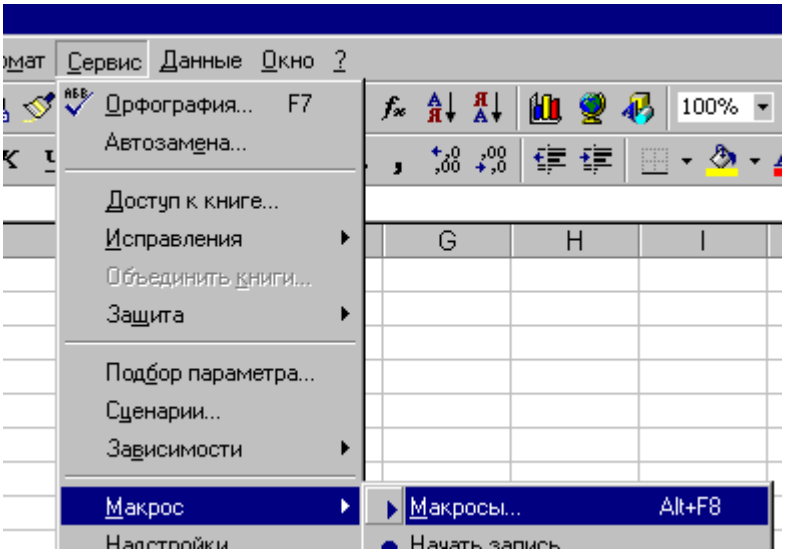
VBA для Microsoft Excel

Шаг	Содержание	Шаг	Содержание
1	Первый макрос	38	Зачем нужна рабочая область
2	Объектная модель Excel	39	Автозапуск и шаблоны
3	Коллекции в VBA	40	О многозадачности Windows и циклах
4	Коллекция Workbooks в Excel	41	Подключаем DAO
5	Далее про Workbooks в Excel	42	Готовим данные
6	Имена ячеек и адресация в Excel	43	Готовим форму
7	Запись макросов и что это дает	44	Считаем
8	Коллекция Sheets	45	Начало и конец данных
9	Еще о Sheets	46	Доступ к одинаковым элементам управления
10	Использование Range	47	Свойства документов MSOffice
11	Дальше о Range	48	Встроенные свойства документов MSOffice
12	Обработка ошибок VBA	49	Связывание макроса с кнопкой на панели инструментов
13	Объект Err	50	Определяем выделенную ячейку
14	События обьектов	51	Изучаем события Excel Workbook
15	Пользовательские формы	52	Автоматизация на основе СУММЕСЛИ
16	Чтение и запись текстовых файлов	53	Разбор строки стандартными функциями
17	Win32 API и VBA	54	Подробнее о событиях загрузки и выгрузки формы
18	Просмотр объектов	55	Четыре основных метода работы с формой (Load,Unload,Show,Hide)
19	Информация о типе переменной	56	Настройка свойств формы
20	Пользовательские классы	57	Элементы для формы
21	Пользовательские типы	58	Наборы элементов управления
22	For Each	59	Проверка ввода на уровне формы (KeyDown, KeyUp, KeyPress)
23	Работа с каталогами	60	Проверка и настройка ввода в TextBox
24	Использование Automation	61	О MaskEdBox
25	О функции SendKeys	62	Maskedit - Text и ClipText
26	Заполнение списка на форме из таблицы	63	Обработка ошибок в VBA
27	Обмен данными между формой и таблицей	64	Функция автоматической проверки синтаксиса
28	Работа с Датами	65	Выделение диапазона выше текущей ячейки
29	Использование With	66	Движение по диапазону
30	Рекурсия в VBA	67	Движение по ячейкам
31	Работаем с выделенным диапазоном	68	Как сделать XLA ?
32	Перемещение по ячейкам и информация	69	Динамическое создание меню
33	Встроенные диалоговые окна	70	Нефть, таблицы и как делать не надо
34	Архитектура программ VBA	71	Нефть, таблицы и как делать не надо, продолжение
35	Дополнительные компоненты	72	Как создать свою функцию
36	Где хранятся настройки панелей инстументов	73	Выделенный диапазон Выше ячейки, второй метод
37	Создание приложений с использованием Excel		

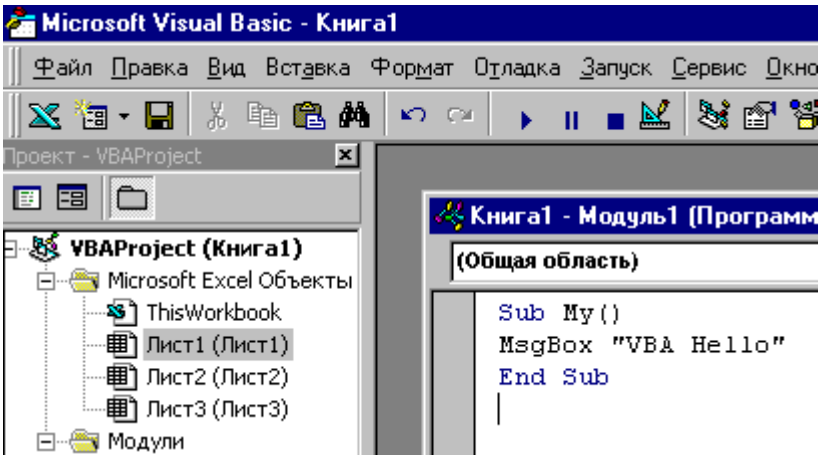
[Меню]

Шаг 1 - Первый макрос

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

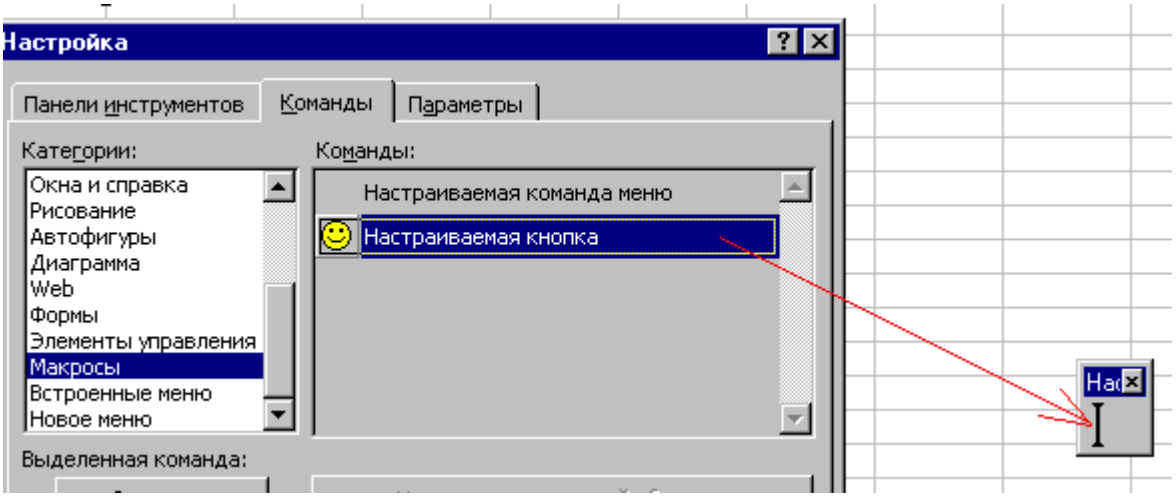


Появится табличка "макросы". В поле имя введите "**MY**", а место нахождения в поле "Находится" выберите - "Это книга". Кнопка "Создать" станет активной нажмите её. Появится редактор **VBA** введите код, как на рисунке ниже.



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 инструметнов 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.

Меню

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 обьектами приложения. Вот именно обьектами и управляет наше приложение. В нашем случае, если упростить иерхическую архитектуру, то это выглядит так.

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

```
                    Cell
                    .....
```

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

Меню

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 колекции имеют 4 метода:

```
Add
Remove
Count
Item
```

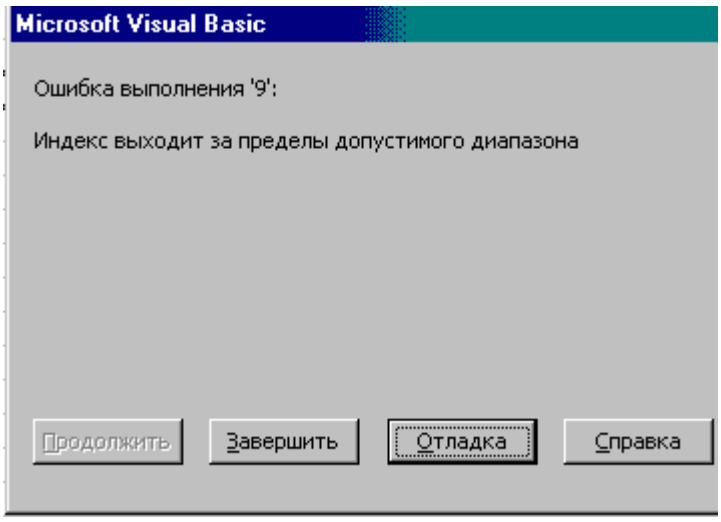
Было бы наверно логично подумать, что в **MS OFFICE** есть специализированный класс колекций, на подобии шаблонов в **C++**. Но это не так. Для каждого типа объектов объявляется своя коллекция. Так же есть некоторые коллекции, которые отличаются названием методов. Всё это немного странно, но что сделаешь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

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.



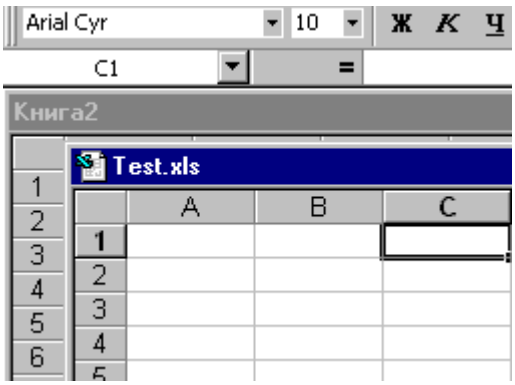
Обработку ошибок мы расмотрим позже.

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++** :-))

Меню

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 исвестен. Он возвращает количество элементов коллекций. Как и на рисунке сверху у меня две книги, о чем и сообщит мне окно сообщения.

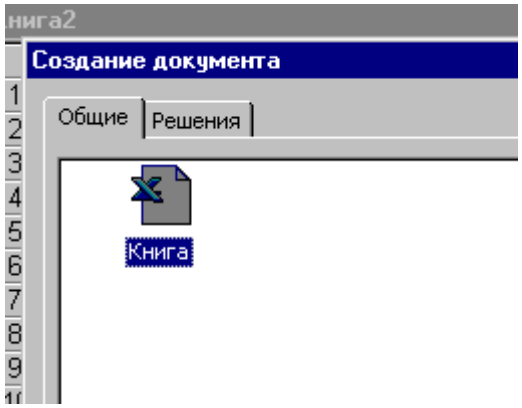
Данная коллекция обладает некоторой спецификой. Это связанно с тем, что книги могут храниться в файлах. Поэтому при работе с этой коллекцией предусмотрено две функции добавления **Add** и **Open**:

```
Add(template)
```

Добавляется книга на основе некоторого шаблона. Шаблоном может выступать, как настояший 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
- Пользовательсякие меню, макросы и панели инструментов
- номера и типы листов
- стили строк и колонок
- текст, даты формулы . можно некторые 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)
```

Меню

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 колекции называется **Close**. Но для удаления необходимо получить доступ к элементу.

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

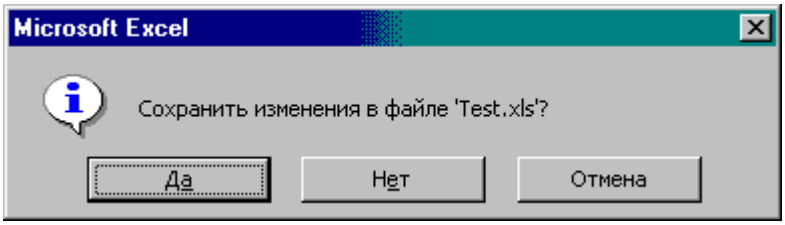
В таком варианте закроются все книги:

```
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 обьекта **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
```

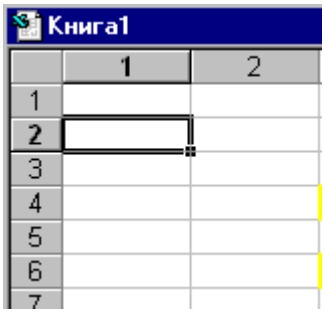
Меню

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.

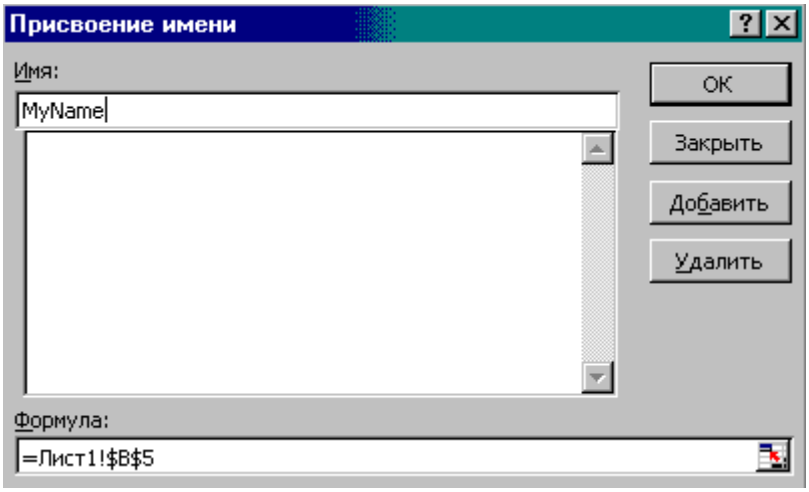
Стиль **A1**:

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 **OK.**

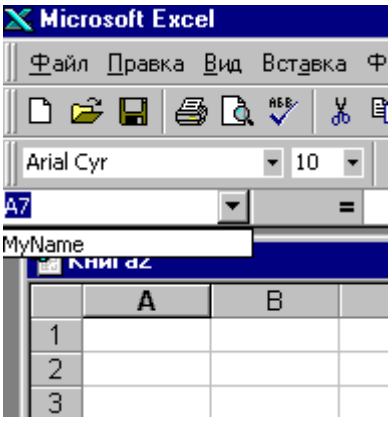


После присваивания имени вы введите число в эту ячейку, а в другой создайте формулу:

```
=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:

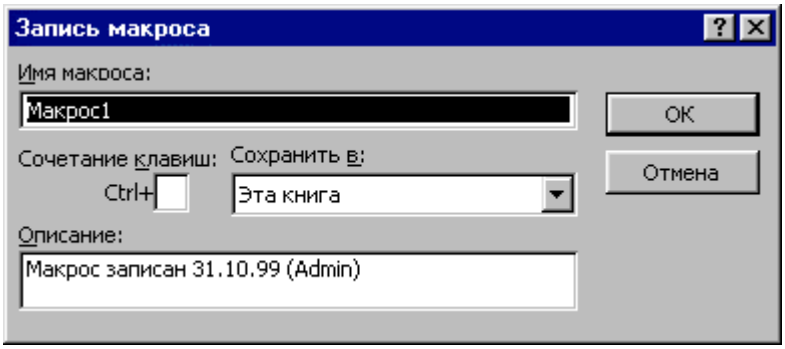


И здесь так же можно быстро переместиться к ячейке с заданным именем. Выберите её из списка и где бы она не находилась Вы окажетесь там :-)

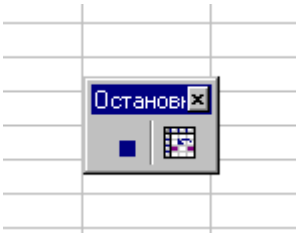
Меню

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 кноку **OK**. В результате у вас появится значек, который говорит о том, что идет запись. Вообще при записи макросов рекомендуется пользоваться клавишами, но я, например, и мышкой пользуюсь и записывается. Итак, появится значек.



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 -> Максросы**, 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.

Меню

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 свойтсво **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
```

Меню

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 обьекты 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
```

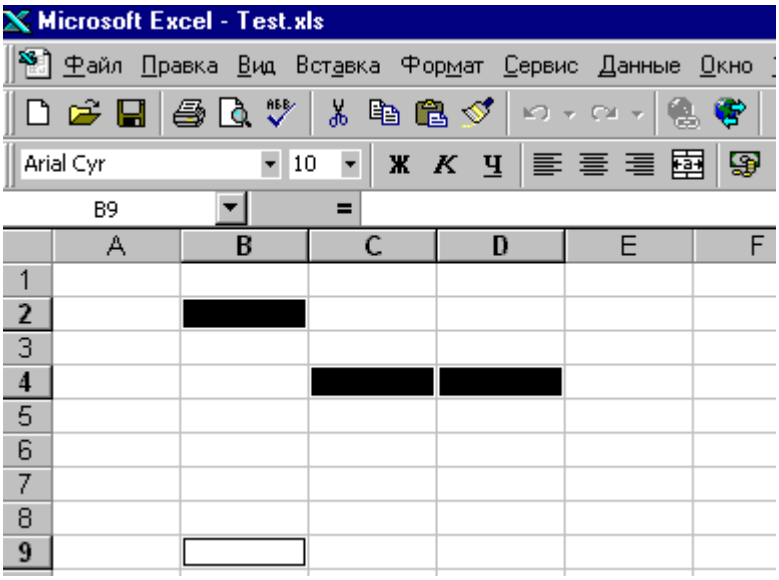
Программным путем можно и выделять листы. Это метод **Select**. У него один параметр типа **BOOL**, если он установлен в **TRUE**, то происходит выделения листа, а если **FALSE**, то выделение объединяющее. Выделите другой лист и запустите следующий макрос.

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

Меню

Шаг 10 - Использование Range

Мы с вами будем использовать это свойтсво 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 кноку 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 диапазовами можно проводить большое количество операций. Вот для программной реализации этих возможностей и служит свойтсво **Range**. Оно есть у очень многих обьектов **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 дожны быть активными, иначе возникнет ошибка. Кроме того, наверно, надо перед кадым 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 ячейкм 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 обьект 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
```

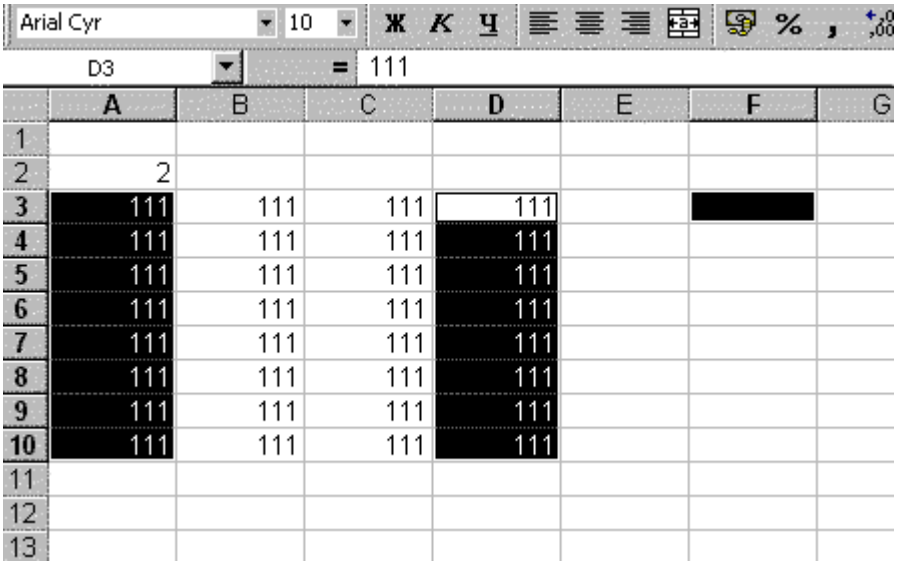
Меню

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 количестов 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 расказать 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
```

Меню

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
```

Меню

Шаг 13 - Объект Err

Да, странное совпадение, 13 шаг и зловещий обьект **Err**, от которого одни неприятности. Этот обьект 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 гарницу 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 очишает. 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 обьект автоматически очистится после ..

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

При отладке или специально в программе вы и сами можете сгенирировать 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
```

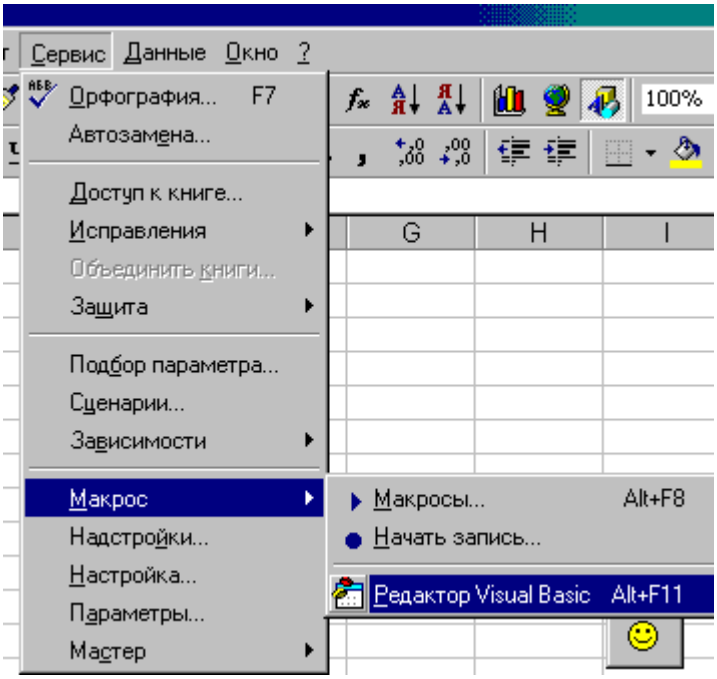
Меню

Шаг 14 - События обьектов

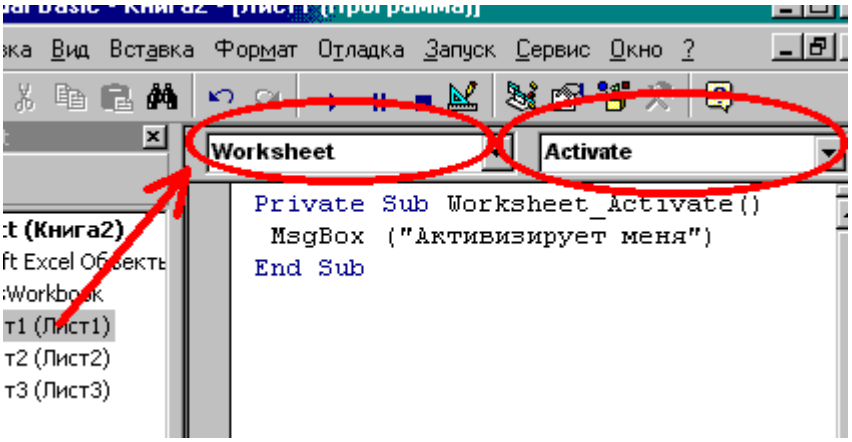
Обработать можно события следующих обьектов **Excel**:

```
Application
WorkBoor
WorkSheet
Chart
```

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



Появится редактор **VBA**. Нас интересует список обьектов в окне **VBAProject**. Выбирите **Лист1** и два раза шелкните по нему. Появится белое окно. Вам нужно выбрать обьект 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 обьектов вниз, но и вверх. Вот то же событие активации обрабатывается и на верхнем уровне.

