

Secteur Tertiaire Informatique
Filière Etude - Développement.

Activité Développer des composants d'interface
Etape Programmer des formulaires et des états
Séance Tester le composant à partir du jeu d'essai

Déboguer un programme C# avec Visual Studio

Accueil

Apprentissage

Période en
entreprise

Evaluation



CODE BARRE

SOMMAIRE

I	Introduction	4
II	Fonctions du débogueur	4
III	Mise en œuvre du débogueur.....	4
III.1	Le projet Triangle.....	4
III.2	Mode Debug / Mode Release	5
III.2.1	Compilation en mode Release.....	5
III.2.2	Compilation en mode Debug	5
III.3	Mettre en place un point d'arrêt	6
III.3.1	Où placer le point d'arrêt ?	6
III.3.2	Placer un point d'arrêt.....	6
III.4	Suivre au pas à pas	7
III.5	Tracer la valeur d'une variable	8
III.6	Tracer la pile des appels.....	9

I INTRODUCTION

Microsoft Visual Studio est un IDE (Integrated Développement Environment). De fait il intègre toutes les outils nécessaires au développeur, parmi ceux-ci le débogueur.

Le débogueur est un outil puissant qui permet de tester un programme et y traquer tous les dysfonctionnements probables que génère l'imperfection humaine à laquelle le meilleur développeur est soumis.

II FONCTIONS DU DEBOGUEUR

Les fonctions les plus importantes que l'on peut attendre d'un débogueur sont les suivantes :

- Poser un point d'arrêt – Il s'agit d'arrêter le programme à un endroit précis du code ou sur une condition déterminée (valeur critique d'une variable par exemple), pour en contrôler l'exécution.
- Suivre au pas à pas – Il s'agit de contrôler l'exécution du programme, instruction par instruction pour vérifier si le fonctionnement est conforme à celui attendu.
- Tracer la valeur des variables – Certaines valeurs critiques des variables peuvent entraîner des dysfonctionnements dans le programme
- Tracer la pile des retours – Un programme est composé d'un nombre important de sous programmes et de fonctions qui s'invoquent l'un, l'autre. La pile des retours permet de visualiser l'imbrication de ces invocations.

III MISE EN ŒUVRE DU DEBOGUEUR

III.1 LE PROJET TRIANGLE

Pour vous aider à la mise en œuvre du débogueur, un programme écrit en C# vous est fourni. Celui-ci permet d'évaluer les caractéristiques d'un triangle à partir des longueurs des trois cotés saisies au clavier.

Le code de ce programme est fourni dans le fichier compressé **Triangle_CSharp.zip**. Ce fichier doit être décompressé dans le répertoire où se trouvent les projets de développement gérés par Visual Studio – en principe **Mes documents\Visual Studio 2005\Projects**.

Pour ouvrir Visual Studio pour le projet **Triangle**, double-cliquez le fichier **Triangle.sln**.

Le projet **Triangle** contient 3 fichiers de code :

- **Program.cs** – Ce fichier contient notamment la fonction **Main** qui constitue le point d'entrée du programme. La première instruction de cette fonction est la première qui sera exécutée au lancement du programme.

- **Form1.cs** – Ce fichier contient le code de la boîte de dialogue, notamment la fonction associée à l'événement CLICK du bouton **Valider** (celle qui est exécutée lorsque l'utilisateur du programme clique le bouton **Valider** avec la souris) : **buttonValider_Click**.
- **Triangle.cs** – Ce fichier contient le code du triangle proprement dit et notamment un certain nombre de fonctions permettant d'évaluer les caractéristiques du triangle (équilatéral, isocèle, rectangle, etc.) en fonctions de la longueur des trois cotés qui le composent.

III.2 MODE DEBUG / MODE RELEASE

Visual Studio permet d'optimiser le code généré à la compilation pour qu'il puisse être exécuté plus rapidement et qu'il prenne moins de place en mémoire et sur le disque dur.

Malheureusement ce code optimisé ne permet plus de faire une relation univoque entre les instructions C# rédigées par le développeur et le code binaire qui est généré. Le débogage est alors impossible.

C'est pourquoi Visual Studio propose deux modes :

- Le mode **Release** qui contient le code optimisé tel qu'il devra être fourni à l'exploitation.
- Le mode **Debug** qui contient un code plus lourd, mais qui permet de déboguer le programme.

Remarque : Le mode **Release** et **Debug** sont les deux modes intégrés par défaut à la création du projet. Visual Studio permet aux développeurs de créer et gérer des modes supplémentaires pour prendre en compte des environnements techniques différents (32 ou 64 bits, type de processeurs, etc.).

III.2.1 Compilation en mode Release

Pour passer en mode **Release** cliquer la commande de menu **Générer/Gestionnaire de Configuration**, puis choisissez le mode **Release** dans la combo (par défaut, c'est le mode **Debug** qui est activé).

Générez le code de la version **Release** en cliquant la commande **Générer/Générer la solution** (touche **F6**). Le programme généré **Triangle.exe** se trouve dans le répertoire **Triangle/bin/Release** du projet.

III.2.2 Compilation en mode Debug

Pour passer en mode **Debug** cliquer la commande de menu **Générer/Gestionnaire de Configuration**, puis choisissez le mode **Debug** dans la combo.

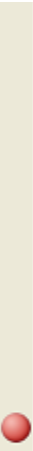
Générez le code de la version **Debug** en cliquant la commande **Générer/Générer la solution** (touche **F6**). Le programme généré **Triangle.exe** se trouve dans le répertoire **Triangle/bin/Debug** du projet.

III.3 METTRE EN PLACE UN POINT D'ARRÊT

III.3.1 Où placer le point d'arrêt ?

De façon générale, on place le point d'arrêt sur l'instruction que l'on suspecte de générer le dysfonctionnement. Cependant, certains bogues sont redoutables à trouver.

Le bon sens voudrait que l'on place le point d'arrêt sur la première instruction du programme. Dans le cas du programme Triangle, il s'agit de l'instruction ci-dessous :



```
using System.Windows.Forms;

namespace Triangle
{
    static class Program
    {
        /// <summary>
        /// Point d'entrée principal de l'application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new FormTriangle());
        }
    }
}
```

Cependant, mettre un point d'arrêt sur cette instruction du programme, puis le suivre au pas à pas va durer indéfiniment... sans jamais aboutir à l'instruction qui a causé le bug. Il suffit d'essayer pour s'en convaincre.

En effet, une application Windows est constituée d'une multitude de petits programmes, appelés *handlers*, dont l'exécution est déclenchée par la réception d'événements.

Dans le cas du programme **Triangle**, le programme à tester est le *handler* associé à l'événement CLICK du bouton **Valider**. Il suffit de placer le point d'arrêt sur la première instruction de la fonction **buttonValider_Click** dans le fichier **Form1.cs**.

Remarque : Le programme s'arrête AVANT que l'instruction sur laquelle est positionné le point d'arrêt soit exécutée.

III.3.2 Placer un point d'arrêt.

Pour placer un point d'arrêt, il suffit de cliquer sur la marge de gauche, en face de l'instruction sur laquelle on veut s'arrêter. Le point d'arrêt est alors matérialisé par un cercle rouge dans la marge. L'instruction est elle-même surlignée en rouge comme le montre la figure ci-dessous.

```

using System.Windows.Forms;

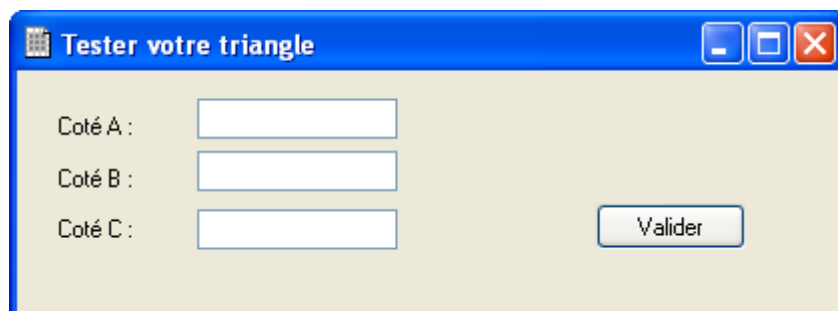
namespace Triangle
{
    public partial class FormTriangle : Form
    {
        public FormTriangle()
        {
            InitializeComponent();
        }

        private void FormTriangle_Load(object sender, EventArgs e)
        {
        }

        private void buttonValider_Click(object sender, EventArgs e)
        {
            Triangle t = this.construireTriangle();
            if (t != null && !t.IsTriangle())
            {
                afficheMessage("ce n'est pas un triangle", false);
            }
            else if (t != null && t.IsTrianglePlatIsocele())
            {
            }
        }
    }
}

