего объекта. Например, Вы нажимаете на кнопку мышкой. Нажатие состоит из кучи сообщений - мышка двигается, клавиша вниз, клавиша вверх и другие. При этом обрабатывается последовательность сообщений и система делает вывод о сообщении более высокого уровня - нажата кнопка. В результате вызовется метод. То есть сообщения нажатия на кнопку вызывает метод **Click** этой кнопки.

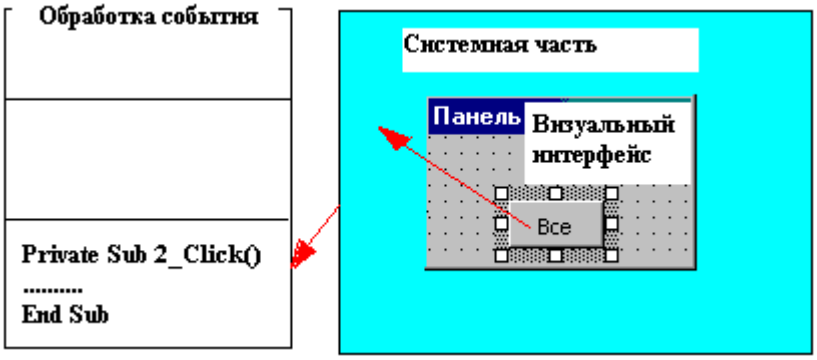
```
Private Sub CommandButton2_Click()
      ......
End Sub
```

Вообще-то сообщения примерно так и работают и в реальном мире. Сообщите жене, что у Вас появилась другая женщина и у жены будет вызван соответствующий метод. Реализация этого метода зависит от конструкции объекта жена :-). Или позвоните 03 и сообщите адрес со словами пожар. То же будет реакция. Вообщем как в жизни. Только для получения реакции нужно послать сообщение.

Модель **VBA** подразумевает три составляющих:

- Визуальная
- Системная
- Обработчик событий

Посмотрите рисунок ниже.



Визуальная часть это то, что видно на экране, т.е. интерфейс пользователя. Это окна диалога, кнопки, списки и т.д. При работе с программой пользователь постоянно её теребит - нажимает кнопки, двигает окна и еще производит кучу действий. Он использует интерфейсные объекты (элементы управления) для генерации событий. В ответ на это системная составляющая, которая включает в себя:

- средства операционной системы
- средства языка программирования

определяет соответствующее событие и формирует сообщение объекту (вызывает метод объекта). Обработчик событий это код, который будет вызван при возникновении события.

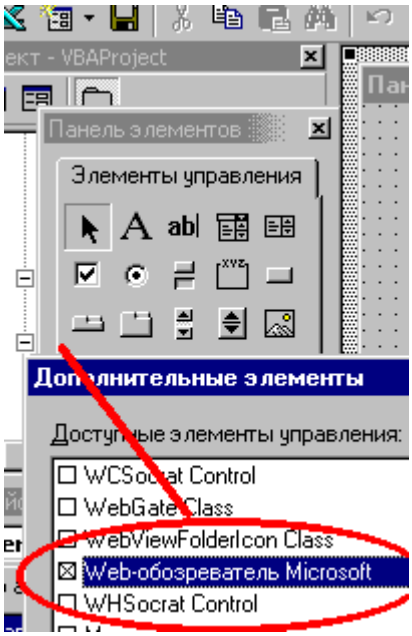
VBA для **OFFICE** полностью соответствует этой концепции. Офис предоставляет Вам средства интерфейса, **VBA** реакцию на события. Вы проектируете интерфейс и реакцию. Системная часть Вас не волнует. Это на совести разработчика **VBA** и **OFFICE**.

Рассуждать о преимуществах и недостатках данной системы можно долго. Только идея здесь следующая. Операционная система и реализация среды программирования может меняться (она и меняется 3.1, 95, 98 etc.) , меняется **VBA** (95 , 97 etc.), а вроде как ваши программы от этого вообще не зависят. Например, если в следующей версии **WINDOWS** кнопка будет допустим голографическая, то ваша программа будет с ней работать :-). Вам придется при необходимости добавить новые методы.

