‘Mates8’ usage in VB6

In version v8.3.0 there have been several changes.

New ‘Config’ class allows private input and output configuration: for rounding; imaginary symbol, decimal or other numeric base output; fractions; input case sensitivity; etcetera. Now there is no shared assignments to ‘MathGlobal8’ members, but to an instance of ‘Config’ class.

Method ‘matrixParser.parse’ is no more a shared method so, an instance is needed to invoke the method.

Output methods like ‘toStringExpr(cfg as Config)’ or ‘toStringComplex(cfg as Config)’ employ the Config instance parameter to know how to formatter the output data.

Please find an explanation in the following code’s instructions and comments:

```
1. Imports mates8VBA
2. Imports cfg = mates8VBA.MathGlobal8
3.
4. Module Module1
5.
6.     Sub Main()
7.         testMates8VBA()
8.     End Sub
9.     Sub testMates8VBA()
10.         Try
11.             Dim N As Int32 = 1
12.             Dim mP As New matrixParser
13.             Dim strQuery As String = "2*2"
14.             Dim oVars As VarsAndFns = Nothing
15.             Dim sOut As String = ""
16.             Dim strVarsAndFns As String = ""
17.             Dim cfg As New Config
18.
19.             ' Set configuration for input data:
20.             cfg.bEngNotation = True ' exponents multiples of 3 (10.3e3, 1.2e6, ...)
21.             cfg.bIgnoreSpaces = True
22.             cfg.bCaseSensitive = True
```

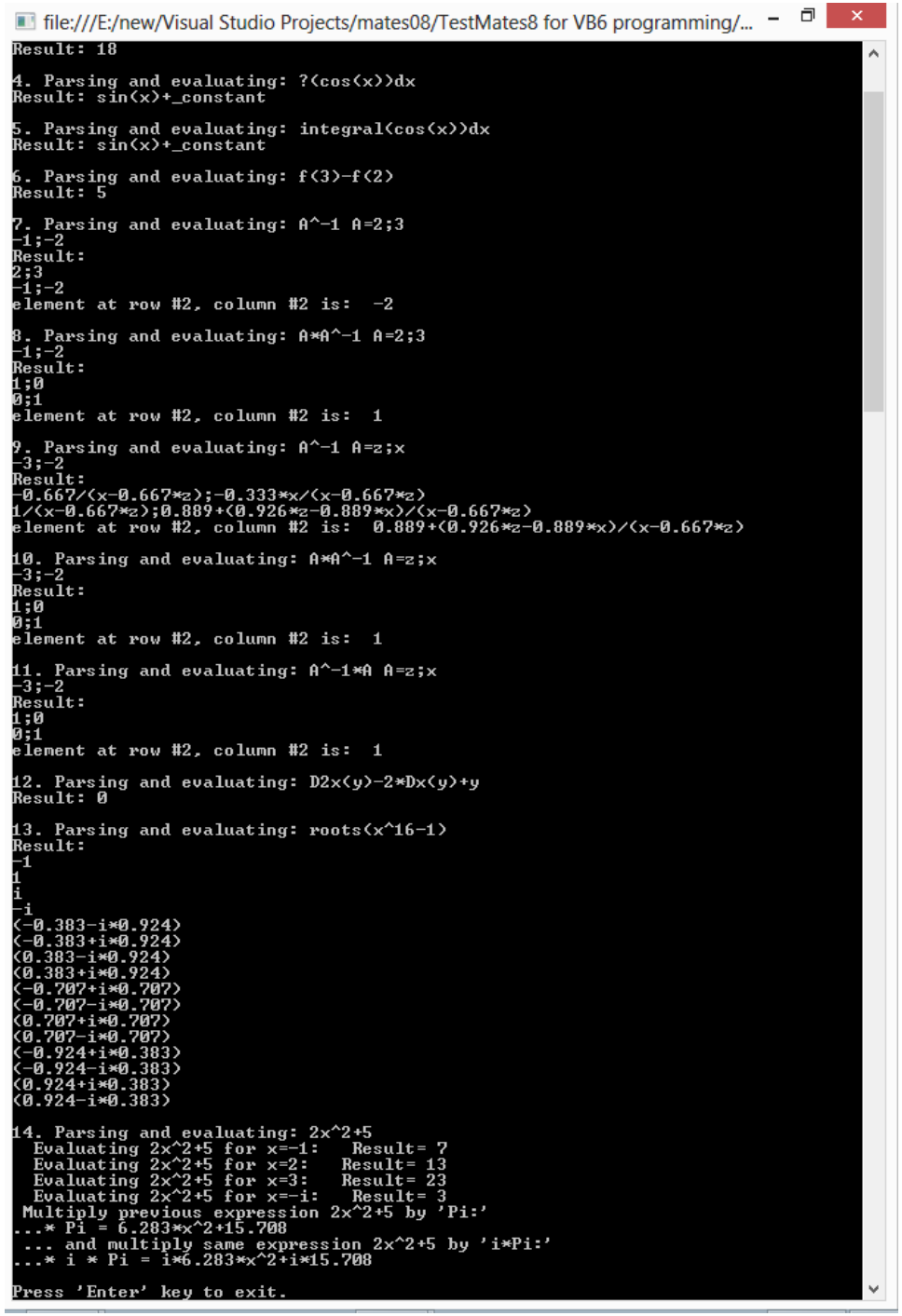
```
23.             cfg.degreesType = MathGlobal8.degreesType.radians
24.
