Découverte de l'IDE et premier exemple

Découverte du code et de l'environnement de développement :

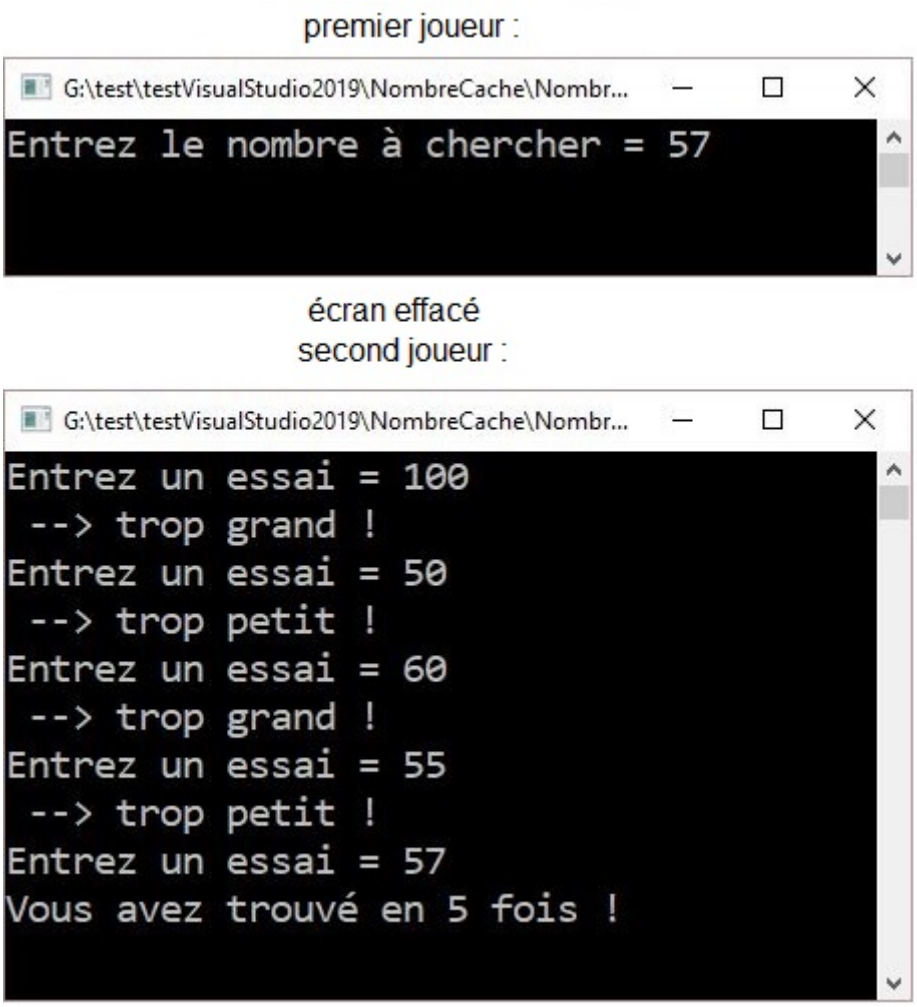
Vous allez coder une petite application qui va vous permettre de découvrir un peu mieux l'environnement de développement et les instructions de base de la programmation en langage C# (C Sharp).

But du premier exemple :

Le jeu du nombre caché se joue à deux joueurs.

- Le premier joueur saisit un nombre puis sa saisie est effacée.
- Le second joueur va ensuite tenter de trouver le nombre en saisissant des essais.
- A chaque essai le message **trop grand** ou **trop petit** est affiché.
- Dès que la valeur est trouvée, le nombre d'essais est affiché.

Les captures ci-contre montrent un exemple de fonctionnement.



Etape 1

Saisie de la valeur à chercher

La première étape de votre programme consiste à demander au premier joueur de saisir une valeur et de mémoriser cette valeur.

Vous savez déjà afficher un message à l'écran et attendre une saisie. Commencez à écrire dans le 'main' :

```
Console.Write("Entrez le nombre à chercher = ");
```

```
Console.ReadLine();
```

test

Faites un test d'exécution. Remarquez que l'affichage se fait mais le curseur se positionne non pas à la suite mais sous la ligne affichée.

Pour changer cela, il suffit d'utiliser **Write** à la place de **WriteLine**.

Faites la modification.

test Cette fois le curseur se positionne bien à la suite du message.

Pour le moment, la saisie est possible mais n'est pas mémorisée. Il faut enregistrer la saisie dans une **variable**. Vous êtes libre du nom donné à la variable en respectant certaines règles (cela sera préciser plus tard).

```
Console.Write("Entrez le nombre à chercher = ");
valeur = Console.ReadLine();
```

Remarquez que **valeur** est souligné en rouge pour indiquer une erreur.

Placez le curseur sur **valeur** sans cliquer : vous allez voir le même message qui apparaît en bas **Le nom 'valeur' n'existe pas dans le contexte actuel**.

Avant d'être utilisée, une variable doit être déclarée pour que l'ordinateur réserve une place mémoire pour elle. La déclaration consiste à préciser son nom précédé de son type (une variable peut être de type entier, réel, chaîne, ...).

```
// déclaration
int valeur;

// saisie du nombre à chercher
Console.Write("Entrez le nombre à chercher = ");
valeur = Console.ReadLine();
```

Remarquez l'ajout de **commentaires** : ce sont les lignes qui commencent par ***%%**. A la suite, vous pouvez écrire ce que vous voulez. Cela permet d'expliquer le code. Il est alors plus facile à relire et comprendre. L'ordinateur ne tient pas compte des commentaires. Suite à la déclaration de **valeur**, cette fois une autre erreur est apparue : **Impossible de convertir implicitement le type 'string' en 'int'**. En effet, une saisie à la console retourne toujours un type **string** (une chaîne de caractères) quelle que soit l'information saisie. **valeur** doit être de type **int** (entier) car on va faire des comparaisons de nombres. Il est possible de changer le type de la saisie de la façon suivante : <code c#> valeur = int.Parse(Console.ReadLine()); </code> Cela signifie qu'on demande de **parser** (changer le type) de l'information entre parenthèses (donc ici, de ce qui est saisi) en type **int**. Cette fois il n'y a plus d'erreur. Après la récupération de la valeur à chercher, il faut effacer l'écran avant que le second joueur ne commence. Cela se fait avec l'instruction suivante : <code c#> Console.Clear(); </code> Enfin, comme on l'a vu précédemment, prenez l'habitude de laisser en fin de programme la ligne suivante, qui permet d'éviter la fermeture directe de la fenêtre de commande, en fin d'exécution du programme : <code c#> Console.ReadLine(); </code> Au final, vous devriez obtenir ceci : <code c#> // déclaration int valeur; // saisie du nombre à chercher Console.Write("Entrez le nombre à chercher = "); valeur = int.Parse(Console.ReadLine()); Console.Clear(); Console.ReadLine(); </code> <WRAP center round todo > **test** Vérifiez si vous arrivez à saisir puis si l'écran s'efface. </WRAP> {{ :bloc1:csharp_02.jpg |}} ===== Etape 2 ===== ==== Saisie d'un premier essai ==== Maintenant que l'écran est effacé, le second joueur va pouvoir commencer. Le principe va être le même que précédemment : * il faut afficher un message * et mémoriser la saisie dans une nouvelle variable. Il faut utiliser une autre variable pour ne pas perdre la valeur d'origine qui est à trouver. A la suite du code actuel (mais toujours avant le Console.ReadLine final), ajoutez : <code c#> // saisie du premier essai Console.Write("Entrez un essai = "); essai = int.Parse(Console.ReadLine()); </code> Remarquez que **essai** est soulignée en rouge. C'est normal et vous devriez savoir pourquoi.: la variable n'est pas déclarée. Ajoutez sa déclaration, au même niveau que 'valeur' : <code c#> // déclaration int valeur, essai; </code> Lorsque plusieurs variables sont de même type, il est possible de les déclarer sur la même ligne. Vous pouvez aussi faire une ligne par déclaration. Maintenant, il n'y a plus d'erreur. <WRAP center round todo> **Test** Vous pouvez saisir la valeur à chercher, puis l'écran s'efface et vous pouvez saisir un premier essai. </WRAP> {{ :bloc1:csharp_03.jpg |}} ===== Etape 3 ===== ==== Comparaison de l'essai avec la valeur à trouver ==== Une fois l'essai saisi, il faut le comparer avec la valeur d'origine pour afficher le message **trop grand** si l'essai est supérieur à la valeur d'origine, ou **trop petit** si l'essai est inférieur à la valeur d'origine. Il est donc nécessaire de comparer **essai** avec **valeur**. Vous allez ainsi découvrir un nouveau principe de codage : **l'alternative (la condition)**. A la suite du code actuel, ajoutez : <code c#> // test de l'essai par rapport à la valeur à chercher if (essai > valeur) { Console.WriteLine(" --> trop grand !"); } else { Console.WriteLine(" --> trop petit !"); } </code> L'alternative **if** contient une condition (entre parenthèses) et un ou 2 blocs : * le premier bloc s'exécute uniquement si la condition est vraie, * le second bloc (dans le **else**) s'exécute uniquement si la condition est*

fausse. La seconde partie (le **else**) est optionnelle. Dans le cas présent, les 2 parties sont importantes pour afficher le message correspondant. L'exécution se poursuit alors normalement après l'alternative. <WRAP center round todo> **test** Vérifiez que vous obtenez bien le message (**trop grand** ou **trop petit**) correspondant à ce qui est attendu. Par exemple, si la valeur est 57 et l'essai est 100, vous devriez obtenir **trop grand**). Même image que l'écran précédent avec une étape supplémentaire (par rapport au code ajouté) : affichage du premier résultat ("--> trop grand !") </WRAP> {{ :bloc1:csharp_04.jpg |}} ===== Etape 4 ===== ==== Saisie d'un nouvel essai ? ==== Il sera nécessaire de saisir plusieurs essais avant de trouver la bonne valeur, sans savoir à l'avance le nombre d'essais à saisir. Il n'est donc pas question de répéter plusieurs fois les lignes de codes précédentes : il faut faire en sorte que le programme se charge de cette répétition. Vous allez ainsi découvrir un nouveau principe de codage : **l'itération (la boucle)** qui permet de répéter plusieurs fois une ou plusieurs lignes de code, tant qu'une **condition** est vérifiée. Sur quoi doit-on boucler ? Sur la saisie de l'essai et la comparaison avec la valeur d'origine. Dans notre cas on veut boucler tant que la valeur n'a pas été trouvée, donc le test doit ressembler à ceci : <code c#> while (essai != valeur) { // instructions qui vont se répéter } </code> Le signe **!=** correspondant à **différent** en mathématique. Le test se faisant dès le début, il faut qu'un premier essai soit saisi. Puis, dans la boucle, il faut intégrer le test déjà écrit. Mais ensuite, toujours dans la boucle, il faut saisir un nouvel essai : <code c#> // saisie du premier essai Console.Write("Entrez un essai = "); essai = int.Parse(Console.ReadLine()); while (essai != valeur) { // test de l'essai par rapport à la valeur à chercher if (essai > valeur) { Console.WriteLine(" --> trop grand !"); } else { Console.WriteLine(" --> trop petit !"); } // saisie d'un nouvel essai Console.Write("Entrez un essai = "); essai = int.Parse(Console.ReadLine()); } </code> Juste après la saisie d'un nouvel essai, l'exécution revient sur la ligne du **while** est compare **essai** avec **valeur**. Si les 2 sont égaux, alors l'exécution sort de la boucle, sinon, le contenu de la boucle est exécuté à nouveau. <WRAP center round todo> **test** Vérifiez qu'à chaque essai saisi, le bon message s'affiche. Une fois la valeur trouvée, il ne se passe pour le moment rien, mais surtout, il n'est pas demandé de saisir un nouvel essai. </WRAP> {{ :bloc1:csharp_05.jpg |}} ===== Etape 5 ===== ==== Valeur trouvée ==== Après la boucle, et avant que la fenêtre ne se ferme, ce serait bien d'afficher un message pour dire que la valeur a été trouvée. Ajoutez la ligne de code correspondante : <code c#> // valeur trouvée Console.WriteLine("Vous avez trouvé"); </code> <WRAP center round todo> **test** Cette fois, après avoir saisi plusieurs essais, puis la bonne valeur, vous obtenez bien le message **Vous avez trouvé**. </WRAP> {{ :bloc1:csharp_06.jpg |}} ===== Etape 6 ===== ==== Affichage du nombre de tentatives ==== Le but est aussi d'afficher le nombre de tentatives qui ont été nécessaires pour trouver la valeur. Il faut une autre variable, qu'on va appeler **nbre** et qui doit compter le nombre de tentatives. Comme il y a au moins une tentative (un premier essai saisi avant la boucle), on va directement affecter 1 à cette variable en début de programme, on dit qu'on **initialise la variable**. Donc, au niveau de la déclaration, ajoutez la variable nbre et initialisez-là à 1, comme ceci : <code c#> // déclaration int valeur, essai, nbre = 1; </code> Arrêtons-nous quelques instants sur cette ligne : * cette écriture est un raccourci ; * les 3 variables étant de même type, le type n'est précisé qu'une seule fois et les variables sont séparées par des virgules. De plus il est possible de déclarer et initialiser des variables en même temps. Cette ligne de code pourrait être écrite en plusieurs fois comme ceci (inutile de modifier le code, c'est juste pour que vous compreniez). <code c#> // déclaration int valeur; int essai; int nbre; nbre = 1; </code> Ces deux blocs de code sont équivalents. <WRAP center round info> Remarquez au passage que **nbre** est souligné en vert : ce n'est pas une erreur mais une information. Si vous mettez la souris dessus sans cliquer, vous obtenez le message "La variable 'nbre' est assignée, mais sa valeur n'est jamais utilisée". </WRAP> Effectivement on ne l'utilise pas encore. Il faut compter le nombre d'essais, donc il faut ajouter 1 dans **nbre** à chaque fois que l'on saisit un nouvel essai, donc dans la boucle. L'ajout de 1 peut se faire de la façon suivante : <code c#> nbre = nbre + 1; </code> On affecte dans 'nbre' sa valeur actuelle + 1. Il existe un raccourci d'écriture très utilisé dans de nombreux langages héritiers du C (C like) : <code c#> nbre++; </code> Vous allez ajouter cette ligne de code en fin de boucle (dernière ligne de code de la boucle), comme ceci : <code c#> // compteur d'essais nbre++; </code> Il reste à afficher le nombre d'essais dans le message final. Modifiez la ligne de code correspondante (c'est faux mais c'est pour que vous compreniez) : <code c#> // valeur trouvée Console.WriteLine("Vous avez trouvé en nbre fois !"); </code> <WRAP center round todo> **test** Vérifiez qu'au final, vous obtenez le message **Vous avez trouve en nbre fois !** </WRAP> Ce n'est pas ce que l'on veut : on veut que ce soit le contenu de la variable **nbre** qui soit affiché. Par exemple, si on a trouvé en 4 fois, on aimerait obtenir : **"Vous avez trouvé en 4 fois !"**. Tout ce qui est entre guillemets est affiché en l'état. Si on veut afficher une variable, il ne faut pas la mettre entre guillemets. Il est possible de construire une chaîne de caractère en concaténant (collant) des chaînes et des contenus de variables. Voici donc la ligne de code correcte : <code c#> // valeur trouvée Console.WriteLine("Vous avez trouvé en "+nbre+" fois !"); </code> <WRAP center round info 60%> **test** Cette fois, vous allez obtenir le bon message, avec le nombre d'essais affichés. </WRAP> {{ :bloc1:csharp_07.jpg |}} ===== Etape 7 ===== ==== Nettoyage du code ==== Rappelez-vous que du code a été généré automatiquement dès le début de la création du projet, en particulier les **using** : {{ :bloc1:csharp_08.jpg |}} <WRAP center round info> Les **using** permettent d'accéder à certaines fonctionnalités. </WRAP> Vous remarquerez qu'excepté la première ligne, les autres **lignes sont grisées**. Cela vient du fait que seul le premier **using**' est utilisé : il permet d'utiliser Console. Les autres n'étant pas utilisés dans votre programme, vous pouvez les supprimer. Cette phase consiste aussi à contrôler qu'il n'y a pas de variable non utilisée (qui apparait en gris). Ce n'est pas le cas ici. <WRAP center round info> **test** Vérifiez que le programme fonctionne toujours correctement. </WRAP> {{ :bloc1:csharp_09.jpg |}} ===== Etape 8 ===== ==== Ajout de cartouche ==== Il nous reste à ajouter un tout dernier détail dans le programme : un **cartouche** qui est un commentaire particulier, de plusieurs lignes et qui a la particularité de pouvoir être interprété par l'ordinateur pour générer automatiquement la documentation technique du programme. Même si le cartouche permet d'apporter des information complémentaire, la création de la documentation technique est nécessaire mais abordée plus tard. * Placez le curseur tout en haut à droite du fichier et validez pour créer une ligne vide. * Ajoutez le cartouche suivant (en changeant l'auteur et la date) Remarquez que, dès la saisie de ** / **, tout le programme se met en vert, car tout est considéré comme un commentaire, tant que le cartouche n'est pas fermé. En validant après **/%%, une ligne se crée, commençant par * , vous pouvez alors écrire à la suite. Quand au final, vous écrivez / collé à *, le cartouche se ferme et le reste du code reprend ses couleurs d'origine : <code c#> / * Jeu du nombre caché * author : votre nom * date : 23/05/2020 */ using System; </code> Toutes ces modifications ne doivent rien changer à l'exécution du programme. Voici le programme complet.

test Vérifiez que le programme fonctionne toujours correctement.

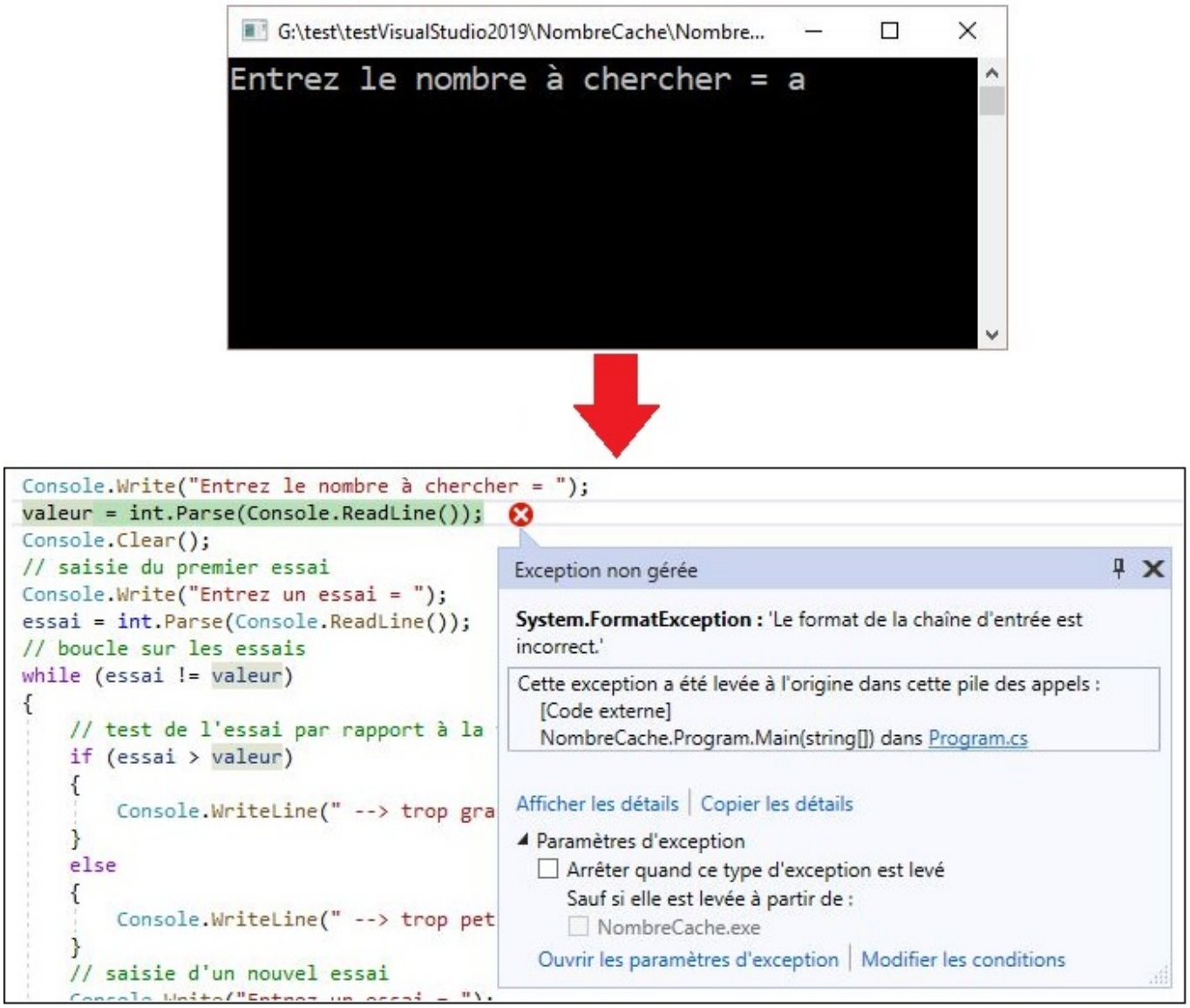
```
/**
 * Jeu du nombre caché
 * author : Emds
 * date : 23/05/2020
 */
using System;

namespace NombreCache
{
    0 références
    class Program
    {
        0 références
        static void Main(string[] args)
        {
            // déclaration
            int valeur, essai, nbre = 1;
            // saisie du nombre à chercher
            Console.Write("Entrez le nombre à chercher = ");
            valeur = int.Parse(Console.ReadLine());
            Console.Clear();
            // saisie du premier essai
            Console.Write("Entrez un essai = ");
            essai = int.Parse(Console.ReadLine());
            // boucle sur les essais
            while (essai != valeur)
            {
                // test de l'essai par rapport à la valeur à chercher
                if (essai > valeur)
                {
                    Console.WriteLine(" --> trop grand !");
                }
                else
                {
                    Console.WriteLine(" --> trop petit !");
                }
                // saisie d'un nouvel essai
                Console.Write("Entrez un essai = ");
                essai = int.Parse(Console.ReadLine());
                // compteur d'essais
                nbre++;
            }
            // valeur trouvée
            Console.WriteLine("Vous avez trouvé en "+nbre+" fois !");
            Console.ReadLine();
        }
    }
}
```

===== Etape 9 ===== ==== Tests de l'application ==== Lorsqu'un programme est termné, il est important de le tester. En réalité, les tests doivent se faire après chaque fonctionalité, ce que vous avez fait. Mais les tests finaux sont importants. Le but est de voir si le programme fonctionne quelle que soit la situation. Il est généralement impossible de tester tous les cas possibles : ici, on ne peut pas essayer tous les nombres ! Le but est plutôt de tester les possibilités classiques, les cas particuliers et les comportements inattendus. Dans l'exemple actuel, il faut donc tester les cas suivants : * trouver la valeur en plusieurs essais (cas classique) ; * trouver la valeur au premier essai (cas particulier car le premier essai a été traité différemment, avant la boucle) ; * saisir un caractère au lieu d'un nombre (comportement inattendu).

test Essayez les 2 premiers cas qui ne doivent pas poser de problème. Essayez ensuite le dernier cas en tapant une lettre au lieu d'un nombre (dans la valeur, mais le problème sera identique dans l'essai).

Le programme s'arrête violemment et une erreur apparait directement dans le code (voir capture ci-contre). La ligne concernée est mise en évidence : <code c#> valeur = int.Parse(Console.ReadLine()); </code> et le message d'erreur dit : **System.GormatException : 'Le format de la chaîne d'entrée est incorrect.'**. La ligne de code permet de réaliser une saisie, puis de la transformer (**transtyper**) en type **int** (entier) pour pouvoir l'affecter à la variable **valeur** qui est de type **int**. Le problème vient du fait qu'une lettre est saisie et que le programme tente de la convertir en **int**, ce qui n'est pas possible. Pour éviter l'arrêt brutal du programme, il faut soit éviter la conversion (ce qui n'est pas **naturel** ici, car on veut chercher un nombre), soit **capturer l'erreur**.



==== Tests de l'application (capture d'erreur) ==== Voilà comment fonctionne la capture d'erreur : <code c#> try { *contenant le code qui peut poser problème } catch {* contenant le code à exécuter, en cas d'erreur } </code> Commençons par la première saisie (valeur). L'idée est de boucler tant que l'utilisateur n'a pas saisi une valeur correcte (un nombre entier). Dans la boucle, pour éviter une erreur, le code de saisie et de transtypage va être mis dans le 'try'. Si le 'catch' s'exécute, c'est qu'il y a une erreur et un message sera affiché. Cela suppose de savoir quelle condition mettre au niveau de la boucle : on veut boucler tant que la saisie n'est pas correcte. Pour cela utilisons une variable 'correct' de type booléen (qui ne peut prendre que les valeurs true ou false). Voici le code à compléter (juste en dessous des déclarations actuelles, et en récupérant les 2 lignes de code qui concernant la saisie de 'valeur') : <code c#> bool correct = false; *saisie du nombre à chercher while (!correct) { try { Console.Write("Entrez le nombre à chercher = "); valeur = int.Parse(Console.ReadLine()); correct = true; } catch { Console.WriteLine("Erreur de saisie : saisissez une nombre entier"); } } Console.Clear();* </code> Le signe "!" devant 'correct' *signifie "non" : donc on boucle tant que "non correct", ce qui est équivalent à "correct == false". Dans le 'try', dès que l'ordinateur tombe sur une erreur, il va directement au 'catch', donc la ligne de code "correct = true" ne s'exécutera que s'il n'y a pas d'erreur à la ligne précédente. Remarquez une nouvelle erreur : sur la ligne de code "while (essai != valeur)", la variable 'valeur' est soulignée en rouge, avec le message d'erreur "Utilisation d'une variable locale non assignée 'valeur'". Cela vient du fait que 'valeur' peut très bien ne rien recevoir, car son affectation est dans un 'try'. L'ordinateur n'est pas assez 'intelligent' pour comprendre que le programme va justement boucler tant qu'une valeur correcte n'a pas été saisie. Donc, pour éviter cette erreur, il suffit d'initialiser 'valeur' : autant le faire au niveau des déclaration : affectez-lui la valeur 0 (comme vous aviez affecté 1 à 'nbre').*

test

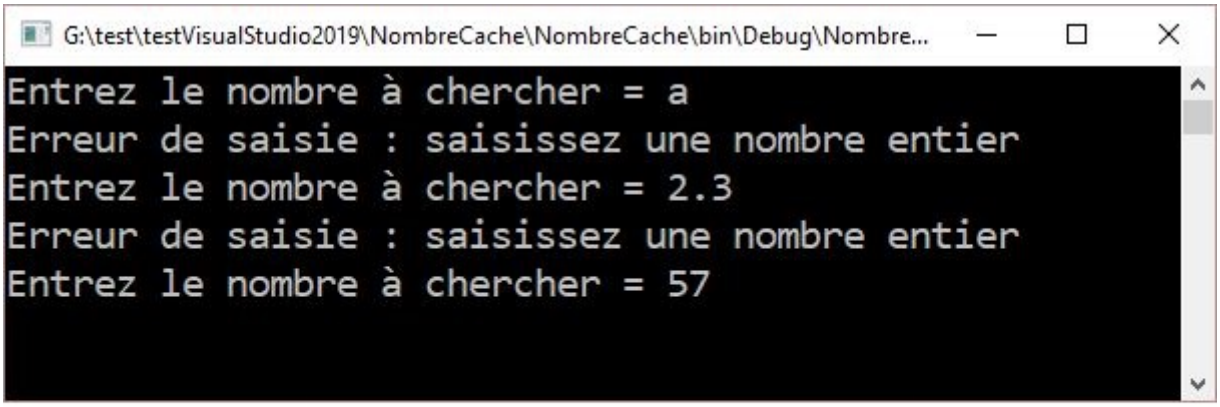
Vérifiez qu'en saisissant une lettre, le message d'erreur s'affiche et la saisie est redemandée. Vous pouvez faire l'erreur plusieurs fois. Saisissez enfin un entier : le jeu continue normalement.

Faites les mêmes modifications pour la saisie de l'essai (attention, il y a une saisie avant la boucle, et une dans la boucle) : vous pouvez bien sûr utiliser la même variable 'correct', mais sans oublier de lui réaffecter la valeur 'false' à chaque fois avant la boucle sur la saisie.

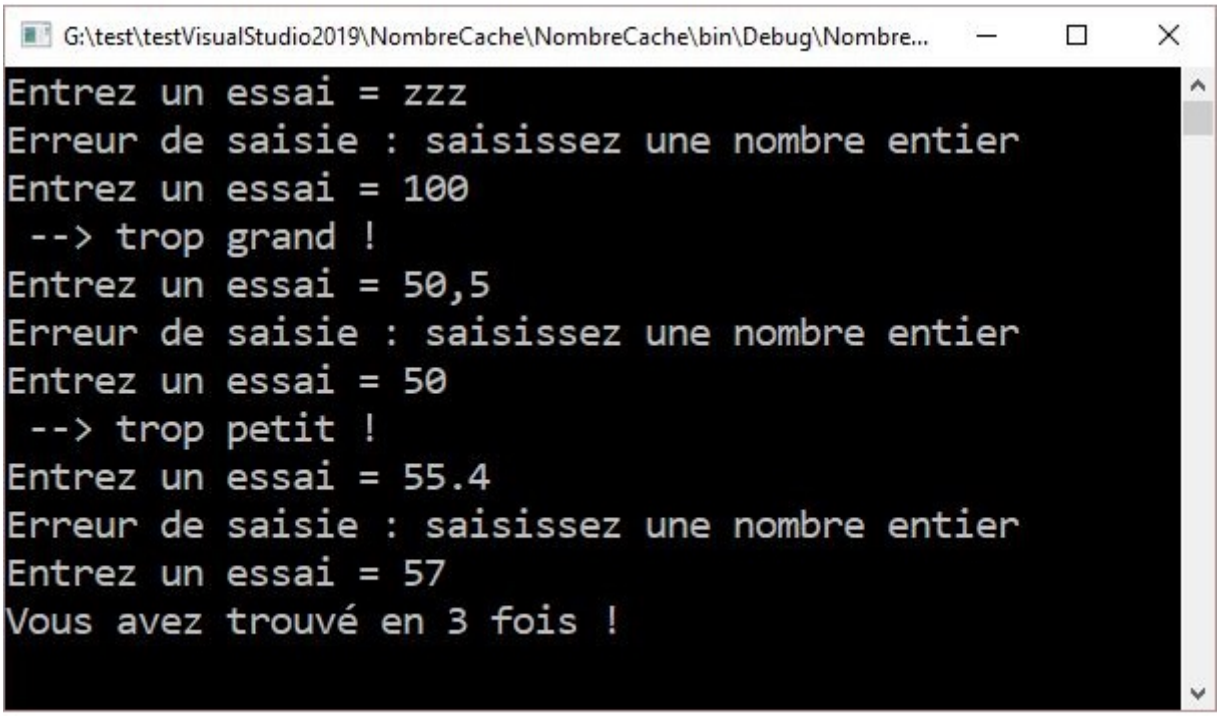
test

Vérifiez que tout le programme fonctionne correctement. Essayez de saisir une lettre lors du premier essai, et aussi lors des essais suivants. Vérifiez aussi que le nombre d'essais, au final, est correct (l'ordinateur n'a pas compté les erreurs de saisie).

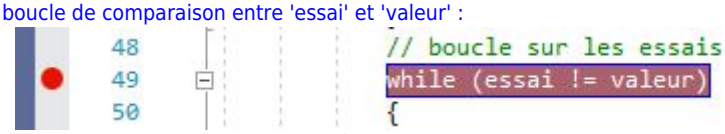
Vous avez remarqué que du code se répète : vous apprendrez plus tard comment optimiser le code pour éviter ce genre de répétition.



écran effacé

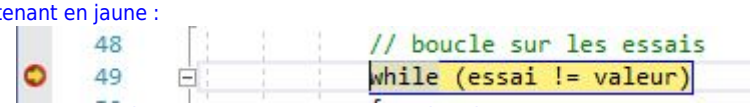


===== Etape 10 ===== ==== Outil de débogage : ==== Tous les IDE possèdent un outil de débogage qui permet une exécution pas à pas du programme avec vérification de l'évolution des variables. Même si le programme est opérationnel, voyons comment fonctionne cet outil. === Point d'arrêt : === Il est possible de placer un ou plusieurs points d'arrêt dans un programme. Lors du débogage, le programme va s'exécuter normalement jusqu'au point d'arrêt, puis il se met en attente. Pour placer un point d'arrêt, cliquez dans la marge de gauche au niveau de la ligne de code concernée. Un point rouge va apparaître. Pour l'enlever, il suffit de cliquer dessus à nouveau. Placez un point d'arrêt au niveau de la grande boucle de comparaison entre 'essai' et 'valeur' :

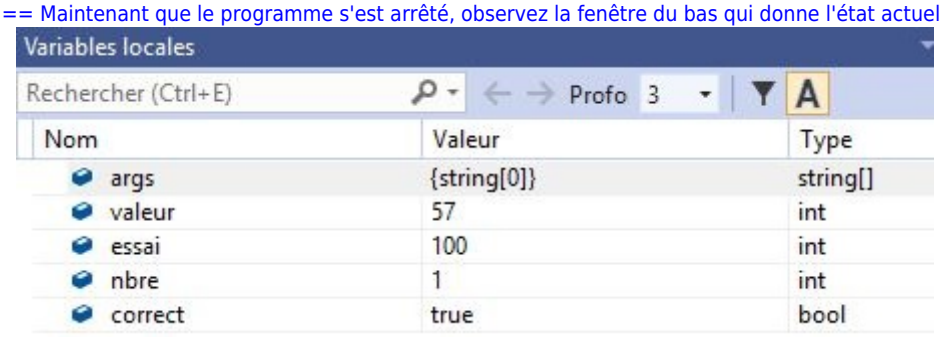


La ligne du point d'arrêt apparaît en rouge. === Démarrage du débogueur : === Il est alors possible de lancer le débogueur : menu “Déboguer > Démarrer le débogage”. La fenêtre d'exécution s'ouvre normalement. Saisissez une valeur et validez. L'écran s'efface. Saisissez un premier essai (différent de valeur) et validez. Cette fois le programme se met en attente et, dans le code, remarquez que la

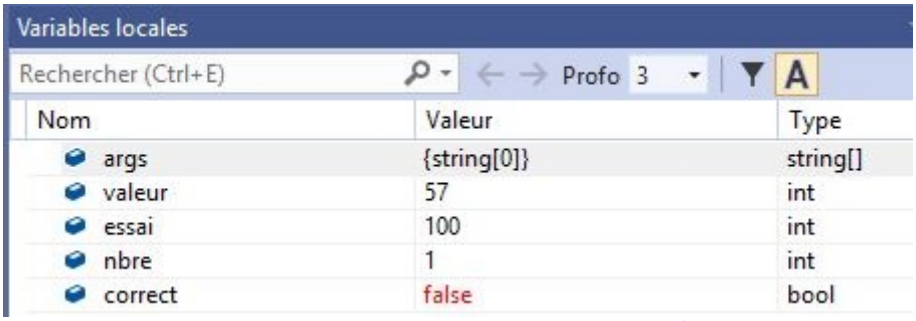
ligne du point d'arrêt est maintenant en jaune :



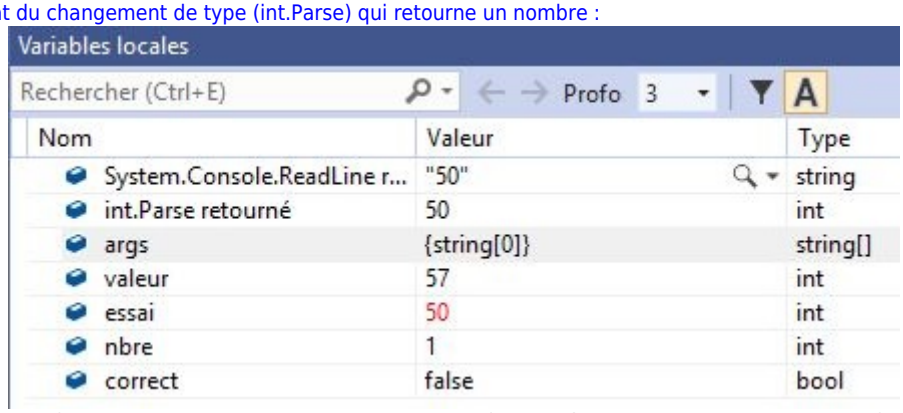
A partir de là, vous allez pouvoir consulter l'état des variables et continuer à exécuter le programme ligne par ligne, à votre rythme. === Etat des variables : === Maintenant que le programme s'est arrêté, observez la fenêtre du bas qui donne l'état actuel des variables :



Ne tenez pas compte de 'args'. Vous retrouvez les 4 variables du programme : valeur, essai, nbre, correct. La seconde colonne contient les valeurs actuelles (ici, j'ai tapé 57 pour 'valeur' et 100 pour le premier essai). La troisième colonne donne les types des variables. === Exécution pas à pas : === Vous allez maintenant avancer dans l'exécution, ligne par ligne. Dans le menu Debogueur, remarquez qu'il y a plusieurs possibilités de faire du “pas à pas”. On va pour le moment utiliser le “pas à pas principal”, que vous pouvez obtenir aussi avec F10. Appuyez sur F10 (ou passez par le menu). La ligne jaune d'exécution a avancé d'une instruction. Elle est maintenant sur l'accolade, et la ligne du point d'arrêt est redevenue rouge. Continuez à avancer avec F10 : le test va être évalué entre essai et valeur et, suivant le résultat, la ligne jaune va se possitionner dans la première partie ou la seconde (le else). Avancez encore une fois : le message “trop petit” ou “trop grand” s'affiche dans la fenêtre d'exécution. Avancez encore : lorsque vous aurez passé l'instruction “correct = false”, vous remarquerez le changement de valeur de 'correct' dans la fenêtre des variables :



Cela permet de repérer les changements qui viennent de se faire. Continuez d'avancer jusqu'à passer la ligne de saisie de l'essai : cette fois il n'y a plus de ligne jaune et la fenêtre d'exécution est repassée au premier plan, en attente d'une saisie. Saisissez une valeur et validez : vous retournez à nouveau dans le code avec la ligne jaune. Remarquez aussi, dans la fenêtre des variables, que la valeur de 'essai' est en rouge, et aussi 2 informations complémentaires : la valeur saisie (Console.ReadLine) où l'on remarque bien que c'est une chaîne (entre guillemets) et le résultat du changement de type (int.Parse) qui retourne un nombre :



=== Exécution rapide ou arrêt : === A tout moment, il est possible d'arrêter le débogage (en cliquant sur le carrêt rouge, ou menu “déboguer > arrêter le débogage”). Il est aussi possible de reprendre une exécution normale (F5 ou menu “Déboguer > Continuer”) : le débogueur va alors avancer jusqu'à la fin du programme ou jusqu'au point d'arrêt suivant, s'il y en a un. Essayez.