

Caractères posant problème avec le logiciel Visual Studio

En plus des caractères suivants, officiellement interdits par Windows dans un nom de fichier :

Un nom de fichier ne peut pas contenir les caractères suivants :
\\;*?"<>|

vous devez faire attention à ne pas utiliser le caractère **dièse « # »** dans un **nom de dossier ou de fichier** (cela pose problème autant pour le logiciel *Visual Studio* que pour les hyperliens et les raccourcis).

Etant donné qu'on programme en langage « Visual C# » (prononcez « vɪʒuəl sɪʃarp »), on pourrait être tenté de nommer un fichier ou un dossier « C# ». Au lieu de ceci, vous pouvez le nommer « CSharp », sachant que le signe *dièse* se traduit par *sharp* en anglais.

De plus, n'utilisez jamais d'**apostrophe « ' »** ni d'**espace « »** dans le logiciel *Visual Studio* dans un nom de **solution, namespace, classe, formulaire, fichier, etc** !

Introduction à l'informatique

Informatique

- Définition : traitement automatique de l'information.
- Exemple : conversion d'un texte manuscrit en document numérique à l'aide de la reconnaissance optique de caractères (OCR).

Programme

- Définition : série d'instructions que l'ordinateur exécute pour accomplir une tâche spécifique.
- Exemple : un programme de calculatrice qui effectue des opérations mathématiques.

Logiciel ou Application

- Définition : ensemble de programmes conçus pour réaliser des tâches spécifiques pour l'utilisateur.
- Exemple : Microsoft Word pour le traitement de texte, ou une application de messagerie comme WhatsApp.

Différents niveaux de complexité

- Programme exécuté sur :
 - un appareil électronique contenant un PIC
 - un ordinateur personnel (PC = personal computer)
 - un système d'exploitation (OS = operating system) avec interface graphique (GUI = graphical user interface) (p. ex. Windows)
 - une machine virtuelle (p. ex. Java)
 - une plateforme d'exécution (p. ex. .Net Framework)

Programmeur

- Définition : personne qui écrit et maintient des programmes informatiques.
- Rôle : créer des logiciels, résoudre des bugs, et améliorer les fonctionnalités.

Programmateur

- Définition : appareil participant à des tâches de programmation qui permet la copie ou la programmation des puces électroniques. Ce dispositif est utilisé pour changer l'état des mémoires non volatiles d'un circuit logique programmable tels que les PROM, EPROM, EEPROM, PIC, Flashes...
- Rôle : transférer le programme développé par le programmeur dans un microcontrôleur pour qu'il puisse être exécuté.
- Exemple : PICKit de Microchip

Utilisateur

- Définition : personne qui utilise le logiciel ou l'application.
- Rôle : interagir avec le programme pour accomplir des tâches, comme envoyer des emails ou jouer à des jeux.

Bêta-testeur

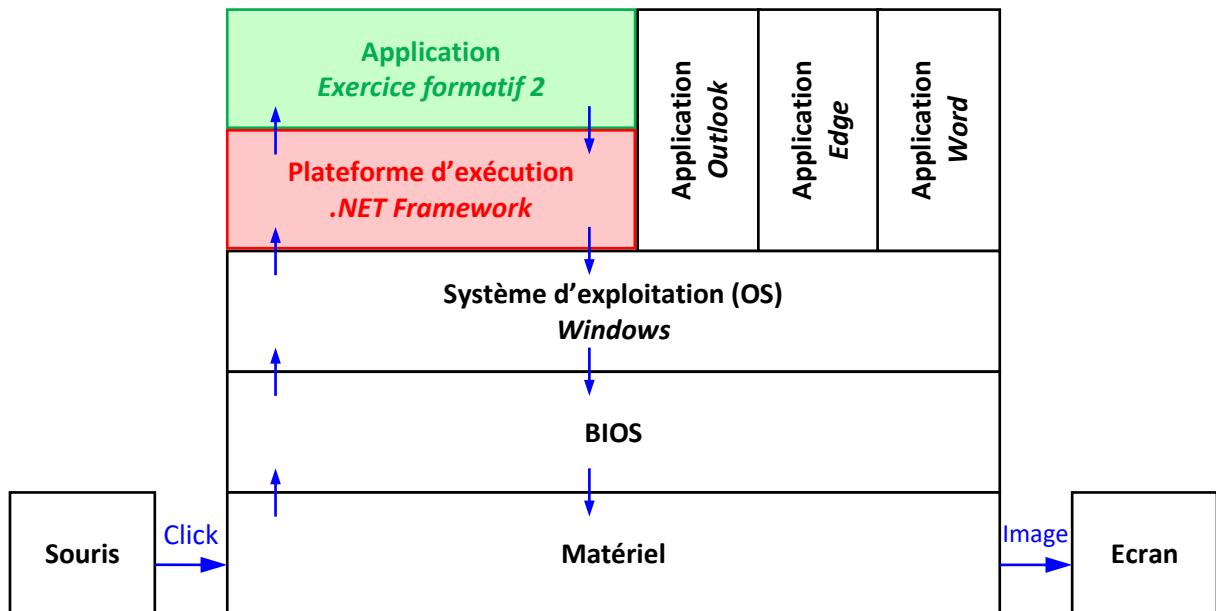
- Définition : personne qui teste les logiciels avant leur sortie officielle pour trouver et signaler des bugs.
- Rôle : utiliser le logiciel dans des conditions réelles pour aider à améliorer sa qualité avant la mise sur le marché.

Conclusion

- Vous allez endosser plusieurs rôles :
 1. Programmeur, pour concevoir des applications
 2. Bêta-testeur, pour vérifier les applications que vous aurez conçues
 3. Utilisateur, pour vous rendre compte de et améliorer l'ergonomie de vos applications
- Résumé : comprendre ces termes de base est essentiel pour débuter en informatique.

Couches logicielles avec .NET

L'utilisateur clique sur le bouton de la souris et l'application affiche une image à l'écran :



Quelques explications :

- En bleu : le flux des messages lorsque l'utilisateur clique sur le bouton de la souris et que l'**application** (en vert) affiche un résultat dans un label visible sur l'écran
- BIOS : Basic Input Output System (système élémentaire d'entrée/sortie) c'est un ensemble de fonctions, contenu dans la mémoire morte (ROM ou Flash-EEPROM) de la carte mère d'un ordinateur, lui permettant d'effectuer des opérations élémentaires lors de sa mise sous tension, par exemple la lecture d'un secteur sur un disque. Par extension, le terme est souvent utilisé pour décrire l'ensemble du micrologiciel de la carte mère. Il fournit un ensemble de services permettant de faire le plus possible abstraction de la couche matérielle et fait ainsi le lien entre les opérations « standardisées » de l'OS et les opérations « individuelles » du matériel. Plus d'infos sur http://fr.wikipedia.org/wiki/Basic_Input_Output_System
- OS : Operating System (système d'exploitation) c'est un ensemble de programmes qui sert d'interface entre la partie matérielle et les applications. Plus d'infos sur http://fr.wikipedia.org/wiki/Système_d'exploitation
- **.NET Framework** (dire « dot NET ») (en rouge) : nom de la plateforme d'exécution développée par Microsoft pour exécuter des applications .NET (qu'on développe avec Visual Studio). Elle a pour but de faciliter la tâche des développeurs en proposant une approche unifiée à la conception d'applications Windows ou Web. Elle permet une exécution dans un environnement sécurisé et une compatibilité avec différents systèmes d'exploitation. Plus d'infos sur http://fr.wikipedia.org/wiki/Framework_.NET

Qu'est-ce qu'un langage de programmation

- L'**humain** parle un langage qui lui est propre (français, anglais, Schwiizertüütsch, ...)
- Le **processeur** comprend un langage qui lui est propre (le langage machine, du binaire)
- Un **langage de programmation** est un langage compréhensible par l'humain **ET** par le processeur (après avoir été traduit automatiquement)
- Niveaux langagiers :

Niveau	Application au langage naturel
Lexical	Vocabulaire, orthographe
Syntaxique	Grammaire, règles
Sémantique	Signification, sens de la phrase
Pragmatique	Interprétation en tenant compte du contexte

- Langage de programmation « de **bas niveau** » ([Wiki](#)) :
 - Exemple : assembleur ([Wiki](#))
 - 1 instruction en assembleur = 1 instruction en langage machine
 - Difficilement compréhensible par l'humain, ne nécessite aucune traduction, compréhensible par le processeur
 - Dépendant du matériel
- Langage de programmation « **évolué** » ([Wiki](#))
 - Exemples que vous connaissez : C, VC#
 - 1 instruction en langage évolué = plusieurs instructions en langage machine
 - Facilement compréhensible par l'humain, nécessite une traduction (automatique) pour le processeur
 - Proche des langues parlées (anglais)
 - Indépendant du matériel (grâce à une HAL [Wiki](#))
 - Procédural : il suffit de décrire les opérations à effectuer pour résoudre un problème

- Pour que le code source soit compréhensible par **plusieurs** humains, il faut y ajouter des **commentaires** !
- Processus de **traduction** :
 - **Compilation** (traduction de tout le code source en 1 fois, lors du développement, par l'environnement de programmation) à la manière d'un traducteur qui traduit une lettre
 - **Interprétation** (traduction instruction par instruction, lors de l'exécution, par l'environnement d'exécution)
à la manière d'un interprète qui traduit un dialogue en direct
 - Différences : **vitesse de traduction**, **vitesse d'exécution**, **portabilité**
- Types de langage (paradigme de programmation [Wiki](#)) :
 - Programmation procédurale (langages **structurés** : **C**, Pascal, Modula, Basic, Ada, Cobol, Delphi, Fortran, Perl, PHP, ...)
 - Programmation par objet (langages **orientés objets** : **VC#**, Smalltalk, Java, C++, Eiffel, Python, ...)

Comment...

- **écrire un code source en langage évolué**
- **le traduire en langage machine**
- **le faire exécuter par un processeur**
- Environnement de développement intégré ([IDE](#))
(IDE connus : Microsoft Visual Studio, Microchip MPLAB)
 - Editeur de code source (et éditeur visuel)
 - Bibliothèques
 - Editeur de liens
 - Vérifications lexicale et syntaxique
 - Compilateur
 - Débogueur (vérification sémantique) en mode pas-à-pas
(bug = insecte, vermine : [Babbage](#))
 - Générateur de fichier exécutable (+ déploiement)

3 questions à Copilot (GPT-5)

Quelles sont les IA spécialisées dans la génération de code source informatique ?

Copilot :

Il existe plusieurs IA spécialisées dans la génération de code source, allant des modèles généralistes aux outils très ciblés pour certaines tâches. Voici les principales catégories et exemples :

1. Modèles généralistes pour la génération de code

Ces IA sont capables de générer du code dans plusieurs langages et pour divers contextes :

- **GitHub Copilot** (basé sur OpenAI Codex) : intégré à VS Code, GitHub, etc. Très utilisé pour l'assistance en temps réel.
- **OpenAI GPT-4/5 avec Code Interpreter** : capable de générer, corriger et expliquer du code dans de nombreux langages.
- **Google Gemini (anciennement Bard)** : propose aussi des fonctionnalités de génération et d'explication de code.
- **Anthropic Claude** : bon pour la génération de code avec explications détaillées.

2. Outils spécialisés pour le développement logiciel

- **Tabnine** : IA orientée complétion de code, optimisée pour la productivité.
- **Codeium** : alternative à Copilot, avec support multi-langages.
- **Amazon CodeWhisperer** : conçu pour les développeurs AWS, avec intégration cloud.

3. IA pour la génération de code à partir de spécifications

- **AlphaCode (DeepMind)** : orienté résolution de problèmes complexes (compétitions de programmation).
- **Codex** : très performant pour transformer des instructions en code exécutable.

4. IA pour la génération de code bas-niveau ou optimisé

- **ChatGPT avec mode développeur** : pour scripts, automatisations, et optimisation.
- **LLM spécialisés en C/C++ ou Rust** (ex. StarCoder, CodeGen).

5. IA pour la génération de code visuel ou low-code

- **Builder.ai, OutSystems, Mendix** : génèrent des applications sans écrire tout le code manuellement.

Est-il intelligent d'utiliser une IA pour générer du code source ?

Copilot :

Très bonne question ! **Utiliser une IA pour générer du code peut être intelligent... mais sous certaines conditions.** Voici une analyse complète :

Avantages

1. Gain de temps

- Génération rapide de boilerplate, tests unitaires, scripts répétitifs.
- Automatisation des tâches simples (CRUD, configuration CI/CD).

2. Accessibilité

- Permet aux non-développeurs de créer des prototypes.
- Aide à apprendre en proposant des exemples concrets.

3. Productivité accrue

- Les IA comme Claude, Copilot ou Codeium comprennent le contexte et suggèrent des solutions optimisées.
- Réduction des erreurs syntaxiques.

⚠ Risques et limites

1. Qualité et sécurité

- Le code généré peut contenir des failles (injections, mauvaises pratiques).
- Nécessité d'une **revue humaine** avant mise en production.

2. Dépendance excessive

- Risque de perte de compétences si on ne comprend pas le code généré.
- Difficulté à maintenir le code sans IA.
- Trop s'appuyer sur l'IA peut freiner l'apprentissage et la créativité.

3. Compréhension

- Si on ne comprend pas le code, on risque des bugs ou des failles.

4. Propriété intellectuelle

- Certains outils peuvent s'appuyer sur des données open source avec licences spécifiques.

🔍 Quand est-ce intelligent ?

• Oui, si :

- Tu l'utilises pour **accélérer** des tâches répétitives.
- Tu **valide et teste** le code avant déploiement.
- Tu gardes une **compréhension claire** du fonctionnement.

• Non, si :

- Tu délègues **entièvement** la logique métier à l'IA sans contrôle.
- Tu ignores les aspects **sécurité et conformité**.

✓ Bonnes pratiques

- **Toujours relire et tester** le code généré.
- **Utiliser l'IA comme assistant**, pas comme remplaçant.
- **Compléter avec des revues de code** et des tests automatisés.
- **Former les étudiants/développeurs** à comprendre ce qui est généré.

👉 Conclusion : L'IA est un **assistant puissant**, mais pas un remplaçant. Elle doit être intégrée dans un workflow avec **supervision humaine**, tests et bonnes pratiques.

Est-il indiqué ou contre-indiqué d'utiliser une IA pour générer du code source en tant qu'apprenant débutant en programmation informatique ?

Copilot :

Excellente question ! La réponse dépend de **l'objectif pédagogique** et de **la manière dont l'IA est utilisée**. Voici une analyse claire :

Indiqué (avantages pour un débutant)

1. **Apprentissage par l'exemple**
 - Voir du code généré aide à comprendre la syntaxe et les structures.
 - Permet de comparer ses propres solutions avec celles proposées par l'IA.
 2. **Gain de temps sur les tâches répétitives**
 - Génération de boilerplate (ex. configuration de projet).
 - Focus sur la logique plutôt que sur les détails syntaxiques.
 3. **Découverte de bonnes pratiques**
 - Certains outils proposent des solutions optimisées et commentées.
 4. **Aide à la correction**
 - L'IA peut identifier des erreurs et suggérer des améliorations.
 5. **Feedback immédiat**
 - L'IA peut expliquer le code qu'elle génère, ce qui renforce la compréhension.
-

Contre-indiqué (risques pour un débutant)

1. **Apprentissage passif**
 - Si tu te contentes de copier-coller sans comprendre, tu ne développes pas tes compétences.
 - Risque de dépendance : tu ne sauras pas coder sans IA.
 - Si on se contente de copier-coller sans comprendre, on n'apprend pas réellement.
 2. **Manque de réflexion algorithmique**
 - L'apprentissage de la logique est essentiel avant d'automatiser.
 3. **Compréhension superficielle**
 - L'IA peut masquer la complexité, ce qui freine la capacité à résoudre des problèmes seul.
 4. **Complexité prématuée**
 - L'IA peut générer du code trop avancé (design patterns, optimisations) qui dépasse ton niveau actuel.
 5. **Qualité variable**
 - Le code généré peut contenir des erreurs ou ne pas respecter les standards.
 6. **Erreurs non détectées**
 - Sans savoir tester ou relire, tu peux intégrer des bugs ou des failles.
-

