CNED

B1.2-PARTIE-I-SQ1-S1

Liberté • Égalité • Fraternité
RÉPUBLIQUE FRANÇAISE

MINISTÈRE
DE L'ÉDUCATION
NATIONALE, DE
L'ENSEIGNEMENT
SUPÉRIEUR ET DE
LA RECHERCHE

www.cned.fr

Les cours du CNED sont strictement réservés à l'usage privé de leurs destinataires et ne sont pas destinés à une utilisation collective. Les personnes qui s'en serviraient pour d'autres usages, qui en feraient une reproduction intégrale ou partielle, une traduction sans le consentement du CNED, s'exposeraient à des poursuites judiciaires et aux sanctions pénales prévues par le Code de la propriété intellectuelle. Les reproductions par reprographie de livres et de périodiques protégés contenues dans cet ouvrage sont effectuées par le CNED avec l'autorisation du Centre français d'exploitation du droit de copie (20, rue des Grands Augustins, 75006 Paris)

CNED, BP 60200, 86980 Futuroscope Chasseneuil Cedex, France - © CNED

Table des matières

1 ACCUEIL 3

1.1 OBJECTIFS 3

2 SITUATION PROFESSIONNELLE 4

2.1 VOTRE TÂCHE 4

3 PARTIE 1 - APPORTS MÉTHODOLOGIQUES 5

3.1 CE QU'IL FAUT SAVOIR 5
3.2 INSTALLATION ET CONFIGURATION DE VISUAL STUDIO 5
3.3 PREMIER EXEMPLE 21

4 PARTIE 2 - APPLICATION 40

4.1 APPLICATION 40
4.2 VOTRE TÂCHE 41
4.3 ETAPES À SUIVRE 42
4.4 CORRECTION 44

5 EVALUER SES ACQUIS 44

5.1 EVALUER SES ACQUIS : IDE 45
5.2 EVALUER SES ACQUIS : DÉBOGAGE 45
5.3 EVALUER SES ACQUIS : TICKET D'INCIDENTS 45

6 COMPLÉTER LES SAVOIRS 46

6.1 COMPLÉTER LES SAVOIRS 46

7 SYNTHÈSE 46

7.1 SYNTHÈSE DE LA SÉANCE 46

8 GLOSSAIRE 47

9 DOCUMENTATION 50

9.1 PREMIER EXEMPLE 50
9.2 SITUATION PROFESSIONNELLE 50
9.3 GUIDE INSTALLATION VISUAL STUDIO 50
9.4 FICHE SAVOIRS 50
9.5 TEST 50

10 VERSION IMPRIMABLE 51

1 Accueil

1.1 Objectifs

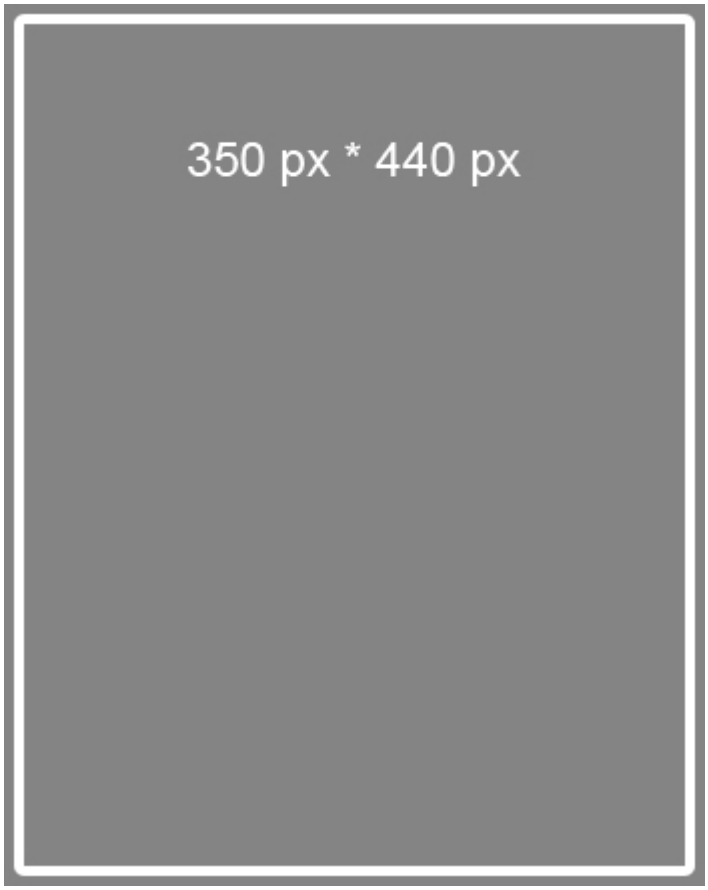


Image illustrative de développeurs devant un écran

1.1.1 Rappel de la mission

Votre travail consiste à rechercher l'origine et les causes de différents bugs dans les applications informatiques de votre client. Ces applications sont développées dans un langage procédural.

1.1.2 Ce que nous allons aborder

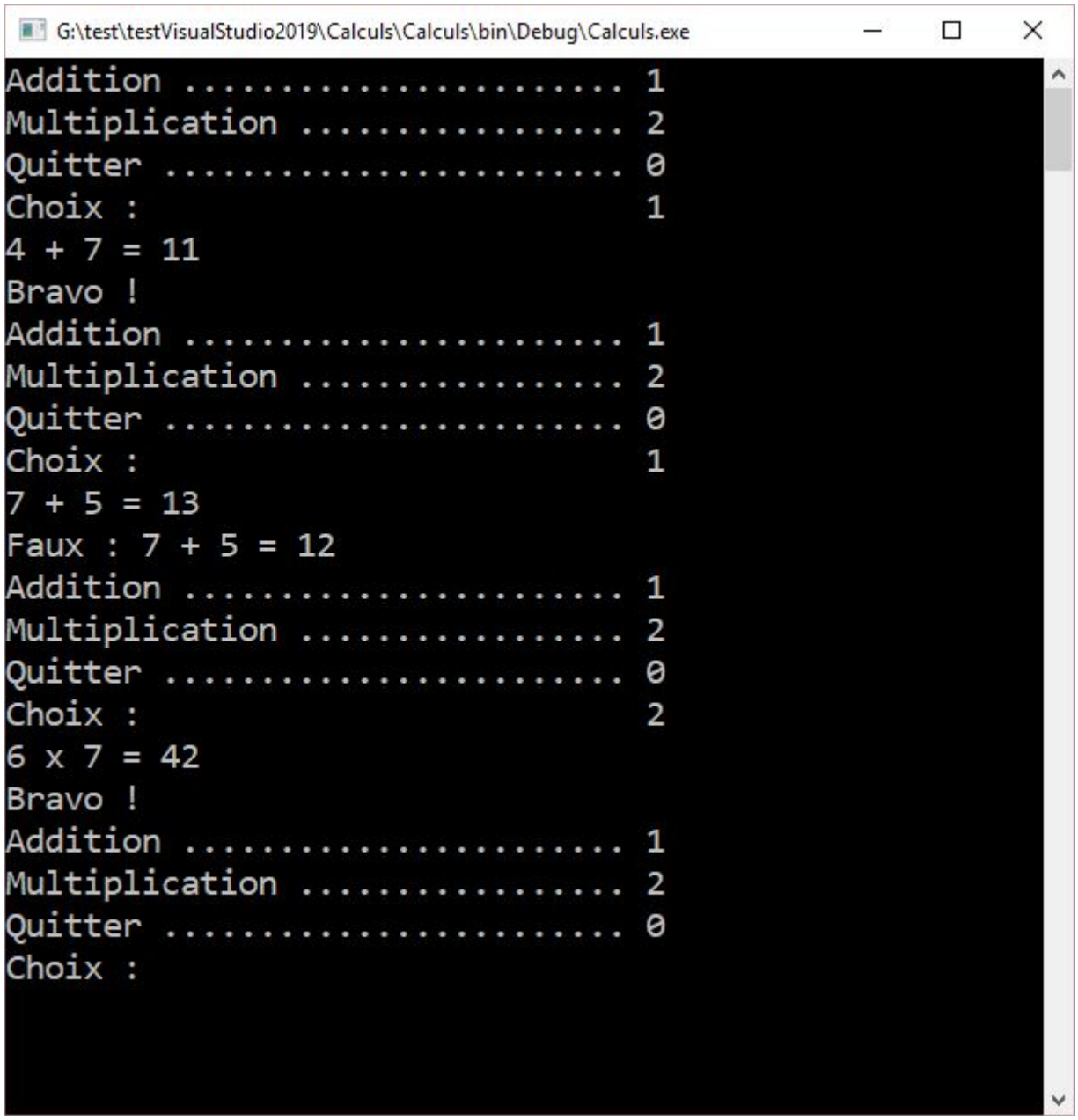
- Installer et configurer un IDE (integrated development environment) contenant un outil de débogage pour l'aide à la recherche d'erreurs (Visual Studio).
- Comprendre le fonctionnement de l'IDE et de l'outil de débogage.
- Dans un contexte professionnel, apprendre à répondre à une demande de correction d'une application existante.

1.1.3 Temps de travail indicatif

3 heures

2 Situation professionnelle

2.1 Votre tâche



Capture montrant l'exécution du programme :
- affichage du menu avec 3 choix (1 addition, 2 multiplication, 0 quitter)
- saisie du choix 1, proposition d'addition, saisie de la bonne réponse, affichage de "Bravo !"
- réaffichage du menu
- saisie du choix 1, proposition d'addition, saisie d'une réponse fausse, affichage de "Faux" suivi de la bonne réponse.
- réaffichage du menu

- saisie du choix 2, proposition de multiplication, saisie de la bonne réponse, affichage de "Bravo !"
- réaffichage du menu

2.1.1 Situation professionnelle

2.1.1.1 Présentation du contexte :

L'entreprise cliente réalise des applications d'apprentissage et de tests de connaissances dans des environnements divers.

2.1.1.2 Présentation de l'application :

Un petite application est en cours de création. Dans un premier temps réalisée dans un environnement non graphique, elle permet de contrôler les connaissances en calculs mathématiques basiques (addition, multiplication).
Pour le moment, comme le montre la capture ci-contre, un menu s'affiche, offrant le choix de contrôler ses connaissances en addition ou en multiplication. Suivant le choix, une opération est proposée et il faut saisir la réponse. Si la réponse est correcte, "Bravo !" est affiché. Dans le cas contraire, "Faux" est affiché, suivi de la réponse correcte. Dans tous les cas, le menu est à nouveau affiché, permettant de faire un nouveau test.

2.1.1.3 Présentation des dysfonctionnements :

Suite à des tests réalisés à la demande du chef de projet, un ticket d'incidents a été généré et vous est confié (vous pouvez le télécharger ici*).
Vous êtes en charge de corriger les dysfonctionnements, puis de rendre le code corrigé avec le rapport d'incidents complété (rapport d'incidents à compléter téléchargeable ici).

*les dysfonctionnements signalés dans le ticket d'incidents, sont rappelés dans le rapport d'incidents.

3 Partie 1 - Apports méthodologiques

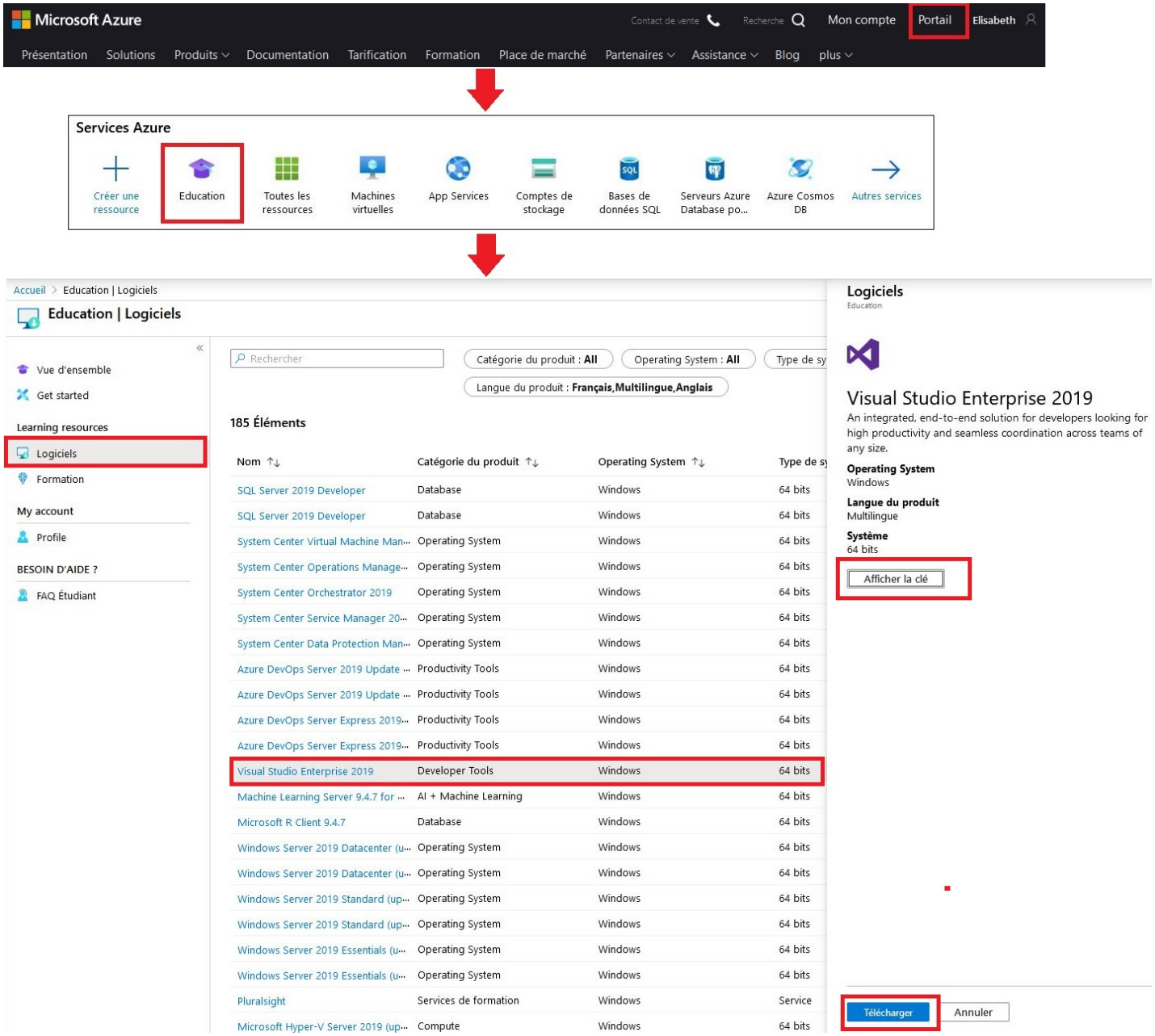
3.1 Ce qu'il faut savoir

Pour réaliser cette tâche, vous devez avoir acquis les savoirs suivants :

- Installation et configuration de l'IDE Visual Studio. Pour cela suivez le mode opératoire présenté dans les pages suivantes.
- Découverte de la programmation à travers un premier exemple. Ce premier exemple est disponible à la suite de l'installation et de la configuration de l'IDE Visual Studio.

3.2 Installation et configuration de Visual Studio

3.2.1 Téléchargement de Visual Studio

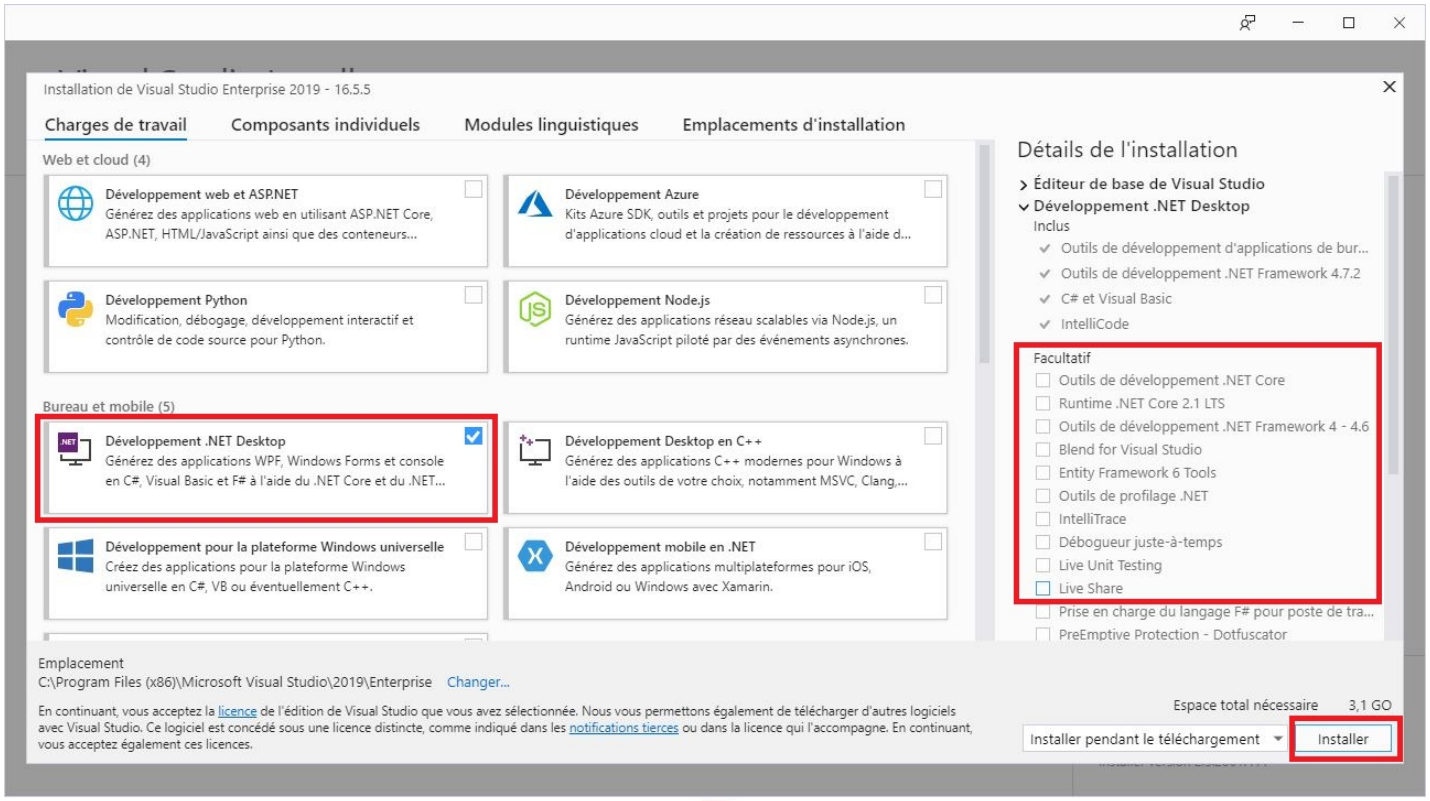


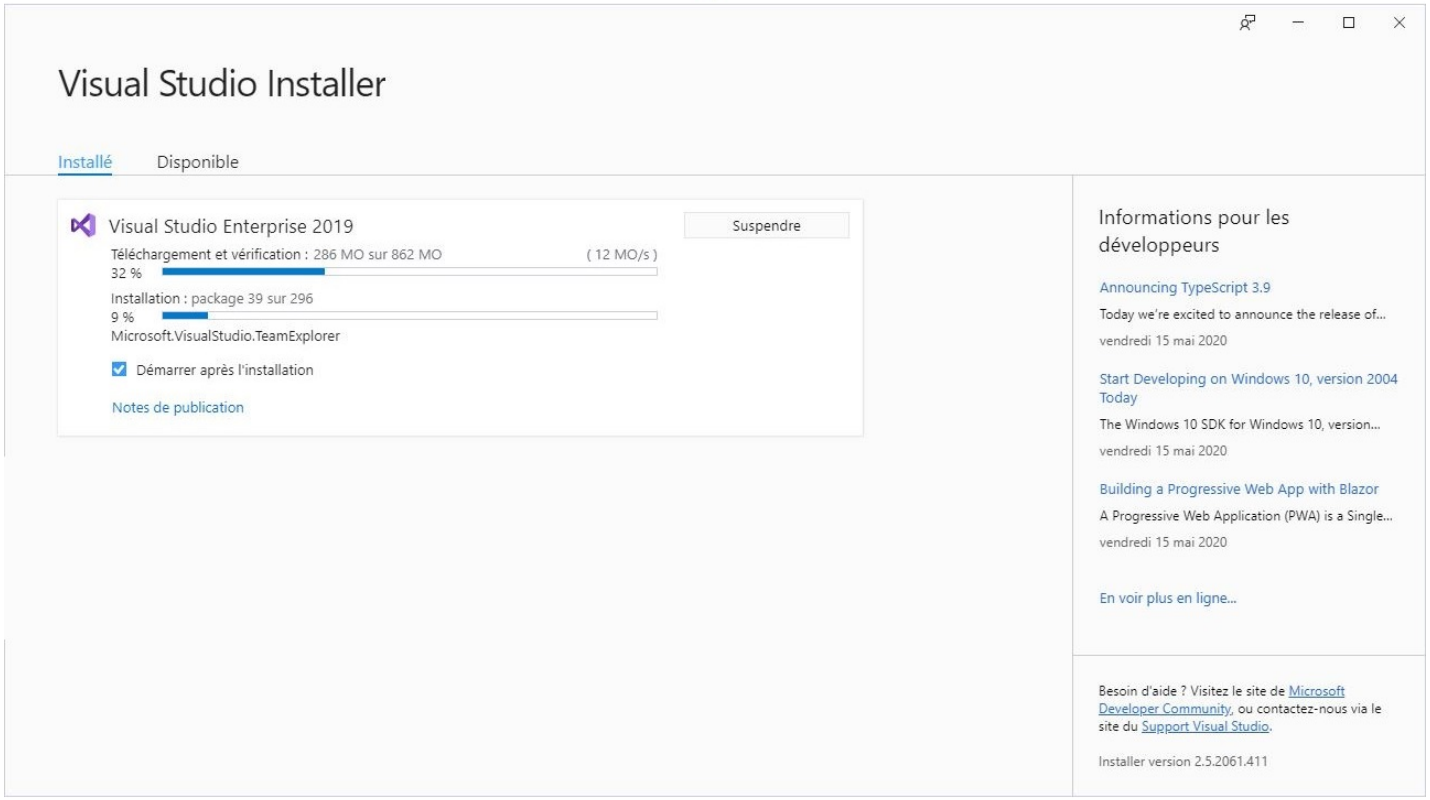
3 captures qui se suivent :
capture 1 : menu de Microsoft Azure une fois connecté, choix "Portail" entouré.
Capture 2 : barre d'icônes de "Services Azure", choix "Education" entouré.
Capture 3 : écran de "Educations" avec, à gauche "Logiciels" entouré, au milieu (dans la liste des logiciels), "Visual Studio Entreprise 2019" entouré, à droite, dans le détail du logiciel "Visual Studio Entreprise 2019", le bouton "Afficher la clé" entouré, et en bas, le bouton "Télécharger" entouré.

3.2.2 Téléchargement de Visual Studio 2019 Entreprise :

1. Allez sur votre compte Azure
2. Sélectionnez "Portail" > "Education" > "Logiciels" > "Visual Studio Entreprise 2019"
3. Cliquez sur "Afficher la clé" et notez la clé (vous en aurez besoin pour enregistrer le produit)
4. Cliquez sur "Télécharger"

3.2.3 Installation de Visual Studio





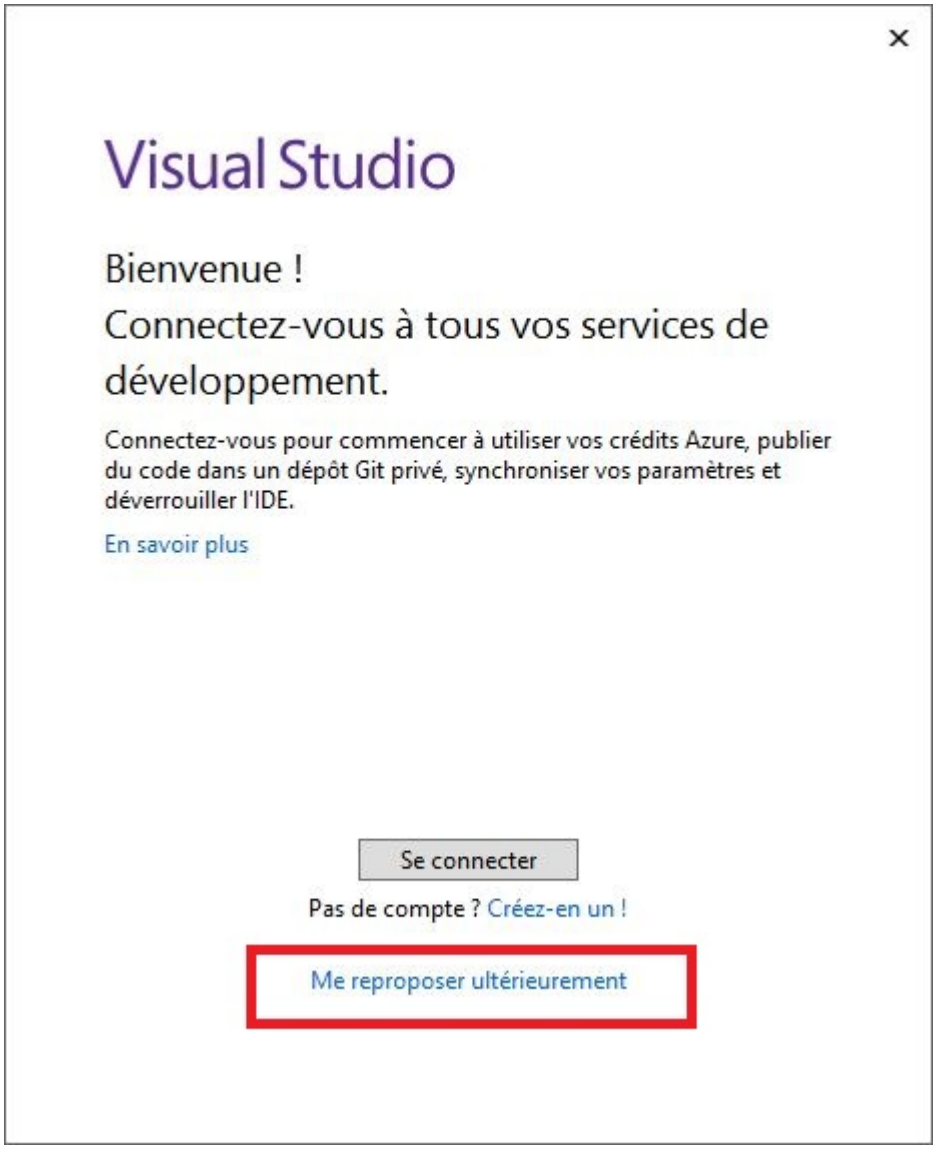
2 captures qui se suivent :
Capture 1 : premier écran d'installation avec "Développement .NET Desktop" coché et à droite, partie "Détails d'installation", toutes les options facultatives décochées, enfin, bouton "Installer" entouré.
Capture 2 : écran "Visual Studio Installer" montrant la progression de l'installation.