25.             ' the following are examined only when
26.             ' 'ToString" type methods are invoked:
27.             cfg.bFractions = True
28.             cfg.bRounding = True ' round to 3 decimals
29.             cfg.bDetail = False ' no detailed info
30.             cfg.Base = MathGlobal8.outputBase.decimal
31.
32.             ' set config. into mP, mostly for
33.             ' input data as in toStringXXXX() methods
34.             ' cfg is needed to be passed as parameter (except for matrixParser
35.             ' mP.toString()). These are toStringExpr(cfg), toStringMatrixParser(cfg),
36.             ' toStringMtx(cfg), toStringPoly(cfg), toStringComplex(cfg),
37.             ' for expressions, matrices, polynomials and complex numbers,
38.             ' respectively.
39.             mP.setConfig(cfg)
40.
41.
42.
43.             sOut = sOut & CStr(N) & ". Parsing and evaluating: " & strQuery
44.             mP.parse(CStr(strQuery), "", Nothing)
45.             If Len(mP.errMsg) Then
46.                 sOut = mP.errMsg
47.             End If
48.             sOut = sOut & vbCrLf & "Result: " & mP.ToString & vbCrLf
49.             sig(sOut, N)
50.
51.             strQuery = "2*x+3*x"
52.             sOut = sOut & CStr(N) & ". Parsing and evaluating: " & strQuery ' 2*x+3*x
53.             mP.parse(strQuery, "", Nothing)
54.             If Len(mP.errMsg) Then
55.                 sOut = mP.errMsg
56.             End If
57.             sOut = sOut & vbCrLf & "Result: " & mP.ToString & vbCrLf ' Result: 5*x
58.             sig(sOut, N)
59.
60.             ' Prefix for hexadecimal numbers is &h, for octal &o, binary prefixed by &b
61.             ' 255 as hexadecimal (&hFF),  logical optor. AND, 15 as hexa.(&hF)
62.             ' logical operators valid are "and", "or", "xor", "not", "nand", "nor"
63.             strQuery = "(&hff AND &hf)+3" ' ...and add 3 (decimal)
64.             sOut = sOut & CStr(N) & ". Parsing and evaluating: " & strQuery
65.             mP.parse(strQuery, "", Nothing)
66.             If Len(mP.errMsg) Then
67.                 sOut = mP.errMsg
68.             End If
```

```
69.             sOut = sOut & vbCrLf & "Result: " & mP.ToString & vbCrLf ' ' Result: 18
70.             sig(sOut, N)
71.
72.             strQuery = "∫(cos(x))dx"
73.             sOut = sOut & CStr(N) & ". Parsing and evaluating: " & strQuery
74.             mP.parse(strQuery, "", Nothing)
75.             If Len(mP.errMsg) Then
76.                 sOut = mP.errMsg
77.             End If
78.             sOut = sOut & vbCrLf & "Result: " & mP.ToString & vbCrLf ' Result: sin(x) + _constant
79.             sig(sOut, N)
80.
81.
82.             strQuery = "integral(cos(x))dx"
83.             sOut = sOut & CStr(N) & ". Parsing and evaluating: " & strQuery ' Integral(cos(x))dx
84.             mP.parse(strQuery, "", Nothing)
85.             If Len(mP.errMsg) Then
86.                 sOut = mP.errMsg
87.             End If
88.             sOut = sOut & vbCrLf & "Result: " & mP.ToString & vbCrLf ' Result: sin(x) + _constant
89.             sig(sOut, N)
90.
