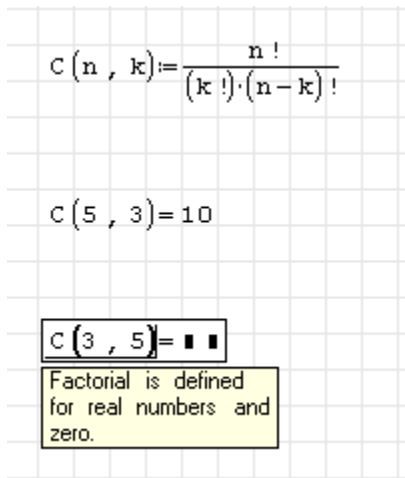


## Writing your first plug-in for Smath Studio Desktop

This tutorial uses a free development tool provided by Microsoft: Visual Basic Express 2010. This tutorial could be easily adapted for use with Visual Basic Express 2008 (or even the C# express editions). Please feel free to do so.

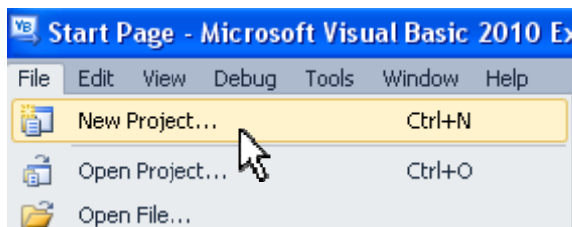
1. First, go to <http://www.microsoft.com/express/windows/> to download and install Visual Basic Express 2010. Follow the onscreen instructions to install to your system. (You will be prompted as to whether you would like to install SQL Server Express and Silverlight, but it is not necessary to install these components.)
2. For this plug-in, we will try to create a plug-in to realize a combinations function that achieves what is shown below:


$$C(n, k) := \frac{n!}{(k!)(n-k)!}$$
$$C(5, 3) = 10$$
$$C(3, 5) = \blacksquare \blacksquare$$

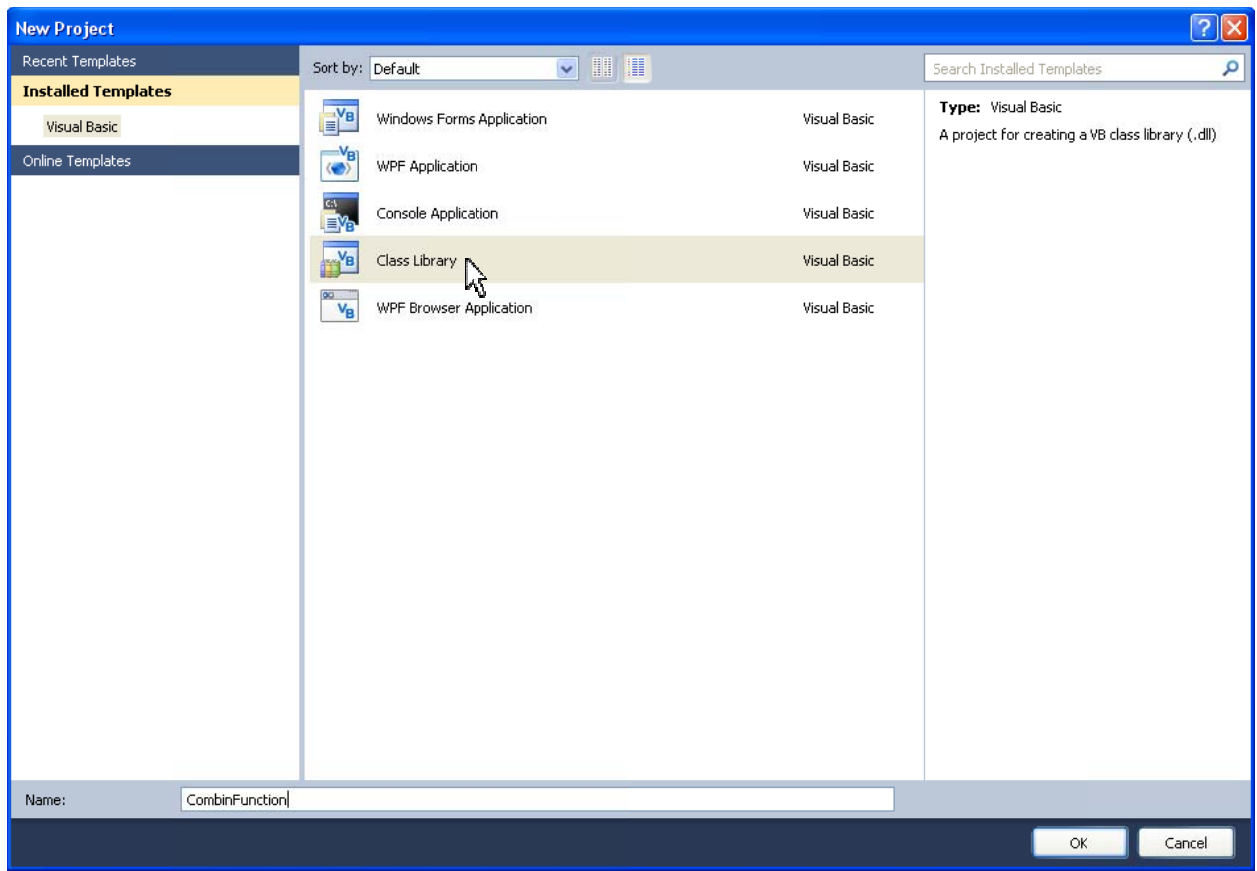
Factorial is defined for real numbers and zero.

The finished function syntax will be in the form:  $\text{combin}(n, k) =$

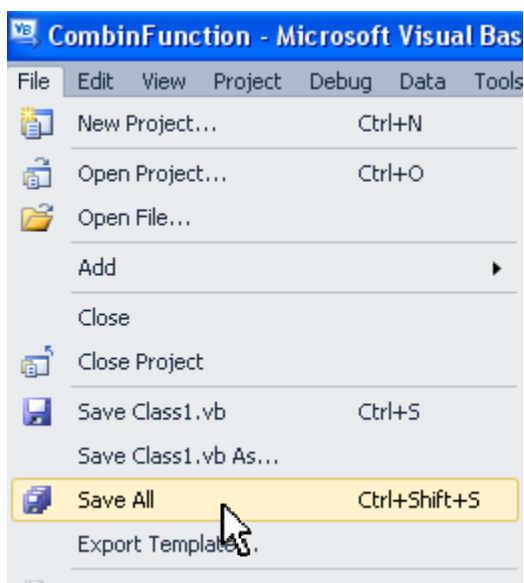
3. Once Visual Basic Express is installed, open Visual Basic Express and click on *File* → *New Project*



4. In the New Project dialog, click on “*Class Library*” and type in a name for the class library. In this case, we will call it “*CombinFunction*”. Then click OK.



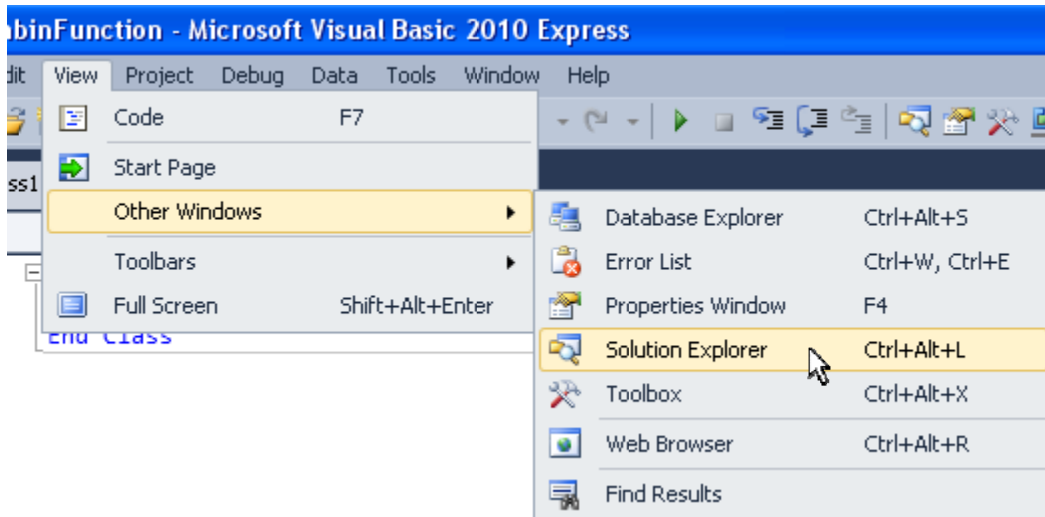
5. Before proceeding lets save the project, do this by clicking on *File* → *Save All*



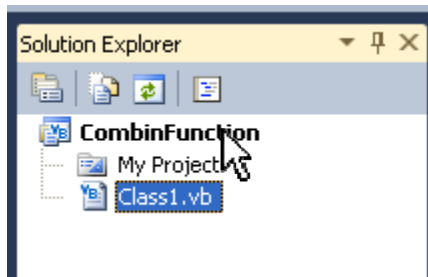
In the dialog that comes up, type in a name for the project and click Save.

IMPORTANT: Be sure to save your project periodically as you work on this tutorial!

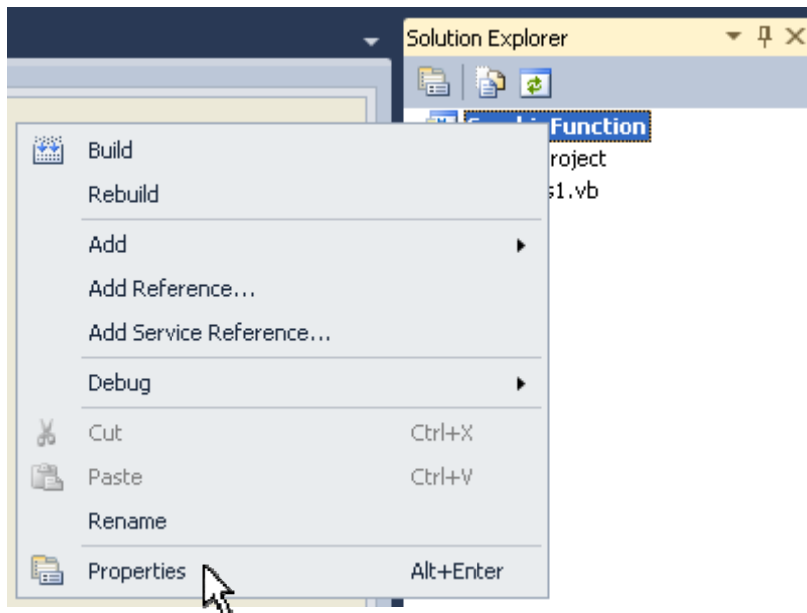
- Next, make the Solution Explorer visible (if it is not visible already) by clicking on *View* → *Other Windows* → *Solution Explorer*



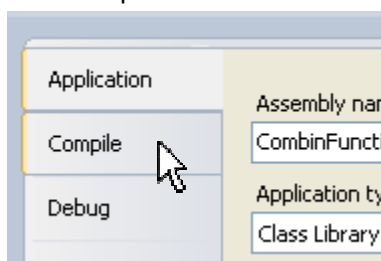
- In the Solution Explorer (which should be on the right side of the screen), right-click on the *CombinFunction* project name (or whatever you named your project).



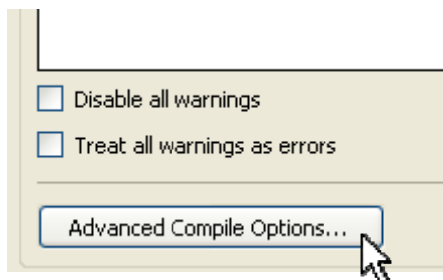
and then click on *Properties* in the drop down.



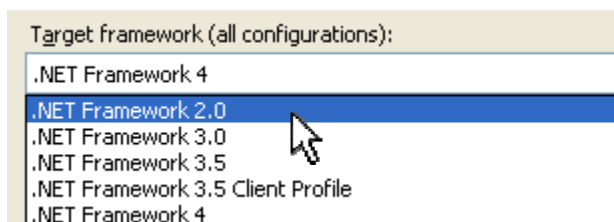
8. In the Properties window click on the *“Compile”* tab,



and then click on the *“Advanced Compile Options”* button at the bottom,

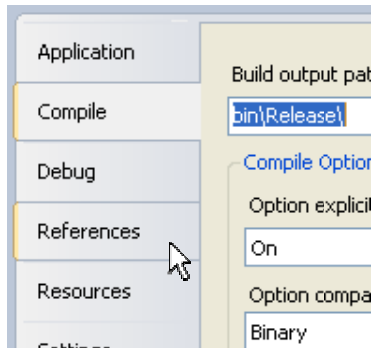


9. In the bottom drop down called *“Target framework”* select *“.NET Framework 2.0”* and then click OK.

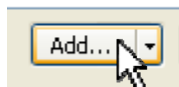


In the dialog that asks if you are sure you want to change the target framework, click “Yes”.

