Positional systems converter

Khashimov Akmalkhon

1. Description

In this project, I created a positional systems converter that can convert from any integer base from -36 to -2 and 2 to 36, Fibbonaci base, and 2's complement.

Usage: first, you need to enter the number into the input box at the top, then choose the base you are converting from, then base you are converting to, and toggle the "Normalize" button if you want to normalize the result.

If you want to save the results into the spreadsheet, use the "Save results" button: it will automatically create a new spreadsheet and correspondingly save the input, parameters, and results.

2. Methodology

In this section I will explain the main takeaways parts of my code.

The converter requires several helpful functions for its work: trimming zeros, fulfilling zeros, checking the input for digit correspondence with the picked base from which we convert.

Trimming zeros (Trim0s)

The function takes a string variable as input, and checks whether in the left and right part of it there are zeros using the functions Right and Left in combination, to access specific letter in the string, as we see in the following code:

```
Do While (Right(Left(temp_val, 2), 1) = "0" And Left(temp_val, 2) <> "0.")
        temp_val = Right(temp_val, Len(temp_val) - 1)
Loop
```

Thus, it removes the zeros and returns the result.

Fulfilling zeros (Fullfil0s)

This function takes a string variable and a byte variable as input and checks if the string length complies with the length sent as byte variable input, which we can see in:

```
Do While Len(init_val) < mylen
        init_val = "0" & init_val
Loop
```

If the length is insufficient long, then the string variable is concatenated with string "0" using & operator.

Checking digits (CheckDigits)

This function takes a string variable and a byte variable as input. It checks every letter in the string provided and converts the letter into the number if it is possible, as we see it in the following code:

```
If (digit > 64 And digit < 91) Then
digit = digit - 55
ElseIf (digit > 47 And digit < 91) Then
digit = digit - 48
ElseIf (ctr = Len(init_val) And digit = 45) Then
digit = 65 - 55
Else
CheckDigits = -1
Exit Function
End If
```

As we take the ASCII number of the letter, then to get the digit of the letter we need to subtract 55, and 48 in case if the letter is provided in more than base 10.

It is also obvious that I used the following below code to access each letter specifically in the string:

```
digit = Asc(Left(Right(init_val, ctr), 1))
```

The function uses Exit Function method to stop the function if it found incorrespondence, as we see here:

```
If (digit >= from_base) Then
CheckDigits = 0
Exit Function
```

The function returns **-1** for an unknown letter found in the string input, **0** if the digit provided in the input is greater than the maximum digit used for the base converted from, **1** if the letter is compliant with the base.

Convert10

In its core, the converter takes the base it converts from and transforms it into base 10, next the converted value with base 10 is converted to the base the user requires.

This function is used to convert from any base from 2 to 36 to any base from 2 to 36.

Takes variant(string) variable, and two byte variables as input, from and to bases.

It does the following: it creates a decimal (base 10) integer variable; checks each digit with the method used in the *CheckDigits* function, from right to left; and continuously sums up each digit multiplied by the from base at the power of the position in the string starting from 0 from left to right, as we see in the following code in the loop:

```
decim = decim + digit * (from_base ^ (ctr - 1))
ctr = ctr + 1
```

Thus, converting from the given base to base 10. If the base we want is already 10, then return the result. If the input was initially negative, negate the result.

```
If (to_base = 10) Then
   If (neg = 1) Then
      Convert10 = "-" & decim
   Else
      Convert10 = decim
   End If
```

In any other case, we transform the base 10 to the base given to the converter, using the remainder method, as we see in the code below:

```
Do While decim <> 0
   digit = decim Mod to_base
   'REMAINDER IS A DIGIT
   If (digit < 9) Then
         result = digit & result
   Else
      result = Chr(digit + 55) & result
   End If
   decim = decim \ to_base
   ctr = ctr + 1
Loop
```

And the negation:

```
If (neg = 1) Then
   Convert10 = "-" & result
Else
   Convert10 = result
End If
```

Finally, returning the result as a string.