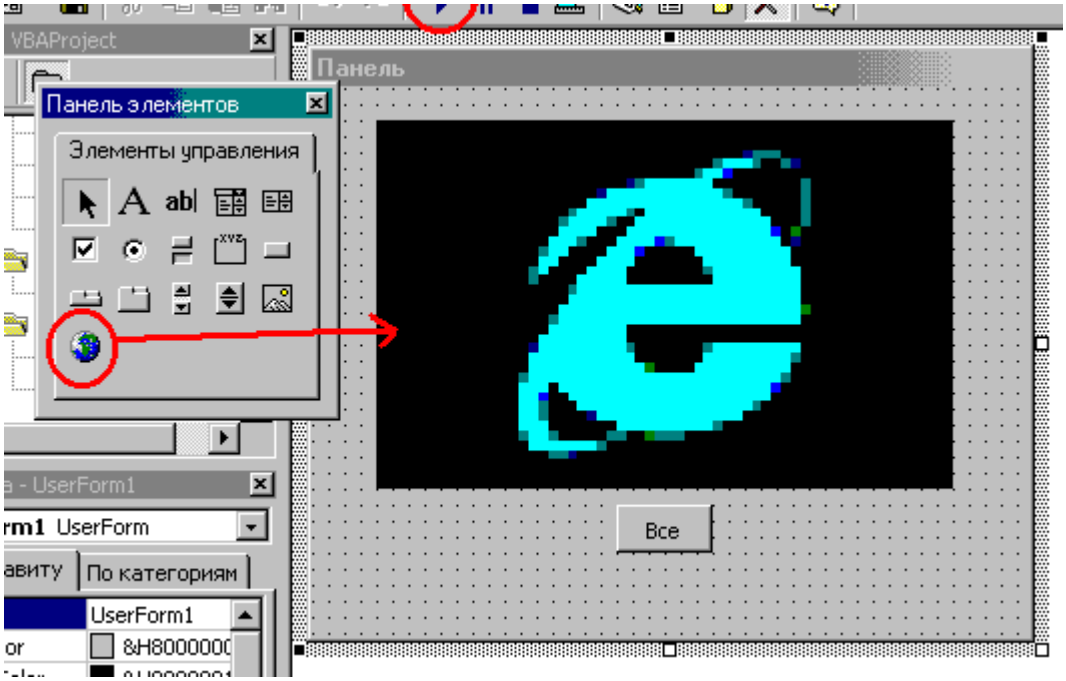
Визуальный он потому, что код рождается от визуального интерфейса. То есть стройте интерфейс потом только код реализации. Бейсик потому, что он и есть Бейсик с дополнительными возможностями. Представляете как далеко смотрели в будущее наши учителя, по школьной программе изучается Бейсик.

Шаг 35 - Дополнительные компоненты

Когда вы редактируете диалоговое окно, вы видите далеко не все компоненты, которые есть в системе. Для получения полного списка вам необходимо щелкнуть правой кнопкой на Панели инструментов и выбрать "Дополнительные компоненты". Выбрать можно любой, но мне понравился **Web**-обозреватель, его я и подключил.



В панели компонент появится земной шар. Выберите его и поместите на диалоговую панель.



Два раза щелкните на кнопки и привяжите код к событию нажатия.

```
Private Sub CommandButton2_Click()
        WebBrowser1.GoHome
End Sub
```

Вернитесь к редактированию диалоговой панели. Её тут же можно запустить на исполнение. Смотрите на рисунке обведено кружком. Установите соединение с Интернетом. Нажмите на кнопку. Загрузится страница **HOME**. Там про то, что "Добро пожаловать".

Как видите подключение и использование дополнительных компонент дело в принципе несложное. Это только в принципе :-).

Шаг 36 - Где хранятся настройки панелей инструментов

Это очень интересный вопрос. Вот пример. Я создал настройки панелей инструментов как на картинке.

Всё это я сохранил в файле. Теперь открывая **Excel** такой вид будет у всех книг. Значит информация о настройках панелей инструментов где-то хранится. Конечно !!!

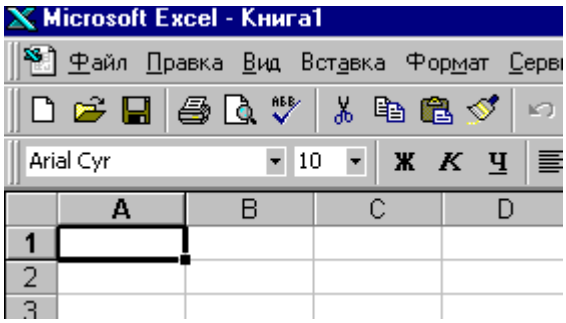
Параметры модифицированных панелей инструментов хранятся в файле.