3.2.4 Installation de Visual Studio 2019 entreprise :

1. Double cliquez sur le fichier téléchargé pour lancer l'installation
2. Dans la première fenêtre, sélectionnez juste "Développement .NET Desktop"
3. Désélectionnez tout ce qui est facultatif, à droite
4. Cliquez sur "Installer"
5. Dans la seconde fenêtre, attendez la fin de l'installation
6. En fin d'installation, il est demandé de redémarrer l'ordinateur : cliquez sur "Redémarrer"

3.2.5 Création d'un projet

<u>Bienvenue</u>



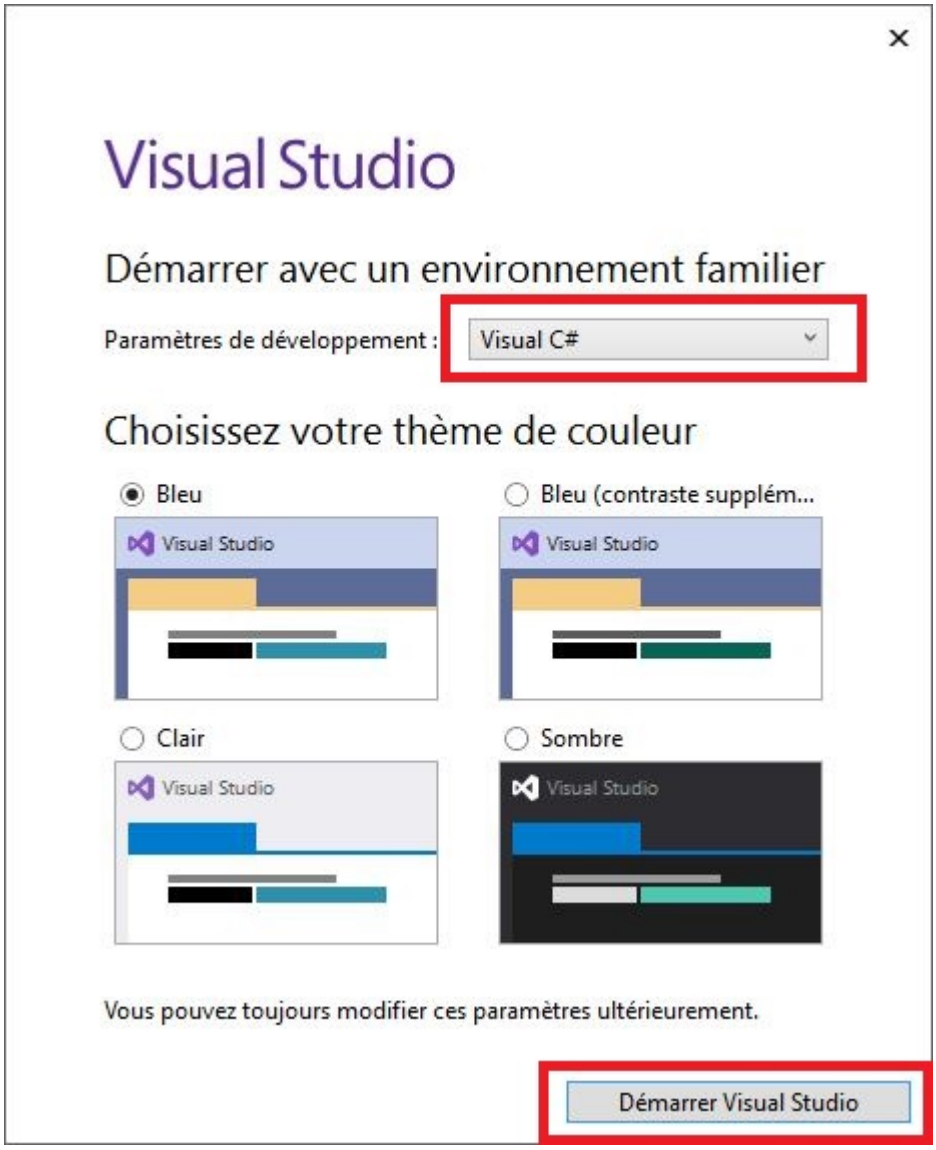
Écran de bienvenue avec "Me proposer ultérieurement" entouré.

Lancez Visual Studio 2019 Entreprise et suivez bien toutes les étapes en cliquant sur chaque miniature ci-dessous.

Etape 1 : écran de bienvenue

Lorsque la première fenêtre de bienvenue apparait (ci-contre), cliquez directement sur "Me proposer ultérieurement". Plus loin, vous verrez comment enregistrer le produit afin de ne plus avoir cette demande et surtout pour ne pas être limité dans l'utilisation de Visual Studio. En effet, Visual Studio est un produit commercialisé. C'est grâce à l'abonnement souscrit par le CNED que vous pouvez le récupérer gratuitement, dans le cadre de votre formation.

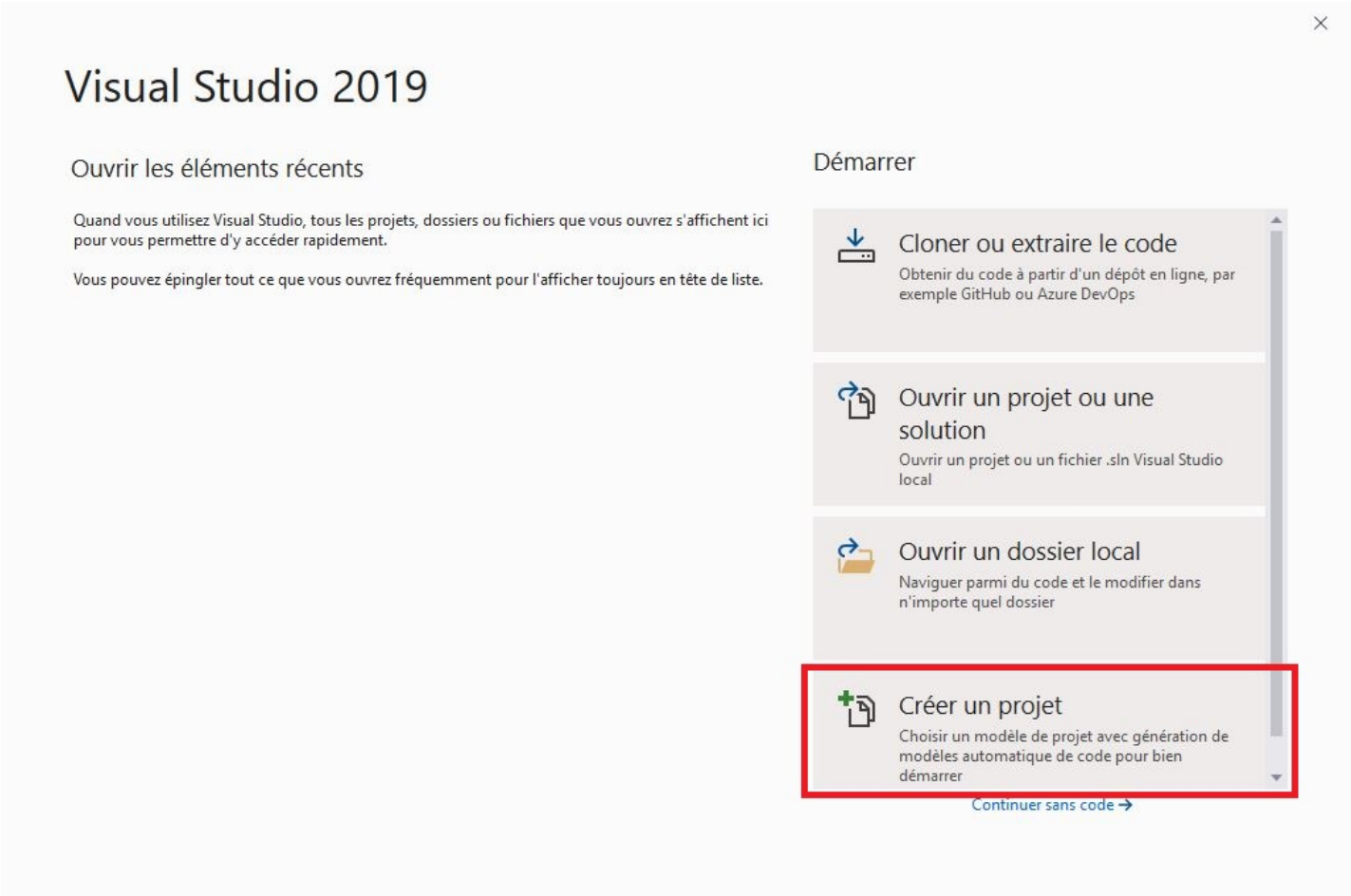
<u>Configuration</u>



Écran de configuration avec le choix "Visual C#" entouré et le thème "Bleu" sélectionné, puis le bouton "Démarrer Visual Studio" entouré.

Etape 2 : configuration de l'environnement
La nouvelle fenêtre permet de paramétrer l'environnement.
Au niveau du combo "Paramètres de développement", sélectionnez "C#" puisque tous les projets seront faits dans ce langage.
Il est possible de choisir l'aspect graphique de la fenêtre : ici, j'ai gardé le choix par défaut, mais vous pouvez prendre un autre choix. Vous pourrez changer cette configuration par la suite.
Cliquez ensuite sur "Démarrer Visual Studio".

<u>Ouverture ou création</u>



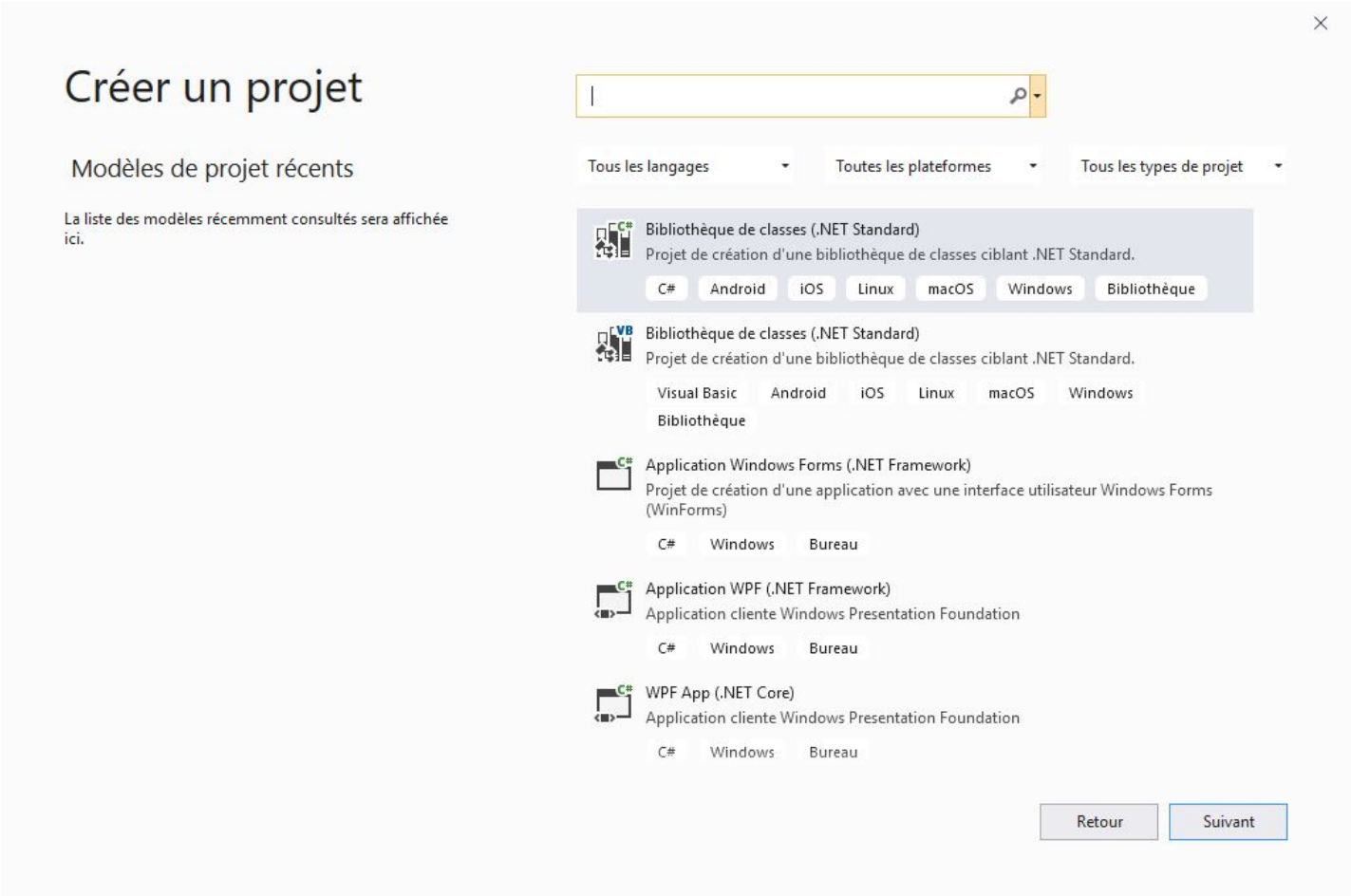
Écran qui permet d'ouvrir un projet existant ou de créer un projet : zone "Créer un projet" entourée.

Etape 3 : Ouverture ou création d'un projet

A cette étape, il est possible, entre autres, d'ouvrir un projet existant, d'ouvrir un dossier ou bien sûr de créer un nouveau projet.

Cliquez sur "Créer un nouveau projet".

Type de projet

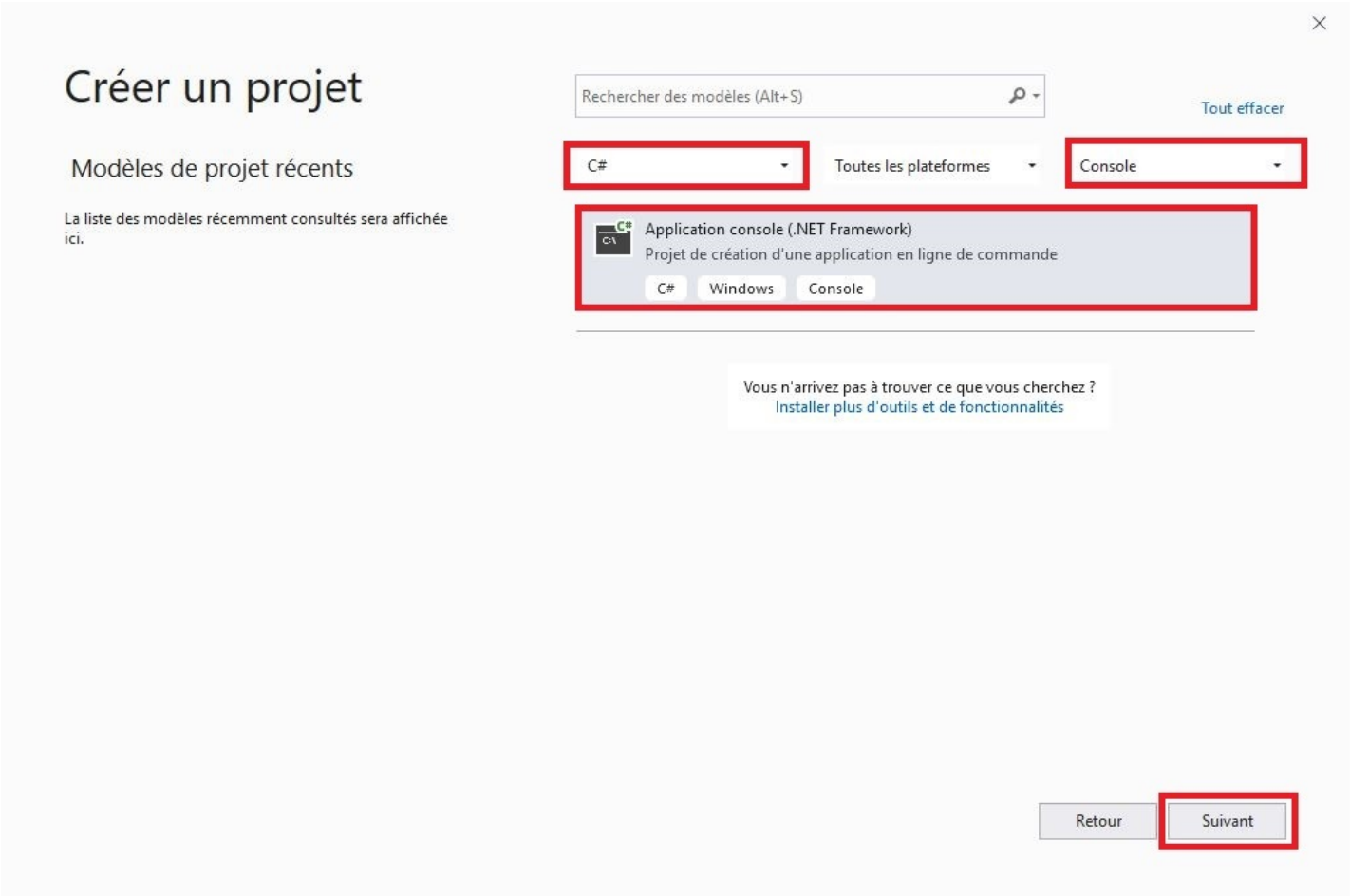


Écran "Créer un projet" qui montre la liste des types de projets possibles.

Etape 4 : Choix du type de projet

Il existe un grand nombre de types de projet.
Nous n'allons travailler qu'avec deux types de projets : d'abord les applications console en C# et, plus tard, les applications "Windows Forms" en C#.
Pour le moment, regardez juste la liste proposée et ne faites aucune sélection.
Passez à la miniature suivante pour savoir comment gérer plus facilement une sélection de type de projet.

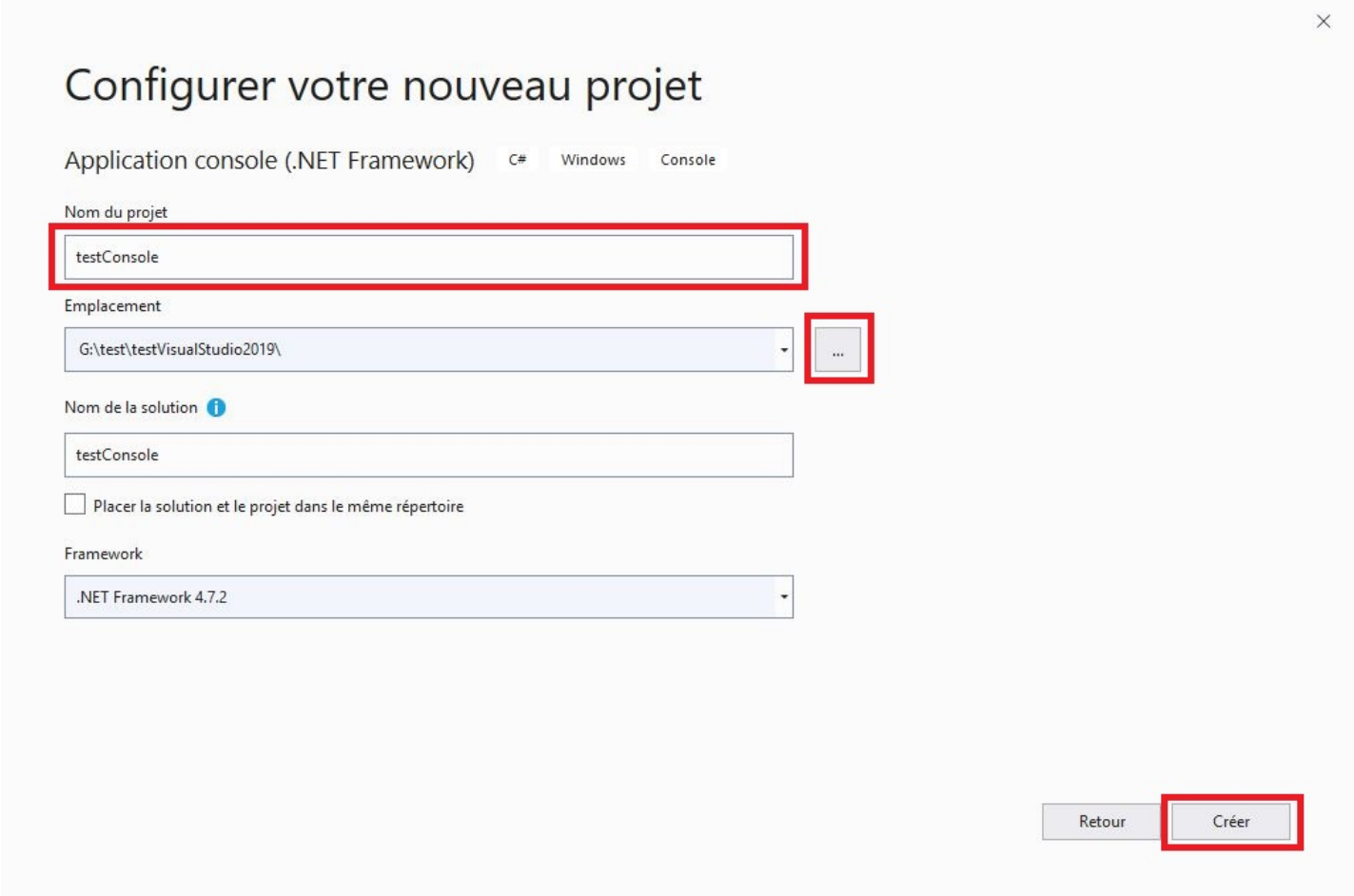
Type de projet (suite)



Écran "créer un projet" avec "C#" sélectionné dans le combo "tous les langages" et "console" sélectionné dans le combo "Tous les types de projets".
Il ne reste plus que "Application console (.NET Framework)".

Etape 5 : Aide au choix du type de projet
Il est possible de sélectionner un type de projet directement dans la liste, mais il est aussi possible d'affiner la sélection en utilisant les 3 combos.
Comme montré ci-contre, sélectionnez le langage (C#), et le type de projet (Console).
Remarquez que maintenant, il n'y a plus qu'un élément dans la liste : "Application console (.NET Framework)".
Cliquez sur cet élément pour le sélectionner, puis cliquez sur "Suivant".

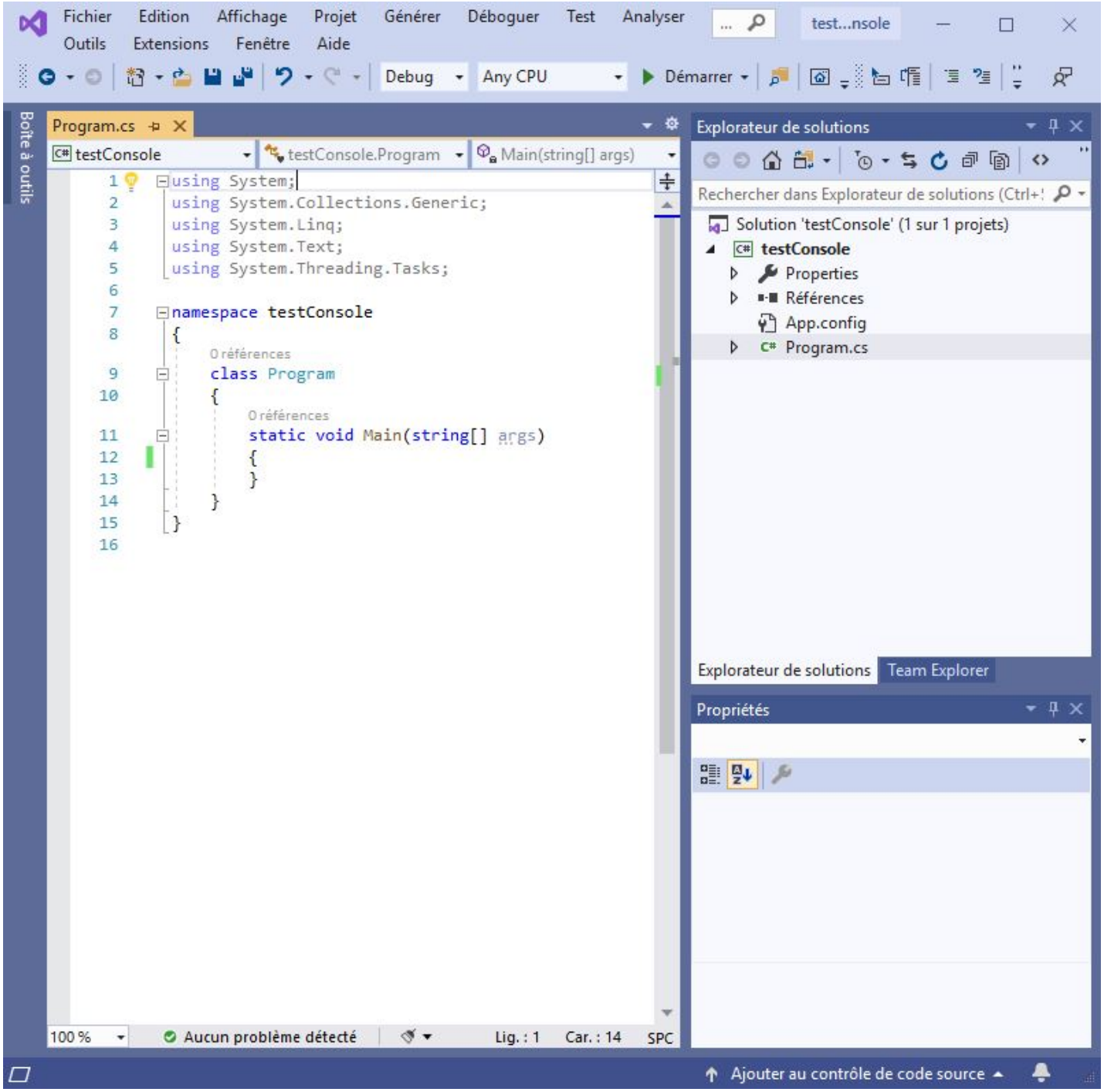
Nom du projet



Écran "Configurer votre nouveau projet" avec "testConsole" saisi dans la zone "Nom du projet", le bouton "..." entouré au niveau de "Emplacement" et le bouton "Créer" entouré.

Etape 6 : Choix du nom et du chemin du projet
Il reste enfin l'étape où il faut donner un nom au projet et préciser son emplacement.
Dans la zone de texte "Nom du projet", tapez "testConsole".
Au niveau "Emplacement", cliquez sur "..." et sélectionnez un dossier, préalablement créé sur votre disque, dans lequel vous allez placer vos projets.
Ne modifiez pas les autres informations et cliquez sur "Créer".

3.2.6 Fenêtre initiale



Fenêtre de Visual Studio montrant plusieurs zones : en haut le menu et la barre d'icônes, au centre le fichier Program.cs avec une partie du code généré, à droite l'explorateur de solutions, et en bas à droite les propriétés.

3.2.7 Présentation de la fenêtre initiale du projet

Suite à la demande de création du projet, la fenêtre de travail apparait.

3.2.7.1 Contenu de la fenêtre :

La fenêtre contient :
- une zone principale contenant le code du fichier Program.cs
- une zone à droite "Explorateur de solutions" contenant la liste des fichiers de la solution (on peut voir le fichier Program.cs)
- une zone à droite en bas "Propriétés" qui contiendra les propriétés des objets sélectionnés (pour le moment, on ne s'en occupera pas)
- en haut, les menus et icônes de raccourcis qui seront présentés au fur et à mesure des besoins
Les différentes zones peuvent être déplacées, ouvertes, fermées, adaptées en taille.

3.2.7.2 Notion de bloc de code :

Une partie du code a déjà été automatiquement générée. Les différents éléments d'un projet seront expliqués en temps utile, mais pour le moment, sachez juste qu'un programme se découpe en blocs, repérables par un entête de bloc et des accolades (ouvrantes et fermantes) pour déterminer le contenu du bloc.

3.2.7.3 Notion d'indentation :

Chaque bloc peut contenir d'autres blocs : dans ce cas, vous pouvez remarquer des décalages vers la droite. Cela permet de mieux repérer qui contient quoi. En programmation, ces décalages s'appellent aussi des "indentations". Normalement l'IDE les gère automatiquement, mais il faudra cependant surveiller qu'elles sont toujours correctes. L'ordinateur ne fait pas attention aux indentations quand il exécute un programme (sauf pour certains langages comme le Python), mais il est important de les respecter car sinon un programme devient vite illisible pour un programmeur.

3.2.7.4 Où écrire votre code :

A partir du code généré, le code que vous allez ajouter doit être écrit dans les accolades du bloc "static void Main(string[] args)".

3.2.8 Enregistrement du produit

Connectez-vous à Visual Studio

Visual Studio vous permet de planifier vos projets, de collaborer avec votre équipe et de gérer votre code en ligne quel que soit l'emplacement où vous vous trouvez.

En savoir plus

Connectez-vous pour commencer à utiliser vos crédits Azure, publier du code dans un dépôt Git privé, synchroniser vos paramètres et déverrouiller l'IDE.

Se connect Pas de compte ? Créez-en un !

Tous les comptes

Se connecter...

Visual Studio

Enterprise 2019

Licence : Version d'essai 30 jours (pour évaluation seulement)
La période d'évaluation a expiré.

⚠ Votre version d'essai a expiré. Connectez-vous pour prolonger la version d'essai ou obtenir une licence à l'aide d'un compte en ligne ou d'une clé de produit.

Rechercher une licence mise à jour

Déverrouiller avec une clé de produit (Product Key)

Quitter Visual Studio

16.0.30104.148 D16.5

Entrez une clé de produit

La clé de produit de 25 caractères se trouve dans le package Microsoft Visual Studio ou sur le site de l'abonné Visual Studio.

Clé de produit :

Appliquer

Connectez-vous à Visual Studio

Visual Studio vous permet de planifier vos projets, de collaborer avec votre équipe et de gérer votre code en ligne quel que soit l'emplacement où vous vous trouvez.

En savoir plus

Connectez-vous pour commencer à utiliser vos crédits Azure, publier du code dans un dépôt Git privé, synchroniser vos paramètres et déverrouiller l'IDE.

Se connect Pas de compte ? Créez-en un !

Tous les comptes

Se connecter...

Visual Studio

Enterprise 2019

Licence : Clé de produit appliquée

Fermer

16.0.30104.148 D16.5

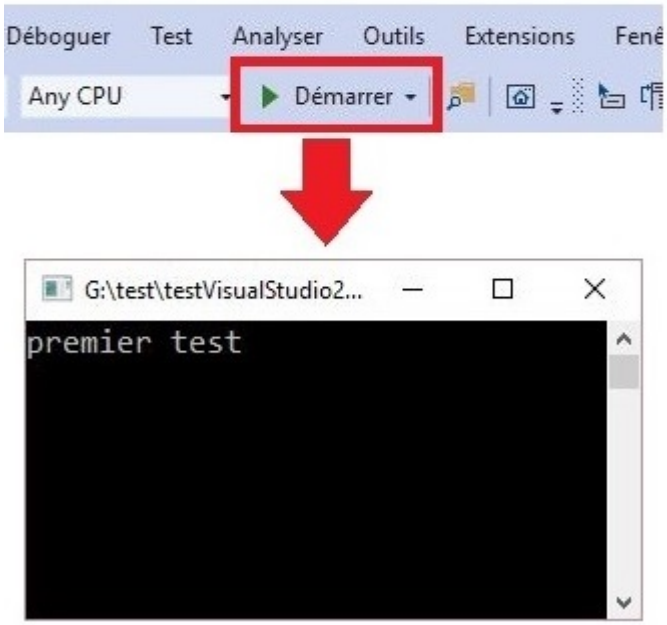
3 captures :
Capture 1 : fenêtre "Connectez-vous à Visual Studio" avec le lien "Déverrouiller avec une clé de produit (Product Key)" entouré.
Capture 2 : popup "Entrez une clé de produit" avec la zone de saisie et le bouton "Appliquer".
Capture 3 : retour à la fenêtre précédente qui précise "Licence : Clé de produit appliquée", avec le bouton "Fermer" entouré.

Avant de poursuivre, il est conseillé d'enregistrer la clé de Visual Studio pour pouvoir travailler tranquillement.

1. Allez dans le menu Aide > Enregistrer le produit pour obtenir la fenêtre du haut.
2. Cliquez sur "Déverrouiller avec une clé de produit (Product Key), vous allez obtenir la petite fenêtre du milieu.
3. Copiez la clé que vous avez précédemment récupérer sur Azure, au moment du téléchargement puis cliquez sur "Appliquer".
4. Normalement, vous allez obtenir la fenêtre du bas, avec le message "Licence : Clé de produit appliquée".
5. Cliquer sur "Fermer".

3.2.9 Premier test

```
0 références
class Program
{
    0 références
    static void Main(string[] args)
    {
        Console.WriteLine("premier  test");
        Console.ReadLine();
    }
}
```



3 captures :
Capture 1 : code contenu dans "main" :
Console.WriteLine("premier test");
Console.ReadLine();
Capture 2 : barre d'icônes avec "Démarrer" entouré
Capture 3 : résultat de l'exécution dans une fenêtre de commande, montrant "premier test"

3.2.9.1 Indentation :

Placez le curseur à droite de l'accolade ouvrante sous "static void Main(string[] args)" et validez. Normalement une ligne a été insérée et le curseur s'est automatiquement décalé vers la droite pour respecter l'indentation (on est à l'intérieur d'un bloc).

3.2.9.2 Autocomplétion :

Écrivez le code montré dans la capture ci-contre. Remarquez qu'en tapant le code, l'éditeur propose au fur et à mesure différentes possibilités pour vous aider lors de la frappe : c'est "l'autocomplétion".

3.2.9.3 Sensibilité à la casse :

C# est sensible à la casse : il fait la distinction entre minuscule et majuscule. Exemple : "Console" est différent de "console".

3.2.9.4 Explication du code :

Console.WriteLine(...) permet de demander un affichage.
Console.ReadLine() permet d'attendre une saisie au clavier (ici, c'est juste pour éviter que la fenêtre de commande se ferme juste après l'affichage).
Le ";" en fin de ligne marque la fin d'une instruction. Chaque bloc est en effet composé d'autres blocs et/ou d'instructions.

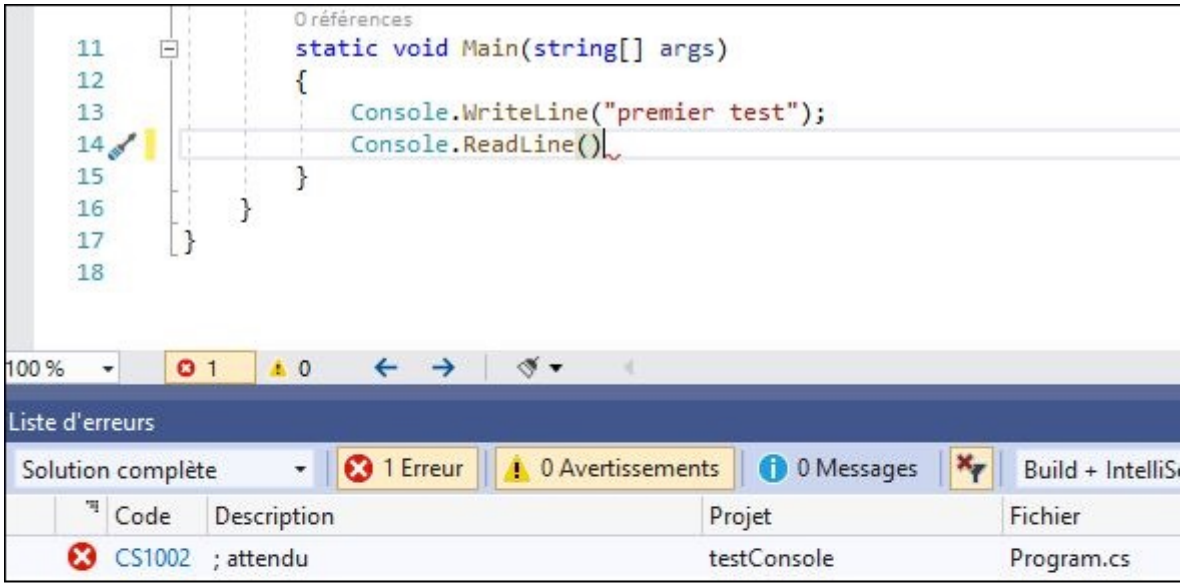
3.2.9.5 Sauvegarde :

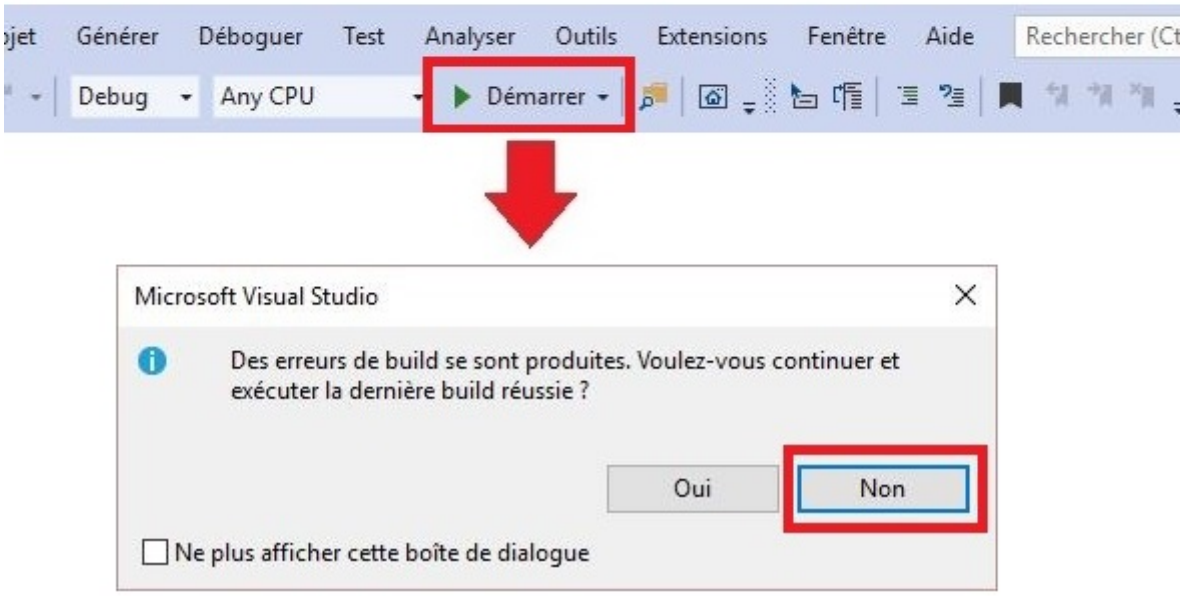
Pensez à régulièrement enregistrer votre travail : menu "Fichier > Enregistrer (Program.cs, ou enregistrer tout)" ou plus rapidement en utilisant le raccourci clavier CTRL-S.

3.2.9.6 Exécution :

Exécutez le programme en cliquant sur la flèche verte suivie du mot "Démarrer". Une fenêtre de commande doit s'ouvrir avec "premier test" affiché. Appuyez sur n'importe quelle touche, la fenêtre va se fermer.

3.2.10 Erreur de compilation





3 captures :
Capture 1 : montre la ligne de code "Console.ReadLind()" sans le ";" final et, à la place, une petite vague rouge.
Capture 2 : barre d'icônes avec "Démarrer" entouré
Capture 3 : popup avertissant qu'il y a des erreurs, proposant d'exécuter le dernier build réussi, et avec les boutons "oui" et "non", "non" étant entouré.

Visual Studio est capable de repérer des erreurs faites lors de la saisie du code, lorsque celui-ci ne respecte pas la syntaxe du langage.

3.2.10.1 Avertissement dans le code :

Imaginons que vous ayez fait une erreur, en oubliant le point-virgule de la fin de la seconde ligne (enlevez-le). Vous remarquez qu'un petit soulignement rouge en forme de vague est apparu à l'endroit de l'erreur.

3.2.10.2 Avertissement dans la liste des erreurs :

De même, dans la fenêtre du bas, le message d'erreur apparait "; attendu", en précisant le nom du fichier et du projet. Si vous double-cliquez sur cette ligne d'erreur, le curseur se place automatiquement sur la ligne de code concernée.

3.2.10.3 Avertissement lors de l'exécution :

Sans corriger l'erreur, enregistrez et lancez l'exécution. Vous allez obtenir la fenêtre d'information comme ci-contre, vous avertissant de la présence d'erreurs. Vous pouvez soit exécuter la version précédente qui n'avait pas d'erreur (avec "oui") soit refuser d'exécuter (avec "non"). Il faut toujours choisir "Non" afin de corriger les erreurs avant de tester à nouveau. Remettez le point-virgule manquant, enregistrez et faites un nouveau test. Normalement tout doit fonctionner.

3.2.11 Téléchargement du guide d'installation

Si vous le souhaitez, vous pouvez également télécharger ce *Guide d'installation et de configuration de L'IDE Visual Studio 2019 Entreprise* au format PDF.
Téléchargez le *Guide d'nstallation et configuration de Visual Studio*.

3.3 Premier exemple

3.3.1 But du premier exemple

3.3.2 Découverte du code et de l'environnement de développement :

Écrire un programme revient à dire à l'ordinateur ce qu'il doit faire. L'ordinateur est à votre service. Vous venez de le voir avec votre premier test. Il faut bien sûr aussi connaître la syntaxe du langage. Vous comprendrez par la suite que ce n'est pas le plus important : on trouve les éléments de syntaxe facilement sur Internet. Il faut surtout bien savoir ce qu'on doit demander.
Vous allez coder une petite application qui va vous perrmettre de découvrir un peu mieux l'environnement de développement et les instructions de base de la programmation.

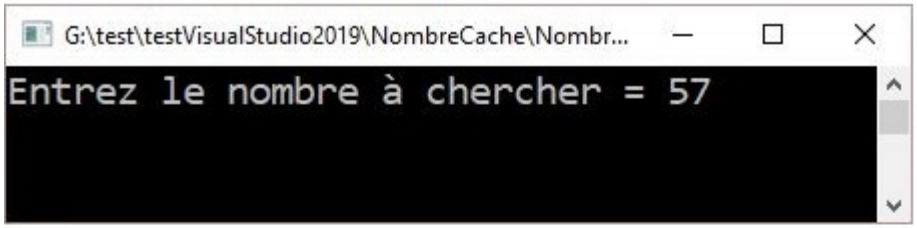
Après la réalisation de la situation professionnelle qui suit ce premier exemple, vous serez invité à approfondir vos connaissances en étudiant la fiche de savoirs : **cette étape est indispensable pour aborder les séances suivantes**.

3.3.2.1 But du premier exemple :

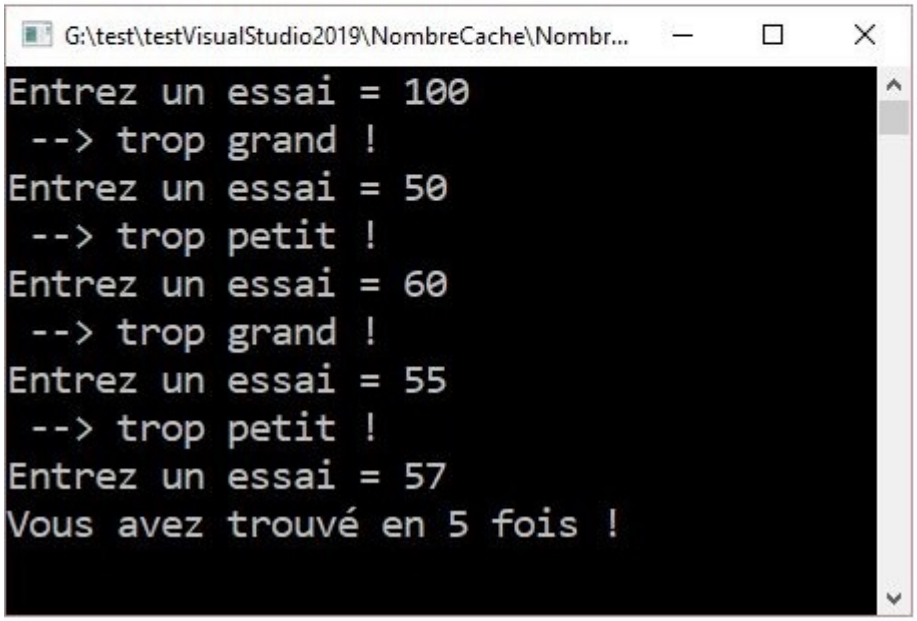
Le jeu du nombre caché se joue à deux joueurs.
Le premier joueur saisit un nombre puis sa saisie est effacée.
Le second joueur va ensuite tenter de trouver le nombre en saisissant des essais. A chaque essai le message

"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.

premier joueur :



écran effacé
second joueur :



Capture 1 : console montrant la première phase du jeu, avec le message "Entrez le nombre à chercher = " et la saisie "57"
Message disant "écran effacé, second joueur :"
Capture 2 : console montrant la suite du jeu avec à chaque fois "Entrez un essai = " suivi de l'essai saisi et le message "--> trop grand !" ou "--> trop petit !", et au final, quand la valeur est trouvé, message "Vous avez trouvé en 5 fois !" (dans l'exemple, il y a eu 5 essais).

3.3.3 Etape 1

3.3.4 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. Commençons ainsi (code à écrire dans le 'main') :

3.3.5 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.

3.3.6 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 (à quelques détails près que vous verrez dans la fiche de savoirs).

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, ...).

Remarquez au passage, 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 impliciteent 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' est 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 :

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 :

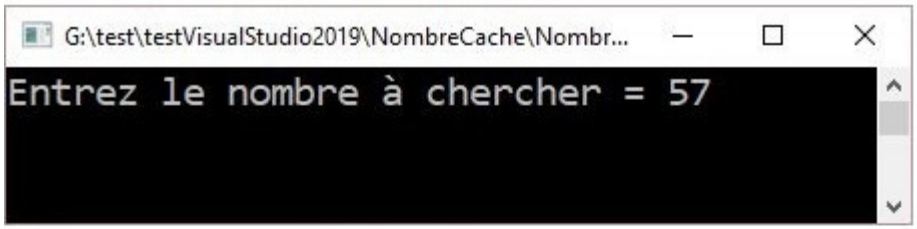
Enfin, comme on l'a vu précédemment, prenez l'habitude de laisser en fin de programme la ligne suivante, qui permet d'éviter la fermeture directe de la fenêtre de commande, en fin d'exécution du programme :

Au final, vous devriez obtenir ceci :

3.3.7 test

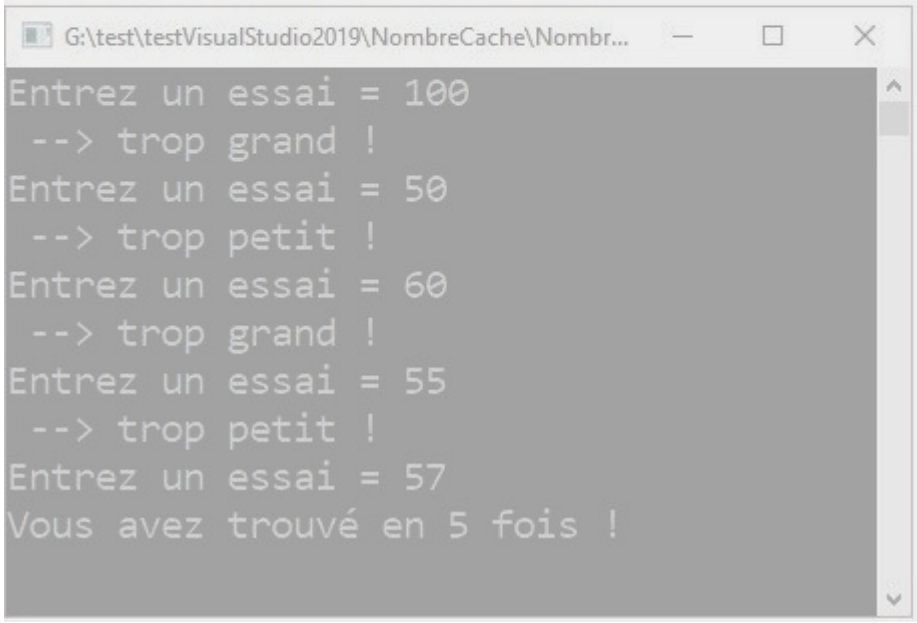
Vérifiez si vous arrivez à saisir puis si l'écran s'efface.

premier joueur :



écran effacé

second joueur :



Même image que l'écran précédent mais en montrant uniquement la partie actuellement codée : saisie de la valeur.
(le reste est en grisé)

3.3.8 Etape 2

3.3.9 Saisie d'un premier essai

Maintenant que l'écran est effacé, le second joueur va povoir commencer.
Le principe va être le même que p récédemment : il faut afficher un message et mémoriser la saisie dans une nouvelle variable. Il faut u tiliser 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 :

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' :

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.

3.3.10 Test

Vous pouvez saisir la valeur à chercher, puis l'écran s'efface et vous pouvez saisir un premier essai.

premier joueur :

```
G:\test\testVisualStudio2019\NombreCache\Nombr...
Entrez le nombre à chercher = 57
```

écran effacé
second joueur :

```
G:\test\testVisualStudio2019\NombreCache\Nombr...
Entrez un essai = 100
 --> trop grand !
Entrez un essai = 50
 --> trop petit !
Entrez un essai = 60
 --> trop grand !
Entrez un essai = 55
 --> trop petit !
Entrez un essai = 57
Vous avez trouvé en 5 fois !
```

Même image que l'écran précédent avec une étape supplémentaire (par rapport au code ajouté) : saisie du premier essai
(le reste est en grisé)

3.3.11 Etape 3

3.3.12 Comparaison de l'essai avec la valeur à trouver

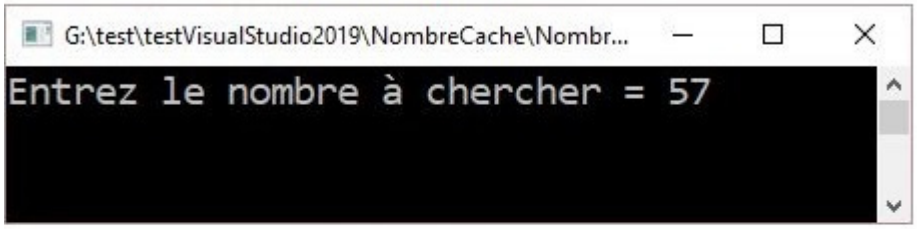
Une fois l'essai saisi, il faut le comparer avec la valeur d'origine pour afficher le message "trop grand" si l'essai est supérieur à la valeur d'origine, ou "trop petit" si l'essai est inférieur à la valeur d'origine. Il est donc nécessaire de comparer 'essai' avec 'valeur'. Vous allez ainsi découvrir un nouveau principe de codage : l'alternative (la condition).
A la suite du code actuel, ajoutez :

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.

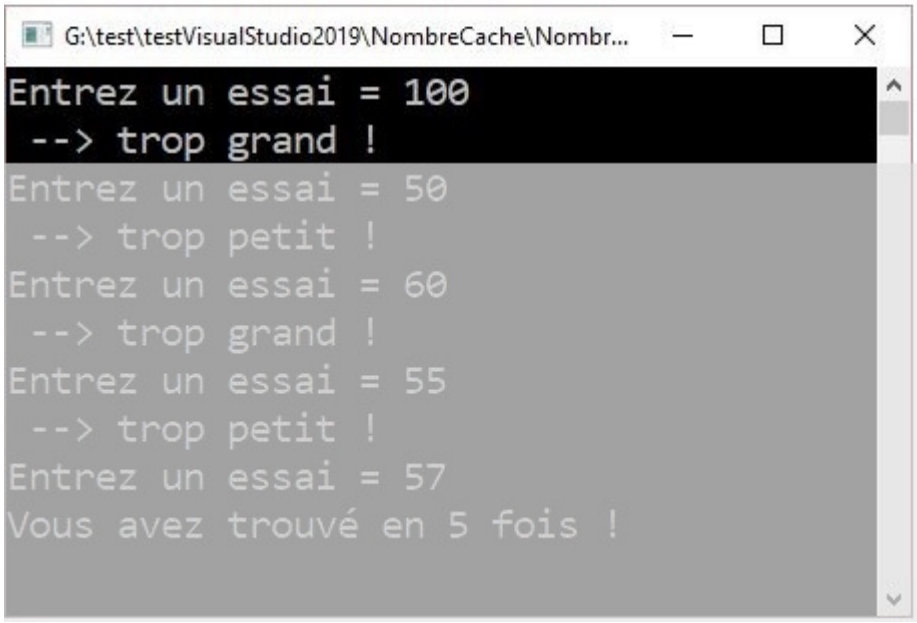
3.3.13 test

Vérifiez que vous obtenez bien le message ("trop grand" ou t"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").

premier joueur :



écran effacé
second joueur :



Même image que l'écran précédent avec une étape supplémentaire (par rapport au code ajouté) : affichage du premier résultat ("--> trop grand !")
(le reste est en grisé)

3.3.14 Etape 4

3.3.15 Saisie d'un nouvel essai ?

Vous imaginez bien qu'il va probablement falloir saisir plusieurs essais avant de trouver la bonne valeur. Et il est impossible de 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 :

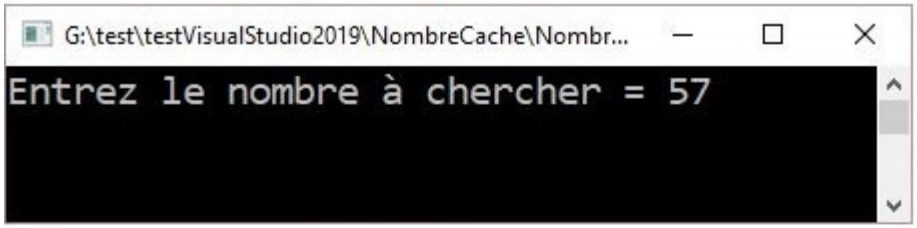
Le signe != correspondant à "différent" en mathématique.
Le test se faisant dès le début, il faut qu'un premier essai soit saisi. Puis, dans la boucle, il faut intégrer le test déjà écrit. Mais ensuite, toujours dans la boucle, il faut saisir un nouvel essai :

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.

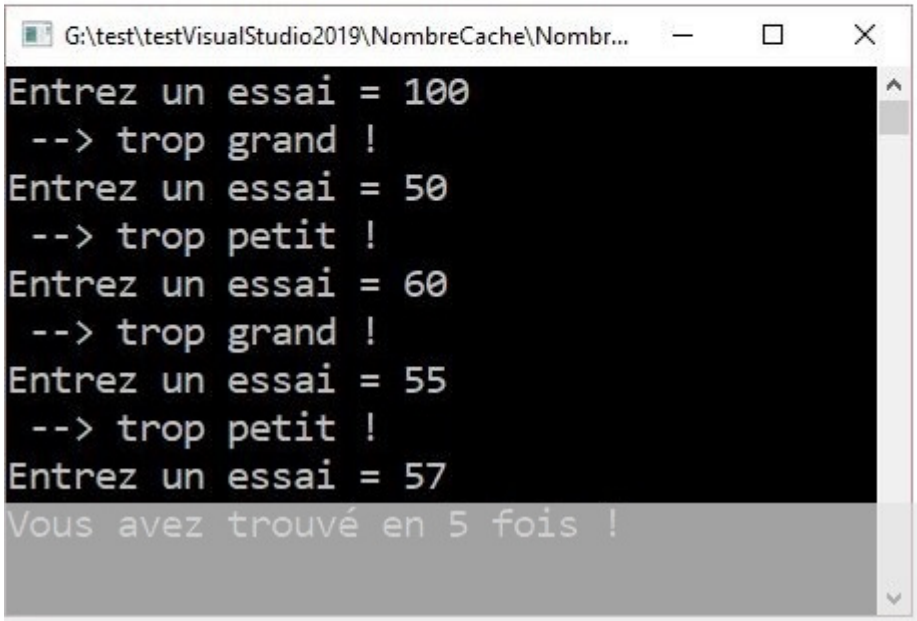
3.3.16 test

Vérufiez qu'à chaque essai saisi, le bon messagr 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.

premier joueur :



écran effacé
second joueur :



Même image que l'écran précédent avec une étape supplémentaire (par rapport au code ajouté) : saisie de tous les essais et affichage de tous les retours jusqu'à la fin.
(le reste est en grisé)

3.3.17 Etape 5

3.3.18 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 :

3.3.19 test

Cette fois, après avoir saisi plusieurs essais, puis la bonne valeur, vous obtenez bien le message "Vous avez trouvé".

premier joueur :

```
G:\test\testVisualStudio2019\NombreCache\Nombr...
Entrez le nombre à chercher = 57
```

écran effacé
second joueur :

```
G:\test\testVisualStudio2019\NombreCache\Nombr...
Entrez un essai = 100
 --> trop grand !
Entrez un essai = 50
 --> trop petit !
Entrez un essai = 60
 --> trop grand !
Entrez un essai = 55
 --> trop petit !
Entrez un essai = 57
Vous avez trouvé en 5 fois !
```

Même image que l'écran précédent avec une étape supplémentaire (par rapport au code ajouté) : affichage "vous avez trouvé"
(le reste est en grisé)

3.3.20 Etape 6

3.3.21 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 :

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).