91.             strQuery = "f(3)-f(2)"
92.             strVarsAndFns = "f(x)=x^2-1"
93.             oVars = Nothing
94.             sOut = sOut & CStr(N) & ". Parsing and evaluating: " & strQuery
95.             mP.parse(strQuery, strVarsAndFns, oVars)
96.             If Len(mP.errMsg) Then
97.                 sOut = mP.errMsg
98.             End If
99.             sOut = sOut & vbCrLf & "Result: " & mP.ToString & vbCrLf ' ' Result: (3*3-1)-(2*2-1)=8-3=5
100.             sig(sOut, N)
101.
102.             cfg.bIgnoreSpaces = False ' carrige return (vbCrLf) must be considered in strVarsAndFns:
103.             mP.setConfig(cfg) ' set new configuration value
104.             strQuery = "A^-1"
105.             strVarsAndFns = "A=2;3" & vbCrLf & "-1;-2" ' watch for row and columns' delimiter
106.             sOut = sOut & CStr(N) & ". Parsing and evaluating: " & strQuery & " " & strVarsAndFns
107.             mP.parse(strQuery, strVarsAndFns, Nothing)
108.             If Len(mP.errMsg) Then
109.                 sOut = mP.errMsg
110.             End If
111.             sOut = sOut & vbCrLf & "Result: " & vbCrLf & mP.ToString
112.             sOut = sOut & vbCrLf & "element at row #2, column #2 is:  " & _
113.                                    mP.ret.exprMtx.getExpr(1, 1).ToStringExpr(cfg) & vbCrLf
114.             sig(sOut, N)
```

```
115.
116.            strQuery = "A*A^-1"
117.            strVarsAndFns = "A=2;3" & vbCrLf & "-1;-2" ' watch for row and columns' delimiter
118.            sOut = sOut & CStr(N) & ". Parsing and evaluating: " & strQuery & " " & strVarsAndFns
119.            mP.parse(strQuery, strVarsAndFns, Nothing)
120.            If Len(mP.errMsg) Then
121.                sOut = mP.errMsg
122.            End If
123.            sOut = sOut & vbCrLf & "Result: " & vbCrLf & mP.ToString
124.            sOut = sOut & vbCrLf & "element at row #2, column #2 is:  " & _
125.                                    mP.ret.exprMtx.getExpr(1, 1).toDouble.ToString & vbCrLf
126.            sig(sOut, N)
127.
128.            strQuery = "A^-1"
129.            strVarsAndFns = "A=z;x" & vbCrLf & "-3;-2" ' watch for row and columns' delimiter
130.            sOut = sOut & CStr(N) & ". Parsing and evaluating: " & strQuery & " " & strVarsAndFns
131.            mP.parse(strQuery, strVarsAndFns, Nothing)
132.            If Len(mP.errMsg) Then
133.                sOut = mP.errMsg
134.            End If
135.            sOut = sOut & vbCrLf & "Result: " & vbCrLf & mP.ToString
136.            sOut = sOut & vbCrLf & "element at row #2, column #2 is:  " & _
137.                                    mP.ret.exprMtx.getExpr(1, 1).ToStringExpr(cfg) & vbCrLf
138.            sig(sOut, N)
139.
140.
141.            strQuery = "A*A^-1"
142.            strVarsAndFns = "A=z;x" & vbCrLf & "-3;-2" ' watch for row and columns' delimiter
143.            sOut = sOut & CStr(N) & ". Parsing and evaluating: " & strQuery & " " & strVarsAndFns
144.            mP.parse(strQuery, strVarsAndFns, Nothing)
145.            If Len(mP.errMsg) Then
146.                sOut = mP.errMsg
147.            End If
148.            sOut = sOut & vbCrLf & "Result: " & vbCrLf & mP.ToString
149.            sOut = sOut & vbCrLf & "element at row #2, column #2 is:  " & _
150.                                    mP.ret.exprMtx.getExpr(1, 1).ToStringExpr(cfg) & vbCrLf
151.            sig(sOut, N)
152.
153.
154.            strQuery = "A^-1*A"
155.            strVarsAndFns = "A=z;x" & vbCrLf & "-3;-2" ' watch for row and columns' delimiter
156.            sOut = sOut & CStr(N) & ". Parsing and evaluating: " & strQuery & " " & strVarsAndFns
157.            mP.parse(strQuery, strVarsAndFns, Nothing)
158.            If Len(mP.errMsg) Then
159.                sOut = mP.errMsg
160.            End If
```

```
161.            sOut = sOut & vbCrLf & "Result: " & vbCrLf & mP.ToString
162.            sOut = sOut & vbCrLf & "element at row #2, column #2 is:  " & _
163.                                   mP.ret.exprMtx.getExpr(1, 1).ToStringExpr(cfg) & vbCrLf
164.            sig(sOut, N)
165.
166.            strQuery = "D2x(y)-2*Dx(y)+y"
167.            strVarsAndFns = "y=x*exp(x)"
168.            sOut = sOut & CStr(N) & ". Parsing and evaluating: " & strQuery
169.            mP.parse(strQuery, strVarsAndFns, Nothing)
170.            If Len(mP.errMsg) Then
171.                sOut = mP.errMsg
172.            End If
173.            sOut = sOut & vbCrLf & "Result: " & mP.ToString & vbCrLf
174.            sig(sOut, N)
175.
176.            strQuery = "roots(x^16-1)"
177.            sOut = sOut & CStr(N) & ". Parsing and evaluating: " & strQuery
178.            mP.parse(strQuery, "", Nothing)
179.            If Len(mP.errMsg) Then
180.                sOut = mP.errMsg
181.            End If
182.            sOut = sOut & vbCrLf & "Result: " & vbCrLf & mP.ToString & vbCrLf ' ' Result: -1, 1, i, ...
183.            sig(sOut, N)
184.
185.            ' Now we want to evaluate "2x^2+5" for x=-1, x=2, x=3 and x=-i
186.            strQuery = "2x^2+5"
187.            sOut = sOut & CStr(N) & ". Parsing and evaluating: " & strQuery & vbCrLf
188.            ' 1) Call parse method:
189.            mP.parse(strQuery, "", Nothing)
190.
191.            ' 2) An AST tree has been created; call the evalExpression() method
192.            '    for each value of x:
193.            sOut = sOut & "  Evaluating " + strQuery + " for x=-1: "
194.            Dim cmplx As Complex = mP.ret.curExpr.evalExpression(New Complex(-1))
195.            sOut = sOut & "  Result= " & cmplx.ToString & vbCrLf ' =7
196.            sOut = sOut & "  Evaluating " + strQuery + " for x=2: "
197.            cmplx = mP.ret.curExpr.evalExpression(New Complex(2))
198.            sOut = sOut & "  Result= " & cmplx.ToString & vbCrLf ' =13
199.            sOut = sOut & "  Evaluating " + strQuery + " for x=3: "
200.            cmplx = mP.ret.curExpr.evalExpression(New Complex(3))
201.            sOut = sOut & "  Result= " & cmplx.ToString & vbCrLf ' =23
202.            sOut = sOut & "  Evaluating " + strQuery + " for x=-i: "
203.            cmplx = mP.ret.curExpr.evalExpression(New Complex(0, -1))
204.            sOut = sOut & "  Result= " & cmplx.ToString & vbCrLf ' =3
205.
206.            sOut = sOut & " Multiply previous expression " + strQuery + " by 'Pi:'"
```

```
207.             sOut = sOut & vbCrLf
208.             Dim product As Expression = _
209.                 (New Expression(Math.PI)) * mP.ret.curExpr
210.             sOut = sOut & "...* Pi = " + product.ToString
211.             sOut = sOut & vbCrLf
212.             N += 1
213.
214.             sOut = sOut & " ... and multiply same expression " + strQuery + " by 'i*Pi:'"
215.             sOut = sOut & vbCrLf
216.             product = _
217.                 (New Expression(New Complex(0, Math.PI))) * mP.ret.curExpr
218.             sOut = sOut & "...* i * Pi = " + product.ToString
219.             sOut = sOut & vbCrLf
220.             N += 1
221.
222.
223.
224.             Console.WriteLine(sOut)
225.         Catch ex As Exception
226.             Console.WriteLine(ex.ToString)
227.         End Try
228.         Console.WriteLine("Press 'Enter' key to exit.")
229.         Console.ReadLine()
230.     End Sub
231.     Sub sig(ByRef sOut As String, ByRef N As Int32)
232.         sOut = sOut & vbCrLf
233.         N = N + 1
234.     End Sub
235.
236.
237. End Module
```


file:///E:/new/Visual Studio Projects/mates08/TestMates8 for VB6 programming/...