WINDOWS\ИМЯ_ПОЛЬЗОВАТЕЛЯ.XLB

или

WINNT\ИМЯ_ПОЛЬЗОВАТЕЛЯ.XLB

Что понимается под именем пользователя? Вот я вхожу в **NT** и к имени **Administrator** ввожу пароль. Поэтому этот файл имеет имя **Administrator.xlb**. Там все настройки. Если вы хотите запуститься с настройками по умолчанию переместите его в другое место и **Excel** сам настроит панели инструментов по умолчанию. Вот смотрите. Сейчас он загружается как на рисунке сверху в смысле панелей инструментов. Переношу файл из каталога **WINNT** на рабочий стол. И вот.



Настраивать панель инструментов можно также из **Visual Basic** используя объект **CommandBar**.

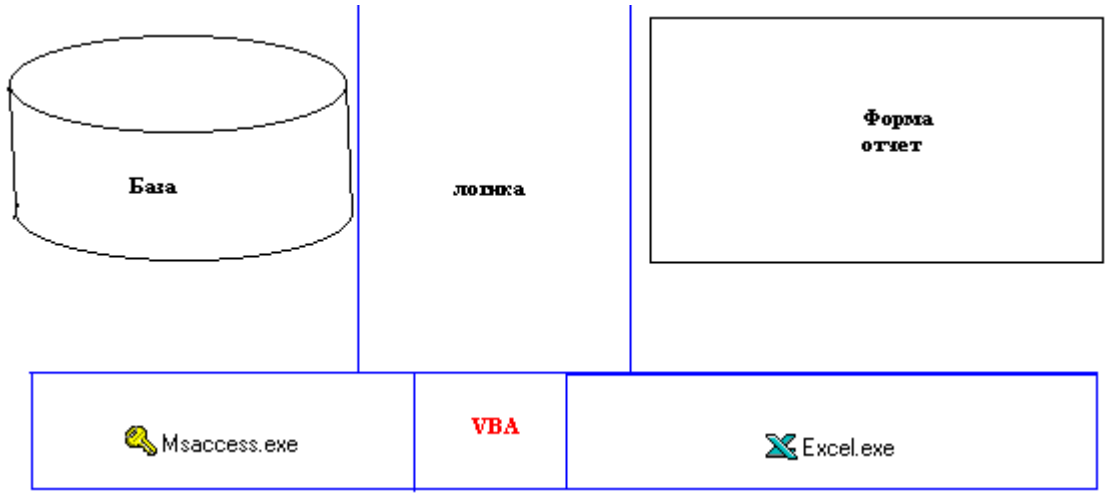
Шаг 37 - Создание приложений с использованием Excel

Итак, нам нужно построить приложение. Любое приложение состоит из 3 частей.

1. Место хранения информации
2. Интерфейс
3. Логика.

Всё это конечно можно реализовать с помощью **Excel**, вопрос какой ценой ? Вообще в этой жизни можно всё. Например, взломать сеть используя только **NotePad**. Можно, пишите в командах процессора, потом переименуйте **txt** в **exe** и готово. Теоретически можно. Вы можете ? Я нет. Вот и о чем речь. Сколько и какой ценой.

Итак мой опыт такой. Неправильное использование инструментов ведет к головной боли программистов. Любое приложение можно сделать, например, и в **Excel**, и в **Access**. Но только реально начнете работать то тут стоп. В **Excel** легко создавать формы и отчеты, считать, но хранить данные тяжело. А **Access** нет проблем с хранением, контролем за информацией, но все остальное труднее. Все просто. Каждый инструмент для своей задачи. Вот мой взгляд.



Как видите данные хранятся в **Access** формы и отчеты в **Excel**, логика реализуется на **VBA**. Обратите внимание, что эта модель не чистая. Все таки часть ответственности за логику ложится и на **Access** и на **Excel**. Например, в **Access** можно установить фильтры на ввод, построить запросы. В **Excel** проводить расчеты. **VBA** является связующим звеном между этими программными продуктами.

Связь между **Access** и **Excel** можно организовать по разному. Например, на основе **DAO**. Но встает вопрос, чей **VBA**, то есть какого программного продукта ? Я склоняюсь к **Excel**. Вот почему. Работа происходит так:

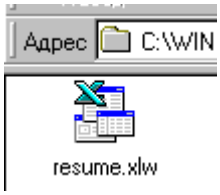
работа с формой
    нужна информация
    запрос к базе
    получение результатов
    возврат на форму

То есть основное количество логических операций производится в интерфейсе. Поэтому удобно пойти от **Excel** туда и поместить код **VBA**. Все, что нужно сделать это научить тесно взаимодействовать **Excel** и **Access**.

Шаг 38 - Зачем нужна рабочая область

Без тебя лето зима
Без тебя метель в июле
        А Студио.

Рабочая область - это набор файлов **Excel**. Она позволяет открывать эти файлы одновременно. Вообщем-то это специальный файл, который содержит информацию о том какие рабочие книги должны быть открыты. Его расширение **XLW**. А выглядит он вот так:



Рабочая область имеет смысл, если у Вас есть несколько рабочих книг связанных между собой. Давайте попробуем. Создавайте рабочий каталог с именем **TestWorkspace**. Теперь создадим файл **Excel** с именем **Test1** и поместим его в рабочий каталог. Давайте занесем информацию. Смотрите ниже. Три участка с фамилиями. Красным это сумма полученная автосуммированием.

H9 = =СУММ(H5:H8)

	A	B	C	D	E	F	G	H
1								
2								
3								
4		Участок 1			Участок 2			Участок 3
5	Петя	123		Оля	44		Бобик	4
6	Коля	233		Маша	33		Шарик	5
7	Вася	445		Наташа	55		Рекс	6
8	Дима	555		Даша	66		Бим	66
9		1356			198			81
10								
11								

Сохраните этот файл и, не закрывая его, создайте новый. Дайте ему имя **Test2.xls** и сохраните в тот же каталог. Теперь мы занесем в него информацию ссылаясь на суммы в первой книге (**Test1.xls**).

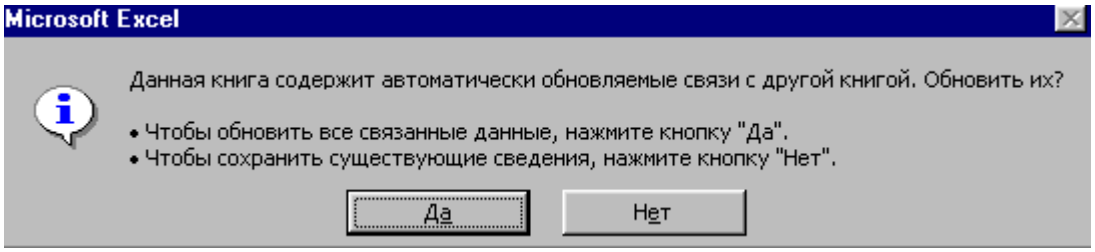
B4 = =[Test1.xls]Лист1!$B$9

	A	B	C
1			
2			
3			
4	Работники (Участок1)	1356	
5	Работницы (Участок 2)	198	
6	Халявщики (Участок 3)	81	
7		1635	

Теперь сохраним рабочую область и проведем эксперименты. Выбираем меню "Файл -> Сохранить рабочую область". И сохраняем в тот же каталог с именем **Test**. Закрывайте **Excel** в вашем каталоге должно быть три файла.

Test1.xls
Test2.xls
test.xlw

Экспериментируем. Откройте **Test2.xls**. Вы получите сообщение:



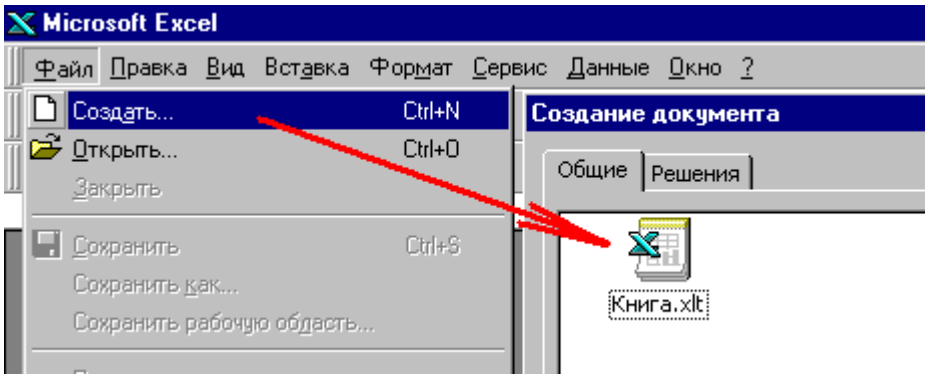
Это нормально. Связи то есть. А вот теперь закройте все и откройте рабочую область. Всё пройдет без сообщений. Конечно откроются сразу все файлы в рабочей области. Это удобно если предполагается обмен данными между многими книгами. При этом всё должно быть динамично.