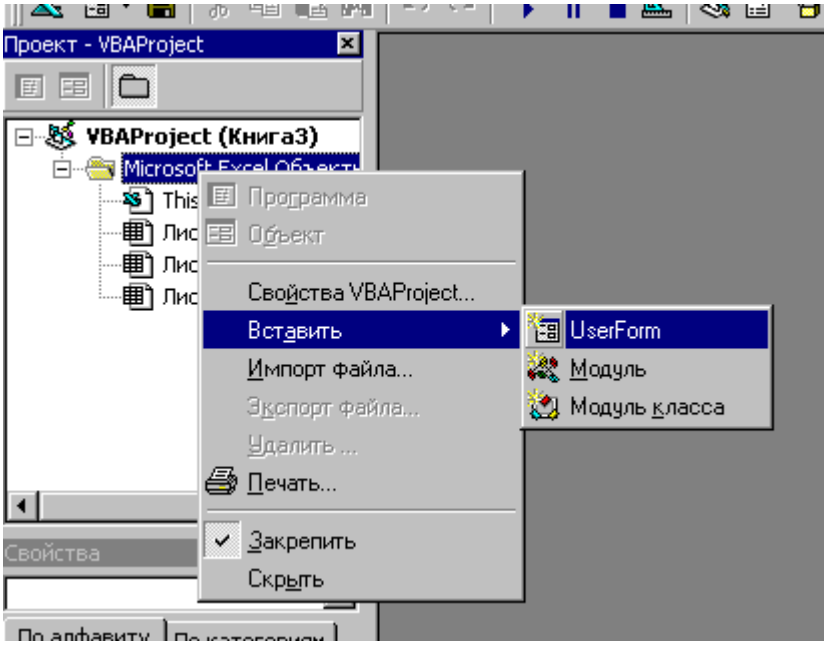
```
Private Sub Workbook_SheetActivate(ByVal Sh As Object)
        MsgBox (Sh.Name)
End Sub
```

Попробуйте его создать и проверить.

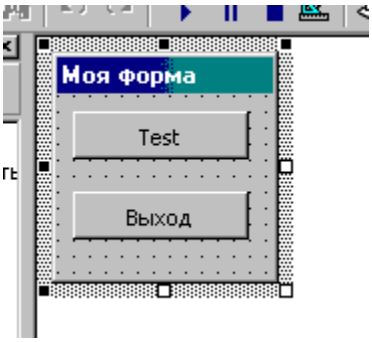
Меню

Шаг 15 - Пользовательские формы

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



У Вас появится диалоговое окно (форма), панель ниструментов, и окно свойств. В окне свойств нас интересует совйство **Caption**, которое позволяет нам изменить заголовок окна. Поменяйте его на осмысленное имя, например, первая форма :-). Разместите на форме кнопки. Выбираете инструмент кнопка. Незнаете 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 шелчок 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 масрос 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.

Меню

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 разделитеть. 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
```

Меню

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 присутсвия 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 упрощеный 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**.

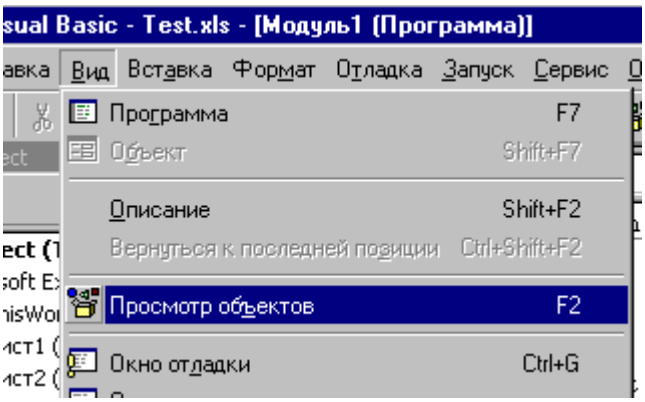
Наверно лучше создавать описания в отдельном модуле и просто его экспортировать в проект, дабы не мучаться. И, наверно, есть уже готовые модули. Но этот метод позволяет Вам подключить любую динамическую библиотеку. Посмотрите в разделе **MFC** шаг за шагом : "Шаг 46 - Dll для Excel".

Меню

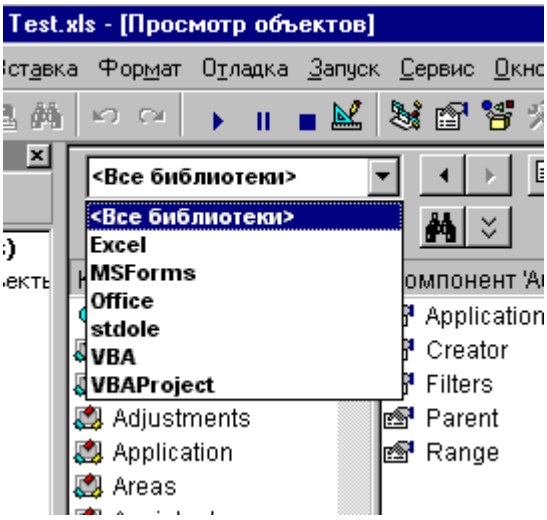
Шаг 18 - Просмотр объектов

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.

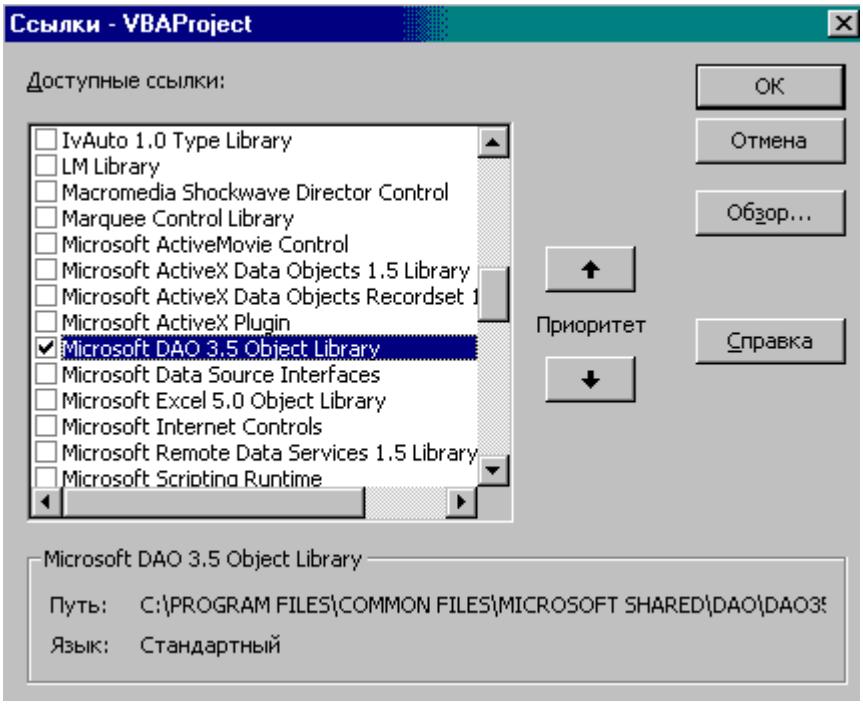
Редактор **VBA** позволяет вам просмотреть доступные Вам обьекты и их свойства. Для этого Вам необходимо запустить **Просмотр объектов** из меню **Вид (F2)**.



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.

Меню

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
```

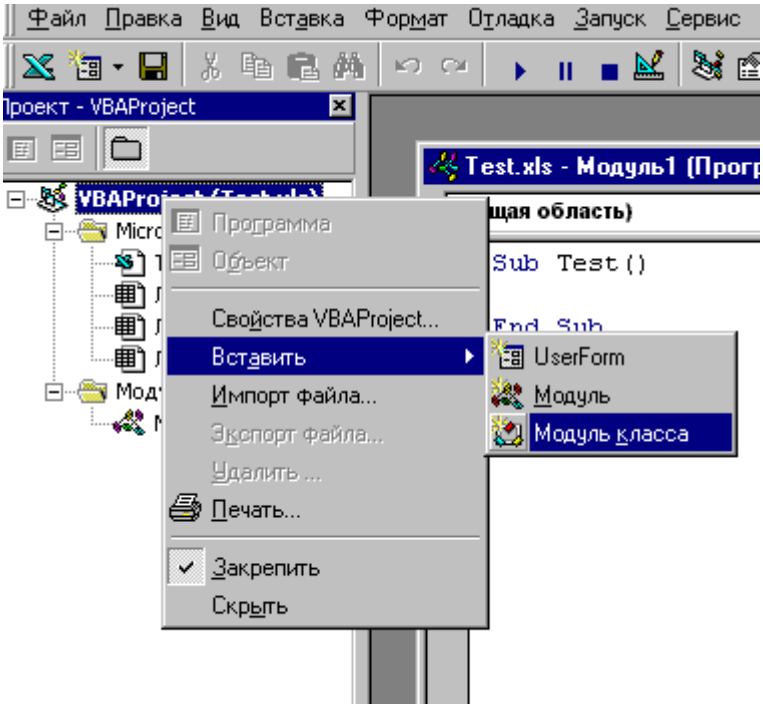
Есть еще ряд подобных функций:

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

Меню

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 шелкнув 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
```

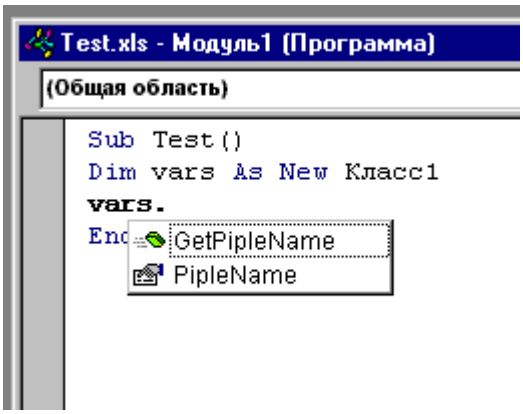
Теперь свойства для получения имени. Пишите ниже:

```
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:

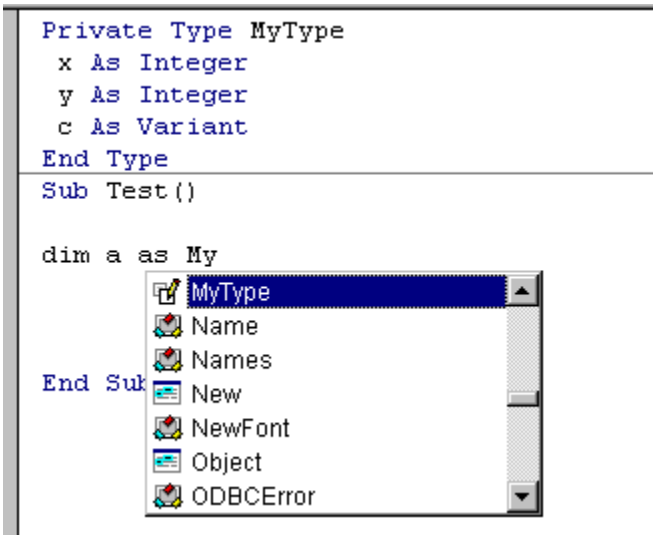
```
Sub Test()
        Dim vars As New Класс1
        vars.GetPipleName
        MsgBox vars.PipleName
        vars.PipleName = "VBA"
        MsgBox vars.PipleName
End Sub
```

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 читате просто так.

Меню

Шаг 21 - Пользовательские типы

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

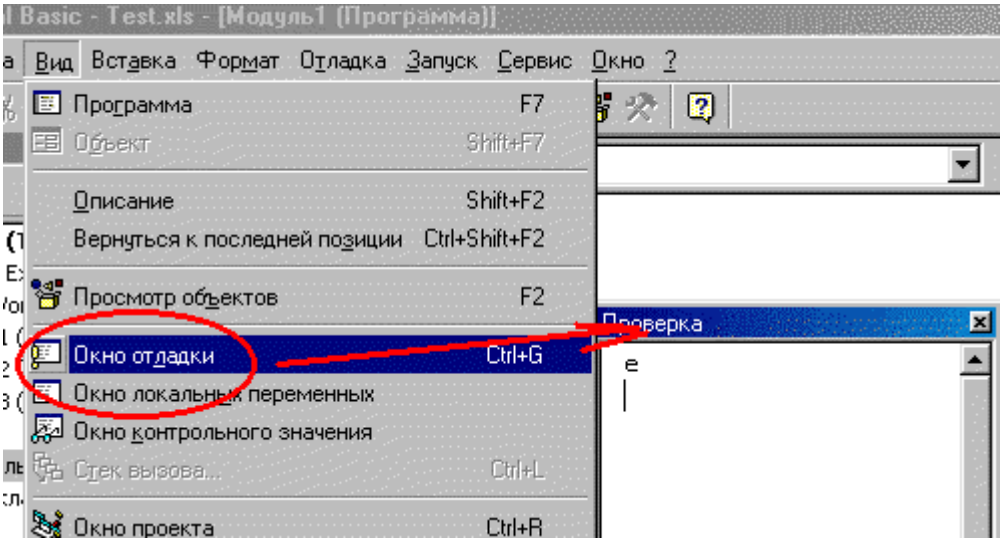


Итак мы создадим тип с элементами **x,y**, типа **Integer** и **c** с типом **Variant**. Ох уж этот **Variant**, он позволяет нам в структуру помещать все, что угодно. Хоть массив. Мы то сделаем. Но разве Вас не впечатляет простота и мощь. Ведь так можно создать структуры любой сложности очень просто.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

Меню

Шаг 22 - For Each

Этот цикл придуман для того, чтобы облегчить дествия 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 кигам.

```
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
```

Меню

Шаг 23 - Работа с каталогами

Первым делом определим откуда запущенно приложение. У обьекта **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**).

Меню

Шаг 24 - Использование Automation

Использование элементов **ActiveX** на базе можели **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. Вобщем это можно назвать построением пользовательских приложений на базе готовых программ.

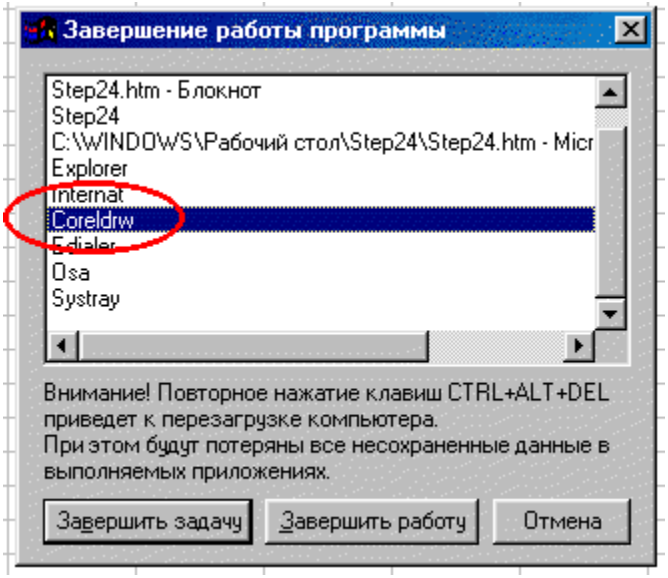
Понятие, которое используеться в основе всех интегрированных систем является служба. **MS OFFICE** обеспечивает все необходимые службы для создания офисных приложений:

- **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
```

Меню

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 обрабочики 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:

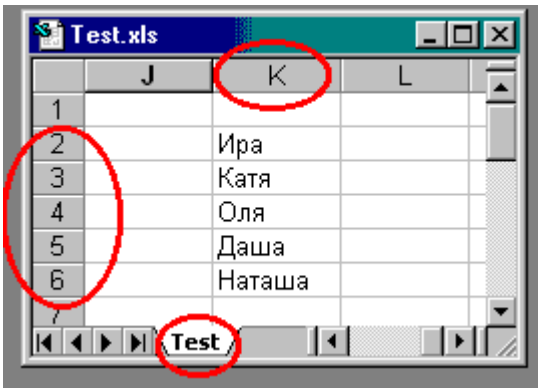
```
Sub Test()
        SendKeys ("{F1}")
End Sub
```

Когда экспериментируете запускайте макрос из активной рабочей книги.

Меню

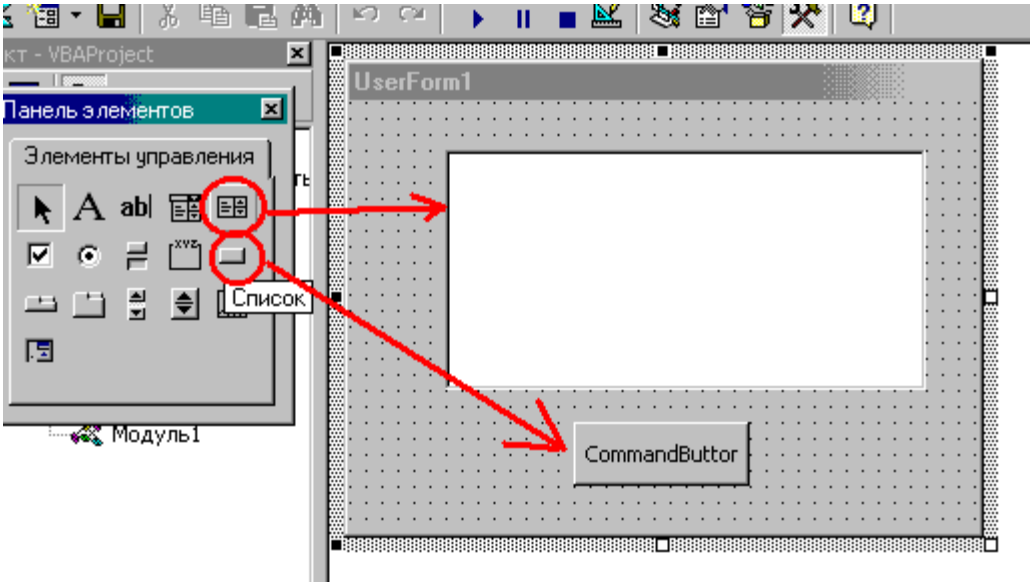
Шаг 26 - Заполнение списка на форме из таблицы

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.



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

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

Что означает "уничтож 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.

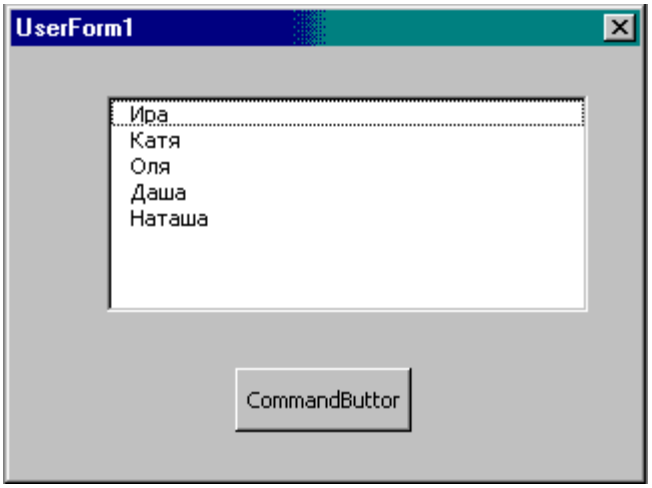
```
Private Sub UserForm_Activate()
        ' Активизируем нужный лист
        Worksheets.Item("Test").Activate
        ' Выбиделяем диапазон
        Dim девушки As Range
        ' Объект выделения
        Set девушки = Range("K2:K6")
        ' Выделяем
        Dim vars As Variant
        ' Пойдем по 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:

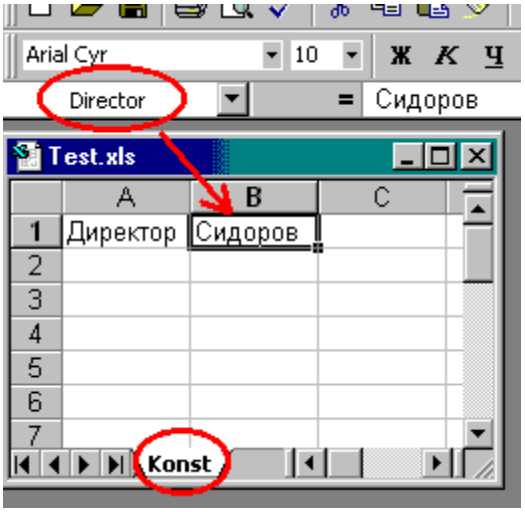


Меню

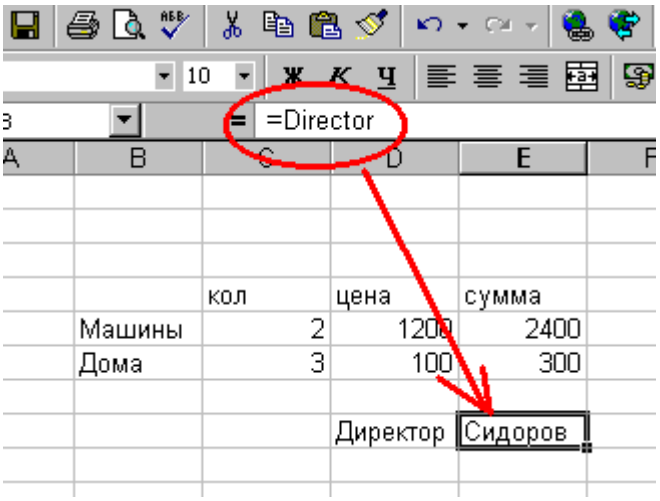
Шаг 27 - Обмен данными между формой и таблицей

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.

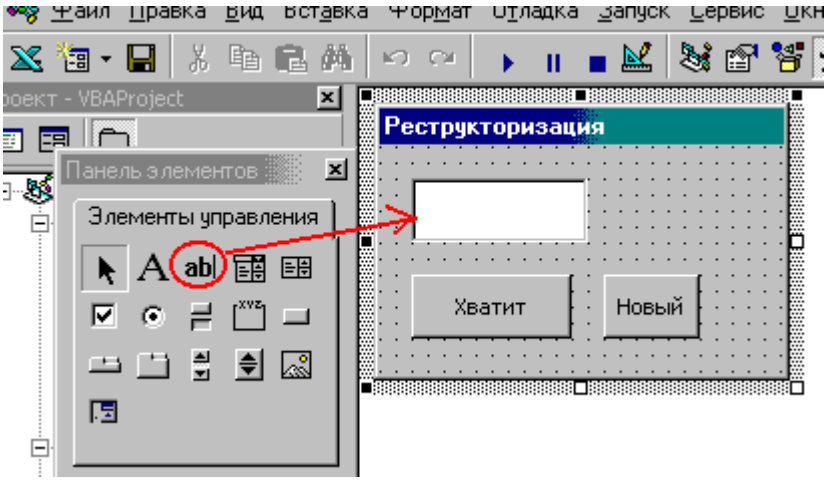
Создаем лист Константы и на нем ячейке даем имя.



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.

Меню

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.

Меню

Шаг 29 - Использование With

Оператор **With** используется для явного указания обьекта, 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
```

Меню

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 интелекта 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.
```

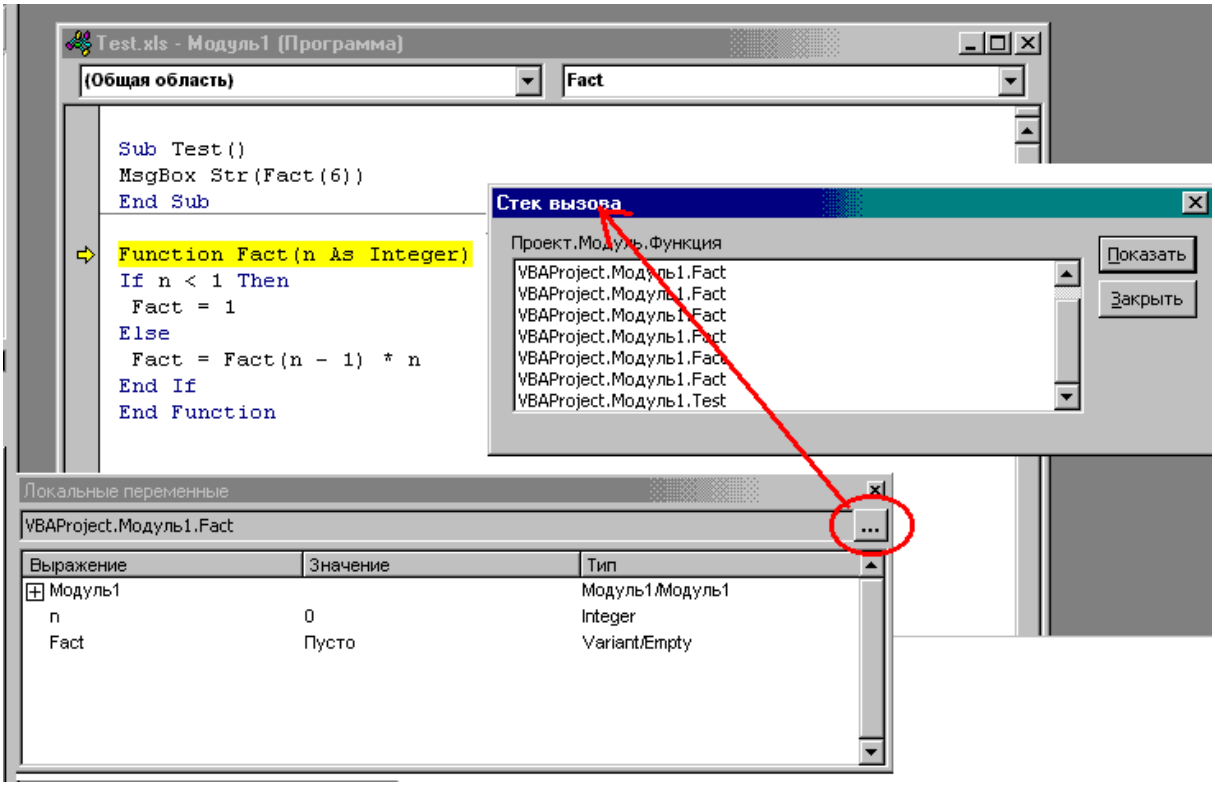
Рекурсивная процедура - это процедура вызывающая сама себя. Классический пример подсчет факториала. Мы то его и реализуем:

```
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 обьектов. 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**.

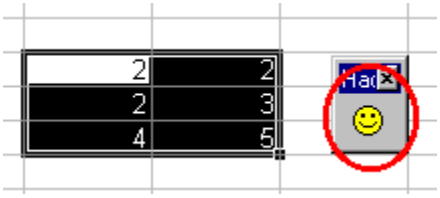


Я тут ставил эксперимент и оказалось, что факториал числа больше 100 можно расчитать, но вот 200 уже нельзя. Переполнение говорит. Вот так.

[Меню]

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()
        ' обьявим переменнуб 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 умноженно 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**.

Меню

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 сроке. Это избавляет Вас от утомительного разбора адресов типа **$A10**. Так как адрес строка придеться 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
```

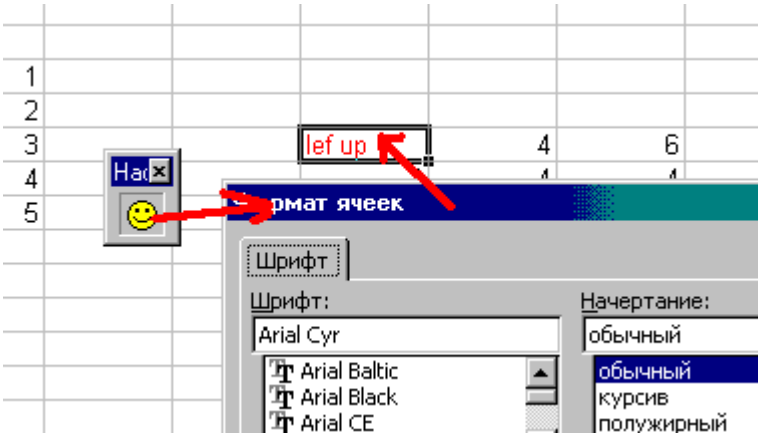
Меню

Шаг 33 - Встроенные диалоговые окна

Excell имеет несколько встроенных диалоговых окон. Несколько это слабо сказано, их более 200. Предназначенны 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.

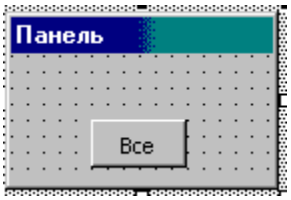
Аргументы могут быть именованными. Ниже пример аналогичен примеру с двумя запятыми.

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

Меню

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. Наример, 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.

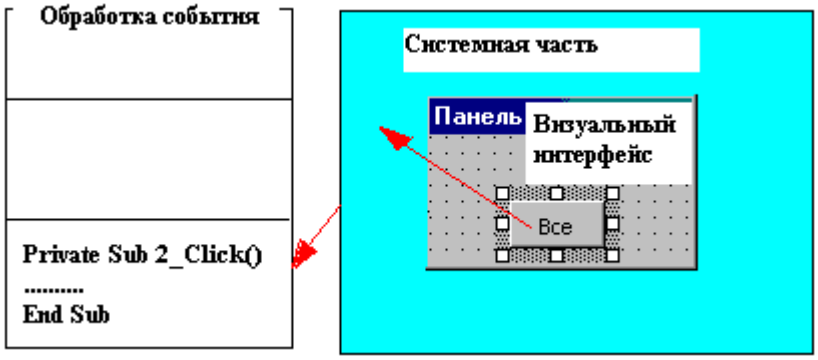
```
Private Sub CommandButton2_Click()
        ......
End Sub
```

Вообще-то сообщения примерно так и работают и в реальном мире. Сообщите жене, что у Вас появилась другая женщина и у жены будет вызван соотвествующий метод. Реализация этого метода зависит от конструкции объекта жена :-). Или позвоните 03 и сообщите адрес со словами пожар. То же будет реакция. Вобщем 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 (элемены управления) для генерации событий. В ответ на это системная составляющая, которая включает в себя:

- средства операционной системы
- средства языка прогарммирования

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

VBA для **OFFICE** полностью соответсвует 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 галлографическая, то ваша программа будет с ней работать :-). Вам придется при необходимости добавить новые методы.

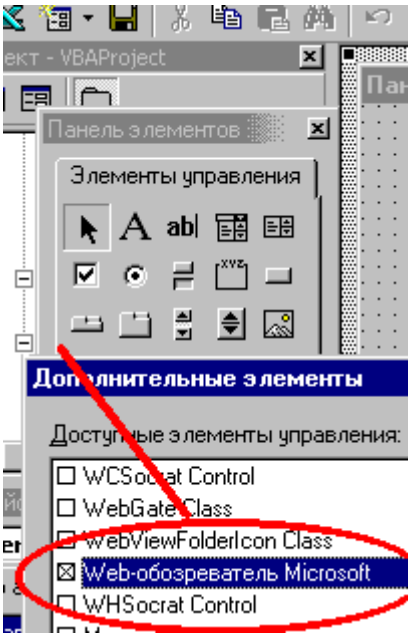
Визульный он потому, что код рождается от визуального интерфейса. То есть стройте интерфейс потом только код релизации. Бейсик потому, что он и есть Бейсик с дополнительными

возможностями. Представляете как далеко смотрели в будущее наши учителя, по школьной программе изучается Бейсик.

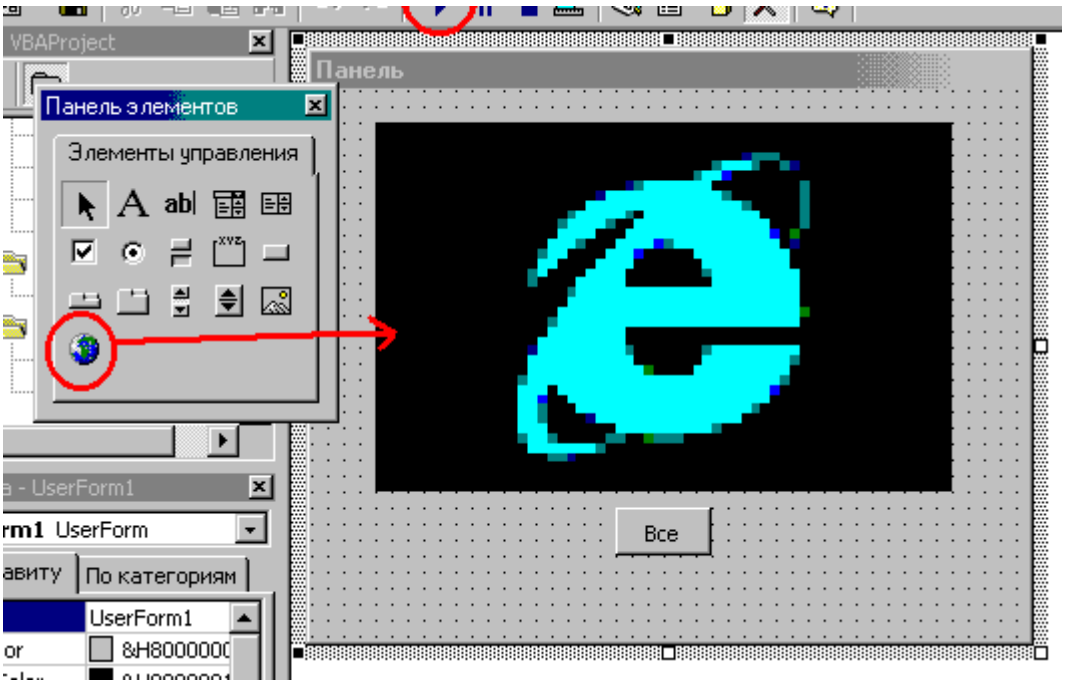
Меню

Шаг 35 - Дополнительные компоненты

Когда вы редактируете диалоговое окно, вы видете 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 привежите код к событию нажатия.

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

```
End Sub
```

Вернитесь к редактированию диалогвой 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 ипользование дополнительных компонент дело в принципе несложное. Это только в принципе :-).

Меню

Шаг 36 - Где хранятся настройки панелей инстументов

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 !!!

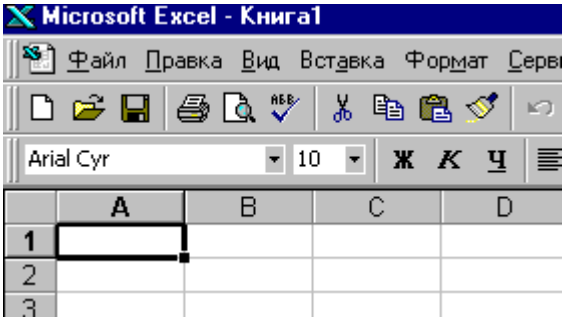
Параметры модифицированных панелей инструментов хранятся в файле.

```
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**.

Меню

Шаг 37 - Создание приложений с использованием Excel

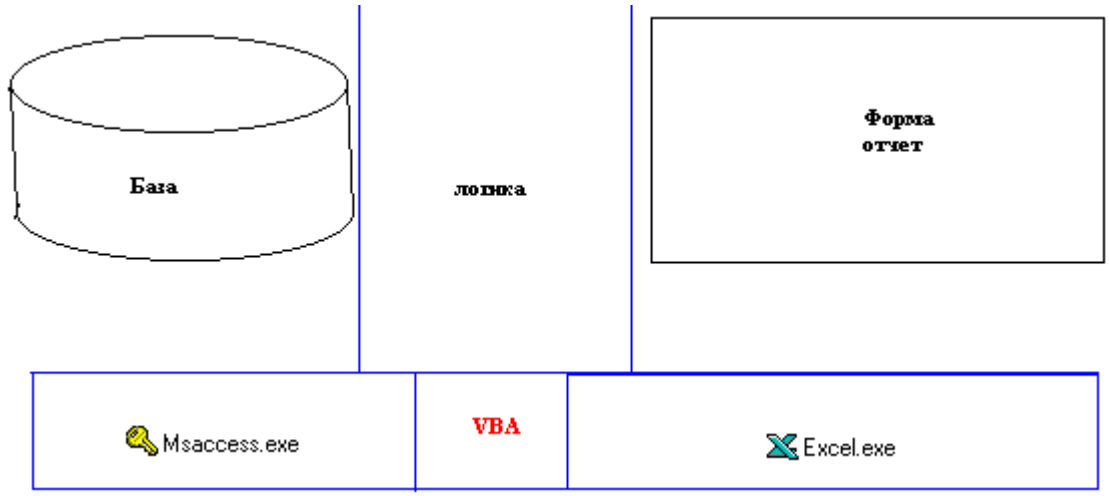
```
Внимание! Это личный взгляд автора на построение
решений на базе Microsoft Office. Притензии идеи
и коментарии принимаются по почте.
```

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 помошью **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. Теоритически можно. Вы можете ? Я нет. Вот и о чем речь. Сколько и какой ценой.

Итак мой опыт такой. Неправильное использование инсрументов 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 с базе
        получение результатов
возврат на форму
```

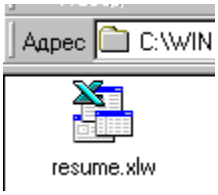
То есть основное количество логических операций производится в интерфейсе. Поэтому удобно пойти от **Excel** туда и поместить код **VBA**. Все, что нужно сделать это научить тесно взаимодействовать **Excel** и **Access**.

Меню

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. Вобщем-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

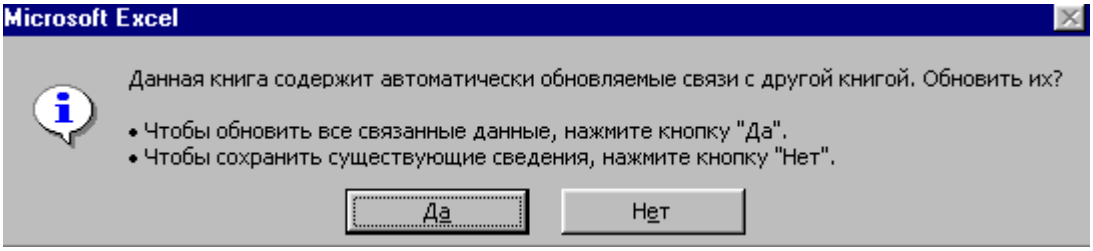
	A	B	C
1			
2			
3			
4	Работники (Участок1)	1356	
5	Работницы (Участок 2)	198	
6	Халявщики (Участок 3)	81	
7		1635	

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.

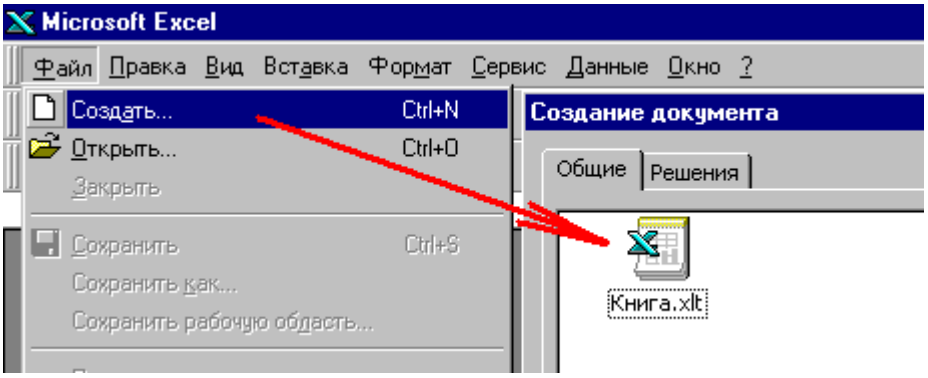
Меню

Шаг 39 - Автозапуск и шаблоны

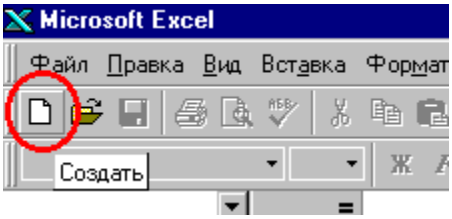
Понимание когда и откуда создается пустой лист и при каких условиях - это залог к автоматизации приложением. Например, вы установили параметры страницы, макросы, шрифты и цвета и хотите, чтобы автоматически создаваемая книга их имела. Это понятие шаблона. Только создается книга двумя разными путями.

- 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.