Result: 18
4. Parsing and evaluating: ?(cos(x))dx
Result: sin(x)+_constant
5. Parsing and evaluating: integral(cos(x))dx
Result: sin(x)+_constant
6. Parsing and evaluating: f(3)-f(2)
Result: 5
7. Parsing and evaluating: A^-1 A=2;3
-1;-2
Result:
2;3
-1;-2
element at row #2, column #2 is: -2
8. Parsing and evaluating: A*A^-1 A=2;3
-1;-2
Result:
1;0
0;1
element at row #2, column #2 is: 1
9. Parsing and evaluating: A^-1 A=z;x
-3;-2
Result:
-0.667/(x-0.667*z);-0.333*x/(x-0.667*z)
1/(x-0.667*z);0.889+(0.926*z-0.889*x)/(x-0.667*z)
element at row #2, column #2 is: 0.889+(0.926*z-0.889*x)/(x-0.667*z)
10. Parsing and evaluating: A*A^-1 A=z;x
-3;-2
Result:
1;0
0;1
element at row #2, column #2 is: 1
11. Parsing and evaluating: A^-1*A A=z;x
-3;-2
Result:
1;0
0;1
element at row #2, column #2 is: 1
12. Parsing and evaluating: D2x(y)-2*Dx(y)+y
Result: 0
13. Parsing and evaluating: roots(x^16-1)
Result:
-1
1
i
-i
(-0.383-i*0.924)
(-0.383+i*0.924)
(0.383-i*0.924)
(0.383+i*0.924)
(-0.707+i*0.707)
(-0.707-i*0.707)
(0.707+i*0.707)
(0.707-i*0.707)
(-0.924+i*0.383)
(-0.924-i*0.383)
(0.924+i*0.383)
(0.924-i*0.383)
14. Parsing and evaluating: 2x^2+5
Evaluating 2x^2+5 for x=-1: Result= 7
Evaluating 2x^2+5 for x=2: Result= 13
Evaluating 2x^2+5 for x=3: Result= 23
Evaluating 2x^2+5 for x=-i: Result= 3
Multiply previous expression 2x^2+5 by 'Pi:'
...* Pi = 6.283*x^2+15.708
... and multiply same expression 2x^2+5 by 'i*Pi:'
...* i * Pi = i*6.283*x^2+i*15.708
Press 'Enter' key to exit.

The code and the image below were taken from a VB6 application.

Due to our impossibility to install a very old VB6 version in the new Windows 8, please find it as originally extracted.

Anyway, the former code, theoretically, should work as well in VB6 and VBA, mainly because otherwise the .tlb file would not have been generated.

```
1.  Private Sub btnStart_Click()
2.     ' INPUT for matrixParser.Parse()
3.     ' ==============================
4.     ' strQuery = the math expression to parse,
5.     ' for example: strQuery="2*2", "2*x+3*x", "?(cos(x))dx", "roots(x^16-1)"
6.     '              or a matrix expression with columns delimited by
7.     '              semicolons and rows by vbCrLf as "A^-1"
8.     ' strVarsAndFns = "" or eventualy variables values or functions
9.     '                 for ex. "x=-1" or  "A=2;3" + vbCrLf + "-1;2"
10.    ' Dim oVars As VarsAndFns = Nothing
11.
12.    ' OUTPUT:
13.    ' =======
14.    ' 1) mP.toString returns the result as a string.
15.    ' 2) mP.retCjo() returns a complex or, eventually, an array of complex.
16.    ' 3) When the result is a matrix
17.    '   xmP.ret.exprMtx.getExpr(row, column) returns the expression
18.    '   contained at a row and column ((0,0) is the first row and columns)
19.    '
20.    '   mP.ret.exprMtx.getExpr(row, column).IsReal will tell
21.    '   if the element's content is a real number and
22.    '   mP.ret.exprMtx.getExpr(row, column).toDouble its value.
23.    '   mP.ret.exprMtx.rows gives the number of rows in the matrix
24.    '   mP.ret.exprMtx.cols gives the # of columns
```

```
25.
26.    On Error GoTo error
27.
28.    Dim cfg As New config
29.    Dim mP As New matrixParser
30.    Dim oVars As VarsAndFns
31.    Dim strQuery, strVarsAndFns, sResult
32.    Dim exprMtx As exprMatrix
33.    Dim expr As Expression
34.    Dim x As New Complex
35.    Dim cmplx As Complex
36.    Dim sOut
37.    Dim N
38.    N = 1
39.    cfg.bIgnoreSpaces = True
40.    ' the following are examined only when
41.    ' 'ToString" type methods are invoked:
42.        '    MathGlobal8.bUseEng = True ' exponents multiples of 3 (10.3e3, 1.2e6, ...)
