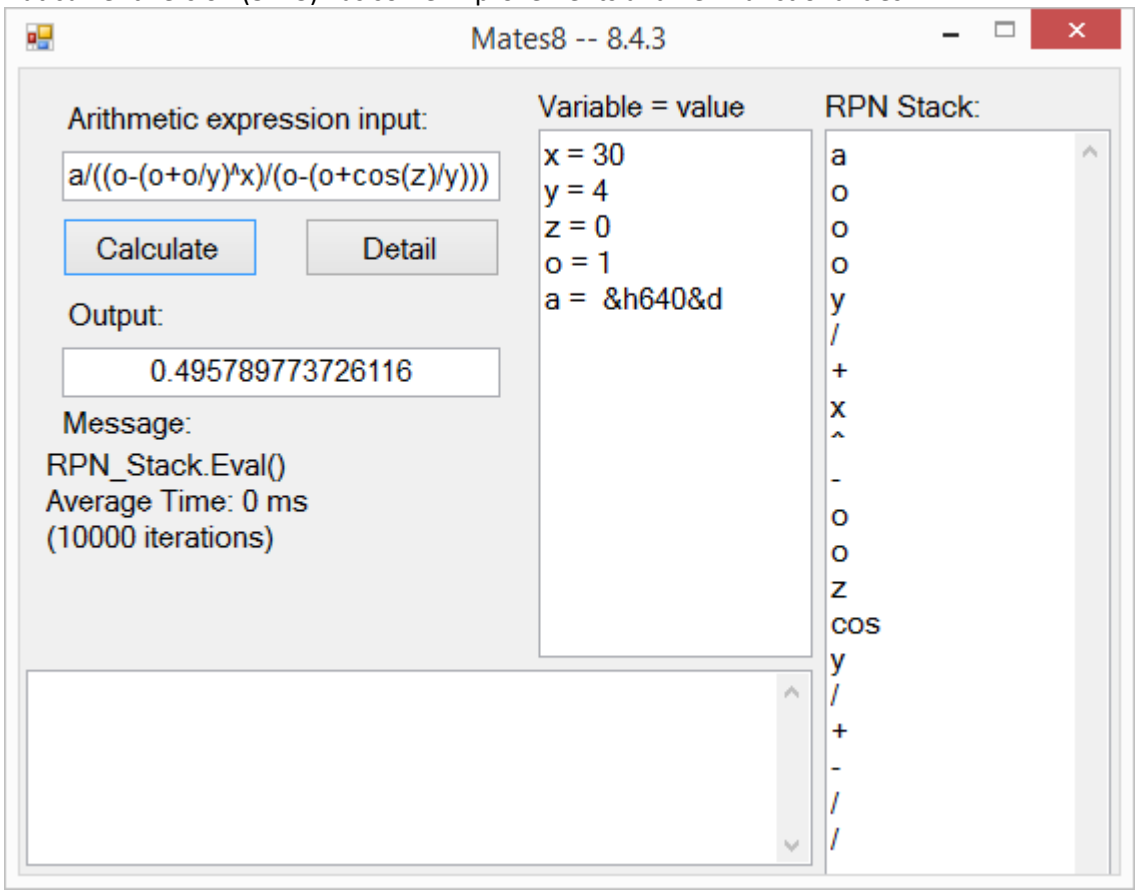
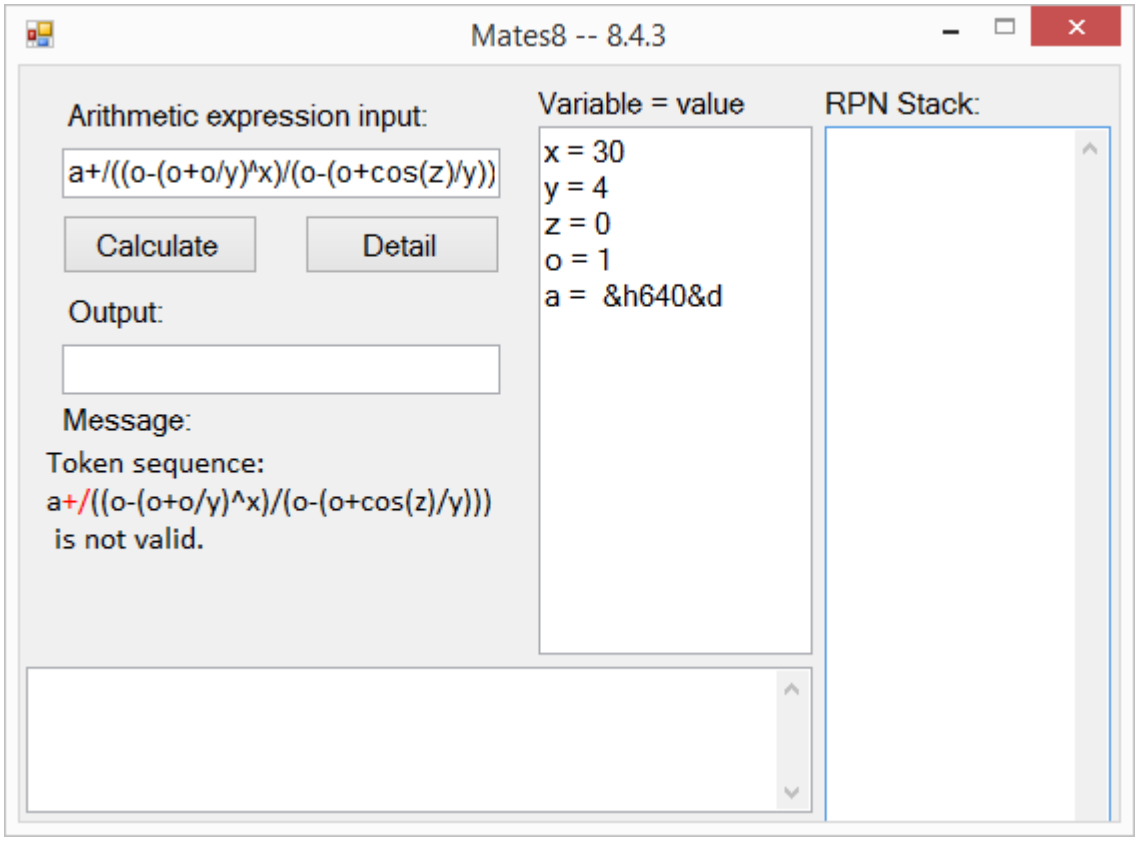
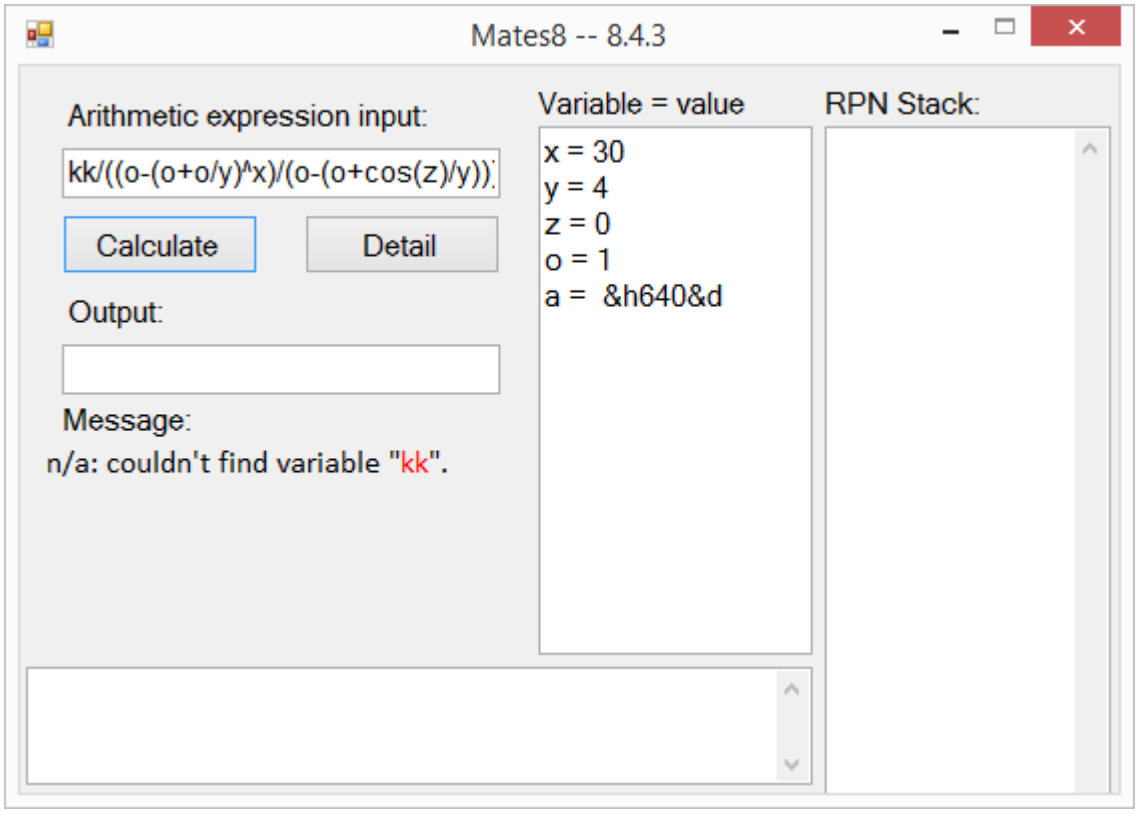
This time classes remain the same four: exprParser, RPN_Stack, Config and msg8.
But current version (8.4.3) has some improvements and new functionalities.

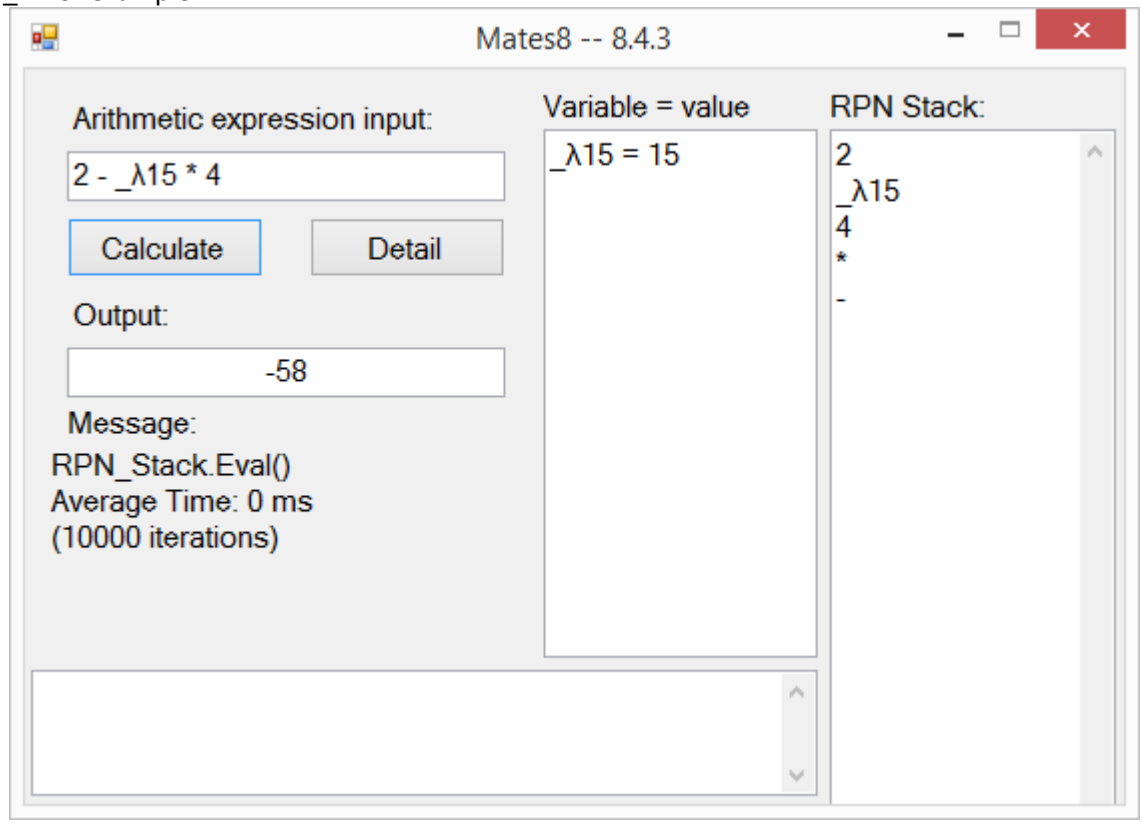