Меню

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 помошью **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
```

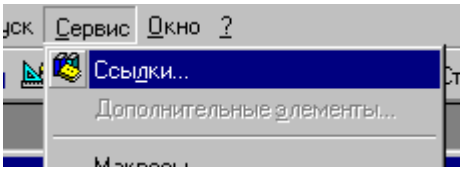
Если Ваша программа тормозит выполнение других программ вспомните об этом шаге.

Меню

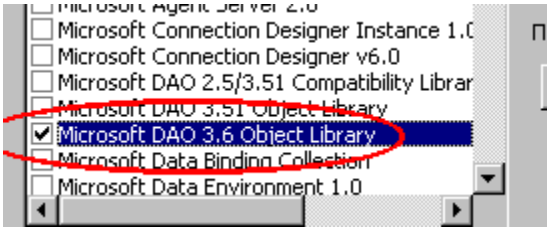
Шаг 41 - Подключаем DAO

Следующая серия шагов будет посвящена проблемме **Excel-Access**
Задача сделать форму Excel, которая будет расчитывать стоимость товара в рублях
из цены в долларах оперируя информацией из **Access** в зависимости от даты.

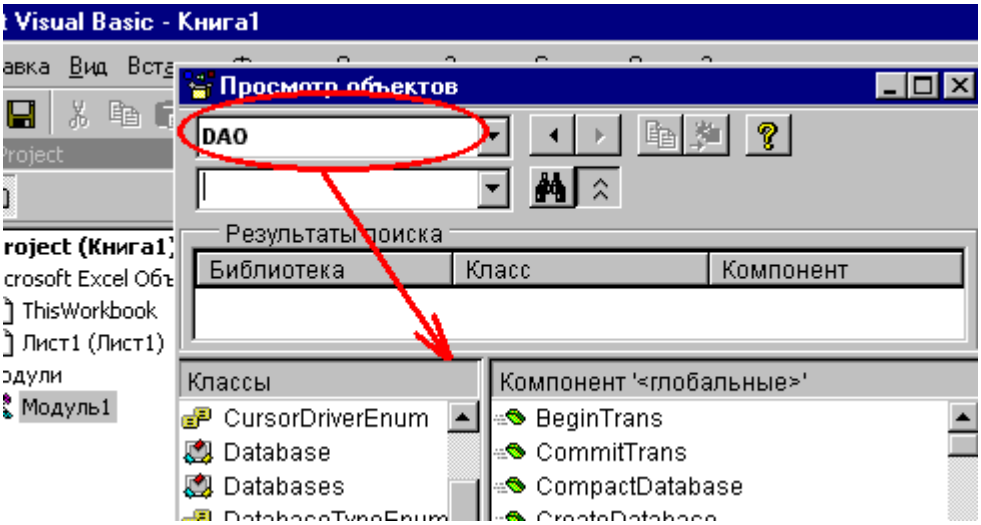
В **Excel** много методов работы с базами данных. Давайте попробуем **DAO** для того, чтобы получить доступ к классам **DAO** нам необходимо их подключить. Это делается из меню "Ссылки":



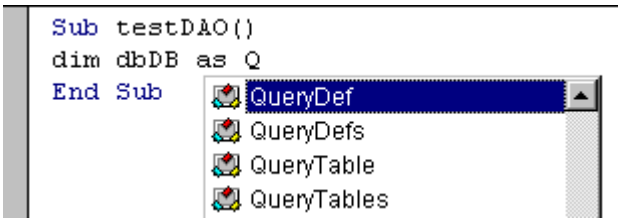
Откроется диалоговое окно, в котором нам надо найти **DAO Object Library**.



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**.

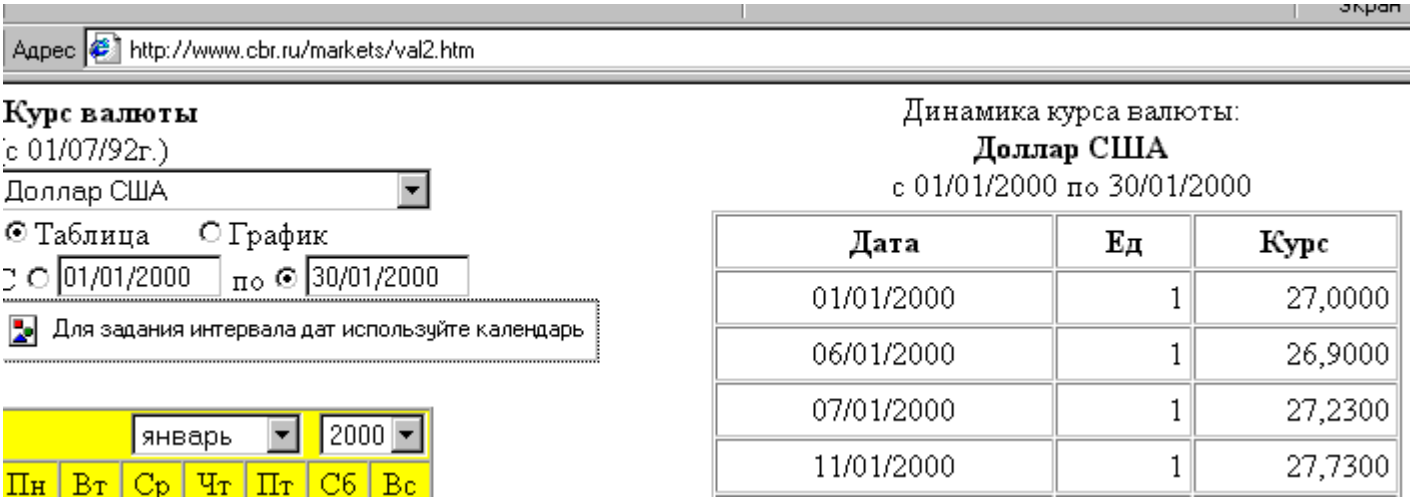


Меню

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:

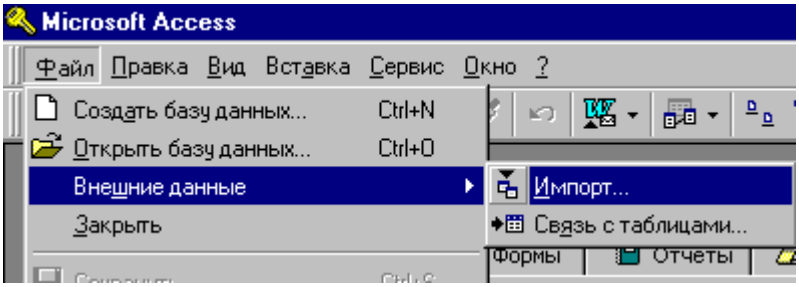


Дата	Ед	Курс
01/01/2000	1	27,0000
06/01/2000	1	26,9000
07/01/2000	1	27,2300
11/01/2000	1	27,7300

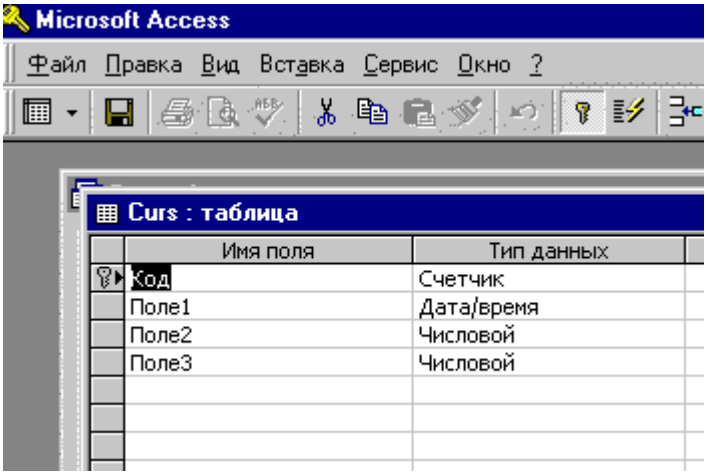
Я просто выделил и скопировал в текстовый файл через буфер обмена. Он примерно такой с именем **Curs.txt**. Он есть в проекте если кому лень.

```
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.



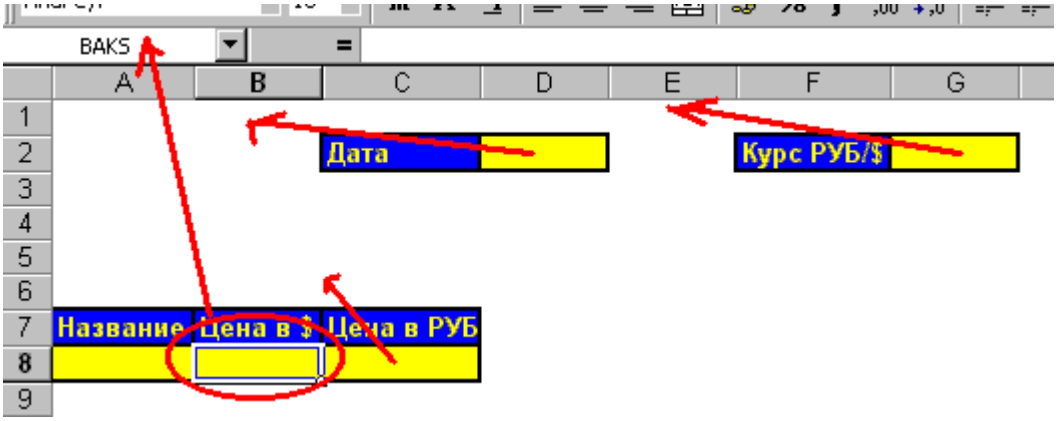
Для особенно ленивых этот файл есть в проекте **curs.mdb**. Только если вы настолько ленивы зачем вообще читаете ???

Меню

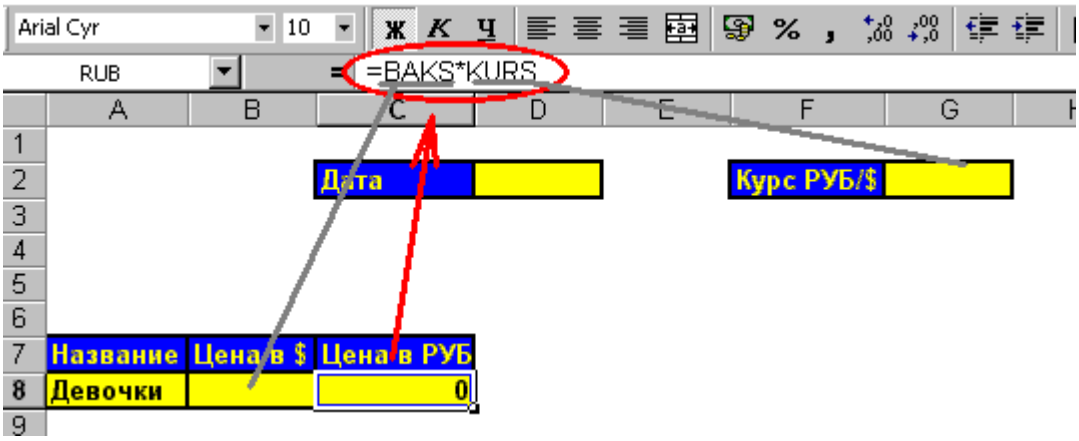
Шаг 43 - Готовим форму

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
```



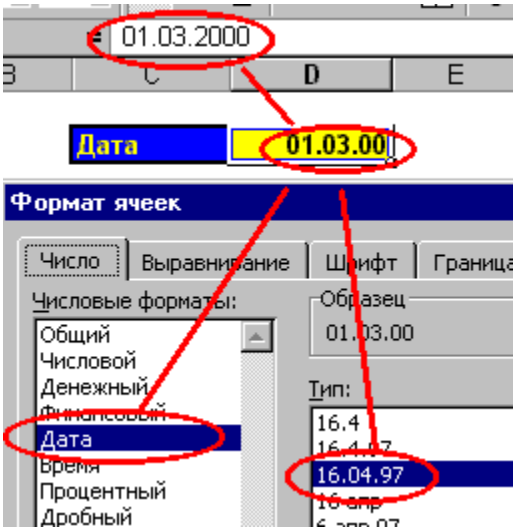
Теперь дело за формулами. Идея ясна как белый день. Стоимость руб = Стоимость в $ * курс. Вот это и запрограммируем:



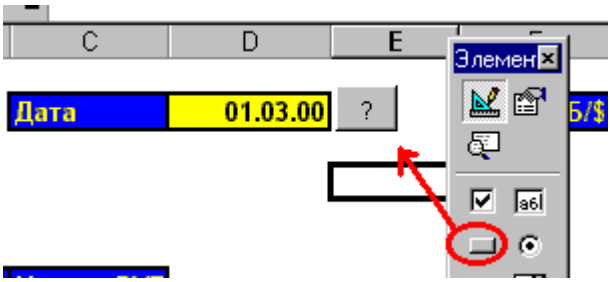
Вводить мы будем цену в долларах, выбирать дату и видеть цену в рублях. Цену в долларах ввести не тяжело :-))). Все остальное посчитается, если будет курс. Курс привязан к дате. То есть мы должны.

- 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.



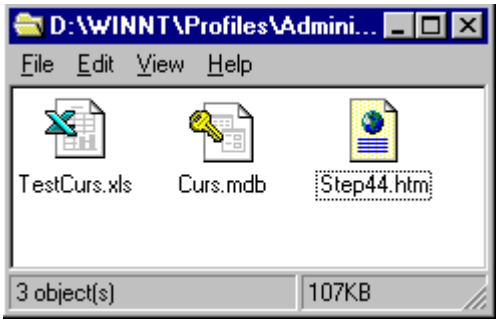
Двойной щелчок создаст макрос. Вот и все на этот шаг. Все готово к программированию. Эта книга есть в проекте, если у Вас что-то неполучилось.

Меню

Шаг 44 - Считаем

```
Голос тихий таинственный
Где ты милый единственный
                        Алсу.
```

Итак, нам нужно связывать **Excel** и **Access**. Где находится **Excel** мы знаем, а вот где файл **MDB** нужно знать и указывать. Но этих проблем можно избежать, если применить такой ход. Если **mdb** будет в том же каталоге, что и **XLS**, то можно создать функцию, которая будет получать этот путь.



Итак мы её поместим в отдельный модуль для этого его еще нужно создать. А вот код.

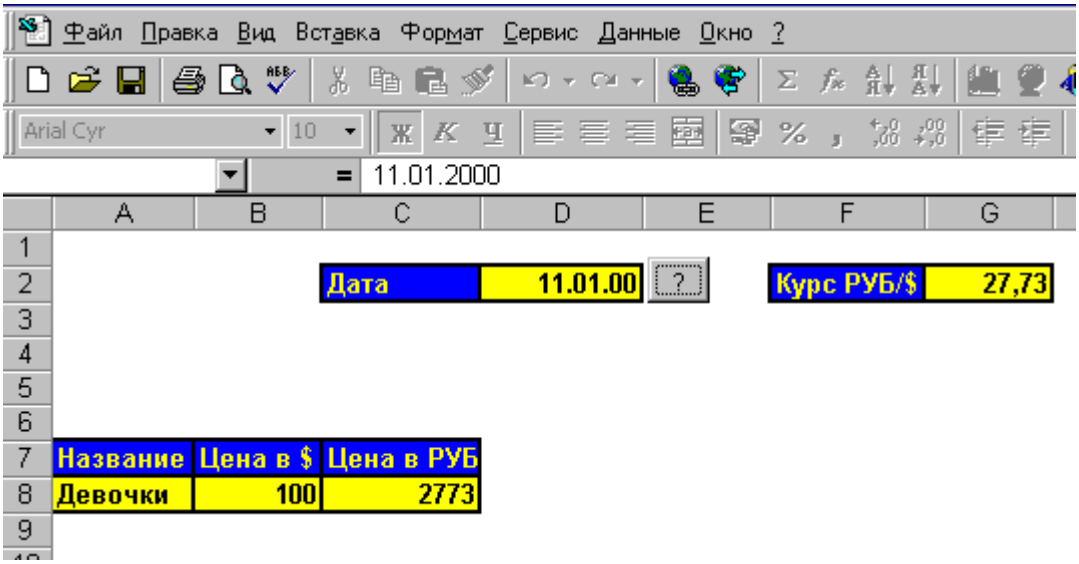
```
Function stDBGetPath() As String
```

```
        Dim stTemp As String
        ' взять путь нахождения активной книги
        stTemp = ActiveWorkbook.Path
        ' прибавить имя базы данных
        stDBGetPath = stTemp + "\" + "curs.mdb"
End Function
```

Вот теперь можно открывать базу данных используя объекты **DAO**. Вот код:

```
Private Sub CommandButton1_Click()
        '---------- объявление переменных ------
        ' переменная типа базы данных
        Dim dbAccess As Database
        ' переменная типа набор записей
        Dim reRecordSet As Recordset
        ' здесь будет SQL запрос
        Dim stSQL As String
        ' переменная типа даты
        Dim daDate As Date

        ' а вдруг ошибка
        On Error GoTo ErrorsDB

        ' ---------- получаем данные из ячейки
        daDate = Range("DATES").Value
        ' ---------- работа с базой ------------
        ' откроем базу данных
        Set dbAccess = OpenDatabase(stDBGetPath)
        ' Строим SQL запрос
        stSQL = "SELECT * FROM[Curs] WHERE[Поле1] =""" & daDate & """"
        ' получаем набор значений
         Set reRecordSet = dbAccess.OpenRecordset(stSQL)
        ' если данные получены тогда занести в ячейку
        If (reRecordSet.RecordCount > 0) Then
                ' поместить значение в ячейку
                Range("KURS").Value = reRecordSet!Поле3
        Else
                MsgBox "Not Found"
        End If

        ' закрываем набор записей
        reRecordSet.Close
        ' закрываем базу данных
        dbAccess.Close
        ' все в норме конец
        GoTo Ends
        ' ошибка где-то однако
ErrorsDB:
        MsgBox "Произошла ошибка"
Ends:
End Sub
```

Поле с датой в файле **Curs.mdb** нужно перевести в текстовый формат. Теперь испытания. Введите дату и цену в $ и нажмите кнопку рядом с датой. Если дата есть, то курс поменяется, иначе получите сообщение, что нет данных **Not Found**. Поменяйте дату и опять нажмите кнопку. Все пересчитается. Вот пример работы:

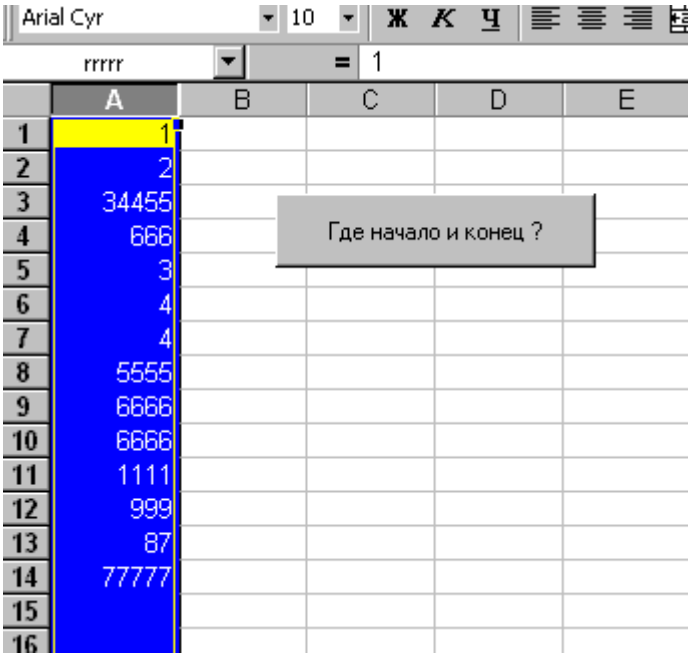


[Меню](#)

Шаг 45 - Начало и конец данных

Если мы захотим изменить наш пример в плане, что нам цена нужна не только одного товара, но и нескольких скажем картошка, лук и апельсины, то возникнет вопрос как это сделать. Ответ это именованный диапазон. Тогда как определить его начало и конец ? В том смысле где находятся цифры ? Вот этим мы и займемся в этом шаге.

Создаем новую книгу и колонку **A** переименовываем в диапазон **rrrrr**.



Так как диапазон есть, теперь к нему можно применять всякие там методы. Выделяем его **Range**, у **Range** есть метод **End** - мол где конец или начало :-))), а у **End** есть адрес вообще-то говоря смотрите !!!

```
Sub TestRange()
        Dim r As Range
        Set r = Range("rrrrr")
        MsgBox (r.Columns.End(xlUp).Address)
        MsgBox (r.Columns.End(xlDown).Address)
End Sub
```

Вот теперь мы и знаем адрес первой и последней ячейки !!!. Привяжем вызов данной процедуры к нажатию кнопки.