43.    cfg.bRounding = True
44.    cfg.bDetail = False
45.    cfg.bBinary = False
46.    cfg.bOctal = False
47.    cfg.bHexadecimal = False
48.    tbOutput.Text = ""
49.    strVarsAndFns = ""
50.    sOut = ""
51.
52.    strQuery = "2*2"
53.    sOut = sOut & CStr(N) & ". Parsing and evaluating: " & strQuery
54.    mP.parse CStr(strQuery), "", oVars
55.    If Len(mP.errMsg) Then
56.       sOut = mP.errMsg
57.    End If
58.    sOut = sOut & vbCrLf & "Result: " & mP.ToString & vbCrLf
```

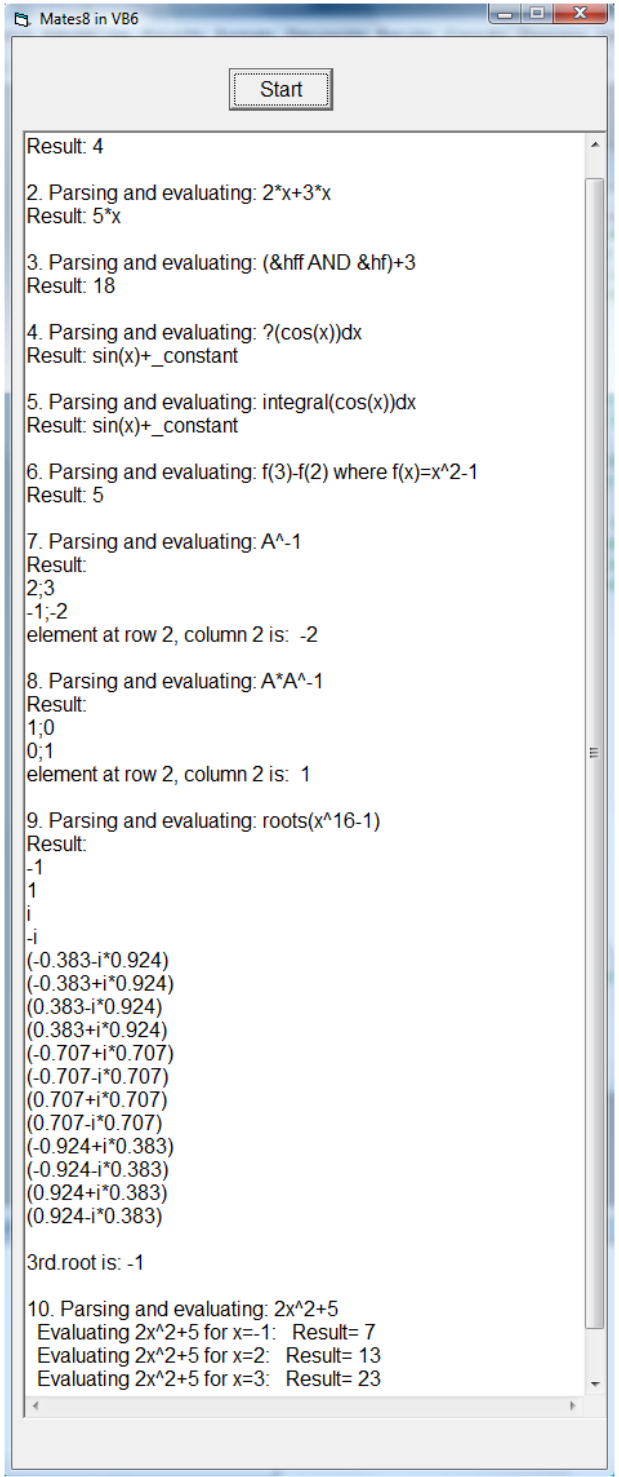
```
59.  sOut = sOut & vbCrLf
60.  N = N + 1
61.
62.  strQuery = "2*x+3*x"
63.  sOut = sOut & CStr(N) & ". Parsing and evaluating: " & strQuery ' 2*x+3*x
64.  mP.parse strQuery, "", Nothing
65.  If Len(mP.errMsg) Then
66.     sOut = mP.errMsg
67.  End If
68.  sOut = sOut & vbCrLf & "Result: " & mP.ToString & vbCrLf
69.  sOut = sOut & vbCrLf
70.  N = N + 1
71.