Шаг 39 - Автозапуск и шаблоны

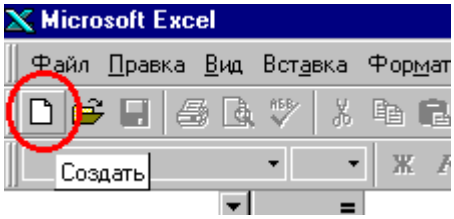
Понимание когда и откуда создается пустой лист и при каких условиях - это залог к автоматизации приложений. Например, вы установили параметры страницы, макросы, шрифты и цвета и хотите, чтобы автоматически создаваемая книга их имела. Это понятие шаблона. Только создается книга двумя разными путями.

- Из каталога шаблонов
- Из каталога автозапуска

Из шаблонов:



Папки автозапуска:



Весь прикол в том, что эти папки разные :-))) Все равно Вы не захотите, чтобы Ваши документы создавались всегда одинаково !!!

C:\Program Files\Microsoft Office\Шаблоны - Здесь шаблоны
C:\Program Files\Microsoft Office\Office\XLStart - А здесь книга на автосоздание

Давайте убедимся, что это разные вещи. Создайте шаблон скажем с желтой верхней строкой и сохраните в шаблоны, тоже самое сделайте только с красной и сохраните в папку автозапуска. Создавайте шаблоны с именами **Книга.xlt**. Закройте **Excel**, а теперь попробуйте два варианта и результат будет разный. Это важно. Важно особенно при программировании, чтобы быть уверенными, что все элементы в книге есть.

Шаг 40 - О многозадачности Windows и циклах

Как Вы знаете **Windows 9x** является многозадачной средой. Это везде пишется. Но на самом деле это далеко не совсем так. То есть в ней нет четкой установки приоритетов. Не верите ? Создайте макрос в **Excel** и запустите вот этот пример.

```
Sub Test()
        For x = 1 To 1000000000000
                Debug.Print x
        Next x
End Sub
```

Любая работа в этот момент будет проблематична. Это связанно с проблемами еще от **Windows 3.1**, тогда при программировании от Вас требовали периодически особенно в процессе длительных циклов передавать управление операционной системе. **VBA** до сих пор не избавлен от этой проблемы. Вам все равно надо это делать. Делается это с помощью **DoEvents**.

Используйте функцию **DoEvents** для передачи управления операционной системе каждый раз прохода цикла. Но она может быть полезна и при дисковых операциях ввода вывода, операциях с **DDE**. Давайте изменим наш пример.

```
Sub Test()
        For x = 1 To 1000000000000
                DoEvents
                Debug.Print x
        Next x
End Sub
```

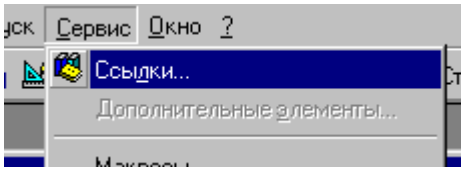
Если Ваша программа тормозит выполнение других программ вспомните об этом шаге.

Шаг 41 - Подключаем DAO

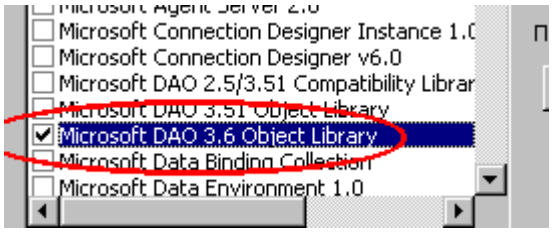
Следующая серия шагов будет посвящена проблеме **Excel-Access**
Задача сделать форму Excel, которая будет рассчитывать стоимость товара в рублях

из цены в долларах оперируя информацией из **Access** в зависимости от даты.

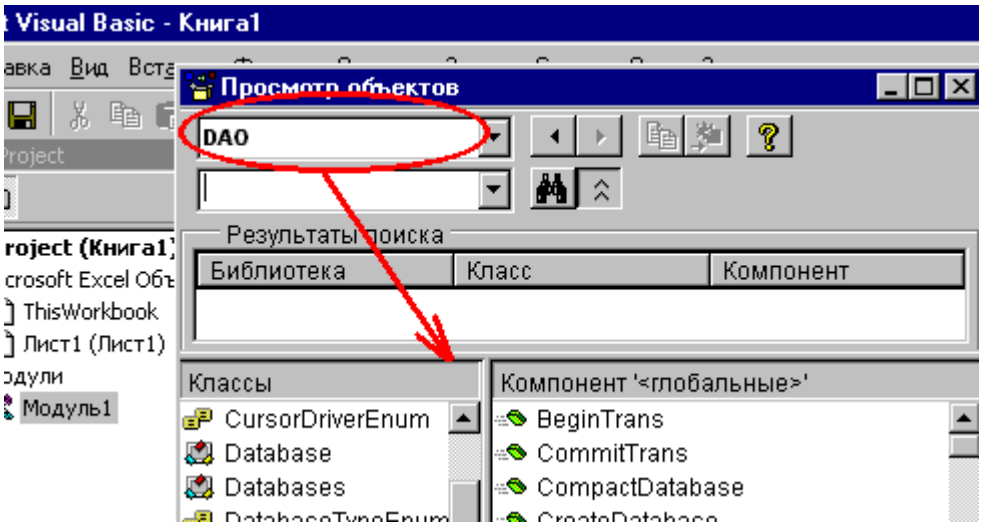
В **Excel** много методов работы с базами данных. Давайте попробуем **DAO** для того, чтобы получить доступ к классам **DAO** нам необходимо их подключить. Это делается из меню "Ссылки":



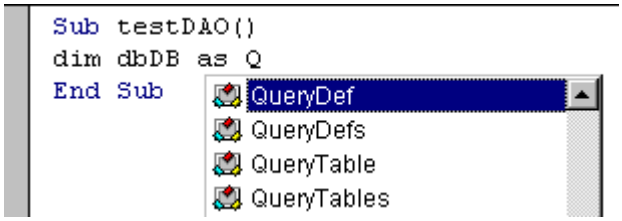
Откроется диалоговое окно, в котором нам надо найти **DAO Object Library**.



С этого момента вы можете многое. Например, просмотреть список классов и их свойств и методов. Сразу скажу, что это надежнее документации. Там есть то, чего нет в описаниях и помощи. Зайдите в меню "Вид -> Просмотр объектов" и выберите **DAO**.



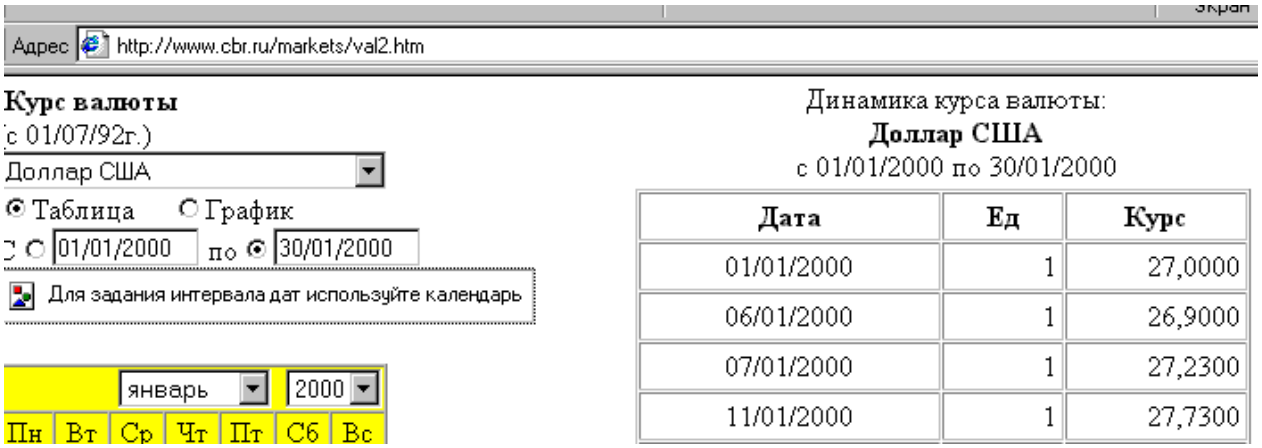
Теперь мы можем использовать классы **DAO**.



Шаг 42 - Готовим данные

Для того, чтобы быть ближе к жизни я пошел на сайт Центрального Банка России за курсом доллара. Вот его адрес, чтобы долго не ходить (http://www.cbr.ru/scripts/daily.asp)

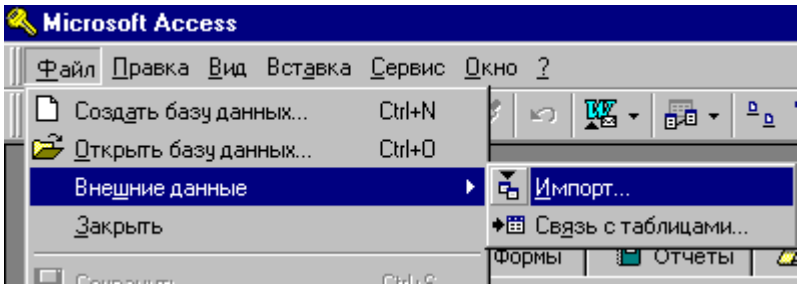
А вот как выглядит страница с курсами:



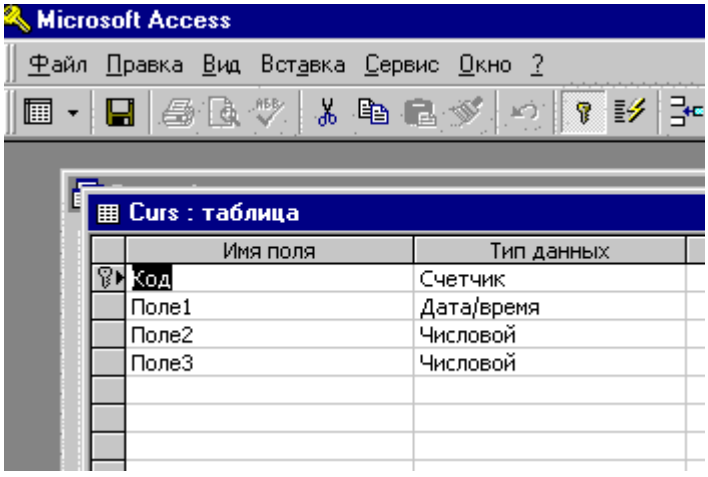
Я просто выделил и скопировал в текстовый файл через буфер обмена. Он примерно такой с именем **Curs.txt**. Он есть в проекте если кому лень.

```
01/01/2000  1  27,0000
06/01/2000  1  26,9000
07/01/2000  1  27,2300
11/01/2000  1  27,7300
12/01/2000  1  28,4400
.....
```

Теперь мы из этого текстового файла сделаем базу данных. Запустим **Access** создадим новую БД с именем **curs** и меню "Импорт":



А дальше "Текстовый файл" и "фиксированной ширины". В результате появится таблица со следующей структурой и заполненными значениями.



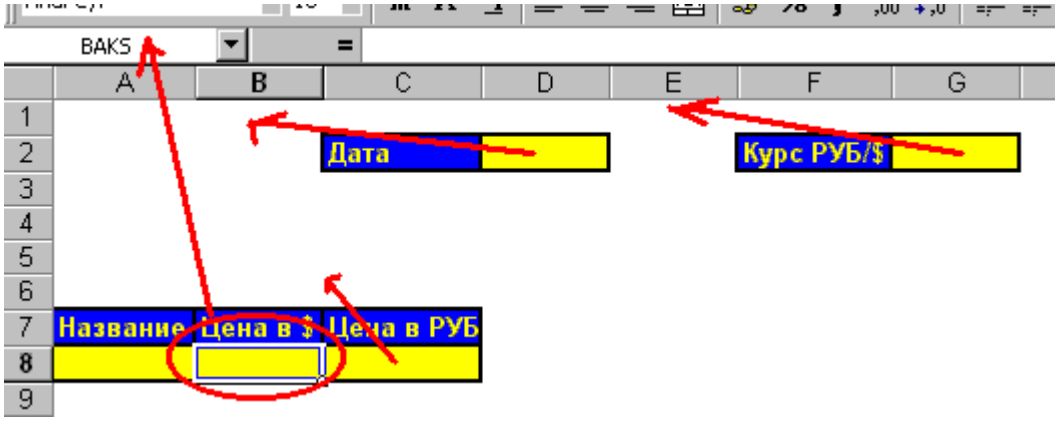
Для особенно ленивых этот файл есть в проекте **curs.mdb**. Только если вы настолько ленивы зачем вообще читаете ???