```
Private Sub CommandButton1_Click()
        TestRange
End Sub
```

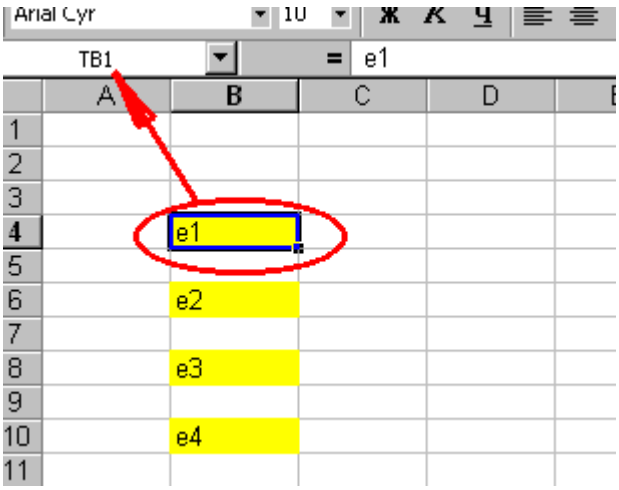
Меню

Шаг 46 - Доступ к одинаковым элементам управления

```
Артем привет!
Есть вот такой код:
        str1.Cells(kod_filiala + 6, 4) = Val(Form2.TextBox1.Text)
        str1.Cells(kod_filiala + 6, 5) = Val(Form2.TextBox2.Text)
        str1.Cells(kod_filiala + 6, 6) = Val(Form2.TextBox3.Text)
        str1.Cells(kod_filiala + 6, 7) = Val(Form2.TextBox4.Text)
        str1.Cells(kod_filiala + 6, 8) = Val(Form2.TextBox5.Text)
        str1.Cells(kod_filiala + 6, 9) = Val(Form2.TextBox6.Text)
        str1.Cells(kod_filiala + 6, 10) = Val(Form2.TextBox7.Text)
        str1.Cells(kod_filiala + 6, 11) = Val(Form2.TextBox8.Text)
Возможно ли все эту писанину заменить на такую
        For i=1 to n
                Условия ......
        end if
т.е. чтобы ТекстБоксы перебирались как массив.
С Уважением, Владимир.
*********************************************
With best wishes, Vladimir(Владимир)
My E-mail stilvlad@chat.ru (text only, koi-8 only, 60 kb),
          stilvlad@mail.ru (all formats).
My Web: http://stilvlad.chat.ru
*********************************************
*Пожалуйста цитируйте всю переписку со мной*
```

Чтож давайте попробуем. Создаем книгу. На ней расположим именованные ячейки.

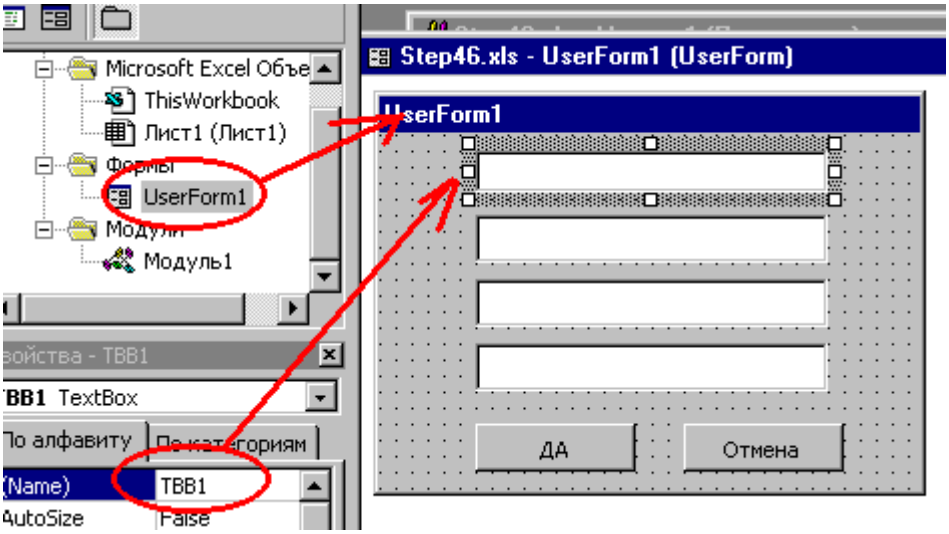
```
TB1
TB2
TB3
TB4
```



Создали форму и на ней элементы **TextBox**. Внимание, сначала создавайте **TextBox**, а потом кнопки, это позволит избежать проверки типов для **Controls**. Имена:

```
TBB1
```

TBB2
TBB3
TBB4



Код по кнопке "ДА":

```
Private Sub CommandButton1_Click()
        ' объект элемент управления
        Dim objObject As Control
        ' объект диапазон
        Dim raRange As Range
        ' строка с адресом диапазона
        Dim stAdders As String
        ' цикл по всем элементам управления
        For x = 0 To UserForm1.Controls.Count - 3
                ' присвоить обьекту элемент управления
                Set objObject = UserForm1.Controls.Item(x)
                ' создать адрес ячейки
                stAdders = "TB" + LTrim(Str(x + 1))
                ' получить диапазон
                Set raRange = Range(stAdders)
                ' присвоить ему значение из элемента управления
                raRange.Value = objObject.Text
        Next x
        ' закрыть форму
        Unload Me
End Sub
```

Код кнопки "нет":

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

Запускной макрос:

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

Вот и все. Попробуйте. Можете загрузить проект и посмотреть.

Меню

Шаг 47 - Свойства документов MSOffice

Аналогично свойству **Tag** форм и элементов управления, свойства документа позволяют программисту сохранять необходимую информацию (о возможности использования свойств, точнее их значений, для поиска документов здесь упоминать не будем - это пользовательская возможность). При необходимости эту информацию можно легко извлечь и использовать.

В VBA для MSOffice определены два типа свойств документов: встроенные (BuiltIn) и пользовательские (Custom). Оба типа свойств организованы в коллекции, соответственно BuiltInDocumentProperties и CustomDocumentProperties, относящиеся к объекту(ам) Document и Template. Нумерация элементов коллекций начинается с единицы.

Встроенные свойства документов автоматически поддерживаются соответствующими приложениями, и без необходимости модифицировать их не стоит, можно считывать и использовать.

Гораздо больший интерес для программиста представляют Custom свойства документа - именно их можно использовать для хранения нужной информации, связанной с документом. Достаточно удобно то, что можно использовать разные типы свойств: строки, числа, даты, и логические поля.

Добавить нужное свойство (здесь и далее будем полагать, что работаем со свойствами активного документа) можно, использовав метод **Add**:

```
ActiveDocument.CustomDocumentProperties.Add _
Name:=PropertyName, _
LinkToContent:=False, _
Value:="", _
Type:=msoPropertyTypeString
```

В этой инструкции:

- **Name:=PropertyName** - имя свойства, может быть сроковым выражением или переменной.
- **LinkToContent:=False**, булево значение, определяющее связь с элементами контейнера в самом документе. Используя False, будет создано статическое свойство, именно оно и интересно. Если установить в True, то надо дополнительно определить LinkSource, указывающий на соотвествующий объект документа.
- **Value:=""**, собственно значение, необязательный параметр.
- **Type:=msoPropertyType**, тип свойства, определены следующие константы типов (в скобках приведены числовые значения констант, тип Long):
  - *msoPropertyTypeBoolean* (2),
  - *msoPropertyTypeDate* (3),
  - *msoPropertyTypeFloat* (5),
  - *msoPropertyTypeNumber* (1), Чем по сути различаются два последних типа - не совсем ясно, по крайней мере оба они способны хранить double число и в стандартном диалог-боксе MSWord у них один тип - Number;
  - *msoPropertyTypeString* (4).

Обязательными элементами являются имя, тип и LinkToContent, значение можно не задавать.

Для считывания некоторого свойства просто пишем:

```
varProperty = ActiveDocument.CustomDocumentProperties.Item(Name)
```

где **Name** имя свойства или номер в коллекции.

Все было бы так просто, если бы не одна маленькая неприятность (по крайней мере, в MSOffice97 это так, а с ним еще долго будут работать пользователи). Неприятность заключается в том, что нет

возможности прямо проверить, создано ли уже свойство с некоторым именем, а обращение к пустому свойству или попытка создания свойства с именем, совпадающим с именем уже определенного, вызывает ошибку (коды соответственно 5 и -2147467259 - не удивляйтесь, а распечатайте err.number!). Но такое поведение можно с легкостью использовать, написав свой обработчик ошибок.

Итак, необходимо присвоить пользовательскому свойству **Test** значение переменной **strString**, при этом не известно, существует ли в данный момент пользовательское свойство (даже если вы его уже когда-то определили, то ведь пользователь мог его удалить, значит такая ситуация является общей, а не частной). Вот фрагмент кода для решения такой задачи.

```
Dim strString as String

strString = "тестовое значение пользовательского свойства Test"
On Error GoTo AddCustomProperty               ' устанавливаем обработчик ошибок
ActiveDocument.CustomDocumentProperties.Item("Test") = strString
On Error GoTo 0                         ' или туда, куда он был установлен ранее

Exit Sub
' ОБРАБОТЧИК(И) ОШИБОК
AddCustomProperty:
        Select Case Err.Number
        Case 5
                ' Этот номер ошибки возникает, если пытаемся писать
                ' в свойство, которое пока не создано
                ActiveDocument.CustomDocumentProperties.Add _
                Name:=PropName, LinkToContent:=False, Value:="",
Type:=msoPropertyTypeString
                Resume ' возвращаем управление в оператор присвоения значения
        Case Else
                ' Вывод сообщений о других ошибках и прерывание программы
                MsgBox(Err.Number & Chr(13) & Chr(13) & Err.Description)
                Exit Sub
End Sub
```

Аналогично, считывание значения пользовательского свойства производится так (тоже ведь никогда нельзя быть уверенным в том, что свойство существует):

```
Dim varProperty as Variant              ' Явное указание типа Variant

On Error GoTo ReadCustomProperty        ' устанавливаем обработчик ошибок
strString = ActiveDocument.CustomDocumentProperties.Item("Test")
On Error GoTo 0                         ' или туда, куда он был установлен ранее
......

Exit Sub
' ОБРАБОТЧИК(И) ОШИБОК
ReadCustomProperty:
        Select Case Err.Number
        Case 5
                ' Этот номер ошибки возникает, если пытаемся читать
                ' свойство, которое пока не создано
                varProperty = "" ' или "Null" или "NoProperty" - как нравится
                Resume Next             ' возвращаем управление следующему оператору
                                ' кстати, вместо возврата строки можно и свойство
создать...
        Case Else
                ' Вывод сообщений о других ошибках и прерывание программы
                MsgBox(Err.Number & Chr(13) & Chr(13) & Err.Description)
                Exit Sub
End Sub
```

Существенно, что при считывании значения свойства производится автоматическое приведение типа (если оно возможно), и считывание, например, числа в стоковую переменную не приводит к ошибке.

Если задан номер пользовательского свойства, то легко можно узнать все его свойства (извините за каламбур), для этого достаточно написать (пример цикла для всех свойств):

```
For Each prop In ActiveDocument.CustomDocumentProperties
        With prop
                MsgBox .Name & "= " & .Value & Chr(13) & _
                        "Application" & " = " & .Application & _
                        "Creator" & " = " & .Creator & _
                        "Parent" & "= " & .Parent
        End With
Next
```

Для всей коллекции пользовательских свойств легко получить число определенных свойств:

```
ActiveDocument.CustomDocumentProperties.Count
```

Очевидно, что индекс в цикле перебора всех свойств может пробегать от 1 до этого значения.

Для ситуации, когда надо создать свойство, можно тоже написать обработчик ошибок, но логика его работы уже будет сильно зависеть от функциональности программы, в которой он используется. Поэтому он здесь не приводится, однако информации для его написания достаточно. Повторим, что код ошибки для ситуации повторного определения свойства с некоторым именем, равен -2147467259 (знак минус!, впрочем, "правильный" код 440).

Меню

Шаг 48 - Встроенные свойства документов MSOffice

Хорошо писать прикладные программы для MSWord - вставил куда-либо поле **{AUTOR}** и получил информацию о том, кто автор документа (иногда, правда, это делать опасно - всплывают прелюбопытные подробности). А в других программах MSOffice? Сколько, например символов в таблице Excel? Так просто не получится... :-(

Для этой задачи может помочь коллекция BuiltInDocumentProperties. В MSOffice97 определено 30 (а не 28, как в документации) свойств, названия которых приведены в таблице. Все эти свойства могут быть прочитаны в любом документе MSOffice, вне зависимости от того, какая программа документ породила. Кроме чтения программным путем, MSWord (это уже упоминалось выше) позволяет вывести значение свойств документа в сам документ при помощи полей, но это к программированию на прямую не относится.

	Название свойства	Назначение
1	Title	Заголовок
2	Subject	Предмет
3	Author	Автор (создавший документ)
4	Keywords	Ключевые слова
5	Comments	Комментарии
6	Template	Шаблон документа
7	Last Author	Тот, кто последний сохранил документ
8	Revision Number	Число входов для редактирования
9	Application Name	Название приложения, обрабатывающего документ

10	Last Print Date	Дата и время последней печати
11	Creation Date	Дата создания
12	Last Save Time	Дата и время последнего сохранения
13	Total Editing Time	Общая продолжительность редактирования (минуты)
14	Number of Pages	Число страниц
15	Number of Words	Число слов
16	Number of Characters	Число символов
17	Security	Секретность
18	Category	Категория
19	Format	Формат
20	Manager	Менеджер
21	Company	Компания
22	Number of Bytes	Размер файла в байтах в момент последнего сохранения
23	Number of Lines	Число строк
24	Number of Paragraphs	Число абзацев
25	Number of Slides	Число слайдов (определено в PowerPoint, в остальных, вероятно, не используется)
26	Number of Notes	Число заметок к слайдам (определено в PowerPoint, в остальных, вероятно, не используется)
27	Number of Hidden Slides	Число скрытых слайдов (определено в PowerPoint, в остальных, вероятно, не используется)
28	Number of Multimedia Clips	Число клипов (определено в PowerPoint, в остальных, вероятно, не используется)
29	Hyperlink Base	База гиперссылок
30	Number of Characters (with spaces)	Общеечисло символов, включая пробелы

Стоит иметь в виду, что при обращении к значению свойства, которое для данного документа не определено, также возникает ошибка выполнения с кодом -2147467259 (знак минус!), что требует аккуратной работы (см. примеры обработчиков ошибок в предыдущем шаге). Также приводит к ошибке попытка записи в свойство, которое данным приложением не поддерживается, что можно установить методом проб и ошибок. Но имена выводятся для всех свойств правильно. Практически полезной может оказаться такая процедура:

```
Sub test()

On Error GoTo NotDefined
' В следующей стороке ActiveDocument стоит заменть на ActiveWorkbook, etc
For Each prop In ActiveDocument.BuiltInDocumentProperties
        With prop
                PName = .Name
                PValue = .Value
                PType = .Type
                Debug.Print PName & " = " & PValue & " [" & PType & "]"
        End With
Next
Exit Sub
' ОБРАБОТЧИК ОШИБОК
NotDefined:
PValue = "Value not defined"
Resume Next
```

```
End Sub
```

Ее польза в том, что можно исследовать, какие свойства определены для данного типа документов, не появились ли новые свойства с выходом новой версии MSOffice (и не потерялись ли старые :-)), какие у них типы. Тип выводится в виде целого числа. Результат ее работы (копия **Debug Window**):

```
Title = Свойства документов MSOffice [4]
Subject =  [4]
Author = DD [4]
Keywords =  [4]
Comments =  [4]
Template = Normal [4]
Last author = DD [4]
Revision number = 4 [4]
Application name = Microsoft Word 8.0 [4]
Last print date = Value not defined [3]
Creation date = 17.03.00 14:13:00 [3]
Last save time = 17.03.00 15:06:00 [3]
Total editing time = 67 [1]
Number of pages = 3 [1]
Number of words = 1183 [1]
Number of characters = 7702 [1]
Security = 0 [1]
Category =  [4]
Format =  [4]
Manager =  [4]
Company = SBC [4]
Number of bytes = 45056 [1]
Number of lines = 235 [1]
Number of paragraphs = 161 [1]
Number of slides = Value not defined [1]
Number of notes = Value not defined [1]
Number of hidden Slides = Value not defined [1]
Number of multimedia clips = Value not defined [1]
Hyperlink base =  [4]
Number of characters (with spaces) = 8995 [1]
```

Соответствие числа и предопределенных констант типов - ниже:

```
1       msoPropertyTypeNumber
2       msoPropertyTypeBoolean
3       msoPropertyTypeDate
4       msoPropertyTypeString
5       msoPropertyTypeFloat
```

Свойства, недоступные в данном приложении, будут иметь вид: **Number of slides = Value not defined [1]**

Меню

Шаг 49 - Связывание макроса с кнопкой на панели инструментов

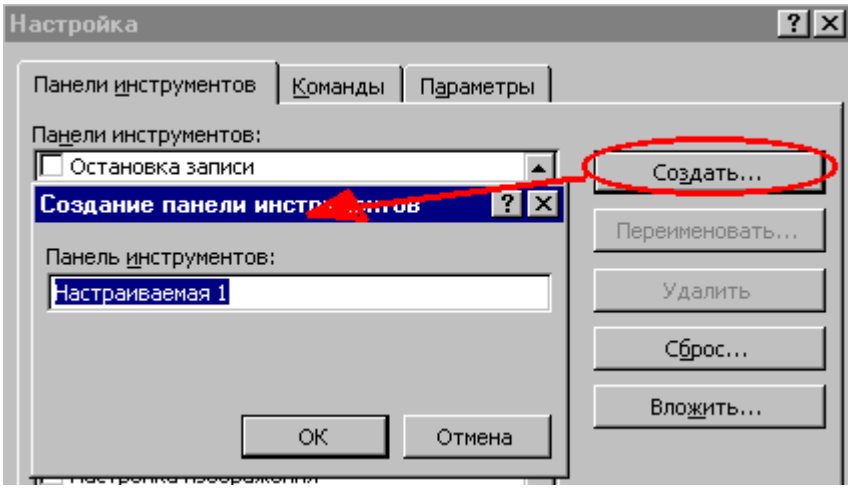
Создавая интерфейс Ваш код на **VBA** в виде макросов можно разместить для доступа пользователей в несколько позиций:

- Меню
- Панели инструментов
- Кнопки формы

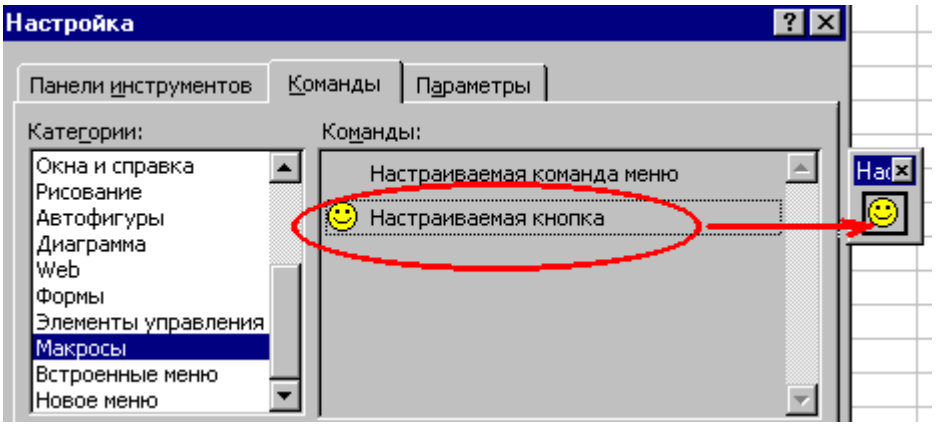
Панели инструментов являются наиболее удобными и быстрыми для доступа. Для начала давайте создадим простой макрос. Ну хотя бы вот такой:

```
Sub Test()
        MsgBox "Hello"
End Sub
```

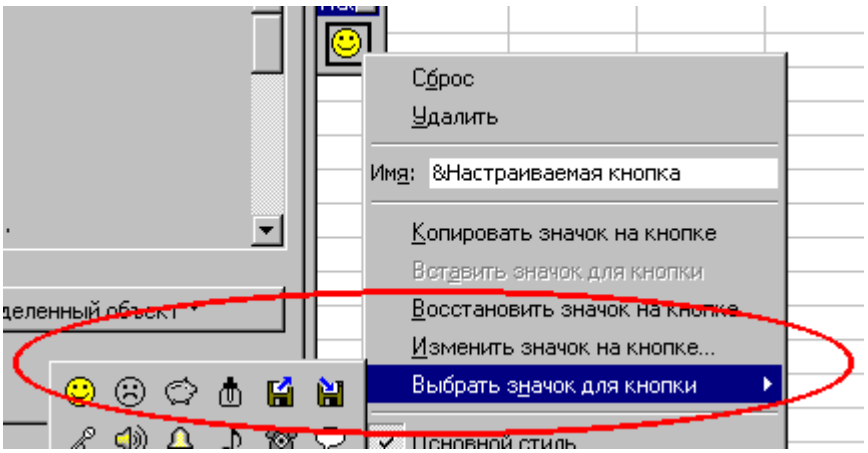
Теперь у нас есть, что связывать. Создать свою панель инструментов можно через меню **Сервис - Настройка**. У Вас по вызову этого пункта меню появляется диалоговое окно с тремя вкладками. Нас интересует сейчас "Панели инструментов", на которой находится кнопка **Создать.**



По нажатию на эту кнопку у Вас появится диалоговое окно с предложением ввести имя. После этого можно смело жать **OK**. В **Excel** у Вас появится новая панель инструментов. Следующим шагом является помещение на эту панель настраиваемой кнопки. Нам нужно перейти на вкладку "Команды" и в списке "Категории" найти "Макросы", у Вас справа появится значек настраиваемая кнопка. Схватите ее мышкой и перенесите на панель инструментов.



С этого момента у нас есть возможность менять значек на кнопке. Вы можете выбрать готовый или создать свой. Нажмите правую кнопку мыши на кнопке и посмотрите.



После того как выбрали значек можно и привязать макрос. Опять по правой кнопке мыши есть пункт меню "Назначить макрос". Появится стандартное диалоговое окно с выбором макроса. Выберите его. Теперь осталось закрыть диалоговое окно. Если панель нужно отредактировать выберите опять этот пункт меню.

Меню

Шаг 50 - Определяем выделенную ячейку

Давайте посмотрим как определить событие выбора определной ячейки. Первое - это надо определить, что вообще что-то выбрали. Для этого создайте книгу. Запустите редактор **VBA** и щелкните на листе. Для данного листа будет создана функция **SelectionChange**, если ее нет, то у Вас всегда есть возможность ее выбрать. Теперь впишите код сообщения в функцию, чтобы увидеть, что происходит.

