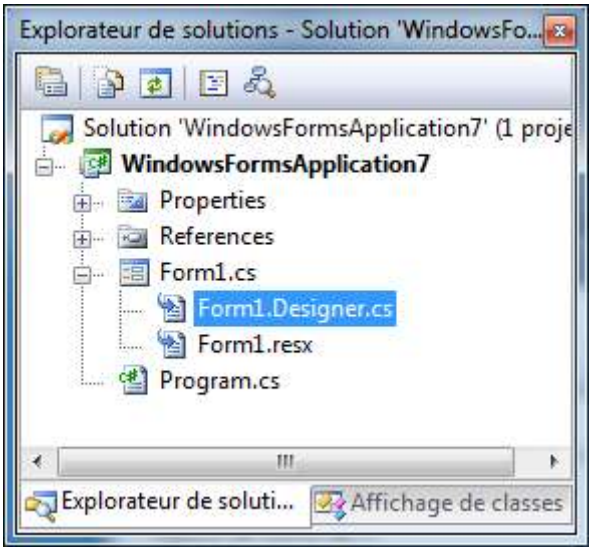
TP Informatique 2 – 4h
C# les types élémentaires et Algorithmique

Pré requis : savoir utiliser Visual studio … un peu !
But :
Consolider la maîtrise de visual
Voir les types en C# (par rapport au C++)
Algorithmique de base

Quelques bases en C#

Lorsqu'on génère une application Windows Form , on a :

- Un onglet Form1.cs [Design] : c'est le designer, le concepteur graphique qui permet de concevoir l'interface graphique de l'application et génère le code.
- Un onglet Form1.cs* : c'est le fichier de code C#, il est en partie généré automatiquement, mais c'est là que le programmeur doit ajouter son code.
- Dans l'explorateur de solution on voit que Form1.cs est aussi composé de Form1.Designer.cs : c'est le code généré par le designer. Allez y jeter un œil, et développez la partie Code généré par le concepteur graphique, car on voit le code pour utiliser les objets graphiques (parfois çà aide !)



Le code de Form1.cs

```
using System;
using System.Collections.Generic;    //les using, ce sont des fichiers qui
using System.ComponentModel;         // qui contiennent des classes
using System.Data;                   // qu'on utilise
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace WindowsFormsApplication7
{
    public partial class Form1 : Form  //vous êtes en train d'écrire
```

```
                                                    // une nouvelle classe
    {
        public Form1()
        {
            InitializeComponent(); // appel de la methode dans Form1.Designer.cs
        }

        private void button1_Click(object sender, EventArgs e)
        {
// ici je met du code de gestion de click button…
        }
    }
}
```

On a vu que les données sont contenues dans des variables. Une variable est une zone en mémoire, on lui donne un nom, on définit son type (la nature de son contenu), et on peut ensuite y stocker des données.

types élémentaires du C# :

nature type	**Nom en C#**	Équivalent .NET	Équivalent C	Explications
Byte / Octet	**byte**	System.Byte	char	Voici le plus petit type disponible pour C#, il peut contenir des valeurs allant de -127 à 128.
Integer 16bits / Entier 16bits	**short**	System.Int16	short	Peut contenir des valeurs allant de -32 768 à 32 767.
Integer 32bits / Entier 32bits	**int**	System.Int32	int ou long	Peut contenir des valeurs allant de -2 147 483 648 à 2 147 483 649.
Integer 64bits / Entier 64bits	**long**	System.Int64	long long	Peut contenir des valeurs allant de -9 223 372 036 854 775 808 à 9 223 372 036 854 775 807.
Character / Caractère	**char**	System.Char	wchar_t	Représente un caractère au format Unicode(2 octets).
Boolean / Booléen	**bool**	System.Boolean	bool (C++)	Une variable de type booléen ne peut contenir que "true" ou "false".
Double / Double	**double**	System.Double	double	Contient un nombre à virgule de double précision allant de -1,79769313486232e308 à 1,79769313486232e308.