Ces deux blocs de code sont équivalents.
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". 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 :

```
nbre = nbre + 1;
```

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) :

```
nbre++;
```

Vous allez ajouter cette ligne de code en fin de boucle (dernière ligne de code de la boucle), comme ceci :

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) :

3.3.22 test

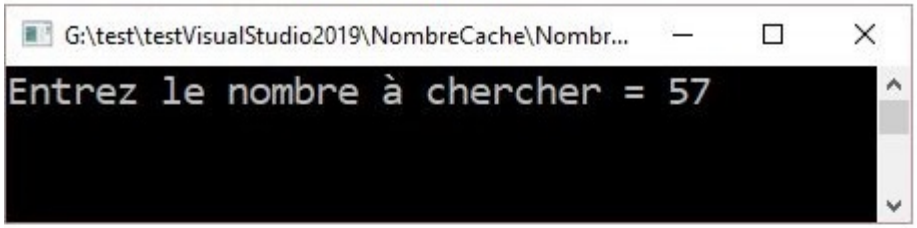
Vérifiez qu'au final, vous obtenez le message "Vous avez trouve en nbre fois !"

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 :

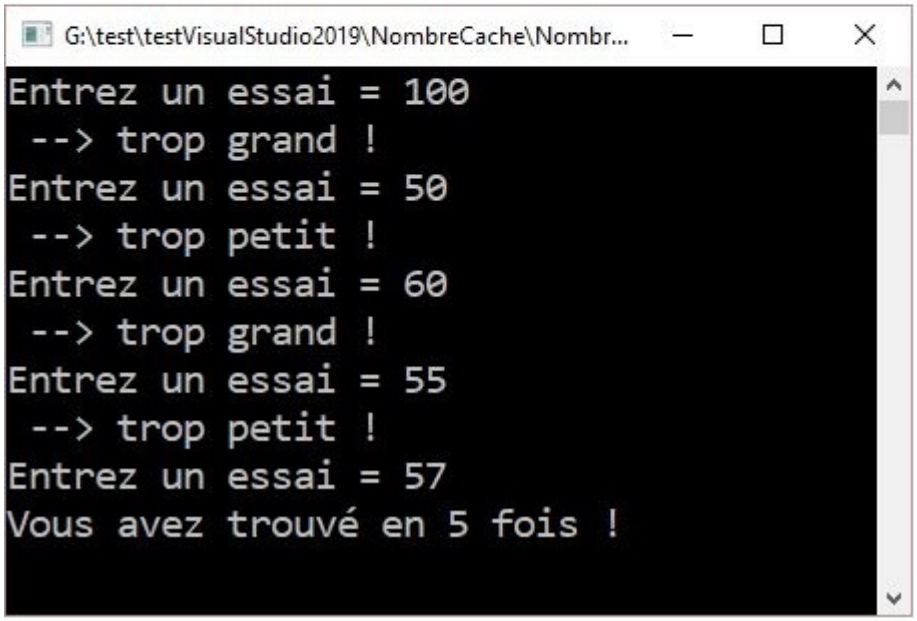
3.3.23 test

Cette fois, vous allez obtenir le bon message, avec le nombre d'essais affichés.

premier joueur :



écran effacé
second joueur :



Même image que l'écran précédent avec une étape supplémentaire (par rapport au code ajouté) : à côté de "vous avez trouvé", affichage de " en 5 fois !"
(le reste est en grisé)

3.3.24 Etape 7

3.3.25 Nettoyage du code

Rappelez-vous que du code a été généré automatiquement dès le début de la création du projet, en particulier les 'using' :

Les 'using' permettent d'accéder à certaines fonctionnalités.
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.

3.3.26 test

Vérifiez que le programme fonctionne toujours correctement.

```
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();
        }
    }
}
```

Code complet de l'applcation, avec une seule ligne de using.

3.3.27 Etape 8

3.3.28 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. Ceci dit, même si on va ajouter le cartouche dans

l'idée d'apporter une information complémentaire, la création de la documentation technique ne sera pas encore abordée. 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) et 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 :

Toutes ces modifications ne doivent rien changer à l'exécution du programme. Retrouvez à droite, le programme complet.

3.3.29 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();
        }
    }
}
```

Code complet de l'application avec en plus, tout en haut, le cartouche.

3.3.30 Etape 9

3.3.31 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 impossble 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 inatendus. 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 prticulier car le premier essai a été traité différemment, avant la boucle)
- saisir un caractère au lieu d'un nombre (comportement inatendu)

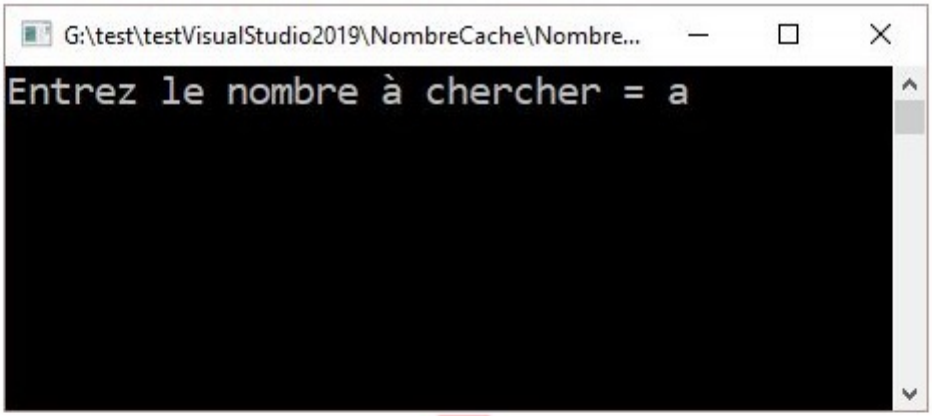
3.3.32 test

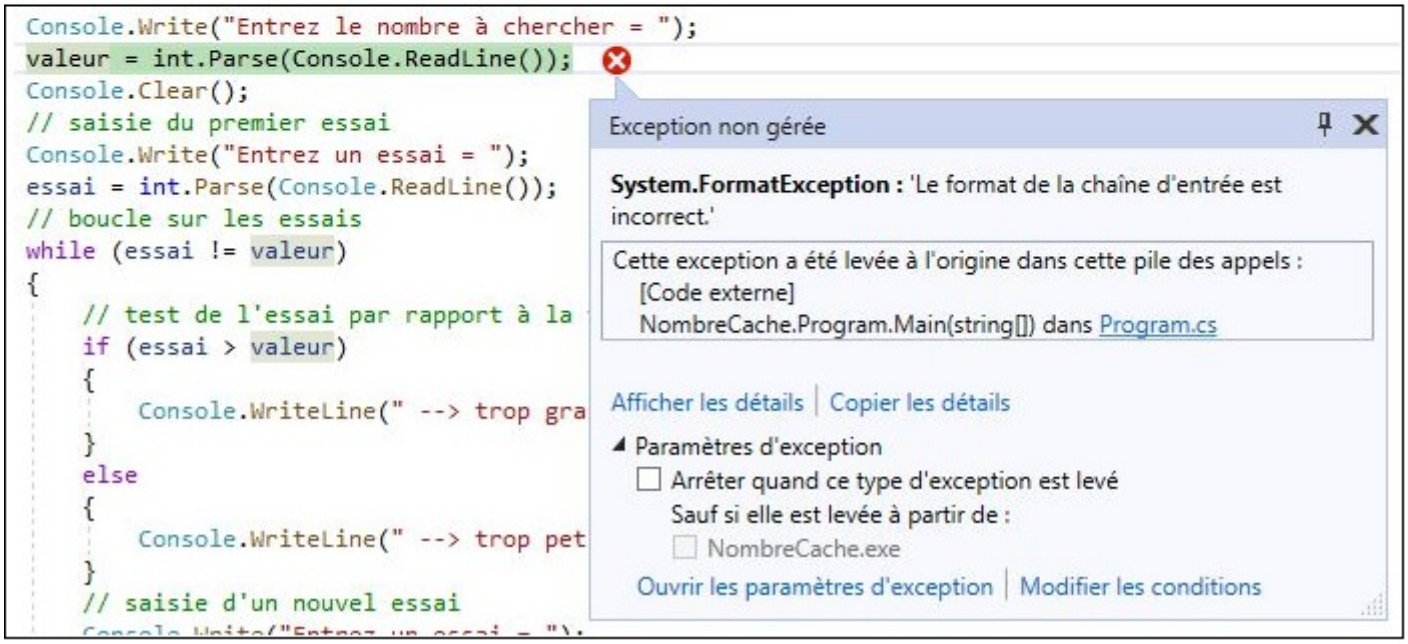
Essayez les 2 premiers cas qui ne doivent pas poser de problème. Essayez ensuite le dernier cas en tapant une lettre au lieu d'un nombre (dans la valeur, mais le problème sera identique dans l'essai).

Le programme s'arrête violemment et une erreur apparait directement dans le code (voir capture ci-contre). La ligne concernée est mise en évidence :

```
valeur = int.Parse(Console.ReadLine());
```

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 (voir explications juste après).





Capture 1 : console montrant "Entrez le nombre à chercher = " suivi de la saisie de la lettre "a".
Capture 2 : fenêtre de code avec une croix rouge sur la ligne concernée par l'erreur et une info-bulle donnant le message d'erreur.

3.3.33 Etape 9 (suite)

3.3.34 Tests de l'application (capture d'erreur)

Voilà comment fonctionne la capture d'erreur :

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') :

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').

3.3.35 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.

3.3.36 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 (dans la séquence 3) comment optimiser le code pour éviter ce genre de répétition.

```
G:\test\testVisualStudio2019\NombreCache\NombreCache\bin\Debug\Nombre...
Entrez le nombre à chercher = a
Erreur de saisie : saisissez une nombre entier
Entrez le nombre à chercher = 2.3
Erreur de saisie : saisissez une nombre entier
Entrez le nombre à chercher = 57
```

écran effacé

```
G:\test\testVisualStudio2019\NombreCache\NombreCache\bin\Debug\Nombre...
Entrez un essai = zzz
Erreur de saisie : saisissez une nombre entier
Entrez un essai = 100
 --> trop grand !
Entrez un essai = 50,5
Erreur de saisie : saisissez une nombre entier
Entrez un essai = 50
 --> trop petit !
Entrez un essai = 55.4
Erreur de saisie : saisissez une nombre entier
Entrez un essai = 57
Vous avez trouvé en 3 fois !
```

Captures montrant des tentatives de saisie (lettres et nombres à virgule) et aucune erreur de code mais à chaque fois le message "Erreur de saisie : saisissez un nombre entier".

3.3.37 Etape 10

3.3.38 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 opératoinnel, voyons comment fonctionne cet outil.

3.3.38.1 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.

3.3.38.2 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.

3.3.38.3 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.

3.3.39 Etape 10 (suite)

3.3.40 Outil de débogage (suite) :

3.3.40.1 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 :

3.3.40.2 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.

3.3.41 Téléchargement du code final

Si vous voulez, vous pouvez télécharger le fichier zip contenant le code complet du premier exemple, à l'étape finale.
Téléchargez l'application NombreCache.

4 Partie 2 - Application

4.1 Application

A partir de la situation professionnelle déjà présentée en début de séance, à votre tour désormais de réaliser en autonomie la mission qui vous a été confiée. Vous trouverez à la page suivante le rappel de la situation professionnelle, puis les étapes à suivre.

4.2 Votre tâche

```
G:\test\testVisualStudio2019\Calculs\Calculs\bin\Debug\Calculs.exe
Addition ........................ 1
Multiplication .................. 2
Quitter ......................... 0
Choix :                          1
4 + 7 = 11
Bravo !
Addition ........................ 1
Multiplication .................. 2
Quitter ......................... 0
Choix :                          1
7 + 5 = 13
Faux : 7 + 5 = 12
Addition ........................ 1
Multiplication .................. 2
Quitter ......................... 0
Choix :                          2
6 x 7 = 42
Bravo !
Addition ........................ 1
Multiplication .................. 2
Quitter ......................... 0
Choix :
```

Capture montrant l'exécution du programme :
- affichage du menu avec 3 choix (1 addition, 2 multiplication, 0 quitter)
- saisie du choix 1, proposition d'addition, saisie de la bonne réponse, affichage de "Bravo !"

- réaffichage du menu
- saisie du choix 1, proposition d'addition, saisie d'une réponse fausse, affichage de "Faux" suivi de la bonne réponse.
- réaffichage du menu
- saisie du choix 2, proposition de multiplication, saisie de la bonne réponse, affichage de "Bravo !"
- réaffichage du menu

4.2.1 Rappel de la situation professionnelle

4.2.1.1 Présentation du contexte :

L'entreprise cliente réalise des applications d'apprentissage et de tests de connaissances dans des environnements divers.

4.2.1.2 Présentation de l'application :

Un petite application est en cours de création. Dans un premier temps réalisée dans un environnement non graphique, elle permet de contrôler les connaissances en calculs mathématiques basiques (addition, multiplication).
Pour le moment, comme le montre la capture ci-contre, un menu s'affiche, offrant le choix de contrôler ses connaissances en addition ou en multiplication. Suivant le choix, une opération est proposée et il faut saisir la réponse. Si la réponse est correcte, "Bravo !" est affiché. Dans le cas contraire, "Faux" est affiché, suivi de la réponse correcte. Dans tous les cas, le menu est à nouveau affiché, permettant de faire un nouveau test.

4.2.1.3 Présentation des dysfonctionnements :

Suite à des tests réalisés à la demande du chef de projet, un ticket d'incidents a été généré et vous est confié (vous pouvez le télécharger ici*).
Vous êtes en charge de corriger les dysfonctionnements, puis de rendre le code corrigé avec le rapport d'incidents complété (rapport d'incidents à compléter téléchargeable ici).

*les dysfonctionnements signalés dans le ticket d'incidents, sont rappelés dans le rapport d'incidents.

4.3 Etapes à suivre

1. Récupérez le code source du projet ici
2. Ouvrez le projet sous l'IDE Visual Studio (en double cliquant sur le fichier sln).
3. Testez les fonctionnalités de base pour vérifier qu'elles sont opérationnelles.
4. Testez les manipulations à l'origine des dysfonctionnements mentionnés dans le ticket d'incidents (téléchargeable ici).
5. Utilisez le débogueur de l'IDE et réalisez une exécution pas à pas pour mieux cerner l'origine des dysfonctionnements.
6. Apportez les corrections nécessaires pour corriger les dysfonctionnements et obtenir les résultats attendus.

7. Compléter le rapport d'incidents (téléchargeable ici), en expliquant l'origine des dysfonctionnements et les corrections apportées (extrait du rapport d'incidents ci-contre). Votre responsable vous a aussi demandé de présenter les différents possibilités de correcitons, lorsqu'il y en avait plusieurs et, dans ce cas, de justifier le choix effectué (la solution retenue). Les informations seront mémiorisées dans une banque de données, dans le cas où le même type de dysfonctionnement apparaîtrait dans d'autres tickets d'incidents.
8. Rendre au client le projet corrigé et le rapport d'incidents complété (étape fictive) : le ticket d'incident est clôturé.

Votre responsable, qui a rapidement regardé le code, vous suggère de vous renseigner sur la structure switch qui permet de réaliser plusieurs alternatives et vous donne le lien suivant, dans le cas où cela pourrait vous servir.

Erreur	**1**
Test effectué	Au niveau du menu, saisie d'un chiffre autre que ceux proposés dans le menu.
Résultat obtenu	Exécution du contrôle de connaissances en multiplication.
Résultat souhaité	Affichage du message "erreur de saisie" et réaffichage du menu.
Origine du dysfonctionnement	
Corrections possibles	
Correction choisie (si plusieurs) avec justification	

Capture montrant à titre d'exemple, juste le tableau de l'erreur 1 du "Rapport d'incidents".

4.4 Correction

Vous pouvez désormais télécharger :

- le code de la correction du projet
- le rapport d'incident complété

5 Evaluer ses acquis

5.1 Evaluer ses acquis : IDE

Exercice 1 - Corrigé à la page 52

Un IDE offre les possibilités suivantes :

- ☐ La colorisation du code
- ☐ La correction automatique des erreurs syntaxiques
- ☐ Le signalement des erreurs syntaxiques
- ☐ La correction automatique des erreurs d'exécution
- ☐ Le signalement des erreurs d'exécution
- ☐ L'outil de débogage
- ☐ L'aide à la saisie (autocomplétion)

5.2 Evaluer ses acquis : débogage

Exercice 2 - Corrigé à la page 52

Au démarrage du débogage, que se passe-t-il ?

- ☐ L'exécution se lance normalement à partir du point d'arrêt
- ☐ L'exécution se lance normalement jusqu'au point d'arrêt
- ☐ L'exécution se fait pas à pas dès le début du programme
- ☐ L'exécution se fait pas à pas à partir du point d'arrêt

5.3 Evaluer ses acquis : ticket d'incidents

Exercice 3 - Corrigé à la page 52

Un ticket d'incidents contient :

- ☐ Le niveau de priorité de la résolution de l'incident
- ☐ L'explication du dysfonctionnement
- ☐ Les étapes à suivre pour résoudre l'incident

- ☐ La personne qui fait la demande
- ☐ Le détail des fonctionnalités à ajouter au programme

6 Compléter les savoirs

6.1 Compléter les savoirs

Pour compléter les savoirs abordés dans cette séance, vous pouvez maintenant consulter la **fiche de savoirs** sur *les bases de la programmation procédurale*. L'étude de cette fiche est **indispensable pour** pouvoir aborder la **séance suivante**.
Consultez la Fiche de savoirs - Les bases de la programmation procédurale.
Téléchargez les codes des corrections des exercices de la fiche de savoirs (consultables aussi à la fin de la Fiche).
Vous pouvez aussi retrouver la correction en vidéo de la plupart des exercices, sur Youtube.
Allez dans la playlist "Bases de la programmatopn (C#)" et suivez les exercices 1 à 30.

7 Synthèse

7.1 Synthèse de la séance

7.1.1 Ce qu'il faut retenir

Pour corriger des dysfonctionnements dans une application, voici le mode opératoire conseillé :

- ouvrir l'application dans un IDE professionnel pour pouvoir exploiter les outils intégrés (entre autres, le débogage) ;
- commencer par tester les les fonctionnalités opérationnelles ;
- reproduire les dysfonctionnements ;
- utiliser le débogueur pour suivre le déroulement du programme et l'évolution des variables, afin de repérer l'origine des dysfonctionnements ;
- apporter les corrections nécessaires ;
- tester l'application pour contrôler que les dysfonctionnements ont été corrigés ;
- remplir le rapport d'incident, le clôturer et rendre l'application opérationnelle.

Il est aussi conseillé de garder une trace de ces correctifs dans une banque de données.

7.1.2 Notions clés

- Ticket d'incident qui présente le ou les dysfonctionnements

- IDE pour coder dans un environnement professionnel
- Debogage pour la recherche d'erreurs (point d'arrêt, exécution pas à pas)

7.1.3 Pour approfondir

- Gestion des incidents (Wikipedia)
- Gérer vos incidents avec le référentiel ITIL sur GLPI (openclassrooms)

8 Glossaire

Affectation

Transfert d'une valeur, variable ou calcul dans une variable (avec compatibilité de type).
En C#, le symbole de l'affectation est '='.
Exemples :

```
correct = true;
nbre = nbre + 1; // raccourci d'écriture : nbre++;
variable1 = variable2;
```

Alternative

Ensemble d'instructions qui s'exécutent sous condition (synonymes : condition, test)
Exemple :

```
if (essai > valeur)
{
  Console.WriteLine(" --> trop grand !");
}
else
{
  Console.WriteLine(" --> trop petit !");
}
```

Application

Ensemble d'instructions formant un tout et pouvant s'exécuter pour répondre à une demande (synonyme : programme).

Autocompletion

Aide en cours de frappe qui affiche les différentes possibilités (mot du langage, variable...) par rapport à ce qui a commencé à être saisi.

Bloc de code

Ensemble d'intructions entourées par délimiteurs (en C#, des accolades).
Les différentes structures (programme, alternative, itératiion, ...) contiennent une bloc de code.

Commentaire

Information normalement ignorée par l'ordinateur, mais qui apporte des explicaitons en clair sur les fonctionnalités du programme.
Les commentaires peuvent être mis à n'importe quel endroit du code.
Il existe 3 types de commentaires. Les voici avec les syntaxes en C# :
Commentaire d'une ligne :

// ce qui suit le double slash est un commentaire, jusqu'à la fin de la ligne
Commentaire de plusieures lignes :
/* tot ce qui est entre le slash-étoile
et le étoile-slash, même sur plusieurs lignres
est un commentaire */
Commentaire de type cartouche :
C'est le seul interprété par l'ordinateur et il ne se positionne pas n'importe où : il est en début de programme (ou avant des modules : notion qui sera vue plus tard).

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

Compilation

Analyse du code pour repérer les erreurs syntaxiques (par exemple une variable utilisée mais non déclarée) puis, en l'absence d'erreur, traduction du code en langage "machine" directement compréhensible par l'ordinateur.

Concaténation

Construction d'une chaîne en ajoutant, les unes à la suite des autres, plusieurs chaînes (valeurs entre guillemets et/ou variables).

Condition

Prédicat qui est soit vrai, soit faux.
Une alternative (if, switch) contient une condition. Une irération (while...) contient une condition.

Débogage

Recherche de dysfonctionnements dans le programme.
La recherche doit se faire à 3 niveaux :
- tests des possibilités classiques (fonctionnement de base de l'application)
- tests des cas particuliers
- tests des comportements inattendus

Déclaration

Déclarer une variable consiste à lui donner un nom et un type. Cette étape est obligatoire en C# (et dans plusieurs langages) avant de pouvoir utiliser la variable.

IDE

Envirronnement de développement intégré (Integrated Development Environment).
Logiciel permettant de coder une application, avec un ensemble d'outils d'aide au codage (colorisation du code, aide en cours de frappe, débogage...).
Visual studio est un IDE.

Indentation

Décalage dans le code pour mettre en évidence les différents blocs.
Exemple :

```
if (essai > valeur)
{
```

```
  Console.WriteLine("Trop grand !")
}
```

Initialisation

Première affectation d'une valeur dans une variable

Instruction

Ordre que l'ordinateur va exécuter : affichage, saisie, affectation, test...

Itération

Ensemble d'instructions qui peuvent s'exécuter plusieurs fois, tant qu'une condition est respectée (synonyme : boucle).
Exemple :

```
while (essai != valeur)
{
  // instructions qui se répètent tant que la condition est vraie.
}
```

main

Module principal qui s'exécute automatiquement dès l'exécution de l'application.
Vous verrez par la suite qu'il existe d'autres types de modules (un module est un regroupement d'instructions qui représente une unité indépendante).

Pas à pas

Le débogueur permet une exécution "pas à pas", c'est à dire en s'arrêtant après chaque ligne (instruction) de code exécutée, pour montrer l'ordre d'exécution des instructions et l'évolution du contenu des variables.

Point d'arrêt

Marque placée sur une ligne du programme, repérée par le débogueur qui arrêtera l'exécution au niveau de cette ligne, permettant ensuite d'avancer pas à pas dans l'exécution du code.

Programme

Ensemble d'instructions formant un tout et pouvant s'exécuter pour répondre à une demande (synonyme : aplication).

Sensibilité à la casse

Distinction entre majuscule et minuscule.
Exemple : les variables 'total' et 'Total' sont différentes.

Test

Ensemble d'instructions qui s'exécutent sous condition (synonymes : alternative, condition)
Exemple :

```
if (essai > valeur)
{
  Console.WriteLine(" --> trop grand !");
}
else
{
  Console.WriteLine(" --> trop petit !");
}
```

Ticket d'incident

(synonyme : rapport d'incident)
Document qui recense des informations sur un dysfoncitonnement repéré (application concernée, demandeur, date, catégorie, niveau de priorité, description du dysfonctionnement...).
Le ticket d'incident est alors confié à un technicien qui a la responsabilité de trouver l'origine du dysfonctionnement et de le corriger. Il complète alors le ticket d'incident et le clôture après avoir rendu une version opérationnelle de l'application.

Transtypage

Changement de type d'une variable ou valeur (synoymes : parse, caste)
Exemples :
int essai = int.Parse(Console.ReadLine))
int essai = int.Parse("23")

Type

Format d'une case mémoire pouvant mémoriser une variable.
Types simples (avec équivalence en C#) :
- entier (int)
- réel (float)
- chaîne (string)
- booléen (bool)

Variable

Zone mémoire nommée et typée, permettant de stocker une information utilisable et modifiable dans le programmes.

Visual Studio

Suite de logiciels de développement pour Windows et mac OS conçue par Microsoft.

9 Documentation

9.1 Premier exemple

Application NombreCache

9.2 Situation professionnelle

Ressources

Corrections

9.3 Guide installation Visual Studio

Guide d'installation de Visual Studio

9.4 Fiche savoirs

Corrections

Programmation procédurale Partie 1 : les éléments syntaxique de base

9.5 Test

Test ZIP

10 Version imprimable

Titre de la ressource

Solutions

Exercice 1 - Page 45

Un IDE offre les possibilités suivantes :

- ■ La colorisation du code
- □ La correction automatique des erreurs syntaxiques
- ■ Le signalement des erreurs syntaxiques
- □ La correction automatique des erreurs d'exécution
- ■ Le signalement des erreurs d'exécution
- ■ L'outil de débogage
- ■ L'aide à la saisie (autocomplétion)

L'IDE offre de nombreux outils pour aider le développeur, et bien plus que ce qui est cité ici, mais ne va pas automatiquement corriger à sa place les erreurs syntaxiques et d'exécution.

Exercice 2 - Page 45

Au démarrage du débogage, que se passe-t-il ?

- □ L'exécution se lance normalement à partir du point d'arrêt
- ■ L'exécution se lance normalement jusqu'au point d'arrêt
- □ L'exécution se fait pas à pas dès le début du programme
- ■ L'exécution se fait pas à pas à partir du point d'arrêt

Au démarrage du débogage, l'exécution se fait normalement jusqu'à ce qu'elle rencontre un point d'arrêt. A partir de là, il est possible d'avancer "pas à pas" dans le programme (en exécutant une ligne de commande à la fois). Il est aussi possible, à tout moment, de demander de continuer l'exécution normalement : elle se fera alors jusqu'au prochain point d'arrêt ou jusqu'à la fin du programme s'il n'y a pas d'autre point d'arrêt rencontré.

Exercice 3 - Page 45

Un ticket d'incidents contient :

- ■ Le niveau de priorité de la résolution de l'incident
- ■ L'explication du dysfonctionnement
- □ Les étapes à suivre pour résoudre l'incident
- ■ La personne qui fait la demande
- □ Le détail des fonctionnalités à ajouter au programme

Le ticket d'incidents donne toutes les informations utiles sur le ou les dysfonctionnaments constatés et qui doivent être résolus. Les étapes de résolution ne sont pas données : c'est le technicien en charge de la résolution du problème qui doit suivre ses propres étapes ou celles imposées par son organisation. Un ticket d'incident n'est pas fait pour demander l'ajout de nouvelles fonctionnalités.