```
Private Sub Worksheet_SelectionChange(ByVal Target As Excel.Range)
        MsgBox Target.Address
End Sub
```

У Вас теперь есть возможность щелкать по ячейкам конкретного листа, я еще раз повторяю и вы получите сообщение с адресом ячейки. Сообщение выбора ячейки обрабатывается локально. А что если у нас надо обрабатывать это сообшение с трех листов и желательно вместе. Для этого нужно создать модуль и в нем процедуру обработки. Я пока поместил туда просто вывод информационного окна.

```
Public Sub Selection_Cell(ByVal Addres As String, ByVal List As String)
        MsgBox Addres + " " + List
End Sub
```

И соотвественно надо пересылать туда данный с каждого листа.

```
Private Sub Worksheet_SelectionChange(ByVal Target As Excel.Range)
        Selection_Cell Target.Address, Target.Worksheet.Name
End Sub
```

Вот с этого момента можно вставлять вызов функции **Selection_Cell** на каждый лист в ответ на реакцию выделения.

Но это еще не все. Дело в том, что передается объект типа **RANGE**, то есть нет разницы передана одна ячейка или диапазон. Но при выделении диапазона внутрь него может попасть и наша ячейка. Итак, нам надо бы анализировать диапазон. Давайте договоримся, что нам надо обязательно отловить момент выделения ячейки **$A$1**. Вот на всех листах. У нас для этого все есть. Процедура общая, в которой есть адрес. И реакция на выделение на каждом листе. Смотрим код:

```
Public Sub Selection_Cell(ByVal Addres As String, ByVal List As String)
        On Error GoTo Ends
        Dim Test As Range
        Dim Find As Range
        Dim Result As Range
        Dim x As Integer

        Set Find = Range("$A$1")
        Set Test = Range(Addres)

        Set Result = Intersect(Test, Find)
        x = Result.Count
        MsgBox "$A$1"
Ends:
End Sub
```

Идея простая. Мы имеем переданный диапазон **Test** и нужный **Find**. Как узнать, что один включает другой ??? Вызовем операцию пересечения **Intersect**. Результатом будет диапазон, который содержит пересечение. Так вот если искомая нам ячейка в нем есть, то нормально, а если нет, то обращение к **Count** вызовет ошибку. Этим мы и воспользовались.

Меню

Шаг 51 - Изучаем события Excel Workbook

Открытие книги:

```
Private Sub Workbook_Open()
End Sub
```

Активация окна. Вызывается в момент активации книги. Этот момент наступает в тот момент, когда из другой активной книги вы переходите к той, в которой обрабатывается событие. Например, при переключении книги в меню "Окно".

```
Private Sub Workbook_Activate()
End Sub
```

Закрытие окна. Вызывается перед закрытием книги:

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)
End Sub
```

Деактивация книги. Вызывается в момент перехода к другой книге:

```
Private Sub Workbook_Deactivate()
End Sub
```

Активация листа. Вызывается в момент смены листа:

```
Private Sub Workbook_SheetActivate(ByVal Sh As Object)
End Sub
```

Деактивация листа. Вызывается в момент смены листа для того листа, с которого уходят:

```
Private Sub Workbook_SheetDeactivate(ByVal Sh As Object)
End Sub
```

Смена ячейки. Вызывается в момент смены диапазона или при редактировании ячейки:

```
Private Sub Workbook_SheetChange(ByVal Sh As Object, ByVal Target As Excel.Range)
End Sub
```

Пересчет данных. Вызываться при пересчете данных, обычно пересчет связан с редактирование данных, поэтому ранее вызывается **SheetChange** практически всегда за исключением ручного пересчета.

```
Private Sub Workbook_SheetCalculate(ByVal Sh As Object)
End Sub
```

Переход к ячейке. Вызывается при переходе от одной ячейки к другой:

```
Private Sub Workbook_SheetSelectionChange(ByVal Sh As Object, ByVal Target As
Excel.Range)
End Sub
```

Разрешение на печать. Вызывается при выборе меню "Печать":

```
Private Sub Workbook_BeforePrint(Cancel As Boolean)
End Sub
```

Разрешение на сохранение. Вызывается перед сохранением документа:

```
Private Sub Workbook_BeforeSave(ByVal SaveAsUI As Boolean, Cancel As Boolean)
End Sub
```

Создание нового листа. Вызывается в момент создания нового листа. Обычно ведет за собой целую цепочку событий - **Workbook_NewSheet, Workbook_SheetDeactivate, Workbook_SheetActivate**:

```
Private Sub Workbook_NewSheet(ByVal Sh As Object)
End Sub
```

Изменение размера листа. Вызывается в момент изменения размера листа:

```
Private Sub Workbook_WindowResize(ByVal Wn As Excel.Window)
End Sub
```

Активизация окна. Вызывается в момент активизации окна книги, например, при переключении на книгу:

```
Private Sub Workbook_WindowActivate(ByVal Wn As Excel.Window)
End Sub
```

Щелчок правой кнопкой. Вызывается по нажатию правой кнопки мыши:

```
Private Sub Workbook_SheetBeforeRightClick(ByVal Sh As Object, ByVal Target As
Excel.Range, Cancel As Boolean)
End Sub
```

Выводы. Часть событий связана в цепочки. То есть, например, событие **Activate** для одного объекта это событие **Deactivate** для другого объекта. Это работает для Окон, Книг и Листов. Часть событий вызывают за собой следующие события, например, ввод данных приведет к пересчету листа. Или вставка нового листа к событиям активизации. Знание событий может помочь в решении нетривиальных вопросов. Вот так можно запретить печать книги.

```
Private Sub Workbook_BeforePrint(Cancel As Boolean)
Cancel = True
End Sub
```

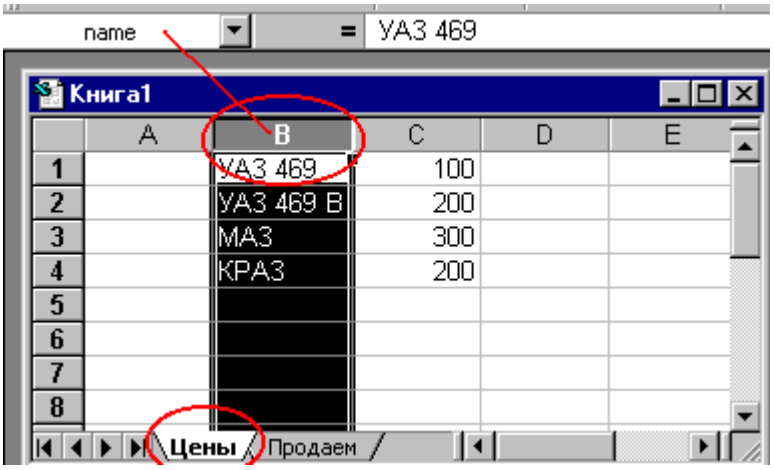
Меню

Шаг 52 - Автоматизация на основе СУММЕСЛИ

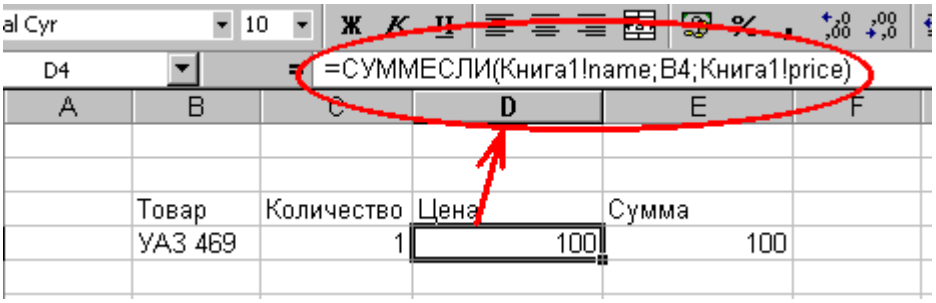
Excel предоставляет огромное количество возможностей и знание этих возможностей значительно может помочь при создании приложений. Давайте рассмотрим обычную ситуацию с созданием счетов. У вас есть список товаров с ценами. Вы готовите счет с набором товаров в разных количествах.

Создаем новую книгу и лист переименовываем в "Цены", а на нем создаем два именованных диапазона. Диапазон **name** и диапазон **price**. В одном будет название товара, во втором цена. Внимание **название товаров должно быть уникальным**. То есть в таком варианте если делать как описано в шаге. Конечно сейчас на складах есть одни и теже товары с разными ценами. Но ведь к названию можно что-то добавить ??? Правда же ??? Именовать нужно целую колонку. Зачем ???

Чтобы спокойно добавлять новые товары и не думать о изменениях. Вот посмотрите рисунок демонструющий суть идеи.

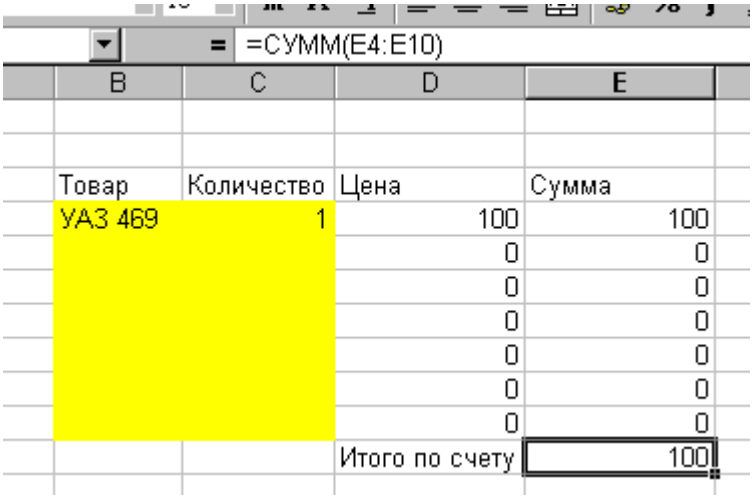


Делаем второй лист с названием "продаем". Здесь мы будем формировать наш счет. Помножить количество на цену и получить сумму, ну это не проблема. А вот как узнать цену исходя из названия ??? Тут нам и поможет **СУММЕСЛИ**, вот смотрите:



Здесь написано, что суммировать по диапазону **price**, если в диапазоне **name** попадется название равное указанному в ячейке **B4**. На самом деле если названия уникальны никакого суммирования не будет, просто выберется нужная цена. Почему диапазон сделан на целую колонку ??? Чтобы спокойно добавлять или удалять названия и ни о чем не думать. И еще если в диапазоне указать что-то типа **A1:A100**, то при растаскивании вниз **Excel** будет менять адреса. А вот если у Вас диапазон не на всю колонку, то при растаскивании формул диапазон не будет меняться.

Ну вот и все. Достаточно ввести название, количество и вы получите общую цену. Надо еще один товар растащите формулу вниз и она будет работать. Теперь при доработке листа вам останется лишь вводить названия и количество.



Меню

Шаг 53 - Разбор строки стандартными функциями

Итак, ситуация простая. У нас в одной ячейке записано имя и фамилия человека. Конечно это не хорошо. Но дело сделано и так оно есть. Строка представляет из себя набор символов, даже если она в ячейке **Excel** :-)

И	В	А	Н	О	В		К	О	Л	Я

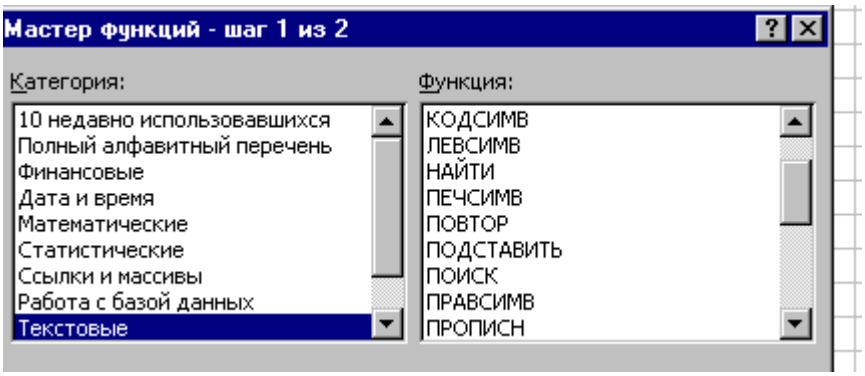
Соотвественно раз это набор символов, то должны быть и функции для определения количества символов. Они есть. Вот она **=ДЛСТР(B5)**, если в **B5** будет та надпить, то она вернет 11. Естественно, что первая буква будет первой :-)

Чтобы разделить имя и фамилию нам надо найти символ, на котором заканчивается имя. Этим символом почти всегда является пробел. Должна быть функция, которая умеет искать место символа в строке. Она тоже есть, **=НАЙТИ(" ";B5;1)** вернет первую позицию, в которой находится пробел. В данном случае 7.

Для отбора строки нам надо выбрать символ от начала до пробела и от пробела до конца. Начало мы знаем - это 1, пробел знаем - это 7, конец тоже - это 11. Осталось найти только функцию вырезания.

```
=ПСТР(B5;1;D5)
=ПСТР(B5;D5;C5)
```

Эта функция вырезает из строки символы от указанного значения до указанного. Соотвественно от начало до пробела - это фамилия, а от пробела до начала - имя. Все функции есть в мастере "функция" в разделе "текстовые".



Вывод отсюда простой. Многие задачи можно сделать и без функций работы со строками **VB(A)**. И, наверно, разумный компромис между **VBA** и возможностями **Excel** - это и есть мастерство создания приложений на базе **Excel.**

Меню

Шаг 54 - Подробнее о событиях загрузки и выгрузки формы

Форма в **VBA** это каркас приложения. Как добавлять форму в Ваш проект смотрите "Шаг 15 - Пользовательские формы". В основном события формы по ее инициализации и деинициализации разворачиваются в таком порядке:

```
Initialize
```

```
Load
Activate
Deactivate
QueryUnload
Unload
Terminate
```

Но форма в **VBA** и **VB** различаются. Давайте сравним:

VBA	VB
UserForm_Initialize()	Form_Initialize()
Нет	Form_Load()
UserForm_Activate()	Form_Activate()
UserForm_Deactivate()	Form_Deactivate()
UserForm_QueryClose(Cancel As Integer,CloseMode As Integer)	Form_QueryUnload(Cancel As Integer, UnloadMode As Integer)
нет	Form_Unload(Cancel As Integer)
UserForm_Terminate()	Form_Terminate()

Получается, что событие **Load** и **Unload** в **VBA** не обрабатываются.

Событие Initialize

Событие **Initialize** (Инициализация) обычно используется для подготовки к работе приложения или формы **UserForm**. Переменным присваиваются исходные значения, а положение или размеры элементов управления могут быть изменены для согласования с данными, заданными при инициализации. Это событие появляется до загрузки формы и ее отображения. Это событие появляется во время загрузки формы. Давайте в него напишем код:

```
Private Sub UserForm_Initialize()
MsgBox "UserForm_Initialize"
End Sub
```

А теперь две функции, которые вызывают **Load**:

```
Sub Test()
        Load UserForm1
        Call Test2
End Sub

Sub Test2()
        Unload UserForm1
        Load UserForm1
End Sub
```

В результате окно с информацией о инициализации будет на экране два раза. Так же это событие сгенерирует событие **Show**, так как первый раз для работы с формой ее нужно загрузить. Еще это событие может быть вызвано, если в форме определена функция общего назначения. Вызов этой функции опять приводит к загрузке формы.

```
Private Sub UserForm_Terminate()

End Sub

Public Sub MyMessage()
        MsgBox "MyMessage"
End Sub
```

А вот так можно вызвать:

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

Итак, событие **Intialize** вызывается только один раз для формы перед ее загрузкой.

Событие Load

Нет ее в **VBA**, а вообще в **VB** здесь можно что-то сделать перед выводом формы на экран.

Событие Activate и Deactivate

Событие **Activate** происходит, когда объект становится активным окном. А становится активным окном он может в двух случаях. Это в результате **Show**, когда форма становится видимой на экране и в результате получения фокуса. Событие **Deactivate** (Деактивизация) происходит, когда объект более не является активным окном. Эти события генерируются только при переключении между окнами одного приложения. Если вы перейдете в другую программу и вернетесь в **Excel**, то эти события не будут сгенерированы.

Событие QueryClose

Это событие генерируется для того, чтобы получить у пользователя разрешение на закрытие формы. В зависимости от кода возврата форма закрывается или нет.

```
Private Sub UserForm_QueryClose(Cancel As Integer, CloseMode As Integer)
        If CloseMode <> 1 Then Cancel = 1
                UserForm1.Caption = "Close не будет работать. Выберите меня!"
        End IF
End Sub
```

Событие Unload

Нет его, точно так же как и **Load**, а вообще используется, если надо что-либо делать при выгрузке формы, например сохранить настройки пользователя.

Событие Terminate

Данное событие происходит, когда все ссылки на экземпляр объекта удаляются из памяти с помощью присвоения всем переменным, которые ссылаются на данный объект, состояния **Nothing** или когда последняя ссылка на объект выходит за пределы области определения. Это событие идет вслед за **Unload**.

Меню

Шаг 55 - Четыре основных метода работы с формой (Load,Unload,Show,Hide)

Давайте сформулируем краткие описания и после этого посмотрим как они взаимодействуют:

- **Load** - загрузка формы в память
- **Unload** - выгрузка формы из памяти
- **Show** - показ формы на экране
- **Hide** - спрятать форму

Мы помним, что **Show** может сам вызвать **Load**, но если форма загружена, то **Load** не будет вызываться. Поэтому просто надо делать так:

```
Начало работы
Load
Конец работы
Unload
```

Показывать или прятать форму можно с помощью **Show** и **Hide**. Но, если Вы проводите в методе **Initialize** настройку переменных, то будьте уверены, что она будет происходить при загрузке в момент вызова **Load.**

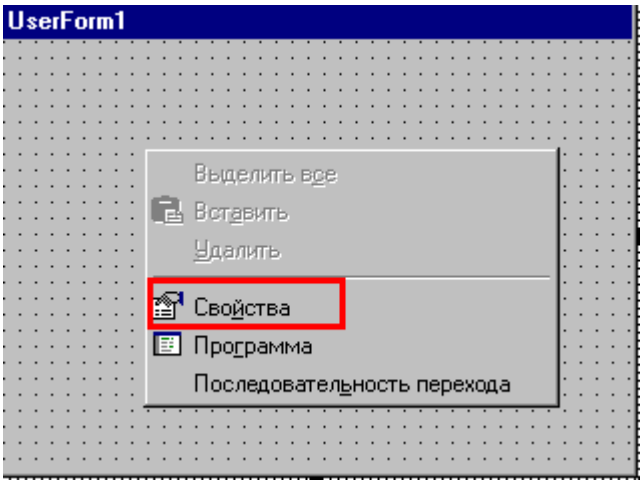
Вот типовые коды:

```
Load
Load UserForm

Unload
Unload UserForm
Из формы
Unload Me

Hide
UserForm.Hide
Из формы
Me.Hide

Show
UserForm.Show
```

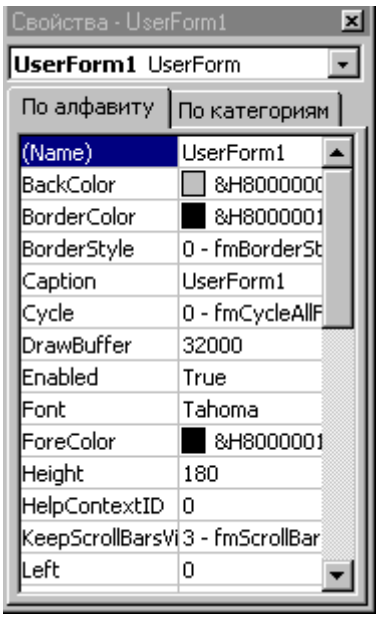
Меню

Шаг 56 - Настройка свойств формы

Свойства формы можно менять, для этого достаточно вызвать меню левой кнопкой на форме.



В результате этого появится окно свойств формы.



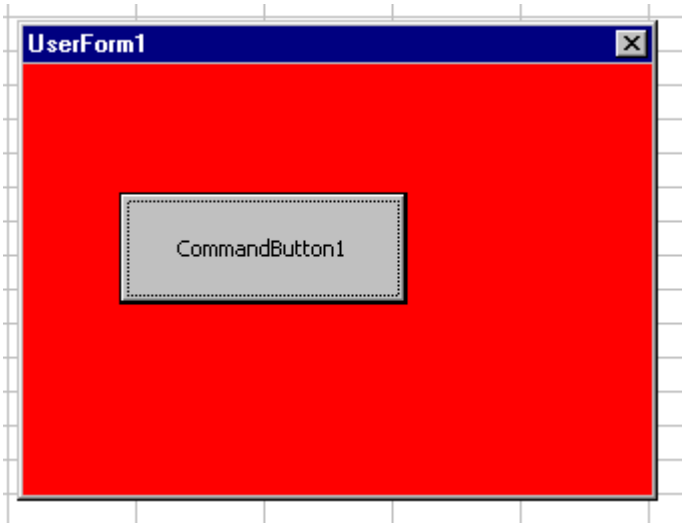
Так же свойства формы можно изменять и в период выполнения. Для этого нужно указать форму, затем через точку название свойства и использовать присваивание для смены свойства. Давайте, например, поменяем цвет формы по нажатию на кнопку на самой же форме.

```
Private Sub CommandButton1_Click()
        UserForm1.BackColor = RGB(255, 10, 10)
End Sub
```

Если запустить эту форму по нажатию на кнопку, то она станет красная.

```
Sub test()
        Load UserForm1
        UserForm1.Show
        Unload UserForm1
End Sub
```

А вот результат.



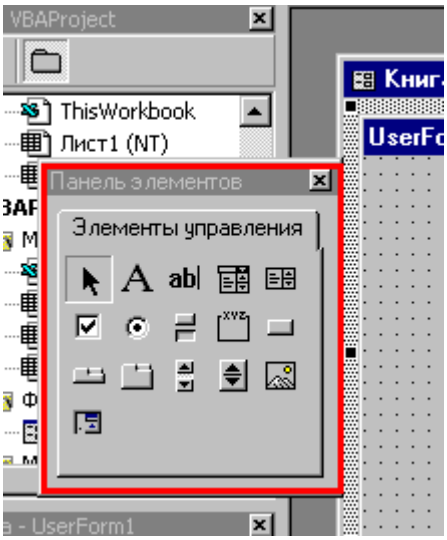
Меню

Шаг 57 - Элементы для формы

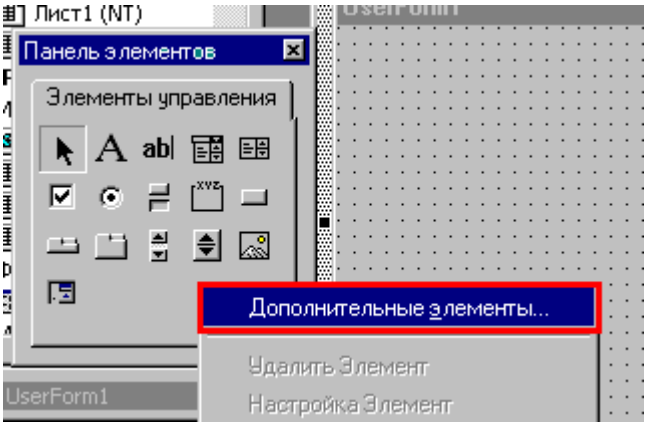
Элемент управления, наверно, вторая главная составляющая интерфейса **VBA** после форм. Элементы управления позволяют Вам добавить в программу функциональность. Их два типа:

- Встроенные - **inherent**
- Нестандартные - **custom**

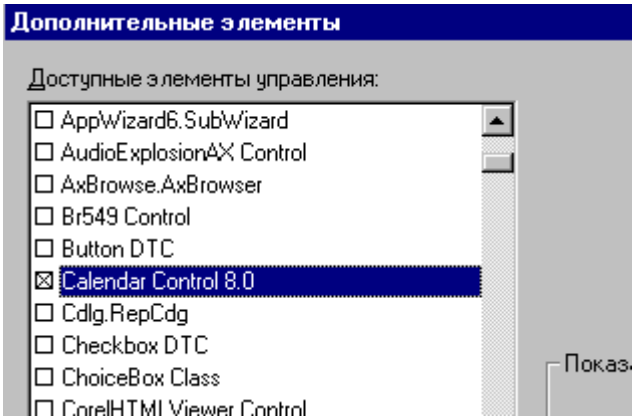
По умолчанию в новом проекте в панели инструментов находятся только нестандартные элементы управления.



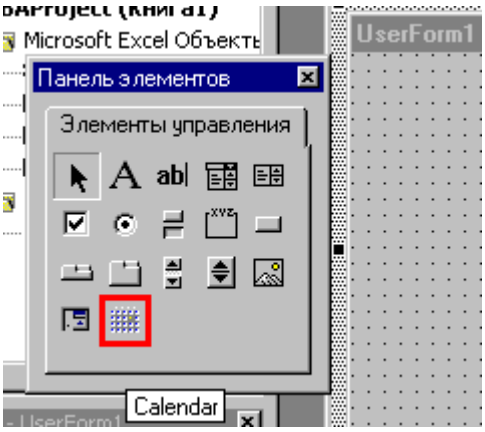
Но у Вас есть возможность добавлять в эту панель элементы **ActiveX**, которые могут существенно увеличить функциональность. Эти элементы в виде файлов **OCX** на данный момент реализовывают практически все. Достаточно нажать правую кнопку мыши на свободном месте в форме элементов управления и появится меню.



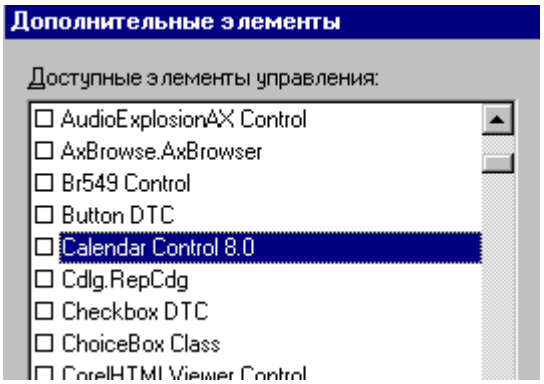
Щелкнув по меню "дополнительные элементы" вы сможете выбрать любой элемент из тех, которые установлены в системе. Например, выберите календарь и нажмите кнопку **OK**.



После этого значок будет в элементах управления.



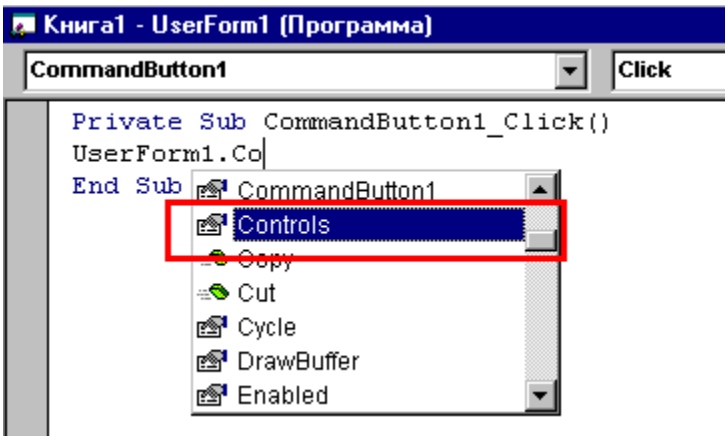
Отсюда его можно помещать на форму. Для удаления его с панели инструментов достаточно снять галочку напротив него в дополнительных компонентах.



Меню

Шаг 58 - Наборы элементов управления

В шаге "Шаг 46 - Доступ к одинаковым элементам управления" я показывал код как можно пройтись по всем элементам управления, но ничего практически не рассказал. Давайте обсудим это подробнее. **VBA** предоставляем массив включающий все элементы на форме с именем массива **Controls**:



У этого массива есть ряд методов, но только одно свойство. Это свойство **Count**. Данное свойство возвращает количество элементов на форме.

```
Private Sub CommandButton1_Click()
        MsgBox UserForm1.Controls.Count
End Sub
```

С помощью этого массива можно ссылаться на элемент по индексу или по имени. У меня на форме два текстовых элемента, вот я их и спрячу двумя способами по нажатию на кнопку.

```
Private Sub CommandButton1_Click()
        Controls(0).Visible = False
        Controls("TextBox2").Visible = False
End Sub
```

Для того, чтобы пробежаться по всем элементам можно использовать цикл **For Each**. Следующий код скрывает все текстовые элементы управления:

```
Private Sub CommandButton1_Click()
        Dim ctrl As Control
        For Each ctrl In Controls
                If TypeName(ctrl) = "TextBox" Then
                        ctrl.Visible = False
                End If
        Next ctrl
End Sub
```

Функция **TypeName** возвращает значение типа **String**, представляющее тип переменной за исключением типа определенного пользователем с помощью **Type**

Меню

Шаг 59 - Проверка ввода на уровне формы (KeyDown, KeyUp, KeyPress)

События:

```
KeyDown
KeyUp
KeyPress
```

Посылаются форме когда производится ввод данных. Событие **KeyPress** возникает когда пользователь нажимает клавишу, у которой есть **ASCII** код. Это событие не сгенирируется при нажатии функциональных клавиш. При том эти события есть как у формы:

```
Private Sub UserForm_KeyPress(ByVal KeyAscii As MSForms.ReturnInteger)

End Sub
```

Так и у элементов управления **TextBox**, например:

```
Private Sub TextBox1_KeyDown(ByVal KeyCode As MSForms.ReturnInteger, ByVal Shift As
Integer)

End Sub
```

А это в свою очередь позволяет проверять ввод. Например, если в поле нужно ввести цифру, то легко проверить, что вводит пользователь. События **KeyUp** и **KeyDown** применяются для специальных функциональных клавиш или комбинаций типа **Ctrl+Del**, так как событие **KeyPress** их не отловит. Удобно использовать данное событие для проверки заполненности полей. Вот форма:

А вот код для нее. В момент нажатия ввода символа в поле проверяется заполненость полей:

```
Private Sub Test()
        Dim ctrl As Control
        Dim bool As Boolean
```

```
        bool = True
        For Each ctrl In Controls
                If TypeName(ctrl) = "TextBox" Then
                        If ctrl.Text = "" Then
                                bool = False
                        End If
                End If
        Next ctrl
        UserForm1.CommandButton1.Enabled = bool
End Sub

Private Sub TextBox1_KeyPress(ByVal KeyAscii As MSForms.ReturnInteger)
        Call Test
End Sub

Private Sub TextBox2_KeyPress(ByVal KeyAscii As MSForms.ReturnInteger)
        Call Test
End Sub

Private Sub TextBox3_KeyPress(ByVal KeyAscii As MSForms.ReturnInteger)
        Call Test
End Sub
```

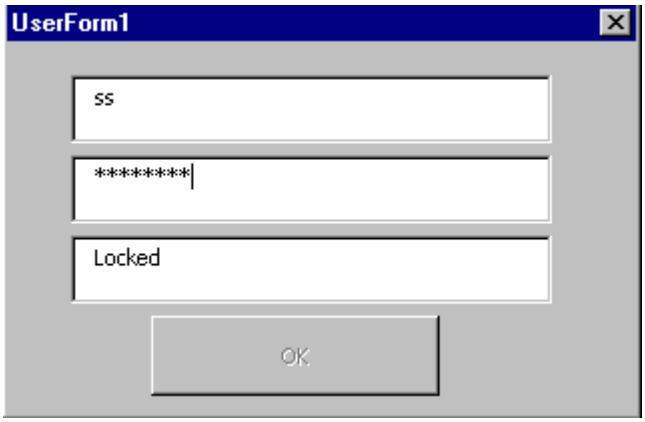
Меню

Шаг 60 - Проверка и настройка ввода в TextBox

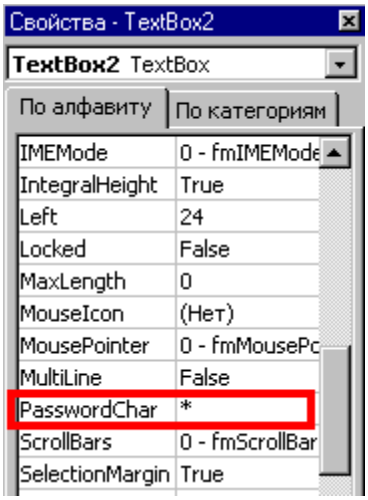
У элемента управления есть несколько свойств, которые позволяют на этапе проектирования ограничить ввод пользователя. Вот они:

```
MaxLength
PasswordChar
Locked
```

Свойство **MaxLength** позволяет ограничить количество символов, которые будут введены. Если пользователь попробует ввести больше, то будет звуковой сигнал. **PasswordChar** не дает возможности просматривать вводимые символы заменяя их на звездочки (*****). Это полезно, как видно из имени, при вводе пароля. Свойство **Locked** определяет может ли пользователь редактировать поле. Посмотрите пример:



В верхнем поле можно ввести только два символа. Во втором поле вместо символов звездочки, а последнее поле нельзя изменить. Кстати в **PasswordChar** не обязательно должна быть звездочка, то есть может быть любой другой знак.



Но в большинстве приложений принята звездочка и смысла пугать пользователя экзотическими знаками наверно нет.

Кстати **Locked** говорит, что поле для чтения, но в некоторых ситуациях его можно открыть для редактирования программным путем.

```
Private Sub CommandButton1_Click()
        TextBox3.Locked = False
End Sub
```

Тоже самое и с **MaxLength**:

```
Private Sub CommandButton1_Click()
        TextBox3.Locked = False
End Sub
```

И **PasswordChar**:

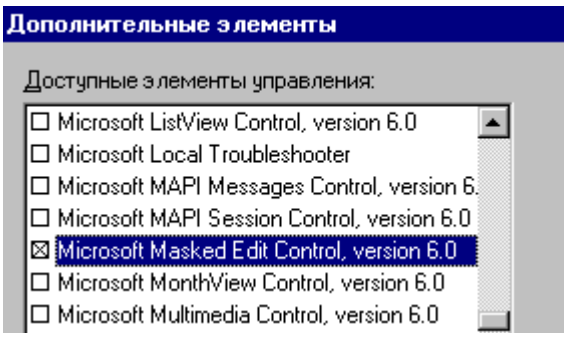
```
Private Sub CommandButton1_Click()
        TextBox2.PasswordChar = "x"
End Sub
```

При изменении свойств данные в полях не пропадают.

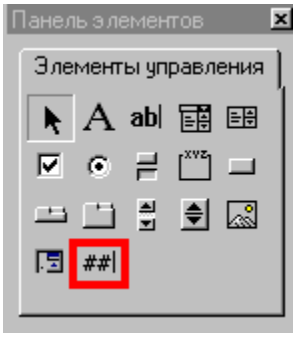
Меню

Шаг 61 - О MaskEdBox

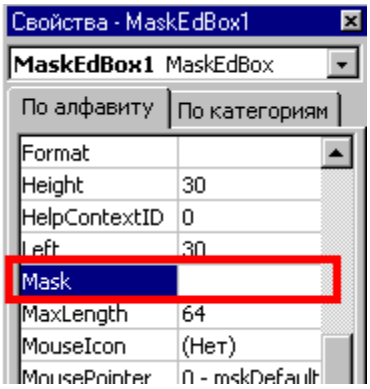
Это элемент ввода, в котором можно использовать маску ввода. Вам нужно ее добавить.



В результате на панели инструментов добавится новый значок.



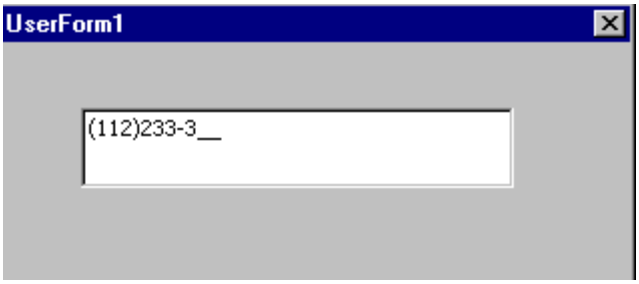
Одно из его важных свойств - это свойство **Mask**:



С помошью этой маски легко задать форматирование текста. Например, вот такое для ввода телефона.

```
(###)###-###
```

Теперь при вводе у Вас будет шаблон:



Для программной очистки необходимо ввести саму маску, например, вот так:

```
Private Sub CommandButton1_Click()
        MaskEdBox1.Text = "(___)___-___"
End Sub
```

Кодов для ограничения ввода много, но вот минимальные:

```
# - число
a - символ
```

Меню

Шаг 62 - Maskedit - Text и ClipText

Эти два свойства позволяют получить данные из элемента **Maskedit**. Отличие этих свойств в том, что поле **Text** возвращает строку вместе с маской, а **ClipText** без маски. Давайте посмотрим. Вот данные введенные в маску.

Давайте посмотрим как они будут возвращаться в случае применения этих двух свойств. Делаем код для кнопок:

```
Private Sub CommandButton1_Click()
        Debug.Print MaskEdBox1.Text
        Debug.Print MaskEdBox1.ClipText
End Sub
```

И смотрим результат:

```
(12)34
1234
```

Данные свойства удобны при помещении данных в базу данных и обратно с маской, которую база данных не поддерживает.

Меню

Шаг 63 - Обработка ошибок в VBA

Обрабатывать ошибки можно тремя способами:

1. Строчная обработка
2. Создание локального обработчика
3. Создание глобального обработчика

Для строчной обработки ошибок применяется функция **On Error Resume Next**, при использовании этой функции выполнение работы программы не прерывается. В этот момент в объект **Err** помешается код ошибки, который можно выяснить через свойство **Number**. После обработки ошибки его необходимо очистить, воспользовавшись методом **clear**:

```
Sub Test()
        On Error Resume Next
        Open "c:\nullfile.nul" For Input As #1
        Select Case Err.Number
        Case 53:
                MsgBox "Not file"
        Case 55:
                MsgBox "Not access"
        End Select
        Err.Clear
End Sub
```

Строчный обработчик можно отключить.

```
On Error Goto 0
```

Локальный обработчик специфичен для конкретной процедуры. То есть для каждой процедуры вы создаете свой обработчик ошибок. Общий вид такой:

```
On Error Goto ErrorHandle
```

```
        код
Exit Sub
ErrorHandle:
        Код обработки ошибки
End Sub
```

Обратите внимание на **Exit Sub**, который предназначен для выхода из процедуры. Если этого не сделать, то код предназначенный для обработки ошибок все равно выполнится, а это недопустимо. После обработки ошибки вы должны возобновить работу программы. Есть три способа:

- Повтор выполнения строки - **Resume**
- Выполнение следующей строки - **Resume Next**
- Закрытие формы - **Unload Me**

При отсутствии обработчика ошибок будет произведен поиск обработчика в вызывающей процедуре, если там его нет, то дальше по цепочке вызовов. Но при этом возобновление выполнения команд будет довольно сложным. Так как **Resume**, **Resume Next** будет работать в зависимости от того в какой процедуре оказалась ошибка.

```
Sub ErrorTest()
        On Error GoTo Error:
        Call Test
Error:
        Select Case Err.Number
        Case 53:
                MsgBox "Not file"
        Case 55:
                MsgBox "Not access"
        End Select
        Err.Clear
End Sub

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

Централизованная обработка ошибок необходима, если ошибки могут возникнуть в разных местах, а обрабатывать лучше в одном. Например, во многих местах программы производится создание файлов. Идея заключается в том, что создание файла всегда производится в одной процедуре где и проводится обработка ошибок.

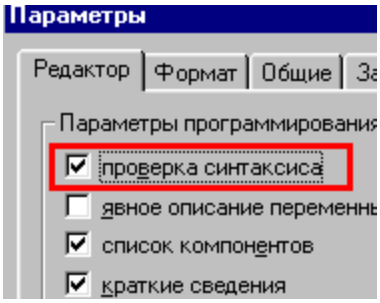
```
Sub ErrorTest()
        Call Test
End Sub

Sub Test()
        On Error GoTo Error:
        Open "c:\nullfile.nul" For Output As #1
        Close #1
        Exit Sub
Error:
        MsgBox "error"
        Err.Clear
End Sub
```

Так вот теперь процедуру **Test** можно вызывать из разных мест и всегда будет произведена одна и та же обработка ошибок. Таким образом удобнее поддерживать процедуры обработки, так как они централизованные.

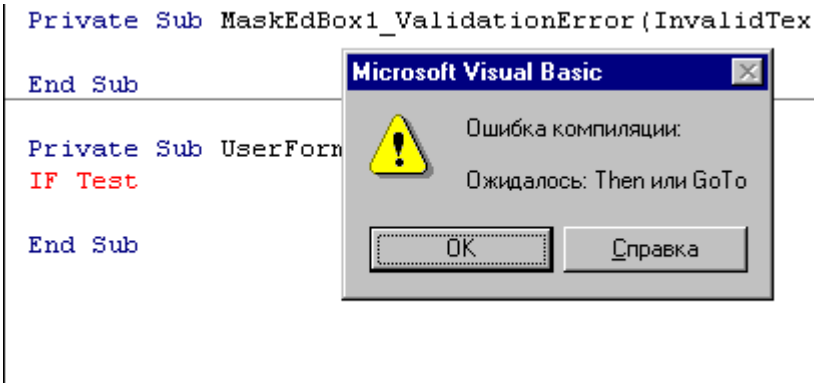
Меню

Шаг 64 - Функция автоматической проверки синтаксиса

Как в **VB**, так и в **VBA** автоматически проверяются синтаксические ошибки, но только в том случае, если проверка включена в **Сервис -> Параметры**.



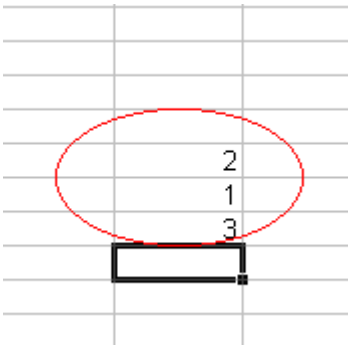
Если эта опция включена, то при проверке синтаксиса Вам будет выводиться сообщение о том, что Вы неправильно пишите. Например, если не завершить **IF**, то сразу после набора строки Вы получите сообщение.



Меню

Шаг 65 - Выделение диапазона выше текущей ячейки

Задача выделить диапазон выше текущей ячейки:

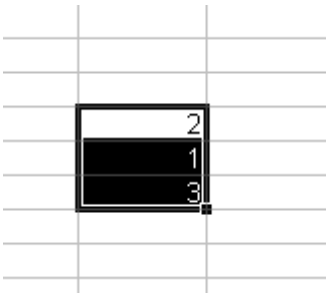


А вот и код с комментариями:

```
Sub SelectColumnData()
        ' что делать при ошибке
        On Error GoTo errors
        ' нижний адрес
        Dim a1 As String
        ' верхний адрес
        Dim a2 As String
        ' диапазон
```

```
        Dim ran As Range
        ' если не верхняя ячейка
        If (ActiveCell.Row <> 1) Then
                ' пойти вверх
                ActiveCell.Offset(-1, 0).Select
                ' взять адрес ячейки
                a1 = ActiveCell.Address
                ' будем подниматься
                For x = 1 To (ActiveCell.Row - 1)
                        ' на одну вверх
                        ActiveCell.Offset(-1, 0).Select
                        ' если не число выход
                        If IsNumeric(ActiveCell.Value) <> True Then
                                ' на одну вниз
                                ActiveCell.Offset(1, 0).Select
                                ' выход
                                GoTo nexts
                        End If
                        ' если пустая
                        If IsEmpty(ActiveCell.Value) = True Then
                                ' на одну вниз
                                ActiveCell.Offset(1, 0).Select
                                ' выход
                                GoTo nexts
                        End If
                Next x
        nexts:
                ' получаем адрес вырехней
                a2 = ActiveCell.Address
                ' строим диапазон
                Set ran = Range(a1 + ":" + a2)
                ' выделяем
                ran.Select
        End If
        ' выходим из процедуры
        Exit Sub
        ' ошибка, зовем на помощь
errors:
        MsgBox "Ошибка сообщите разработчику"
End Sub
```

А вот и результат:



Меню

Шаг 66 - Движение по диапазону

Выделив диапазон может возникнуть задача пройтись по этому диапазону с целью, например, покраски значений определенного диапазона. Смотрим:

```
Sub FullShach()
        For Each c In Range(addressdiap)
                If c.Value > yr1 Then
                        c.Select
```

```
                        With Selection.Interior
                                .ColorIndex = 6
                                .Pattern = xlSolid
                        End With
                        Selection.Font.ColorIndex = yrcolor1
                        If c.Value > yr2 Then
                                c.Select
                                Selection.Font.ColorIndex = yrcolor2
                                If c.Value > yr3 Then
                                        c.Select
                                        Selection.Font.ColorIndex = yrcolor3
                                End If
                        End If
                End If
        Next c
End Sub
```

Основа этой функции цикл **For Each c In Range(addressdiap)**, который будет перебирать ячейки в диапазоне и возвращать каждую ячейку в переменную цикла **c**. Для того, чтобы можно было произвести работу с этой ячейкой ее надо выделить **c.Select.**

Меню

Шаг 67 - Движение по ячейкам

Для движения по таблице используется функция.