Beaucoup de ressemblances C# et C++, attention au byte, char !!
Attention aux divisions avec des entiers : on ne garde que la partie entière
Ex :

```
{
Int oct,oct2=3 ;  //on déclare des variables oct, oct2 de type byte on met 3 dans oct2
oct=oct2 /2 ;    // un calcul … 3/2 = 1!!! Car c'est un entier}
}
```

Eléments pour l'algorithmique

- La séquence : ce sont simplement les instructions les unes derrières les autres.
- La conditionnelle :

```
int v;
          if (v < 0)
          {
              // v negatif
          }
          else
          {
              // v positif
          }
```

- La répétitive, dans un premier temps on se contente du tant que :

```
while (v != 0)
          {
              //on repete
          }
```

Pour les tests :
Egal (==), différent (!=) , inférieur (<) , inférieur ou égal (<=) , etc…

Gestion des exception
On a vu que dans certains cas, l'appel d'une fonction génère une exception qui « plante » le programme. Exemple : Convert.ToDouble(v) génère une exception si v contient une lettre.
Le « plantage » vient du fait qu'on n'a pas géré l'exception.
On va utiliser un try … catch .

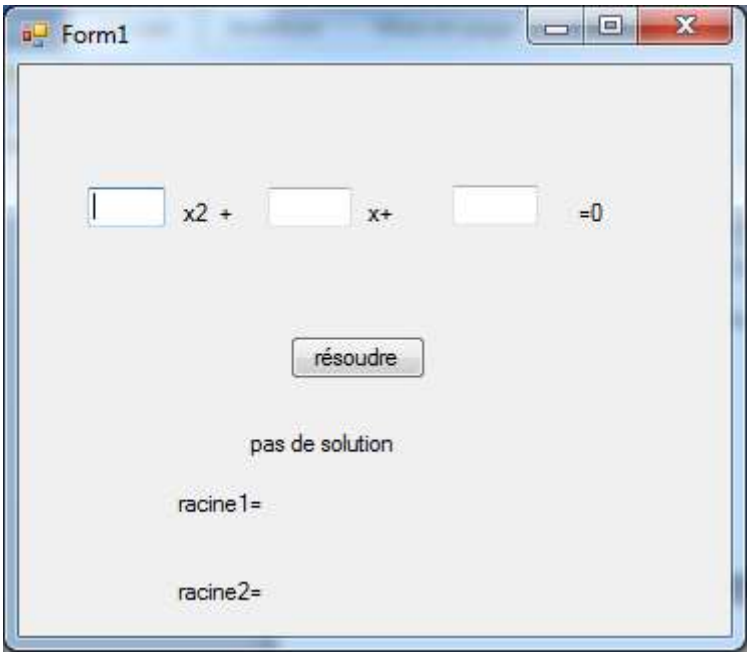
```
double r,v;
            try
            {
                r = System.Convert.ToDouble(this.textBox1.Text);
                v = r * 55;
                this.textBox2.Text = v.ToString();
            }
            catch (Exception err)
            {
                MessageBox.Show("erreur");
            }
```

On voit ici que le programme normal essaie de convertir en double.
Si tout se passe bien, on continue avec v=r*55 et on affiche dans textbox2.
Mais en cas d'erreur, l'exception est attrapée, la conversion n'a pas eu lieu , le programme est dérouté dans le catch , pour traiter l'erreur (ici on affiche un message). Et on sort du catch.

Manipulation.

1°) reprendre le programme de conversion euro/dollar et jouter une gestion des erreurs :
En cas de mauvaise donnée on affiche un message d'erreur, et on met à 0 le Textbox incriminé.

2°) On veut écrire un programme de résolution d'équation du 2ème degrè.



Avec gestion des erreurs en cas de mauvaise saisie.

3°) reprendre le 2°) avec les nombres complexes.

4°) Ecrire une application qui affiche un byte et un bouton qui permet d'ajouter 10 à chaque clic. Faites une trentaine de clic et observez et essayez de justifier.

Critères évaluation :
Autonomie, respect consigne, savoir sauvegarder : 4 pts
Try catch de la conversion : 4pts
Programme équation réel : 6pts
Gestion du cas complexe : 4pts
Débordement du byte : 2pts