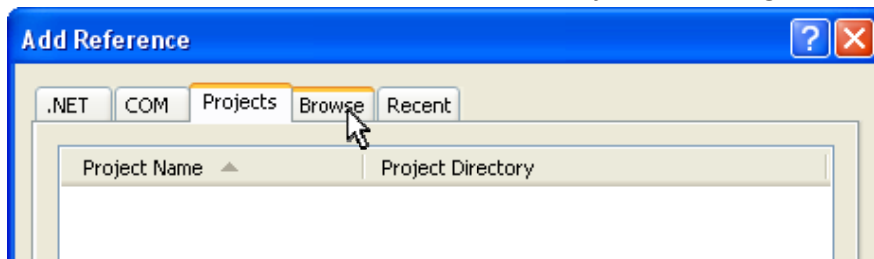
10. Now, return to the Solution Explorer, right-click again on the *CombinFunction* project name (or whatever you named your project) and then click again on Properties in the drop down. In the Properties window, click on the “References” tab.



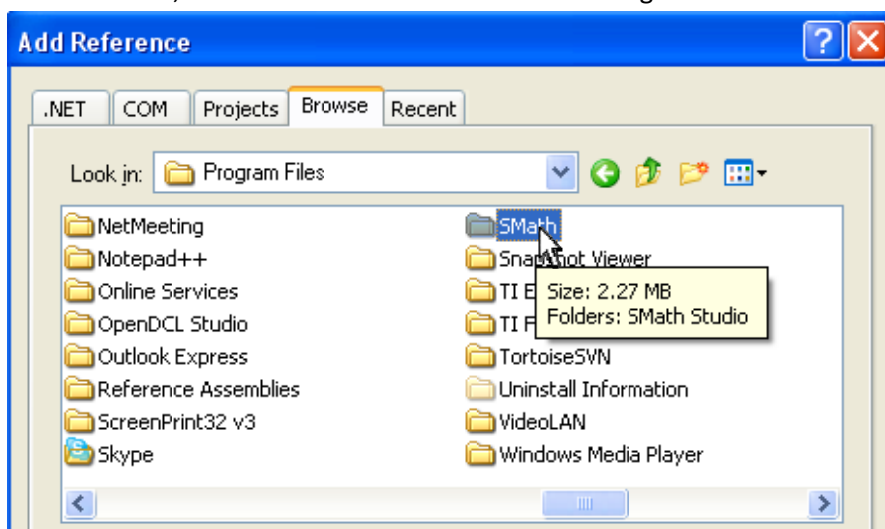
11. Under the References tab, click on the “Add” button,



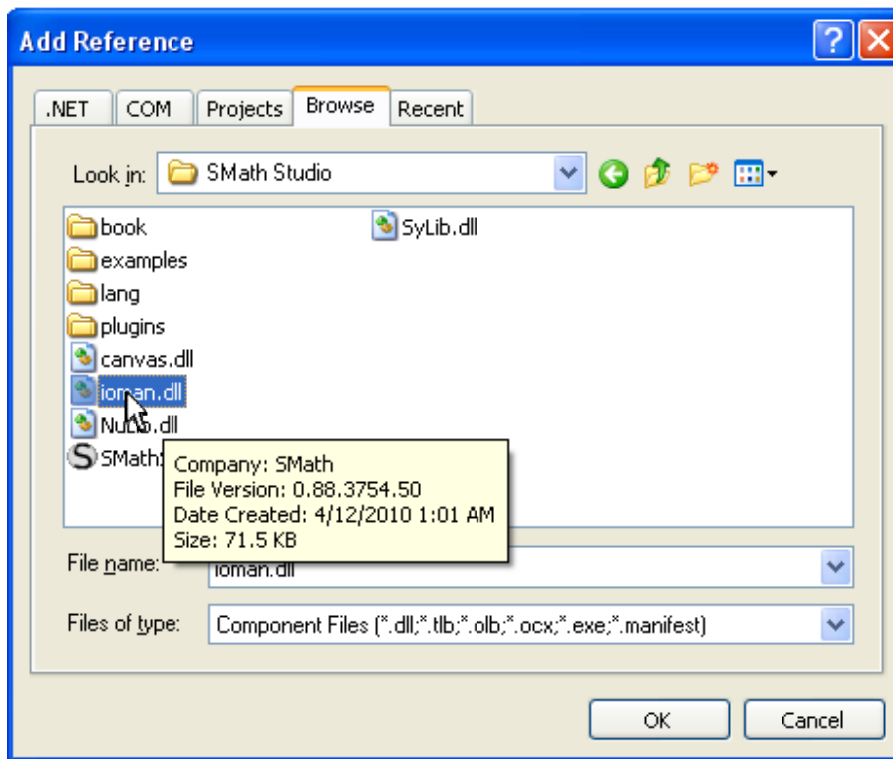
And then click on the “Browse” tab in the “Add Reference” dialog



12. Under this tab, browse to the Smath folder under Program Files.

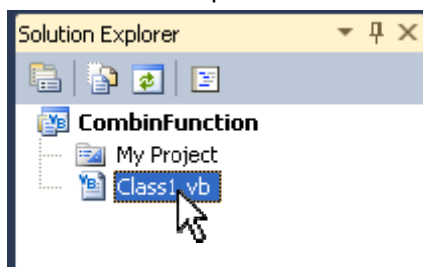


In the Smath folder, under Smath Studio, select the file “*ioman.dll*” and click OK.



13. Repeat steps 10 and 11 for the file “*canvas.dll*”

14. In the Solution Explorer double-click on “*Class1.vb*”



15. In the editing window, above the class definition, type in the following:

```
Imports Smath.Manager  
Imports Smath.Math
```

```
Imports Smath.Manager  
Imports Smath.Math  
  
Public Class Class1  
End Class
```

16. Within the class definition type the following:

*Implements IPluginHandleEvaluation*

```
Imports SMath.Manager  
Imports SMath.Math
```

```
Public Class Class1  
    Implements IPluginHandleEvaluation  
End Class
```

and press enter. This will automatically insert an interface (with the interface members) that must be implemented in the class. (see endnote 1)

17. Next, type in the following:

```
Dim termInfos() As TermInfo  
Dim asseblyInfos() As AssemblyInfo
```

```
Public Class Class1  
    Implements IPluginHandleEvaluation  
  
    Dim termInfos() As TermInfo  
    Dim asseblyInfos() As AssemblyInfo
```

18. Next scroll down the page and find the following subroutine:

```
Public Sub Initialize() Implements SMath.Manager.IPlugin.Initialize  
  
End Sub
```

and type in the lines seen below:

```
Public Sub Initialize() Implements SMath.Manager.IPlugin.Initialize  
    Me.termInfos = New TermInfo() {  
        New TermInfo("combin", TermType.Function, 2, "(n, k) - Returns the  
    }  
End Sub
```

The middle line (that is cut off) that starts with "New TermInfo(...)" has six arguments in parentheses separated by commas. The arguments that are to be inserted are:

- "combin"
- TermType.Function
- 2

- "(n, k) - Returns the number of subsets (combinations) of k elements that can be formed from n elements."
- FunctionSection.Unknown
- True

Don't forget a closing parentheses ")" after the last argument.

19. Next type in more lines of code as seen below:

```
Public Sub Initialize() Implements SMath.Manager.IPlugin.Initialize
    Me.termInfos = New TermInfo() {
        New TermInfo("combin", TermType.Function, 2, "(n, k) - Returns
    }
    Me.assemblyInfos = New AssemblyInfo() {
        New AssemblyInfo("Smath Studio", New Version(0, 88), New Guid(
    }
End Sub
```

The line (that is cut off and that starts with "New AssemblyInfo(...." ) has three arguments in parentheses separated by commas. The arguments that are to be inserted are:

- "Smath Studio"
- New Version(0, 88)

*[note: the number 88 represents the version number of Smath for which you are developing this plug-in. So if you are developing for Smath version 0.88, you insert 88. However, if the version you are targeting is different, enter the appropriate number.]*

- New Guid("a37cba83-b69c-4c71-9992-55ff666763bd")

Don't forget your closing parentheses.

20. Next scroll down to the following subroutine:

```
Public ReadOnly Property TermsHandled As SMath.Manager.TermInfo() Implements SMath.Manager.IPluginHandleEvaluation.TermsHandled
    Get
    End Get
End Property
```

and type in the following within the Get block (see note 2):

*Return Me.termInfos*



```

Public ReadOnly Property TermsHandled As SMath.Manager.TermInfo() Impl
    Get
        Return Me.termInfos
    End Get
End Property

```

21. Now scroll up to the following subroutine:

```

Public ReadOnly Property Dependencies As SMath.Manager.AssemblyInfo() Implements SMath.Manager.IPlugin.Dependencies
    Get
    End Get
End Property

```

and type in the following within the Get block:

*Return Me.asseblyInfos*

```

Public ReadOnly Property Dependencies As SMath.Manager.AssemblyInfo() I
    Get
        Return Me.asseblyInfos
    End Get
End Property

```

22. Next, scroll to the top of the page and find the Implements statement as shown below:

```

Public Class Class1
    Implements IPluginHandleEvaluation

    Dim termInfos() As TermInfo
    Dim asseblyInfos() As AssemblyInfo

```

After “*IPluginHandleEvaluation*”, insert a comma and type the following:

*IPluginLowLevelEvaluation*

```

Public Class Class1
    Implements IPluginHandleEvaluation, IPluginLowLevelEvaluation

```

and then press Enter.

23. Now scroll down and find the following function:

```

Public Function ExpressionEvaluation(ByVal root As SMath.Manager.Term
End Function
End Class

```

and type in the following conditional If statement:

```

Public Function ExpressionEvaluation(ByVal root As SMath.Manager.Term, ByVal args() As SMa
If root.Type = TermType.Function And root.Text = "combin" And root.ChildCount = 2 Then
End If
End Function
End Class

```

24. Now type in the following within the If block:

```

Public Function ExpressionEvaluation(ByVal root As SMath.Manager.Term
If root.Type = TermType.Function And root.Text = "combin" And root
Dim arg1 As Term() = Decision.Preprocessing(args(0), store)
Dim arg2 As Term() = Decision.Preprocessing(args(1), store)
End If
End Function
End Class

```

The preprocessing steps above are needed to correctly prepare the arguments. This means that all possible substitutions will be performed.

25. Next, type the following:

*Dim answer As New List(Of Term)*

```

Public Function ExpressionEvaluation(ByVal root As A
If root.Type = TermType.Function And root.Text
Dim arg1 As Term() = Decision.Preprocessi
Dim arg2 As Term() = Decision.Preprocessi
Dim answer As New List(Of Term)

```

26. Now, we need to compose an expression array formed in Reverse Polish notation (RPN) (see note 3). The mathematical expression:

$$\frac{n!}{k!(n-k)!}$$

is expressed in RPN as: `n ! k ! n k - ! * /`

Thus, type in the following lines to compose the list of terms in RPN:

```
Public Function ExpressionEvaluation(ByVal root As SMath.Manager.Term, ByVal args() As Term) As Boolean
    If root.Type = TermType.Function And root.Text = "combin" And root.ChildCount = 2 Then
        Dim arg1 As Term() = Decision.Preprocessing(args(0), store)
        Dim arg2 As Term() = Decision.Preprocessing(args(1), store)

        Dim answer As New List(Of Term)

        answer.AddRange(arg1) ← n
        answer.Add(New Term([Operator].Factorial, TermType.Operator, 1)) ← !
        answer.AddRange(arg2) ← k
        answer.Add(New Term([Operator].Factorial, TermType.Operator, 1)) ← !
        answer.AddRange(arg1) ← n
        answer.AddRange(arg2) ← k
        answer.Add(New Term([Operator].Subtraction, TermType.Operator, 2)) ← -
        answer.Add(New Term([Operator].Factorial, TermType.Operator, 1)) ← !
        answer.Add(New Term([Operator].Multiplication, TermType.Operator, 2)) ← *
        answer.Add(New Term([Operator].Division, TermType.Operator, 2)) ← /
    End If
End Function
```