```

Lancez le programme (touche F5). La fenêtre du programme s'affiche.



Puis cliquez le bouton **Valider**. Le programme s'arrête sur l'instruction désignée.

```

private void buttonValider_Click(object sender,
{
    Triangle t = this.construireTriangle();
    if (t != null && !t.IsTriangle())
    {
        afficheMessage("ce n'est pas un triangle
    }
}

```

III.4 SUIVRE AU PAS A PAS

Lorsque que le programme est arrêté sur un point d'arrêt, il est possible d'en contrôler l'exécution au pas à pas, c'est-à-dire instruction par instruction.

Certaines instructions – et c’est le cas de l’instruction sur laquelle le programme s’était arrêté – sont elles-mêmes des invocations à des programmes plus complexe. Le débogueur permet trois façons d’effectuer le pas à pas :

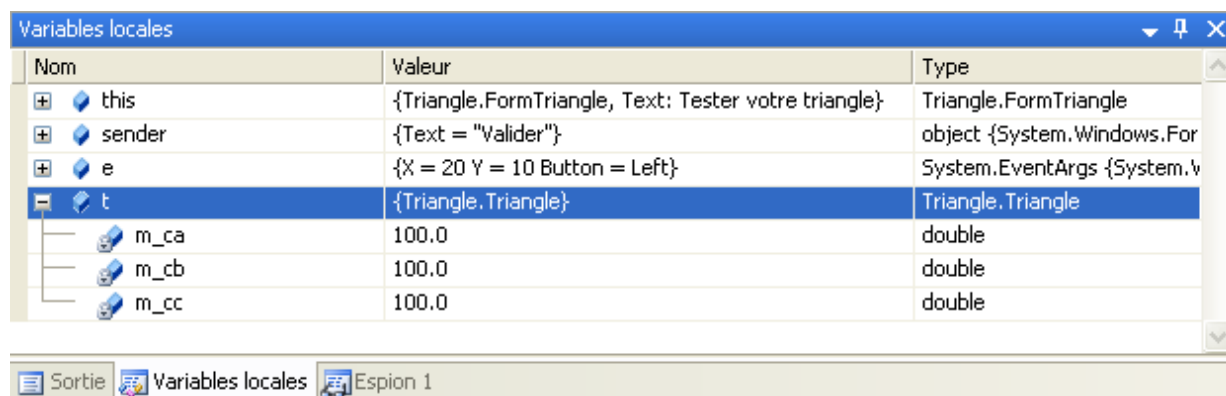
- **Pas à pas principal** (touche F10) – L’instruction pointée est exécuté en totalité, même si celle-ci correspond à l’invocation d’un sous-programme. Le programme s’arrête (si aucun bogue n’a été décelé dans le programme appelé) sur l’instruction qui suit.
- **Pas à pas détaillé** (touche F11) – Si l’instruction correspond à l’invocation d’un sous-programme, le programme s’arrête sur la première instruction de ce sous programme. Si l’instruction est une instruction élémentaire, celle –ci est exécutée et le programme s’arrête à l’instruction suivante.
- **Pas à pas sortant** (touches Maj + F11)– L’exécution du programme reprend pour s’arrêter sur l’instruction qui suit l’appel du sous-programme dans lequel on s’était arrêté auparavant.

