

Analyseur d'expressions mathématiques (*Math Parser*)

(J-C Armici)

Ce document est basé sur un article paru sur www.codeproject.com écrit par M. Cuprjak.

Un analyseur d'expressions mathématiques permet, entre autres, à l'utilisateur d'introduire une expression au moment de l'exécution. L'expression peut ensuite être évaluée pour différentes valeurs des variables. L'analyse et l'utilisation d'une expression mathématique était un procédé jusque là peu performant, car l'expression elle-même n'est pas compilée. Le programme devait faire une analyse lexicale

Profitant du fait qu'il est possible en .NET de lancer la compilation d'un morceau de code source et de l'utiliser immédiatement, la technique montrée ici est très simple. Cependant l'expression doit satisfaire la syntaxe C# en l'occurrence.

L'exemple qui suit évalue une fonction de la forme:

$$y = f(x)$$

c'est-à-dire une fonction à une variable. Pour cela il suffit de travailler avec x et ignorer y. On peut facilement étendre le procédé afin d'analyser des fonctions à plusieurs variables.

Voici ensuite la classe constituant l'analyseur proprement dit:

```
using System;
using System;
using System.Reflection;
using System.Windows.Forms;
using System.Collections;
using System.CodeDom.Compiler;

namespace Parser
{
    public class MathExpressionAnalyseur
    {
        public object myobj = null;
        public ArrayList errorMessages;

        public MathExpressionAnalyseur()
        {
            errorMessages = new ArrayList();
        }
        public bool init(string expr)
        {
            CodeDomProvider cp = CodeDomProvider.CreateProvider("C#");
            CompilerParameters cpar = new CompilerParameters();
            cpar.GenerateInMemory = true;
            cpar.GenerateExecutable = false;
            cpar.ReferencedAssemblies.Add("system.dll");
            CheckExpression(ref expr);
            string src = "using System;" + // construction de la classe
                "class myclass" +
                "{" +
                "public myclass(){}" +
                "public static double eval(double x)" +
                "{" +
                "return " + expr + ";" +
                "}" +
                "};";
            CompilerResults cr = cp.CompileAssemblyFromSource(cpar, src);
            foreach (CompilerError ce in cr.Errors) // ajout des erreurs éventuelles
                errorMessages.Add(ce.ErrorText);

            if (cr.Errors.Count == 0 && cr.CompiledAssembly != null)
            {
                Type ObjType = cr.CompiledAssembly.GetType("myclass");
                try
            }
        }
    }
}
```

```

        {
            if (ObjType != null)
                myobj = Activator.CreateInstance(ObjType);
        }
        catch (Exception ex)
        {
            errorMessages.Add(ex.Message);
        }
        return true;
    }
    else
        return false;
}

private void CheckExpression(ref string expr)
{
    // remplacements simple pour la syntaxe C#
    expr = expr.Replace("ln", "Math.Log");
    expr = expr.Replace("exp", "Math.Exp");
    expr = expr.Replace("sqrt", "Math.Sqrt");
    expr = expr.Replace("sin", "Math.Sin");
    expr = expr.Replace("cos", "Math.Cos");
    expr = expr.Replace("pi", "Math.PI");
}

public double eval (double x)
{
    double val = 0.0;
    Object[] myParams = new object[1] { x };
    if (myobj != null)
    {
        MethodInfo evalMethod = myobj.GetType ().GetMethod ("eval");
        val = (double)evalMethod.Invoke (myobj, myParams);
    }
    return val;
}
}
}

```

Comme on peut le voir la classe **myclass** est générée, compilée et instanciée.

On peut alors utiliser notre analyseur, par exemple de la manière suivante:

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

namespace Parser
{
    public delegate double UnaryFunction(double x);

    public partial class Form1 : Form
    {
        public Form1 ()
        {
            InitializeComponent ();
        }

        private void button1_Click (object sender, EventArgs e)
        {
            bool OKparse = true;
            MathExpressionAnalyseur mp = new MathExpressionAnalyseur();
            try
            {

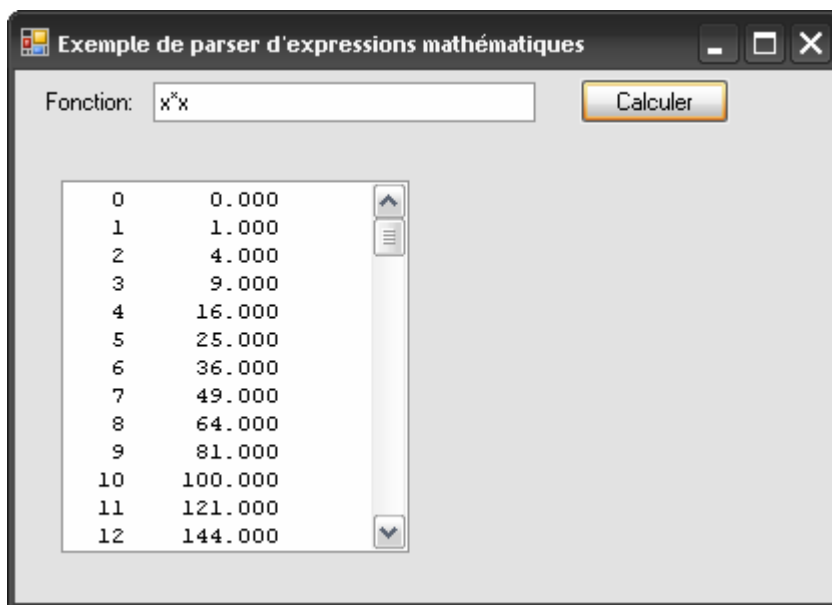
```

```

    OKparse= mp.init(txtFct.Text);
}
catch
{
    MessageBox.Show("Problème dans l'expression");
    return;
}
if (OKparse)
{
    listBox1.Items.Clear();
    for (int i = 0; i < 100; i++)
        listBox1.Items.Add(string.Format("{0,4}    {1,6:#0.000}",
                                           i, mp.eval((double)i)));
}
else
    MessageBox.Show("Problème dans l'expression", "Erreur", MessageBoxButtons.OK);
}
}
}

```

Ce programme a l'aspect suivant:



En combinant cet analyseur d'expressions mathématiques avec un composant de tracée de graphiques comme Zedgraph on peut obtenir des résultats un peu moins sobres:

