Aula 3 – Funções para tratamento de texto, e função para obtenção de dados em arquivos texto com formato livre (e-mails, relatórios, pdfs salvos como texto, etc.). Função para pesquisa fonética.

De novo, vamos começar pelas funções mais básicas, testá-las e depois usá-las em funções mais complexas.

1 – Função para contar as ocorrências de uma dada palavra em um texto.

```
Function ContaPalavra(textoprocurado, NoTexto)1
  If textoprocurado = "" Then
     ContaPalavra = 0
     Exit Function
  End If
  i = InStr(1, NoTexto, textoprocurado, vbTextCompare)
  If i = 0 Then
     ContaPalavra = 0
  Else
    qtd = 0
    Do
       qtd = qtd + 1
       i = InStr(i + 1, NoTexto, textoprocurado, vbTextCompare)
    Loop Until i = 0 Or i > Len(NoTexto)
    ContaPalavra = qtd
  End If
End Function
```

A função aqui apresentada recebe dois strings e devolve o número de ocorrências do primeiro no segundo. O casamento é feito independentemente de maiúsculas ou minúsculas. A grande novidade aqui é a instrução exit function, que é utilizada para encerrar prematuramente a execução do código da função.

Uma outra novidade é a construção "do – loop until condição".
Vamos aproveitar para falar sobre todas as possibilidades do par "do – loop".
Quando queremos que o teste seja feito antes da execução do bloco interno ao comando, colocamos a condição na linha do “do”. Desse modo, existe a possibilidade de o conteúdo não ser executado nenhuma vez; Por outro lado, podemos colocar esta condição na linha do “loop”, e com isso o bloco interno sempre será executado pelo menos uma vez.
A condição pode ser colocada após a palavra “until” (até) ou após a palavra “while” (enquanto). O programador escolhe a palavra que achar mais adequada para tornar o código inteligível. De qualquer forma, existe uma equivalência 100% perfeita entre:
While <condição> e Until not <condição>.
O laço desse programa terminará quando o texto procurado não existir mais no texto total, ou quando a posição de início de pesquisa ultrapassar o tamanho do texto total.

Esta função será brevemente melhorada para permitir que além da independência de caixa (alta ou baixa) haja também uma independência de acentuação (Ângela = ANGELA, e Márcia = marcia). Na realidade faremos até mais: a grafia também não vai importar e exame=ezame=esame.
Para transformar esta rotina em uma pesquisa fonética rudimentar, vamos criar a função “som” conforme código a seguir:

```
Function som(Texto)
   Dim i, orig, dest, trab, resp, ultimo, este
   Static conv(255)
   If conv(1) <> " " Then   ' só faz a primeira vez
      orig = "AÁÀÃÂEÉÈÊ&IÍÌÎOÓÒÕÔUÚÙÛÜBCÇDFGHJKLMNPQRSTUVWXÿYZÑ0123456789,."
      dest = "AAAAAIIIIEIIIIUUUUUUUUUB@@DF#H#@LMNPQR@TUUU@II@N0123456789,."
      For i = 1 To 255
          conv(i) = " "
      Next
      For i = 1 To Len(orig)
          conv(Asc(Mid(orig, i, 1))) = Mid(dest, i, 1)
      Next
   End If
   trab = UCase(Texto) & " "
   resp = ""
   For i = 1 To Len(trab) - 1
      este = conv(Asc(Mid(trab, i, 1)))
      If (este <> " " And ultimo <> este) Or IsNumeric(este) Then
         ultimo = este
         resp = resp & ultimo
      End If
   Next
   som = resp
End Function
```

Esta função nos apresenta uma variável com duas características ainda desconhecidas:
Static – Usamos este comando em vez de Dim, para definir variáveis com a seguinte característica: O seu valor é mantido entre as chamadas à função. É o que ocorre com as variáveis globais, só que neste caso elas só são acessíveis no interior da função (ou sub-rotina)
A segunda característica é que a variável contém muitos valores, que são identificados através do uso de índices.
Esse código não será explicado aqui, mas o que ocorre é o seguinte:
1 – As letras existentes na seqüência "orig" serão substituídas pelas correspondentes na seqüência "dest". Por isso, K, C, Ç, S, X e Z são substituídos pelo mesmo símbolo (@). Por isso exame casa com exame (e casaria também com eçame, só que ninguém vai escrever assim).
2 – As letras dobradas e os espaços são eliminados. Por isso , Collor casará com Color.

Lembre que esta rotina é bem rudimentar. Alguns casos como Christina e Cristina não são previstos aqui, por motivo de simplicidade. O fundamental é que os acentos não importam para a igualdade de textos.

Com essas duas funções anteriores, pode-se facilmente fazer uma terceira, que conta quantas vezes uma palavra existe em um texto, independentemente de sua grafia, ou do uso de acentos.

A função é a seguinte:

```
Function ContaSom(textoprocurado, NoTexto)
  ContaSom = ContaPalavra(som(textoprocurado), som(NoTexto))
End Function
```

Repare que os dois textos são "fonetizados" e depois se usa a função ContaPalavra para ver o número de ocorrências de um no outro.