27. To finish up the function type in the following at the location indicated:

```
result = answer.ToArray()
Return True
```

```
Public Function ExpressionEvaluation(ByVal root As SMath.Manager.Term, ByVal args() As Term) As Boolean
    If root.Type = TermType.Function And root.Text = "combin" And root.ChildCount = 2 Then
        Dim arg1 As Term() = Decision.Preprocessing(args(0), store)
        Dim arg2 As Term() = Decision.Preprocessing(args(1), store)

        Dim answer As New List(Of Term)

        answer.AddRange(arg1)
        answer.Add(New Term([Operator].Factorial, TermType.Operator, 1))
        answer.AddRange(arg2)
        answer.Add(New Term([Operator].Factorial, TermType.Operator, 1))
        answer.AddRange(arg1)
        answer.AddRange(arg2)
        answer.Add(New Term([Operator].Subtraction, TermType.Operator, 2))
        answer.Add(New Term([Operator].Factorial, TermType.Operator, 1))
        answer.Add(New Term([Operator].Multiplication, TermType.Operator, 2))
        answer.Add(New Term([Operator].Division, TermType.Operator, 2))

        result = answer.ToArray()
        Return True
    End If
End Function
```

and then insert the following line after the "End If" statement:

```
Return False
```

```

Public Function ExpressionEvaluation
    If root.Type = TermType.Function
        Dim arg1 As Term() = Decis:
        Dim arg2 As Term() = Decis:

        Dim answer As New List(Of

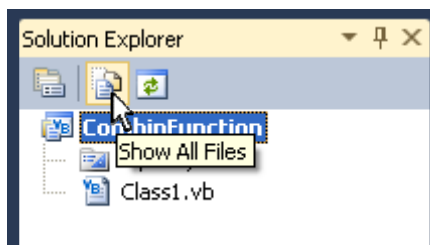
        answer.AddRange(arg1)
        answer.Add(New Term([Operat
        answer.AddRange(arg2)
        answer.Add(New Term([Operat
        answer.AddRange(arg1)
        answer.AddRange(arg2)
        answer.Add(New Term([Operat
        answer.Add(New Term([Operat
        answer.Add(New Term([Operat
        answer.Add(New Term([Operat

        result = answer.ToArray()
        Return True

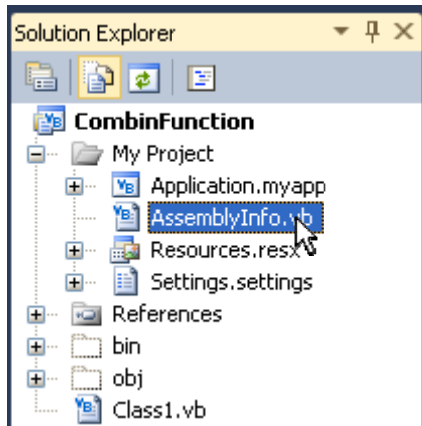
    End If
    Return False
End Function
End Class

```

28. Next, go to the Solution Explorer on the right side of the screen and click on the button “Show All Files”



Then expand the “My Project” directory and double-click on “AssemblyInfo.vb”:



29. Next, we will edit the assembly attributes in the “AssemblyInfo.vb” file within the editing window. Edit the values according to your needs. For this example, we will edit the attributes by substituting the strings in the boxes below:

```
' General Information about an assembly is controlled through the following
' set of attributes. Change these attribute values to modify the information
' associated with an assembly.

' Review the values of the assembly attributes

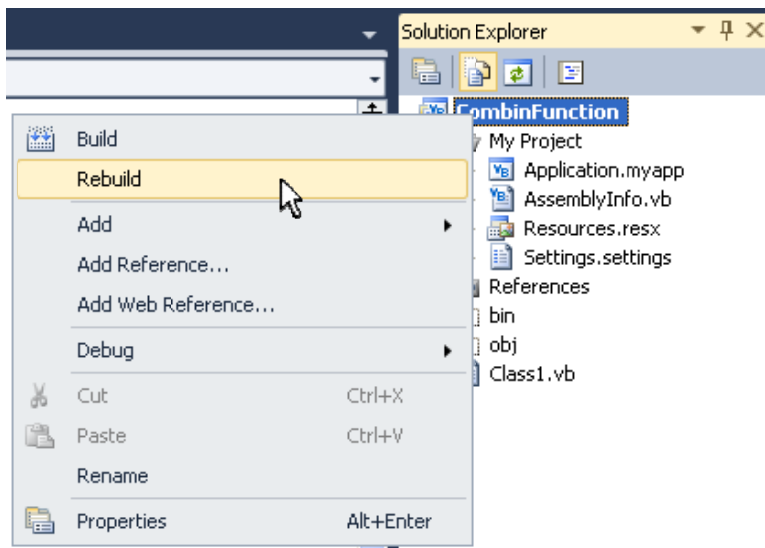
<Assembly: AssemblyTitle("Combinations Function")>
<Assembly: AssemblyDescription("Plugin with Combinations function realization.")>
<Assembly: AssemblyCompany("Andrey Ivashov")>
<Assembly: AssemblyProduct("Combinations Function")>
<Assembly: AssemblyCopyright("Copyright © SMath 2010")>
<Assembly: AssemblyTrademark("")>
```

30. Next, scroll down to the section dealing with built and revision numbers and substitute the strings shown in the boxes below:

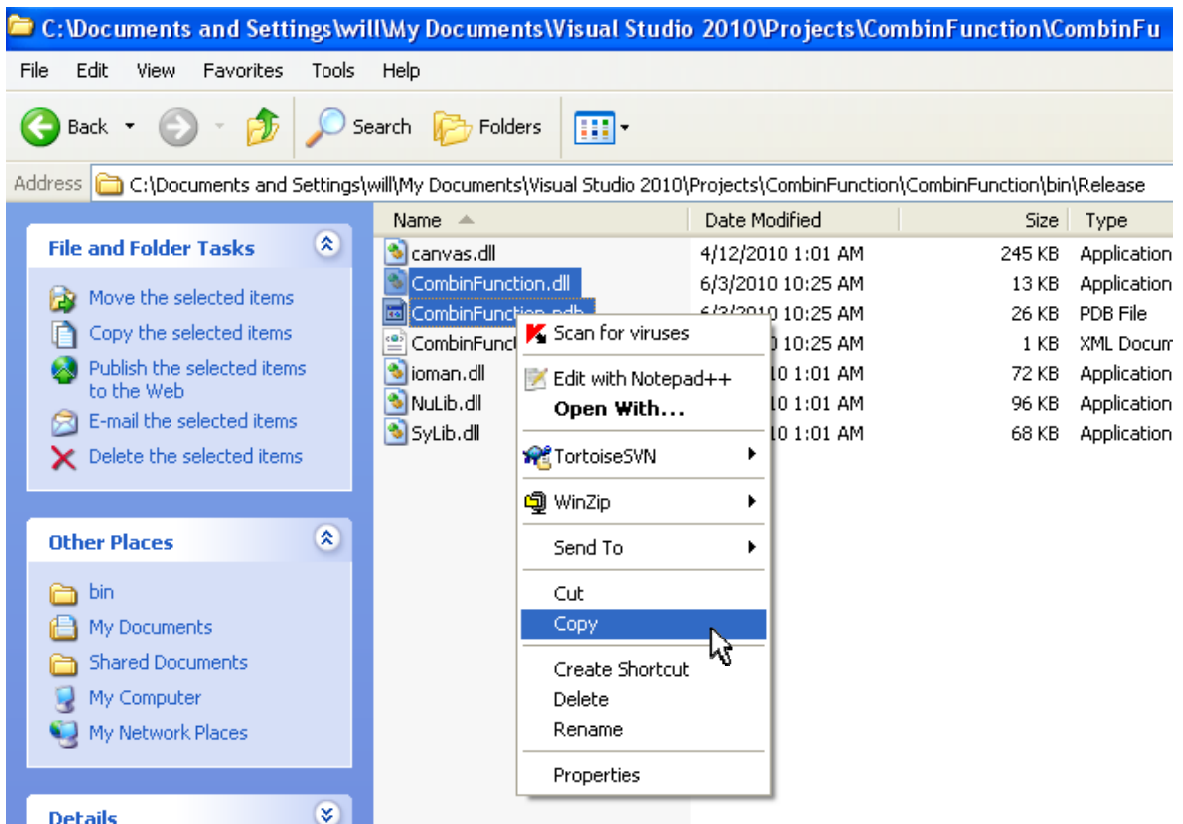
```
' Version information for an assembly consists of the following four values:
'
'     Major Version
'     Minor Version
'     Build Number
'     Revision
'
' You can specify all the values or you can default the Build and Revision Numbers
' by using the '*' as shown below:
' <Assembly: AssemblyVersion("1.0.*")>

<Assembly: AssemblyVersion("1.0.*")>
<Assembly: AssemblyFileVersion("1.0.*")>
```

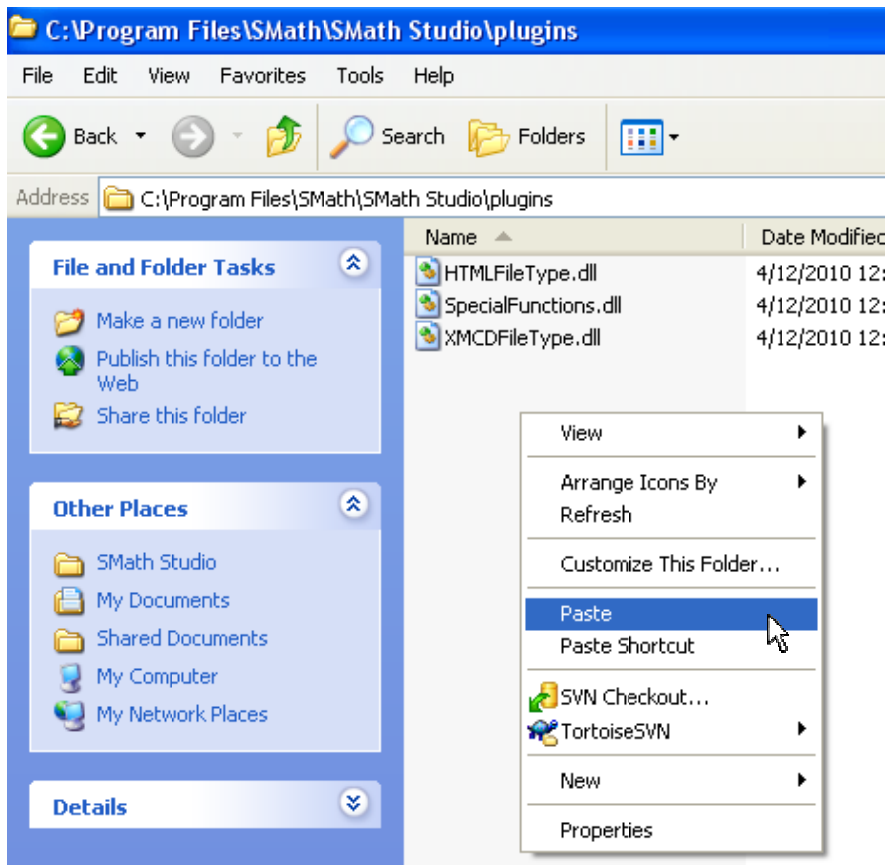
31. Now, go to the Solution Explorer on the right side of the screen and right-click on the “CombinFunction” name and click on “Rebuild”.



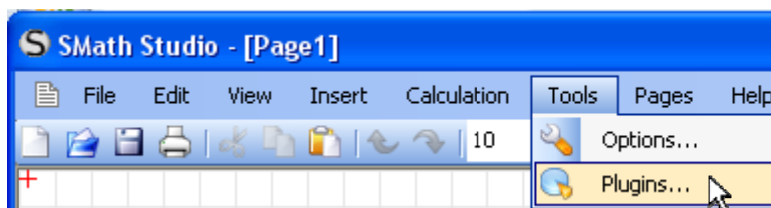
32. Next, using Windows Explorer, in your “My Documents” folder, navigate to “Visual Studio 2010\Projects\CombinFunction\CombinFunction\bin\Release” folder and copy the files *CombinFunction.dll* and *CombinFunction.pdb* that are found in the folder.



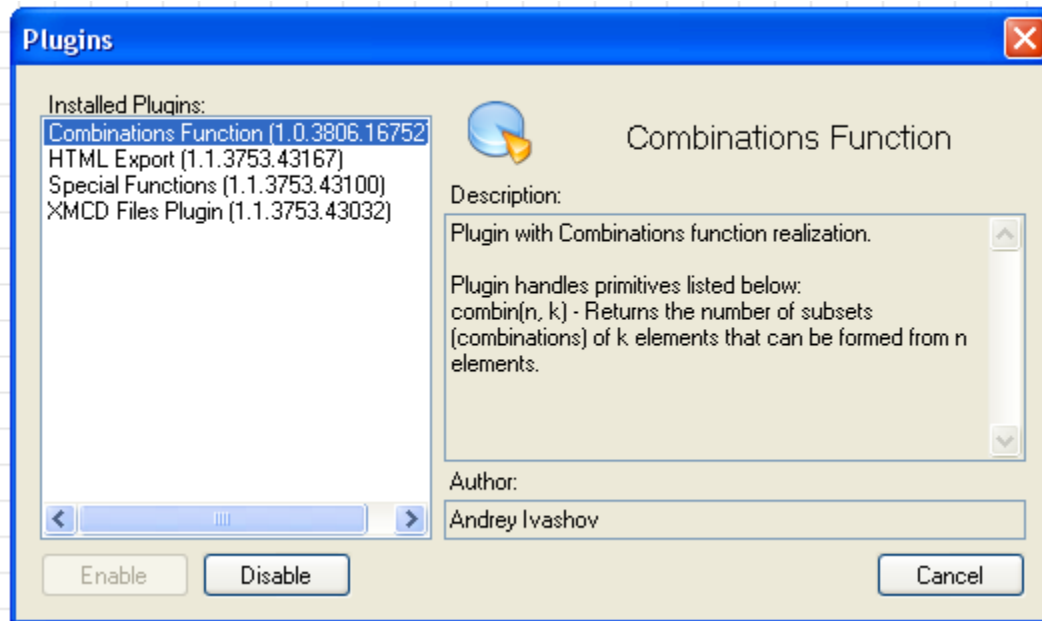
33. Now, using Windows Explorer, in your “Program Files” folder, navigate to “SMath\SMath Studio\Plugins” folder and paste the files into this folder.



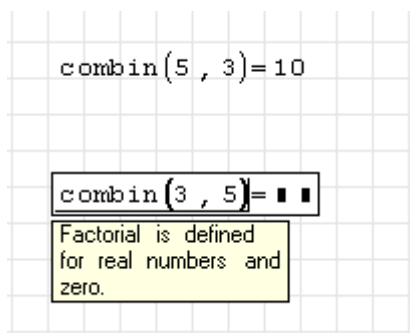
34. Now, let's test our new function. Run Smath Studio and click on *Tools* → *Plugins*



The plug-ins dialog should show our new plug-in along with a description of the plug-in as shown below:



35. You may now test the usage of the function in Smath as shown below:

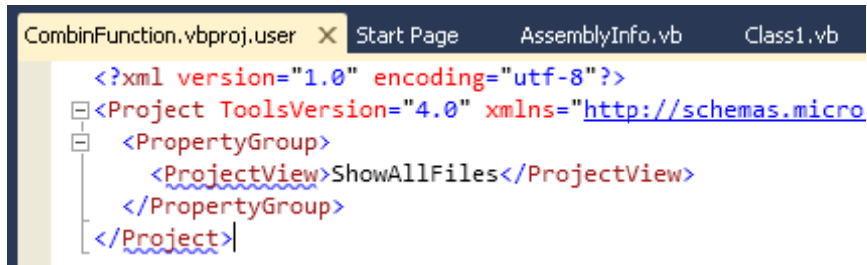


If you see results like this, you have successfully created your first plug-in!

36. However, in the real world, we seldom get by without making mistakes from time-to-time. Let's now show how to debug our plug-in. Typically, you would debug your application before doing steps 31 through 35 that were outlined above. Debugging an application add-in with Visual Studio Express appears to not be as straightforward as in the professional versions of Visual Studio. But below is a workaround that seems to work.

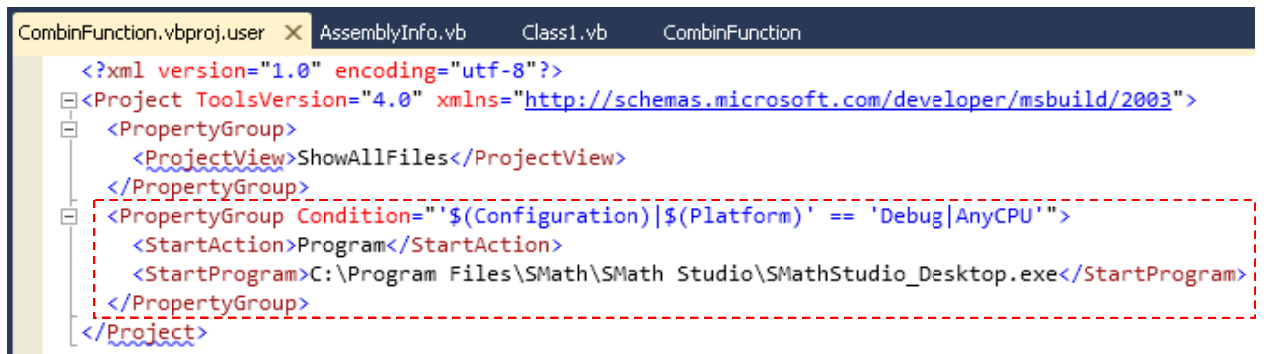
First, click on *File* → *Open* and navigate to the *My Documents* folder and then navigate to the "*Visual Studio 2010\Projects\CombinFunction\CombinFunction*" folder. Within this folder you should find a file called: *CombinFunction.vbproj.user*. (If the file does not exist, you may create it from scratch. It should be a simple text file with a name in the form of: *MyPluginName.vbproj.user*) Open the file by double-clicking. It should look something like this:





```
<?xml version="1.0" encoding="utf-8"?>
<Project ToolsVersion="4.0" xmlns="http://schemas.micro
  <PropertyGroup>
    <ProjectView>ShowAllFiles</ProjectView>
  </PropertyGroup>
</Project>
```