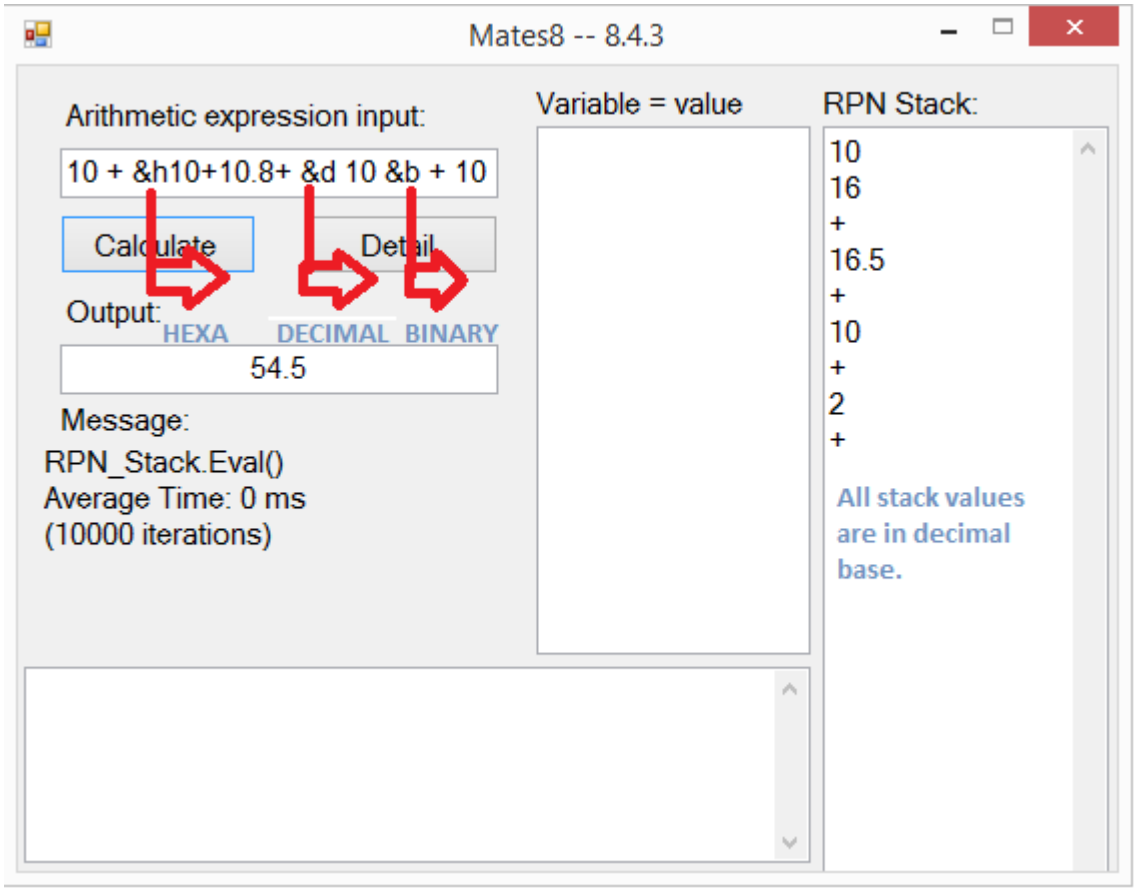
A. Client side.