72.  ' Prefix for hexadecimal numbers is &h, for octal &o, binary prefixed by &b
73.  ' 255 as hexadecimal (&hFF),  logical optor. AND, 15 as hexa.(&hF)
74.  ' logical operators valid are "and", "or", "xor", "not", "nand", "nor"
75.  strQuery = "(&hff AND &hf)+3" ' ...and add 3 (decimal)
76.  sOut = sOut & CStr(N) & ". Parsing and evaluating: " & strQuery
77.  mP.parse strQuery, "", Nothing
78.  If Len(mP.errMsg) Then
79.     sOut = mP.errMsg
80.  End If
81.  sOut = sOut & vbCrLf & "Result: " & mP.ToString & vbCrLf ' ' Result: 18
82.  sOut = sOut & vbCrLf
83.  N = N + 1
84.
85.  strQuery = ChrW(8747) & "(cos(x))dx" ' ChrW(8747) stands for integration char
86.  sOut = sOut & CStr(N) & ". Parsing and evaluating: " & strQuery ' Integral(cos(x))dx
87.  mP.parse strQuery, "", Nothing
88.  If Len(mP.errMsg) Then
89.     sOut = mP.errMsg
90.  End If
91.  sOut = sOut & vbCrLf & "Result: " & mP.ToString & vbCrLf ' Result: sin(x) + _constant
92.  sOut = sOut & vbCrLf
```

```
93.    N = N + 1
94.
95.    strQuery = "integral(cos(x))dx"
96.    sOut = sOut & CStr(N) & ". Parsing and evaluating: " & strQuery ' Integral(cos(x))dx
97.    mP.parse strQuery, "", Nothing
98.    If Len(mP.errMsg) Then
99.       sOut = mP.errMsg
100.   End If
101.   sOut = sOut & vbCrLf & "Result: " & mP.ToString & vbCrLf ' Result: sin(x) + _constant
102.   sOut = sOut & vbCrLf
103.   N = N + 1
104.
105.   strQuery = "f(3)-f(2)"
106.   strVarsAndFns = "f(x)=x^2-1"
107.   Set oVars = Nothing
108.   sOut = sOut & CStr(N) & ". Parsing and evaluating: " & strQuery & " where " & 109.
strVarsAndFns
110.   mP.parse strQuery, strVarsAndFns, oVars
111.   If Len(mP.errMsg) Then
112.      sOut = mP.errMsg
113.   End If
114.   sOut = sOut & vbCrLf & "Result: " & mP.ToString & vbCrLf ' ' Result: (3*3-1)-(2*2-1)=8-
115. 3=5
116.   sOut = sOut & vbCrLf
117.   N = N + 1
118.
119.   cfg.bIgnoreSpaces = False ' carrige return (vbCrLf) must be considered in line 122:
120
121.   strQuery = "A^-1"
122.   strVarsAndFns = "A=2;3" & vbCrLf & "-1;-2" ' watch for row and columns' delimiter
123.   sOut = sOut & CStr(N) & ". Parsing and evaluating: " & strQuery
124.   mP.parse strQuery, strVarsAndFns, Nothing
125.   If Len(mP.errMsg) Then
126.      sOut = mP.errMsg
127.   End If
```

```
128.  sOut = sOut & vbCrLf & "Result: " & vbCrLf & mP.ToString
129.  Set expr = mP.retExprMtx.getExpMtx.getExpr(1, 1)
130.  sOut = sOut & vbCrLf & "element at row 2, column 2 is:  " & _
131.                  expr.toDouble & vbCrLf
132.  sOut = sOut & vbCrLf
133.  N = N + 1
134.
135.  strQuery = "A*A^-1"
136.  strVarsAndFns = "A=2;3" & vbCrLf & "-1;-2" ' watch for row and columns' delimiter
137.  sOut = sOut & CStr(N) & ". Parsing and evaluating: " & strQuery
138.  mP.parse strQuery, strVarsAndFns, Nothing
139.  If Len(mP.errMsg) Then
140.     sOut = mP.errMsg
141.  End If
142.  sOut = sOut & vbCrLf & "Result: " & vbCrLf & mP.ToString
143.  Set expr = mP.retExprMtx.getExpMtx.getExpr(1, 1)
144.  sOut = sOut & vbCrLf & "element at row 2, column 2 is:  " & _
145.                  expr.toDouble & vbCrLf
146.  sOut = sOut & vbCrLf
147.  N = N + 1
148.