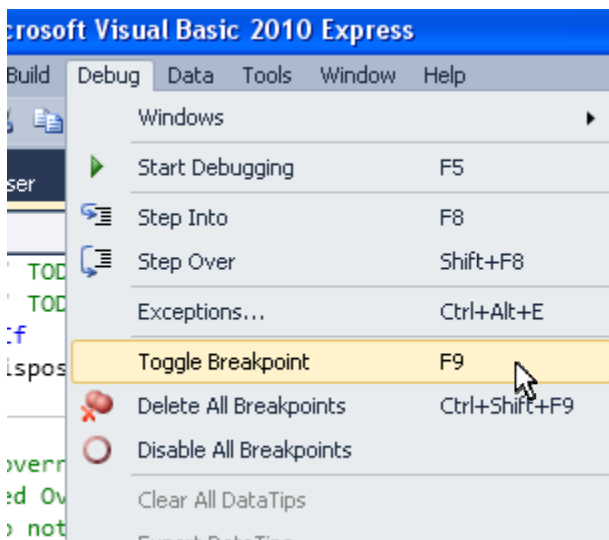
Edit the file by inserting the extra lines shown in the boxes below:



```
<?xml version="1.0" encoding="utf-8"?>
<Project ToolsVersion="4.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <PropertyGroup>
    <ProjectView>ShowAllFiles</ProjectView>
  </PropertyGroup>
  <PropertyGroup Condition="'$(Configuration)|$(Platform)' == 'Debug|AnyCPU'">
    <StartAction>Program</StartAction>
    <StartProgram>C:\Program Files\SMath\SMath Studio\SMathStudio_Desktop.exe</StartProgram>
  </PropertyGroup>
</Project>
```

In the line starting with “<StartProgram>....”, be sure to insert the path of the executable for Smath Studio Desktop for your system. If it was necessary to create the file from scratch, simply replicate what you see in the screenshot above. Finally save the file. It will be necessary to close and then reopen the Visual Studio project for the modified file to be detected and read.

37. Within Visual Basic Express, set a breakpoint at a convenient location. Simply place your cursor in the line at which you wish to set the breakpoint and click on *Debug* → *Toggle Breakpoint* as shown below:



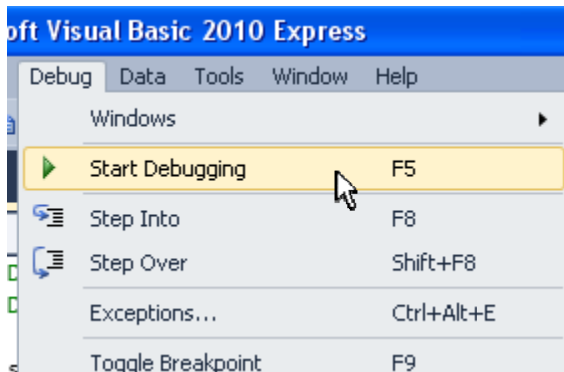
In this case, we will set it at the If statement within the ExpressionEvaluation function as shown in this screenshot:

```
Public Function ExpressionEvaluation(ByVal root As SMath.Manager.Term, ByVal args() As SM
If root.Type = TermType.Function And root.Text = "combin" And root.ChildCount = 2 Then
    Dim arg1 As Term() = Decision.Preprocessing(args(0), store)
    Dim arg2 As Term() = Decision.Preprocessing(args(1), store)

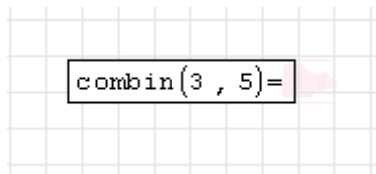
    'Decision.Preprocessing needed to correctly prepare the arguments.
    'This means that all possible substitutions will be performed.

    Dim answer As New List(Of Term)
```

38. Start debugging by clicking on *Debug* → *Start Debugging* as shown below.



When you do this, Visual Studio will automatically start up Smath Studio and pass the focus to Smath. When this occurs, you must attempt to utilize the plug-in you have created for the purpose of debugging it. In this case, we type in the following:



As soon as the "=" is entered, if a breakpoint was set, control and screen focus will return to Visual Studio where you can step through the code, watch variable values, and other debugging tasks.

39. Here are some useful links about how to debug your applications within Visual Studio, Video on using the debugger in Visual Basic:

<http://msdn.microsoft.com/en-us/vbasic/ee672313.aspx>

More information on debugging in Visual Studio may be found at:

<http://msdn.microsoft.com/en-us/library/k0k771bt%28v=VS.100%29.aspx>

Execution Control (stepping through your code):

<http://msdn.microsoft.com/en-us/library/y740d9d3%28v=VS.100%29.aspx>

Breakpoint Overview:

<http://msdn.microsoft.com/en-us/library/5557y8b4%28v=VS.100%29.aspx>

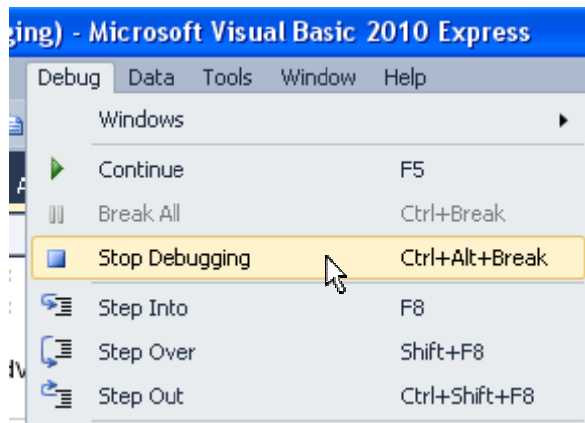
Viewing Data in the Debugger:

<http://msdn.microsoft.com/en-us/library/esta7c62%28v=VS.100%29.aspx>

Edit and Continue:

<http://msdn.microsoft.com/en-us/library/bcew296c%28v=VS.100%29.aspx>

40. To stop debugging, click on *Debug* → *Stop Debugging* as shown below:



When you do this, the instance of *Smath* in which you tested your plug-in will close.

41. Finally, when your plug-in is finished and bug free, you are ready to release your plug-in. This essentially involves repeating steps 31, 32, and 33 outlined above. The only difference is that the *CombinFunction.pdb* file does not need to be copied into the "*SMath\SMath Studio\Plugins*" folder.

Hopefully, this tutorial was enough to get you started making your own plug-ins.

Endnotes:

1. Refer to: <http://msdn.microsoft.com/en-us/library/7z6hzchx%28v=VS.90%29.aspx>
2. Refer to: <http://msdn.microsoft.com/en-us/library/38x6w70d%28v=VS.90%29.aspx>
3. For explanation of Reverse Polish notation refer to:  
[http://en.wikipedia.org/wiki/Reverse\\_polish\\_notation](http://en.wikipedia.org/wiki/Reverse_polish_notation)