Шаг 43 - Готовим форму

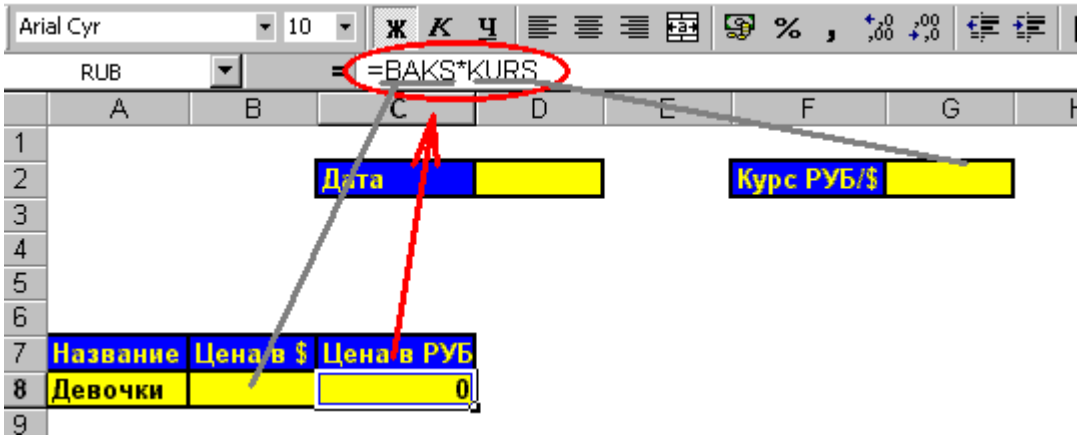
Итак, нам необходимо подготовить форму, к которой мы будем обращаться. Конечно это лист **Excel**. Запустите **Excel**, создайте файл с именем **TestCurs** и оставьте на нем один лист. Теперь на этом листе напишем "Цена товара в $" и "Цена товара в РУБ", также отдельно "Курс РУБ за $". Нужна и ячейка "Дата". И дадим имена ячейкам.

Цена в рублях	RUB

Цена в $	BAKS
Курс	KURS
Дата	DATES



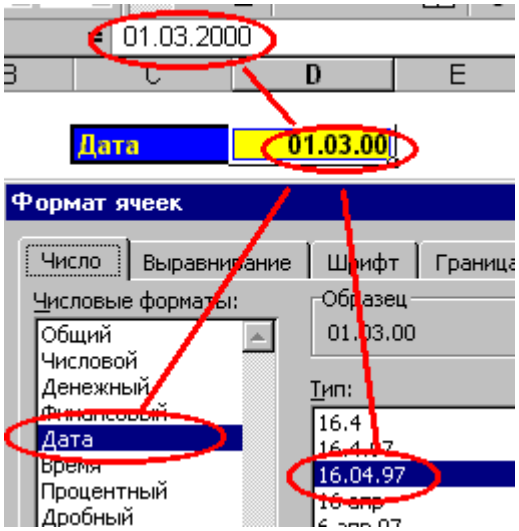
Теперь дело за формулами. Идея ясна как белый день. Стоимость руб. = Стоимость в $ * курс. Вот это и запрограммируем:



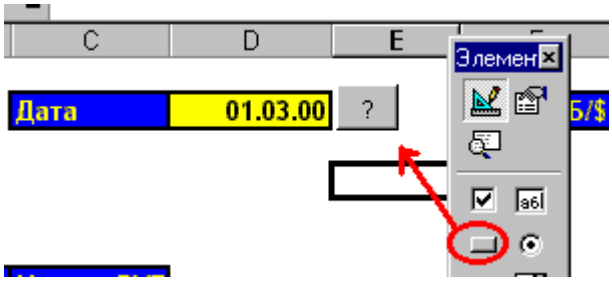
Вводить мы будем цену в долларах, выбирать дату и видеть цену в рублях. Цену в долларах ввести не тяжело :-))). Все остальное посчитается, если будет курс. Курс привязан к дате. То есть мы должны.

- Ввести дату
- Получить курс
- А дальше вводить цену в долларах

Вот теперь надо все подготовить для ввода даты. Надо задать, что это ячейка даты. Выделите её пойдите в "Формат", дальше "Ячейки", потом установите тип ячейки "Дата".



После того как введена дата нужно будет получить курс. Для этого поместим кнопку на лист. Нам нужно в меню "Вид -> Панели инструментов -> Элементы управления" выбрать кнопку и поместить её рядом с датой. Нажимая на неё мы будем получать курс.



Двойной щелчок создаст макрос. Вот и все на этот шаг. Все готово к программированию. Эта книга есть в проекте, если у Вас что-то не получилось.