O Excel tem uma função que troca em um texto uma dada seqüência de caracteres por outra.
=substituir(texto total; pedaço original; novo texto)
Vamos fazer uma extensão aqui: A função **trocar** funcionará no mínimo como a substituir, mas pode ter quantos pares de texto antigo/texto novo queiramos, ou seja, o número de operandos é variável.

```
Function trocar(Texto, ParamArray lista())
    x = Texto
    For i = 0 To UBound(lista) Step 2
        x = Replace(x, lista(i), lista(i + 1))
    Next
    trocar = x
End Function
```

No cabeçalho da função estamos vendo como criar um número variável de operandos, através do uso da palavra reservada ParamArray seguida do nome de um vetor. Como o número de elementos é variável, não precisamos dizer quantos elementos a lista terá, basta colocar o par de parêntesis para indicar que a lista é um vetor.
O primeiro índice de lista() é 0 (zero) e o maior índice é obtido pela função Ubound(lista)
Ubound = Upper Bound, ou limite superior.
Como eles vêm aos pares, o "for" vai incrementar o índice do parâmetro de 2 em 2.

A função replace do VB funciona como a função "substituir" do Excel.
A nossa função trocar apenas estende a replace, para permitir vários pares de parâmetros, em vez de apenas um. Ela apenas vai facilitar a codificação do programa, quando for necessário fazer mais de uma troca num string (por exemplo trocar todos os símbolos de pontuação por espaços)

Agora vamos fazer outra função útil, que existe em algumas linguagens: é a função AllTrim, que difere da função Trim (Arrumar no Excel) que só tira os espaços iniciais e finais de um texto. O Alltrim além disso substitui brancos repetidos por apenas 1 branco. Assim, "A B" vai se transformar em "A B".
O código da função é o seguinte:

```
Function AllTrim(x)
 z = Trim(x)
 Do While InStr(1, z, "  ") > 0  ' loop enquanto tem brancos dobrados
   z = Replace(z, "  ", " ")
 Loop AllTrim = z
End Function
```

A lógica é a seguinte: Enquanto houver 2 brancos seguidos, troca todos eles por apenas 1. O loop é porque se existirem 20 brancos seguidos no meio do string, após o primeiro replace ficam 10, depois 5, depois 3, depois 2 e finalmente 1.

Agora vamos usar as funções AllTrim e Trocar para implementar uma função que, dado um texto e um número n, devolve a n-ésima palavra deste texto.
Para isso, devemos fazer os seguintes passos:

Substituir todos os símbolos de pontuação . , : ; ( ) por espaços (pode-se aumentar esta lista colocando outros símbolos, como = - + etc)
Depois disso, usa-se o AllTrim para obter um texto apenas com palavras e um único branco separando cada uma delas.
Agora usaremos uma função do VB chamada split, que recebe um texto mais um delimitador e coloca todos os trechos delimitados por esse delimitador em um vetor, a partir do elemento (0). Para saber qual é o último elemento criado, usa-se a função Ubound que devolve o índice do maior elemento de um vetor.
Para evitar erros, se for pedida uma palavra além do número de palavras, a função devolverá a última.
Deve-se observar que nós contamos as palavras a partir de 1 e no vetor elas estão a partir do índice 0.
O código da função é:

```
Function Palavra(Frase, ordem)
   novafrase = trocar(Frase, ".", " ", ",", " ", "(", " ", ")", " ", ":", _
                      " ", ";", " ")
   novafrase = AllTrim(novafrase)
   palavras = Split(novafrase, " ")
   If ordem > UBound(palavras) + 1 Then
      Palavra = palavras(UBound(palavras))
   Else
      Palavra = palavras(ordem - 1)
   End If
End Function
```

Agora vamos fazer uma função para obter um valor em um string, sem sabermos em que posição ele se encontra. Apenas sabemos que, se esse valor existir, ele será o primeiro número após um determinado string, que nós conhecemos.
Essa função pode ser considerada "esperta", pois ela responde à seguinte pergunta:
Qual é o salário de Rui?
Responda a partir da leitura do seguinte texto: " ... o professor Rui Medeiros vai ganhar este mês a quantia de R$ 5.123,45 brutos ... " que está registrado na variável Carta01.
A chamada será: salario = ValorApósTexto("Rui", Carta01)
Lendo o texto, você vê claramente que o salário é de 5123,45.
Pois bem, vamos fazer uma rotina que descobre isso.

```
Function ValorAposTexto(textoprocurado, NoTexto)
     arg = UCase(Trim(textoprocurado))
     i = InStr(1, UCase(NoTexto), arg)
     If i = 0 Then
        ValorAposTexto = ""
        Exit Function
     End If
     For i = i + Len(arg) To Len(NoTexto)
         If IsNumeric(Mid(NoTexto, i, 1)) Then
            Exit For
         End If
     Next
     If i > Len(NoTexto) Then
        ValorAposTexto = ""
        Exit Function
     End If
     inic = i
     valor = Mid(NoTexto, i, 1)
     Do
        i = i + 1
        valorok = valor
        valor = valorok & Mid(NoTexto & "*", i, 1)
     Loop While IsNumeric(valor)
     ValorAposTexto = CDbl(valorok)
End Function
```