```
переменная.Offset(RowOffset, ColumnOffset)
```

В качестве переменных может выступать как ячейка так и диапазон (**Range**) удобно пользоваться этой функцией для смещения относительно текущей ячейки.

Например, смещение ввниз на одну ячейку и выделение ее:

```
ActiveCell.Offset(1, 0).Select
```

Если нужно двигаться вверх, то нужно использовать отрицательное число:

```
ActiveCell.Offset(-1, 0).Select
```

Функция ниже использует эту возможность для того, чтобы пробежаться вниз до первой пустой ячейки.

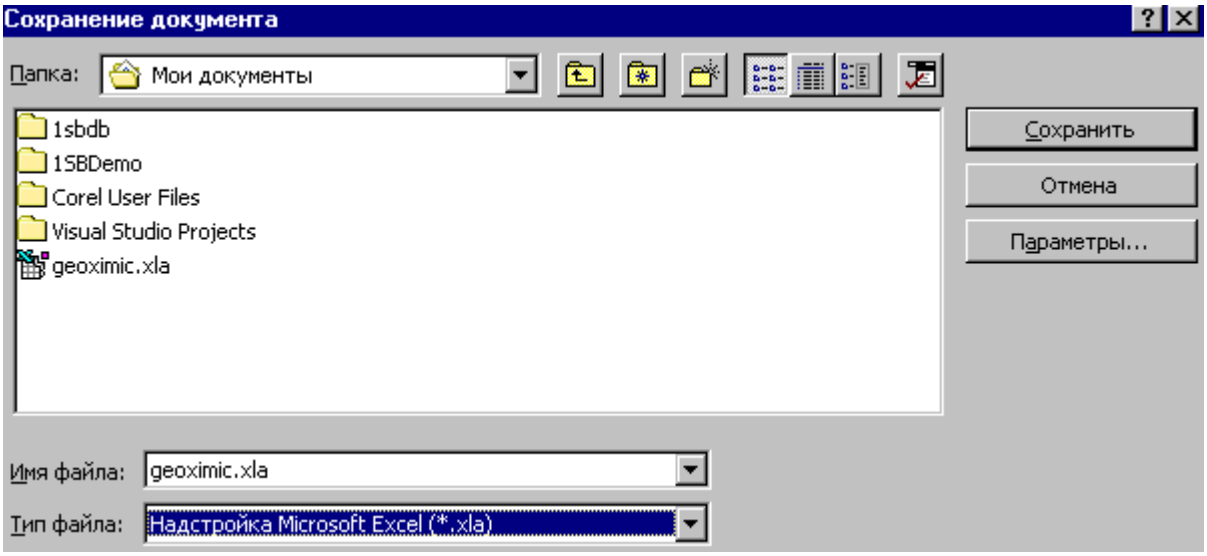
```
Sub beg()
        Dim a As Boolean
        Dim d As Double
        Dim c As Range
        a = True
        Set c = Range(ActiveCell.address)
        c.Select
        d = c.Value
        c.Value = d
        While (a = True)
                ActiveCell.Offset(1, 0).Select
                If (IsEmpty(ActiveCell.Value) = False) Then
                        Set c = Range(ActiveCell.address)
                        c.Select
                        d = c.Value
                        c.Value = d
                Else
                        a = False
```

```
                End If
        Wend
End Sub
```

Меню

Шаг 68 - Как сделать XLA ?

Дополнения к **Excel** делаются на основе обычной книги. Нужно создать книгу, потом написать макросы. После этого сохранить книгу как **XLA**.

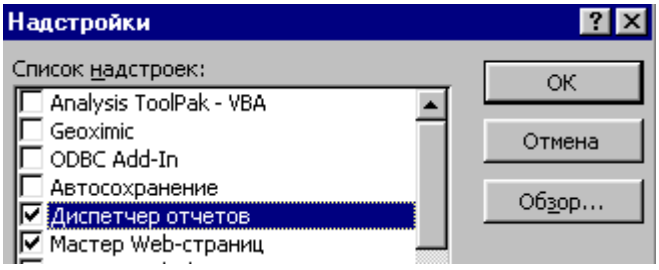


В результате у нас будет файл с расширением **XLA**.

В книге есть несколько событий, которые можно использовать для настройки меню или любых других действий при установке дополнения. Вот они:

```
Private Sub Workbook_AddinInstall()

End Sub

Private Sub Workbook_AddinUninstall()

End Sub
```

Они будут вызываться при установке или удалении дополнения через меню "сервис -> надстройки".



Меню

Шаг 69 - Динамическое создание меню

Смотрим код:

```
Const menuname = "Геохимия"

Private Sub Workbook_AddinInstall()
        ' обработка ошибки
        On Error GoTo errors:
         Dim num As Integer
        ' получаем количество пунктов меню
         num = Application.CommandBars("Worksheet Menu Bar").Controls.Count
        ' добавляем 1 для следующего
        num = num + 1
        Dim a As CommandBarControl
        ' создаем новый пункт меню
        Set a = Application.CommandBars("Worksheet Menu
Bar").Controls.Add(Type:=msoControlPopup, Before:=num)
        ' даем имя
        a.Caption = menuname
        Dim help As CommandBarControl
        Set help = Application.CommandBars("Настраиваемое всплывающее
меню1").Controls.Add(Type:=msoControlButton, Before:=1)
        help.Caption = "Помощь"
        help.OnAction = "Help"
        Dim comms As CommandBarControl
        Set comms = Application.CommandBars("Настраиваемое всплывающее
меню1").Controls.Add(Type:=msoControlButton, Before:=2)
        comms.Caption = "Расчет аномальных значений нормальный закон"
        comms.OnAction = "Anomal"
        Dim commslog As CommandBarControl
        Set commslog = Application.CommandBars("Настраиваемое всплывающее
меню1").Controls.Add(Type:=msoControlButton, Before:=3)
        commslog.Caption = "Расчет аномальных значений логнормальный закон"
        commslog.OnAction = "Anomallog"
        Dim commsbeg As CommandBarControl
        Set commsbeg = Application.CommandBars("Настраиваемое всплывающее
меню1").Controls.Add(Type:=msoControlButton, Before:=4)
        commsbeg.Caption = "Пробежаться по значения"
        commsbeg.OnAction = "beg"
        Exit Sub
        ' что будем делать при ошибке
errors:
        MsgBox Err.Description + " сообщите разработчику"
        Err.Clear
End Sub
```

Основа кода коллекция **CommandBars**, которая отвечает коллекции меню. У этой коллекции есть метод **Add**, после которого меню надо присвоить название и имя макроса.

```
help.Caption = "Помощь"
help.OnAction = "Help"
```

Удаляется меню по имени. Вот код:

```
Private Sub Workbook_AddinUninstall()
        On Error GoTo errors:
        Application.CommandBars("Worksheet Menu Bar").Controls(menuname).Delete
        Exit Sub
        ' что будем делать при ошибке
        MsgBox "не могу удалить пункт меню"
errors:
End Sub
```

Если Вы обратите внимание, то меню устанавливается при подключении расширения **Excel**, а удаляются при его отключении.

Меню

Шаг 70 - Нефть, таблицы и как делать не надо

Вот такое письмо.

```
Добрый день Артем.
Дело в том что с этими данными также нужно производить
другие расчеты и представлять их геологам, в Access'е по
моему это сделать будет немного проблематично да и неудобно
для передачи. так что лучше наверное все же в Excel'е. Ну ты
понял в чем именно суть проблемы да? еще раз повторюсь чтобы
уж не было непоняток, а то может объяснил я не так. имеется таблица.
1 столбец имя скважины, второй насыщение пропластка.
Выглядит это следующим образом:
1210k   Нефть
1210k   Нефть
1210k   НВ
1210k   Неясно
1231    Вода
1231    Вода
1231    Вода
1231    Вода
По скважине 1210к есть насыщение нефть, нв и неясно. соответственно тип
скважины нефтенасыщенный 1231 только вода значит водо-насыщенная и т.д.
в таком же духе. Вот.
С уважением Рустам Сафиуллин
mailto: rustam@geodata.ru
```

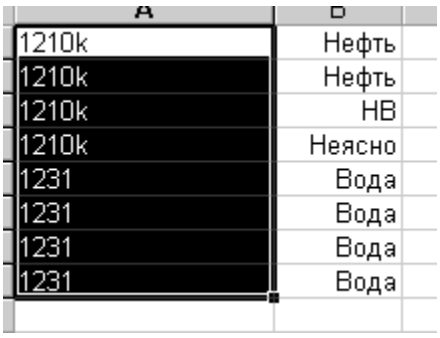
Первая мысль которая родилась у меня после этого письма послать все к черту вместе с автором. Это классическая задача баз данных. Любые расчеты и все такое можно решить с помощью того же **ACCESS** и намного проще. Кроме того большая часть кода ниже будет просто реализация стандартного **SQL** запроса. Кроме того, код подвержен ошибкам в данных ведь Нефт и Нефть не одно и тоже. Но мой опыт работы с геологами показывает что объяснять им что то бесполезно. Можно и на **EXCEL** сделать. Можно неправильно но можно. Итак таблица выгляди вот так.

	A	B
1	1210k	Нефть
2	1210k	Нефть
3	1210k	НВ
4	1210k	Неясно
5	1231	Вода
6	1231	Вода
7	1231	Вода
8	1231	Вода

Сначала нужно определить где начинаются и заканчиваться данные, это написано в "Шаг 45 - Начало и конец данных".

```
Dim allbore As Range ' здесь будет храниться диапазон скважин
' выбрать колонку
Set allbore = Range("A:A")
' только с данными
Set allbore = Range(allbore.Columns.End(xlUp).Address,
allbore.Columns.End(xlDown).Address)
' выделить
allbore.Select
```

А вот результат:



A	B
1210k	Нефть
1210k	Нефть
1210k	НВ
1210k	Неясно
1231	Вода
1231	Вода
1231	Вода
1231	Вода

Теперь нам нужно создать список скважин которые есть, способ один перебрать все записи и уникальные поместить у коллекцию. Как двигаться по диапазону написано в шаге "Шаг 66 - Движение по диапазону".

```
Dim borename As New Collection ' это набор скважин

Sub FindOil()
        Dim allbore As Range ' здесь будет храниться диапазон скважин
        Set allbore = Range("A:A") ' выбрать колонку
        ' только с данными
        Set allbore = Range(allbore.Columns.End(xlUp).Address,
allbore.Columns.End(xlDown).Address)
        allbore.Select                  ' выделить
        For Each bore In allbore        ' бежим по диапазону скважин
            bore.Select                 ' выделяем ячейку
            borename.Add (bore.Value)    ' добавить к коллекцию
        Next bore
End Sub
```

Вот теперь у нас в коллекции все имена скважин. Но они повторяться же. Надо при добавлении проверять есть такое имя в коллекции или нет. Напишем функцию.

```
Function FindElement(name As String) As Boolean
        ' бежим по коллекции
        For Each elem In borename
                ' если имя совпадает вернуть FALSE
                If elem = name Then
                        FindElement = False
                        Exit Function
                End If
        Next elem
        ' нет имени
        FindElement = True
End Function
```

И применим ее:

```
Dim allbore As Range ' здесь будет храниться диапазон скважин

Sub FindOil()
        Set allbore = Range("A:A") ' выбрать колонку
        ' только с данными
        Set allbore = Range(allbore.Columns.End(xlUp).Address,
allbore.Columns.End(xlDown).Address)
        allbore.Select ' выделить
        For Each bore In allbore ' бежим по диапазону скважин
            bore.Select  ' выделяем ячейку
            If FindElement(bore.Value) = True Then
                ' если скважины нет в коллекции
                        borename.Add (bore.Value)    ' добавить к коллекцию
                End If
```

```
        Next bore
End Sub
```

Меню

Шаг 71 - Нефть, таблицы и как делать не надо, продолжение

Итак, теперь нам нужно в соответствии с номером скважины делать выборку. Заведен еще один массив.

```
Dim allbore As Range           ' здесь будет храниться диапазон скважин
Dim alldata As Collection      ' это набор элементов
Dim borename As New Collection         ' это набор скважи
```

Напишем функцию которая будет заполнять коллекцию по именам скважины:

```
Sub SelectBore(s As String)
        Set alldata = New Collection
        For Each bore In allbore ' бежим по диапазону скважин
                bore.Select  ' выделяем ячейку
                If bore.Value = s Then  ' если это та скважина
                        ActiveCell.Offset(0, 1).Select ' вправо
                        alldata.Add (ActiveCell.Value) ' поместить в коллекцию
                End If
        Next bore
End Sub
```

Пробежим по всем скважинам и возмем значение из правой колонки поместив его а массив.

```
Sub FindOil()
        Set allbore = Range("A:A") ' выбрать колонку
        ' только с данными
        Set allbore = Range(allbore.Columns.End(xlUp).Address,
allbore.Columns.End(xlDown).Address)
        allbore.Select ' выделить
        For Each bore In allbore ' бежим по диапазону скважин
                bore.Select  ' выделяем ячейку
                If FindElement(bore.Value) = True Then ' если кважины нет в коллекции
                        borename.Add (bore.Value)   ' добавить к коллекцию
                End If
        Next bore
        For Each elem In borename
                SelectBore (elem)
                Debug.Print (elem)
                For Each data In alldata       ' пробежим по результату для того что бы
                                        ' показать что массив заполнен
                        Debug.Print (data)
                Next data
        Next elem
End Sub
```

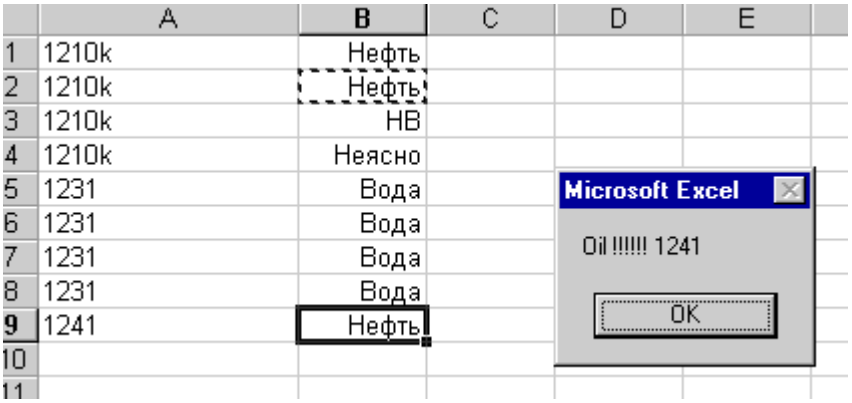
В окне отладки можно увидеть, что массив заполнен значениями соответствующими скважинам на данный момент.

```
Проверка
 1210k
     Нефть
     Нефть
     НВ
    Неясно
 1231
     Вода
     Вода
     Вода
     Вода
```

Ну, а теперь можно исследовать этот массив на пример решения какая скважина. Вообщем то на самом деле я просто повторил работу БД. Сделал выборку. Давайте попробуем сделать простой выводы. Например если упоминается нефть но она нефтенасыщенная иначе пусто.

```
Sub FindOil()

.....................

        For Each elem In borename
                SelectBore (elem)
                Debug.Print (elem)
                For Each data In alldata       ' пробежим по результату для того что бы
                                               ' показать что массив заполнен
                        'Debug.Print (data)
                        If data = "Нефть" Then
                                MsgBox "Oil !!!!!! " + elem
                                Exit For
                        End If
                Next data
        Next elem
End Sub
```

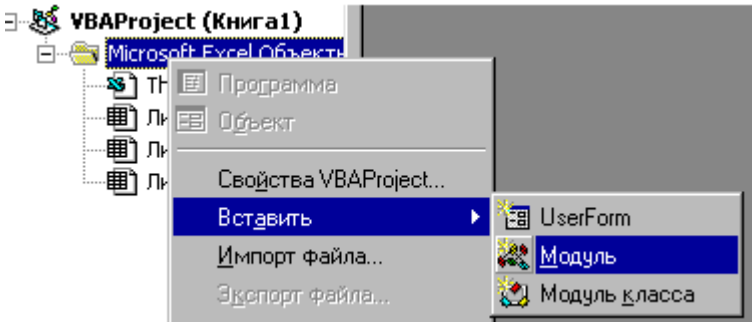
При работе этой функции будет два окна для скважины с нефтью.



Меню

Шаг 72 - Как создать свою функцию

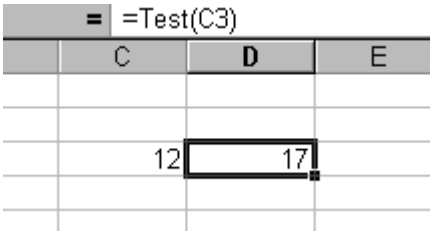
Для того, чтобы создать и использовать свою функцию в **Excel** точно так же как стандартные функции Вам нужно перейти в редактор **VBA** читайте "Шаг 1 - Первый макрос". И добавить модуль.



Потом в модуле Вы создаете функцию:

```
Function Test(i As Integer) As Integer
        Test = i + 5
End Function
```

Все можно переходить на таблицу и использовать:



Функция как родная.

Меню

Шаг 73 - Выделенный диапазон Выше ячейки, второй метод

Добрый день!

Мне очень понравился Ваш сайт. Большое Вам за него спасибо! Особенно привлекательным его делает то, что различные среды программирования Вы рассматриваете не только по отдельности, но также разбираете связь между ними, например, написание **DLL** для Excel в **VC++**.

Однако, некоторые примеры вызывают возражения. Вот, скажем, задача выделить диапазон выше текущей ячейки (см. "Шаг 65 - Выделение диапазона выше текущей ячейки"). По-моему, слишком длинное решение такой простой задачи. Я бы это сделал так, воспользовавшись методом **Union**:

```
Sub SelectColumnData()
        If ActiveCell.Row = 1 Then Exit Sub
        If ActiveCell.Offset(-1, 0) <> "" Then
                ActiveCell.Offset(-1,0).Activate
        If ActiveCell.Row = 1 Then Exit Sub
        Do While Not ActiveCell.Offset(-1, 0) = ""
                Union(Selection, ActiveCell.Offset(-1, 0)).Select
                If ActiveCell.Row = 1 Then Exit Do
        Loop
End Sub
```

Задача очень простая, не правда ли, и решается четырьмя строками кода (за исключением трех **if**, которые отсекают выход за пределы листа). Переменные совсем не нужны, как это очень часто бывает в **VBA**.