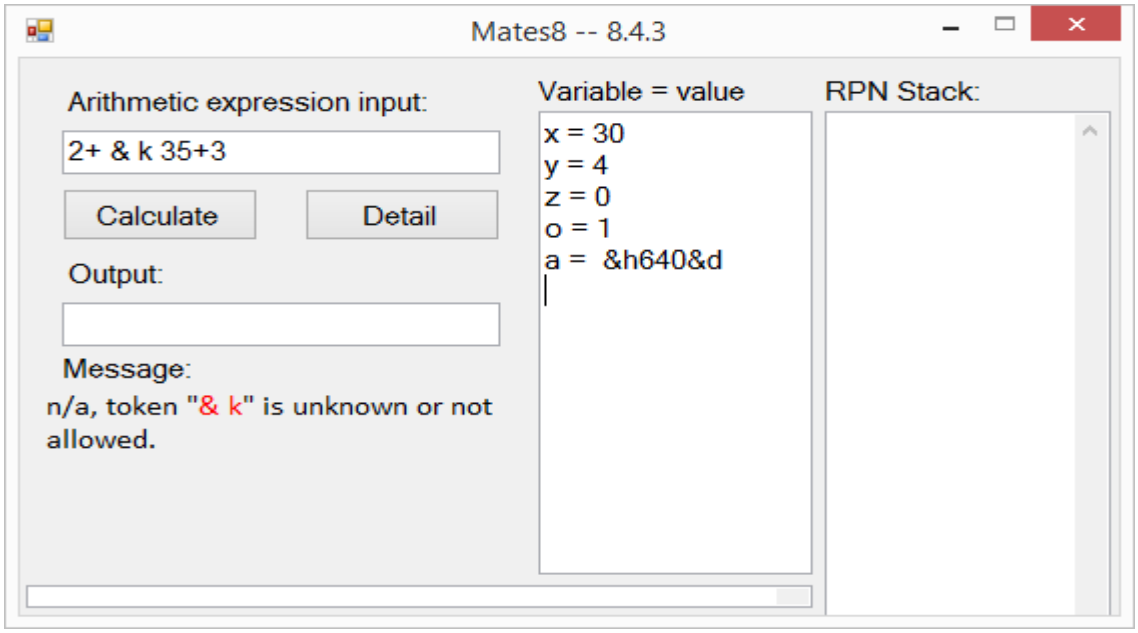


As you may see, now, variables may be one or more characters long. Names may not contain numbers; if there is the need to include numbers or Greek letters, then the name should be preceeded by an underscore "_". For example:



Also (you may skip this paragraph if you will be entering all numbers in decimal base) it is possible to enter numbers in hexadecimal, octal or binary base. To do so, the default decimal base may be overridden by the key word &h for hexadecimal, or &o and &b, for octal and binary respectively. Once a base is overridden, the new base prevails for any number, until another token &h, &d, &o or &b is found.