Complement2

Takes variant(string) variable, and an optional byte variable.

This function converts to and from 2’s complement to/from base 10.

The byte variable ensures what we actually need: from 2’s complement – 1; otherwise - 0.

If we want to convert from 2’s complement, then we check if the string provided starts with 1. In positive case, we reverse each of the bits given in the input, as we see in the following code:

```
Do While index <= Len(init_val)
        tempstr = Left(Right(init_val, index), 1)
        If (tempstr = "0") Then
           inverted = "1" & inverted
        ElseIf (tempstr = "1") Then
           inverted = "0" & inverted
        Else
           MsgBox ("DIGIT/BASE INCORRESPONDENCE")
           Exit Function
        End If
        index = index + 1
    Loop
```

And using function *Convert10*, we convert from base 2 to base 10.

In case if our string does not start with 1, it means that we want to convert from base 2 to 2's complement.

We simply check the length of the provided string, as in the following code :

```
If (Left(init_val & "", 2) = "-0" And Len(init_val) <= 9) Then


    ElseIf (Left(init_val & "", 2) = "-1" And Len(init_val) <= 8) Then


    ElseIf (Left(init_val & "", 1) = "0" And Len(init_val) <= 8) Then


    Else
       MsgBox ("Range of values converting to 2's complement: -127(10-base) to 127(10-base)
")
       Exit Function
    End If
```

Next, we invert each of the bits provided in the string input, just as we did above.

And add 1 to the result we got:

```
       Do While (Left(Right(inverted, index), 1)) = 1
          inverted = Left(inverted, Len(inverted) - index) & "0" & Right(inverted, index - 1)
          index = index + 1
       Loop
```

We use the function *Fullfil0s* to make sure that the result we return is compliant with 8-bit.

And finally, we return the result.

Fibbo

Converts to/from base 10 from/to Fibbonaci base.

Takes variant(string) variable, and optional byte variable, as an indicator if we want to convert to base 10.

I used a LONG variable array of 3 elements, which start from 1, 2, and 3.

First it checks if the input is negative, going by the minus sign on the left.

```
If (Left(init_val, 1) = "-") Then
    negative = 1
    init_val = Right(init_val, Len(init_val) - 1)
End If
```

Next, it checks if we want to convert to base 10, in other words, from fibbonaci base.

In case of from Fibbonaci base, we calculate each fibbonaci number using the following strategy: sum 2 previous fibbonaci numbers, each time calculating which element we are at, so that the element we are at must be the sum of the other 2 elements:

```
        If (index Mod 3 = 1) Then
            a(1) = a(2) + a(3)
        ElseIf (index Mod 3 = 2) Then
            a(2) = a(1) + a(3)
        Else
            a(3) = a(2) + a(1)
        End If
        index = index + 1
```

Thus, summing up all the fibbonaci numbers shown as 1 in the string provided and returning the result.

In the other case, when we want to convert to fibbonaci base, from base 10, we continuously find the biggest fibbonaci number smaller than the number we have as input, continuously putting 1 at the position of the number we found, and 0s until we find the next number, continuously subtracting all the numbers from the initial number, which becomes less and less , and finally 0. We clearly see this strategy in the following code:

```
Do While (temp > 0)
        a(1) = 1
        a(2) = 2
        a(3) = 3
        index = 1
        'Find the biggest Fibo number under the temp every time
        Do While (a((index Mod 3) + 1) <= temp)
            If (index Mod 3 = 1) Then
```

```
            a(1) = a(2) + a(3)
        ElseIf (index Mod 3 = 2) Then
            a(2) = a(1) + a(3)
        Else
            a(3) = a(2) + a(1)
        End If
        index = index + 1
    Loop
    'Until you have all the numbers that sum up to the decimal value
    temp = temp - a(((index + 2) Mod 3) + 1)
    'Get all the indexes into 1
    If (result = "") Then
        result = "1"
    Else
        result = Left(result, Len(result) - index) & "1"
    End If
    'and fill the remaining with 0s
    Do While index <> 1
        result = result & "0"
        index = index - 1
    Loop
Loop
```

If the input was initially negative, put a minus sign in the most left part using concatenation operator &.

```
If (negative = 1) Then
    Fibbo = "-" & result
Else
    Fibbo = result
End If
```

Submit button

The converter, when you press the submit button, first checks if the base it converts from is the same as the base it converts to: in this case it checks the base correspondence of all the digits and prints out the result.

```
If (from_box.Value = to_box.Value) Then
   'Check all the digits if they're corresponding

   If (CheckDigits(InputBox.Value, from_base) = 1) Then
   result_box.Value = InputBox.Value
   Else
   result_box.Value = "digit/base incorrespondance !"
   End If
```

Otherwise, the corresponding from and to base converter is called, divided into 4 parts:

1) from/to Fibbonaci to/from 2's complement
2) to/from Fibbonaci from/to integer
3) to/from 2's complement from/to integer
4) to/from integer from/to integer

Save button

Save button creates a new sheet, after that it uses system variable ActiveCell to address the active cell in the sheet and write the data, continuously offsetting its position and selecting the new one as active cell as we can see here:

```
ActiveCell.Value = InputBox.Value

ActiveCell.Offset(0, 1).Select

ActiveCell.Value = from_box.Value

ActiveCell.Offset(0, 1).Select

ActiveCell.Value = to_box.Value

ActiveCell.Offset(0, 1).Select

ActiveCell.Value = result_box.Value

ActiveCell.Offset(1, -3).Select
```

Normalize button

Togglable button, which converts the result from the base convertion to a normalized.

First it checks the position of the dot in the input using the InStr system function:

```
'Position of the dot
Dim pos As Integer
pos = InStr(temp_str, ".")
```

Next, it checks if the number is negative.

The code makes sure that the position of the dot is always less than 3. To make it, it just moves the dot to the left or to the right, accordingly, decreasing or increasing the exponent of the base the initial result it converted to.

```
If ((pos > 2 And negative = 0)) Then
    temp_str = Left(temp_str, pos - 1) & Mid(temp_str, pos + 1)
    'Turn Pos into Exponent
    pos = pos - 2
    temp_str = Left(temp_str, 1) & "." & Mid(temp_str, 2)
ElseIf (pos > 3 And negative = 1) Then
    pos = pos - 3
    temp_str = Left(temp_str, 2) & "." & Mid(temp_str, 3)
    'Pos = exponent now
ElseIf (pos = 0) Then
    If ((Len(temp_str) > 1 And negative = 0)) Then
        temp_str = Left(temp_str, 1) & "." & Mid(temp_str, 2)
    ElseIf ((Len(temp_str) > 2 And negative = 1)) Then
        temp_str = Left(temp_str, 2) & "." & Mid(temp_str, 3)
    End If
    pos = Len(Mid(temp_str, InStr(temp_str, ".") + 1))
End If
```

And finally constructing everything into one string using concatenation operator &.

```
Normalize = temp_str & " * " & base & " ^ " & pos
```

User form initialization

Empties input box and result box. Adds options to the from base selection and to base selection, focusing the user input on the Input Box, using .SetFocus.

```
With InputBox
    .SetFocus
End With
```

To make the interface more user-friendly, I used labels, pop ups messages when you click special label to find out more about its functions, toggle button “Normalize”, as we can have normalized or not normalized result. Also, I made the choice of bases strictly to be chose only from what is provided, making it even more user friendly and saving results to another sheet for each session of usage of the converter, to not get lost after each usage.

3. Getting started

To get started with the converter just click the button in the first sheet, then the user form appears, and you are ready to go.