Parcourez le *handler* **buttonValider_Click** jusqu’à la dernière instruction en utilisant les touches fonctions décrites ci-dessus.

III.5 TRACER LA VALEUR D’UNE VARIABLE

L’un des intérêts de parcourir un programme pas à pas est de pouvoir contrôler la valeur des différentes variables instruction par instruction.

Par défaut, Visual Studio propose une fenêtre intitulée « **Variables locales** » qui permet de tracer la valeur de toutes les variables en cours d’utilisation par le programme ou le sous-programme dans lequel on s’est arrêté.



Certaines variables peuvent être, elles-mêmes, une agrégation de données composites. C’est le cas, par exemple, de la variable **t**, qui est un **Triangle** composé de trois valeurs de type **double** (nombre virgule flottante) correspondant à la longueur de chacun des cotés du triangle. Un petit **[+]** permet de développer l’affichage des données composites.

Comme le nombre de ces variables peut être important, Visual Studio permet de tracer uniquement les variables souhaitées dans la fenêtre intitulée « **Espion 1** ». Pour ajouter une variable à la liste des variables à surveiller, il suffit de taper son nom dans la colonne **nom** de la fenêtre, ou bien de sélectionner une variable dans le code

et de cliquer la commande **Déboguer/Espion Express** puis le bouton **Ajouter un espion**.

Nom	Valeur	Type
t	{Triangle.Triangle}	Triangle.Triangle
m_ca	100.0	double
m_cb	100.0	double
m_cc	100.0	double

Sortie Variables locales Espion 1

III.6 TRACER LA PILE DES APPELS

Lors de l'exécution d'un programme, il se peut que l'imbrication des appels de fonctions et de sous-programmes soit importante. Ce qui peut rendre difficile le repérage de la source d'un bogue.

Visual Studio propose une fenêtre intitulée Pile des appels permettant d'afficher l'imbrication des invocations successives des fonctions entre elles.

```

public Triangle(double ca, double cb, double cc)
{
    this.m_ca = ca;
    this.m_cb = cb;
    this.m_cc = cc;
}

```

Par exemple, lorsque le programme est arrêté comme le montre la figure ci-dessus, il peut être intéressant de savoir qui a invoqué cette fonction qui permet de construire un triangle. La pile des appels permet de montrer que le constructeur du triangle a été appelé par la fonction **construireTriangle**, elle-même appelée par le *handler* **buttonValider_Click**.

Nom	Langag
Triangle.exe!Triangle.Triangle.Triangle(double ca = 100.0, double cb = 100.0, double cc = 100.0) Ligne 15	C#
Triangle.exe!Triangle.FormTriangle.construireTriangle() Ligne 68 + 0x8e octets	C#
Triangle.exe!Triangle.FormTriangle.buttonValider_Click(object sender = {Text = "Valider"}, System.EventArgs e = {X = 53 Y = 11 Button = ...})	C#
[Code externe]	
Triangle.exe!Triangle.Program.Main() Ligne 19	C#
[Code externe]	

Pile des appels Fenêtre Exécution

Etablissement référent
AFPA-DI-Département Tertiaire

Equipe de conception
LIMOUZIN André-Pierre

Remerciements :

Reproduction interdite

Article L 122-4 du code de la propriété intellectuelle.

«toute représentation ou reproduction intégrale ou partielle faite sans le consentement de l'auteur ou de ses ayants droits ou ayants cause est illicite. Il en est de même pour la traduction, l'adaptation ou la reproduction par un art ou un procédé quelconques.»

**afpa / Direction de l'Ingénierie 13 place du Générale de Gaulle / 93108 Montreuil
Cedex
association nationale pour la formation professionnelle des
adultes
Ministère des Affaires sociales du Travail et de la
Solidarité**

Date de mise à jour jj/mm/aa
afpa © Date de dépôt légal mois année



**afpa / Direction de l'Ingénierie 13 place du Générale de Gaulle / 93108 Montreuil
Cedex
association nationale pour la formation professionnelle des
adultes
Ministère des Affaires sociales du Travail et de la
Solidarité**