B. The insides

1. Class 'exprParser'

```
Private Sub nextExpr()        ' - +
  Try
    nextTerm()
    Do While curOptor = optor.add OrElse curOptor = optor.substract
      Dim oStk As New StackTkn(tokenType.optor, curOptor, _
          0.0, optorPos, optoriTkn, Chr(curOptor))
      nextTerm()
      rpn1.Add(oStk) ' Add the operator to the stack
    Loop
  Catch ex As Exception
    err = ex
  End Try
End Sub

Private Sub nextTerm()  '     * /
  Try
    nextPow()
    Do While curOptor = optor.multiply OrElse curOptor = optor.divide
      Dim oStk As New StackTkn(tokenType.optor, curOptor, _
          0.0, optorPos, optoriTkn, Chr(curOptor))
      nextPow()
      rpn1.Add(oStk) ' Add the operator to the stack
    Loop
  Catch ex As Exception
    err = ex
  End Try
End Sub

Private Sub nextPow() '    ^ !
  Dim sgn As Int32
  Try
    nextToken(sgn)
    Do While curOptor = optor.power OrElse curOptor = optor.modulo
      Dim oStk() As StackTkn = _
        {New StackTkn(tokenType.optor, curOptor, _
          0.0, optorPos, optoriTkn, Chr(curOptor))
        }
      If curOptor = optor.power Then ' ^
        .....
      Else
        ' %
        nextToken(sgn)
        rpn1.Add(oStk(0)) ' Add operator "%" power to the stack:
        sgn = 1
      End If
      nOpnd -= 1
    Loop
    If sgn = -1 Then
      rpn1.Add(New StackTkn( _
          tokenType.chgSgn, 0, 0.0, chgSgnPos, _
          chgSgniTkn, "-"))
    End If
  Catch ex As Exception
    err = ex
  End Try
End Sub
```

As you may know from previous documents, or see in the code snippet, ‘nextExpr()’ calls ‘nextTerm()’, ‘nextTerm()’ calls ‘nextPow()’, and ‘nextPow()’ calls ‘nextToken()’. The execution only exits nextToken under 4 circumstances: the end of tokens has been reached; or an operator, a right parenthesis or an error has been found.

Schematically, nextToken() sub is:

```
  Private Sub nextToken(ByRef sgn As Int32, _
                Optional bHasFn As Boolean = False)
    Dim c As Int32
    Dim bNotUnary As Boolean
    Try
      sgn = 1
      Do
        If iRe < sbExpr.Length Then
          iToken += 1
retry:
          c = AscW(sbExpr.Chars(iRe))
                  If c = 32 Then
                    sbExpr = sbExpr.Remove(iRe, 1) : GoTo retry
                  ElseIf c = 45 OrElse c = 43 OrElse c = 42 OrElse c = 47 _
                  OrElse c = 94 OrElse c = 37 OrElse c = 33 Then '    O P E R A T O R
                          '    O P E R A T O R
                   .....
          ElseIf curBase = numBase.decimal AndAlso _
          ((48 <= c AndAlso c <= 57) OrElse c = 46) Then '     N  U  M  B  E  R
                              .....
          ElseIf ...
                    ..... '       BASE <> DECIMAL ( HEXA, OCTAL, BINARY )
          Else
            .... ' Is a function?
            If iRe < iRe2 Then
              Dim sFnOrVar As String = LCase(sbExpr.ToString.Substring(iRe, iRe2 - iRe))
              Dim iFn As Int32 = Array.IndexOf(Config.vFn, sFnOrVar)
              If iFn > -1 Then
                      '        F  U  N  C  T  I  O  N
              ElseIf Array.IndexOf(vLogOp, sFnOrVar) > -1 Then
                    '  L O G I C A L   O P E R A T O R
              Else
                Dim posConst As Int32 = Array.IndexOf(Config.vConst, sFnOrVar)
                If posConst >= 0 Then
                                '  C O N S T A N T
                Else
                              '       V A R I A B L E
                End If
              End If
            ElseIf c = 91 OrElse c = 40 OrElse c = 123 Then '     LP
                    .....
            ElseIf c = 93 OrElse c = 41 OrElse c = 125 Then '     RP
                    .....
            ElseIf c = 960 Then
                    ....          .'              (PI)
            ElseIf c = 38 Then ' 38="&"
                              .....                    ‘ change default numeric base
            ElseIf c = 39 Then ' 39="'" a comment
              Exit Do ' end of tokens
            ElseIf c = 95 Then ' "_"
              Dim m As Match = reVar2.Match(sbExpr.ToString, iRe)
                    .....
            ElseIf c = 58 OrElse c = 247 Then ' division  : ÷  OPERATORS
            Else  ..... ‘ error
            End If
          End If
          bNotUnary = True
        End If
      Loop While iRe < sbExpr.Length
      If bValidate AndAlso err Is Nothing Then
        Validate(tknGnralType.EOTokens, Chr(c))
      End If
      curOptor = -4 ' End Of Tokens
    Catch ex As Exception
      err = ex
    End Try
  End Sub
```

In words, nextToken() extracts the next token contained in the input string, i.e. in the stringbuilder sbExpr. The integer iRe holds the current position from which sbExpr has to be analyzed and when a token is extracted, iRe is incremented as much as the length of the token.

When there is an omitted operator, for instance sbExpr.toString = “2x”, a first token “2” is extracted and added to the stack, “rpn1.Add(New StackTkn(tokenType.oprnd, ...))”. The second token “x” is extracted and, then, a call to exprParser.Validate() will determine operator “*” is missing. So, “*” will be inserted in sbExpr (“2*x”) and, without modifying iRe, execution will branch to label “retry:” making “*” being the current token been analyzed. The same stands for a missing power “^” operator, for example “2x2” will be read as “2*x^2” and the sequences of tokens will be {“2”,”*”,”x”,”^”,”2”}.

If operands, as numbers, constants and variables are inmediatly added to the stack; operators defer this.

For input “2+3*x”:

```
Private Sub nextExpr()        ' - +
  Try
    nextTerm() ← in the course of this call token “2” is added to the stack  (1)

      (3) returns and curOptor is equal to optor.add, so execution enters the loop

    Do While curOptor = optor.add OrElse curOptor = optor.substract
        (4) an instance of StackTkn, oStk, of the operator “+” is generated:
      Dim oStk As New StackTkn(tokenType.optor, curOptor, _
          0.0, optorPos, optoriTkn, Chr(curOptor))
      nextTerm() ← (5) token “3” is added to the stack, and curOptor contains optor.multiply
      rpn1.Add(oStk) ' (8) Add the operator ”+” to the stack
    Loop ‘ (9) exits the loop (curOptor = - 4) and returns back to the initial caller: exprParse.Parse()
  Catch ex As Exception
    err = ex
  End Try
End Sub

Private Sub nextTerm() '     * /
  Try
    nextPow()← in the course of this call token “2” is added to the stack  (1); in a second call (5) token “3” is added to the stack, and curOptor contains
optor.multiply

    (2) curOptor is equal to optor.add, so the loop is skipped:
    (5) token “3” is added to the stack, and curOptor contains optor.multiply, so execution enters the loop

    Do While curOptor = optor.multiply OrElse curOptor = optor.divide
        (6) an instance of StackTkn, oStk, of the operator “*” is generated:
      Dim oStk As New StackTkn(tokenType.optor, curOptor, _
          0.0, optorPos, optoriTkn, Chr(curOptor))
      nextPow()  ← (5) token “x” is added to the stack, and curOptor contains -4 (end of tokens)
      rpn1.Add(oStk) ' (6) Add the operator ”*” to the stack
    Loop ‘ (7) exits the loop (curOptor = - 4) and returns to (8) in nextExpr()
  Catch ex As Exception
    err = ex
  End Try
End Sub

Private Sub nextPow() '   ^ !
  Dim sgn As Int32
  Try
    nextToken(sgn)
    Do While curOptor = optor.power OrElse curOptor = optor.modulo
        .... FOR “2+3*x” ALWAYS SKIPS THIS LOOP BECAUSE THERE ARE NO POWER OR MODULO OPERATORS
    Loop
  Catch ex As Exception
    err = ex
  End Try
End Sub
```