Analise o código e observe:
Logo na primeira linha temos 2 chamadas à funções embutidas: Tiram-se os espaços antes e depois do texto procurado e transforma-se ele em maiúsculas.
Depois disso, acha-se aonde (se houver) está esta palavra no texto. Se não estiver, devolve-se um nulo (string vazio). Se estiver pula-se o texto achado e procura-se o primeiro dígito no texto a partir daí. Quando acharmos, vamos adicionando caracteres nesse valor até o resultado não ser numérico.Para tanto, usamos a função mid que é semelhante à função ext.texto do Excel.
Imagine a expressão: mid("abcdefg",3,2). Ela devolve "cd" , porque se pegam a partir da posição 3, 2 caracteres.
Na função, se não houver nenhum valor numérico após o texto, devolve-se nulo. Se houver, o texto é convertido para Double. Esta função Cdbl transforma 2.345,67 em 2345.67

Agora que já temos um conjunto de ferramentas básicas, vamos fazer uma função bem especial: Ela procura valores após nomes, não em textos, mas em arquivos!
Como um mesmo arquivo pode conter muitos valores que vamos obter, devemos pensar em uma maneira de não ler o arquivo todo a cada chamada da função, porque senão o tempo de execução da planilha pode ficar tão grande que torna inviável o seu uso.
A estratégia é a seguinte: Usamos uma matriz global (para não perder os dados entre chamadas da função) que vai guardar o conteúdo de cada arquivo referenciado. Desse modo, cada arquivo será lido apenas uma vez a cada execução da planilha.
Além disso, vamos criar mais uma esperteza: vamos procurar o valor após um fonema. Assim, se rui estiver grafado como Ruy ou Márcia for escrito sem acento, o valor será achado.
Na implementação, a maior parte do código é dedicada à tarefa de procurar o conteúdo no arquivo na matriz global e, se não achar, ler o arquivo colocando o seu conteúdo na matriz.
Vamos ao fonte. A variável global deve estar definida no início do módulo. Mostramos aqui imediatamente antes da função por motivos óbvios.

```
Dim DadosExternos(2, 100) As String ' nome + conteúdo do arquivo
Function PegaValorNoArquivo(textoprocurado, filename)
    i = 1
    arq = UCase(filename)
    Do While DadosExternos(1, i) <> "" And arq <> DadosExternos(1, i)
        i = i + 1
    Loop
    If DadosExternos(1, i) = "" Then   ' ainda não leu o arquivo
       DadosExternos(1, i) = arq
       On Error Resume Next
       Open ActiveWorkbook.Path & "\" & arq For Input As #1
       If Err.Number <> 0 Then
            DadosExternos(2, i) = ""
       Else
            On Error GoTo 0         ' para de mascarar erros de programação
            conteudo = ""
            While Not EOF(1)
               Line Input #1, linha
               conteudo = conteudo & " " & linha
            Wend
            DadosExternos(2, i) = som(conteudo)
       End If
       Close #1
    End If
 '  aqui estamos com os dados lidos
    PegaValorNoArquivo = ValorAposTexto(som(textoprocurado), _
                         DadosExternos(2, i))
End Function
```

Analise das novidades apresentadas na função:
Repare que ao definir a matriz dissemos que o tipo de dados é String e não Variant. Isso porque as variáveis Strings são inicializadas com Strings vazios ("").
Ao fazer o loop de pesquisa, procurando se o arquivo já foi lido, pararemos quando achar o nome do arquivo ou quando achar um elemento vazio.
Nesse último caso, vamos abrir, ler todo o arquivo, fechar e depois instalar no vetor.
Definimos que os arquivos estarão no mesmo diretório que o arquivo Excel:
ActiveWorkbook.Path
Para ler um arquivo texto, devemos conhecer 2 coisas:
O comando de leitura é Line Input #<n>, varstring (será lida uma linha do arquivo texto)
Antes de ler, para evitar um erro de programa, testamos a existência de linhas a serem lidas com a função EOF(<n>). EOF é a sigla de End Of File, ou Fim De Arquivo.
O <n> é o número do arquivo dado na instrução open. É usado tanto no line Input quanto no Eof() quanto no Print (visto 2 aulas atrás).

Por motivo de performance, só traduzimos uma vez o texto para a representação do som. Todas as vezes que formos procurar no texto do arquivo basta converter o texto procurado para um som. Isso é importante, porque a função som é bem complexa, e não deve ser utilizada muitas vezes, principalmente se o texto a converter for longo.

Para utilizar essas funções, use a planilha fornecida com a aula. O código VB já está digitado lá, sem erros. Por isso, ignore o que você fez e teste na planilha pronta. Os arquivos de dados para a pesquisa dos valores estão no site, e devem ser colocados no mesmo diretório da planilha. Crie outros arquivos, cometa erros de acentuação e teste bastante as funções. Achando erros no algoritmo em função dos testes de validação, tente corrigir. Um programa sempre pode ter erros, mas deve-se testar bem antes de usar no dia a dia, para minimizá-los.