149.
150.  strQuery = "roots(x^16-1)"
151.  sOut = sOut & CStr(N) & ". Parsing and evaluating: " & strQuery
152.  mP.parse strQuery, "", Nothing
153.  If Len(mP.errMsg) Then
154.     sOut = mP.errMsg
155.  End If
156.  sOut = sOut & vbCrLf & "Result: " & vbCrLf & mP.ToString & vbCrLf ' ' Result: -1, 1, i, ...
157.  sOut = sOut & vbCrLf & "3rd.root is: " & mP.getCplxElem(0).ToString & vbCrLf ' i
158.  sOut = sOut & vbCrLf
159.  N = N + 1
160.
161.  ' Now we want to evaluate "2x^2+5" for x=-1, x=2, x=3 and x=-i
```

```
    strQuery = "2x^2+5"
    sOut = sOut & CStr(N) & ". Parsing and evaluating: " & strQuery & vbCrLf
    ' 1) Call parse method:
    mP.parse strQuery, "", Nothing

    ' 2) An AST tree has been created; call the evalExpression() method
    '    for each value of x:
    Set expr = mP.retExprMtx.getExpMtx.getExpr(0, 0)
    sOut = sOut & "  Evaluating " + strQuery + " for x=-1: "
    x.setReIm -1, 0
    Set cmplx = expr.evalExpression(x)
    sOut = sOut & "  Result= " & cmplx.ToString & vbCrLf ' =7

    sOut = sOut & "  Evaluating " + strQuery + " for x=2: "
    x.setReIm 2, 0
    Set cmplx = expr.evalExpression(x)
    sOut = sOut & "  Result= " & cmplx.ToString & vbCrLf ' =13

    sOut = sOut & "  Evaluating " + strQuery + " for x=3: "
    x.setReIm 3, 0
    Set cmplx = expr.evalExpression(x)
    sOut = sOut & "  Result= " & cmplx.ToString & vbCrLf ' =23

    sOut = sOut & "  Evaluating " + strQuery + " for x=-i: "
    x.setReIm 0, -1
    Set cmplx = expr.evalExpression(x)
    sOut = sOut & "  Result= " & cmplx.ToString & vbCrLf ' =3

    tbOutput.Text = sOut
    Set mP = Nothing
    Exit Sub
error:
    On Error Resume Next
    tbOutput.Text = tbOutput.Text & vbCrLf & Err.Number & " " & Err.Description
```

```
196. End Sub
197.
198. Private Sub Form_Resize()
199.     On Error Resume Next
200.     tbOutput.Width = (Me.Width + tbOutput.Width) / 2 - tbOutput.Left - 50
201.    tbOutput.Height = (Me.Height + tbOutput.Height) / 2 - tbOutput.Top
202.
203. End Sub
```


Mates8 in VB6
Start
Result: 4
2. Parsing and evaluating: 2*x+3*x
Result: 5*x
3. Parsing and evaluating: (&hff AND &hf)+3
Result: 18
4. Parsing and evaluating: ?(cos(x))dx
Result: sin(x)+_constant
5. Parsing and evaluating: integral(cos(x))dx
Result: sin(x)+_constant
6. Parsing and evaluating: f(3)-f(2) where f(x)=x^2-1
Result: 5
7. Parsing and evaluating: A^-1
Result:
2;3
-1;-2
element at row 2, column 2 is: -2
8. Parsing and evaluating: A*A^-1
Result:
1;0
0;1
element at row 2, column 2 is: 1
9. Parsing and evaluating: roots(x^16-1)
Result:
-1
1
i
-i
(-0.383-i*0.924)
(-0.383+i*0.924)
(0.383-i*0.924)
(0.383+i*0.924)
(-0.707+i*0.707)
(-0.707-i*0.707)
(0.707+i*0.707)
(0.707-i*0.707)
(-0.924+i*0.383)
(-0.924-i*0.383)
(0.924+i*0.383)
(0.924-i*0.383)
3rd.root is: -1
10. Parsing and evaluating: 2x^2+5
Evaluating 2x^2+5 for x=-1: Result= 7
Evaluating 2x^2+5 for x=2: Result= 13
Evaluating 2x^2+5 for x=3: Result= 23