Bonne pratique pour un débutant

- **Utiliser l'IA comme un tuteur, pas comme un « fournisseur de solutions » :**
 - Demande à l'IA d'**expliquer chaque ligne**.
 - **Essaie d'écrire toi-même avant de demander une solution.**
 - Utilise-la pour **corriger ton code**, pas pour tout faire à ta place.
 - **Toujours analyser et commenter le code généré** pour comprendre chaque ligne.
 - **Commencer par écrire soi-même**, puis comparer avec la solution IA.
-

Conclusion :

Indiqué si tu l'utilises pour apprendre activement (comprendre, expérimenter, poser des questions). Si l'IA est utilisée comme un outil pédagogique et non comme une béquille.

Elle doit servir à apprendre, pas à remplacer l'effort cognitif.

Contre-indiqué si tu l'utilises pour éviter d'apprendre (copier-coller sans réflexion).

Comment concevoir une solution informatique

(voir l'exemple «Comment concevoir une solution informatique – Exemple»)

1. Bien **analyser** et comprendre le problème (pas la solution !)
2. Réaliser **à la main sur papier**, les étapes nécessaires à trouver le résultat demandé
3. **Analyser** les étapes réalisées à la main
4. Décrire en **mots-clés** une **démarche** à appliquer à n'importe quel cas pour atteindre le résultat demandé
5. Concevoir une solution sous forme d'**algorithme** (**structogramme GNS** ou JSP, pseudo-code, etc.)
6. On peut **affiner** ensuite le structogramme pour tenir compte des **spécificités du langage** utilisé pour le codage qui suivra
7. **Tester** le fonctionnement de l'algorithme (structogramme) **sur papier** en passant par tous les chemins possibles

8. Traduire cet algorithme sous forme d'un **programme** en un langage de programmation adéquat (**commenter** le programme pour qu'il soit compréhensible)
 9. Faire **traduire** le programme du langage évolué au langage compréhensible par le processeur (compiler)
 10. Faire **exécuter** ce programme par le processeur (déboguer)
 11. **Tester** le programme en remplissant un **protocole**
- Attention à **ne pas faire plus** que ce qui est demandé dans le cahier des charges. Les fonctions superflues ont pris du temps de développement, donc du travail, donc un certain coût.
Si un client demande un logiciel (ou un appareil) qui doit comporter 5 fonctions et que vous lui en développez un qui en comporte 10, même si elles sont utiles, il va vous dire que la moitié de votre travail n'était pas demandé et va vous payer la moitié !
Si vous avez de bonnes idées d'amélioration, **proposez-les** au client et **faites modifier le cahier des charges** avant de les implémenter

Comment concevoir une solution informatique - Exemple

Ceci est un exemple illustrant quelques points de la théorie exposée dans le document « *Comment concevoir une solution informatique* ».

1. Bien **analyser et comprendre** le problème (pas la solution !) :

On me demande de créer une application pour résoudre l'équation du 2^{ème} degré. Je devrai appliquer la méthode vue en cours de math pour calculer les 2 racines.

2. Réaliser **à la main sur papier**, les étapes nécessaires à trouver le résultat demandé :

Créez une application permettant de résoudre une équation du 2^{ème} degré

Exemple : $5x^2 - 3x + 4 = 0$, combien vaut x ?
 $a \ b \ c$

$$\Delta = b^2 - 4ac = (-3)^2 - 4 \cdot 5 \cdot 4 = -71$$

Si $\Delta < 0$ alors pas de solution réelle

2^e exemple : $2y^2 + 3y - 4 = 0$
 $a \ b \ c$

$$\Delta = b^2 - 4 \cdot a \cdot c = 3^2 - 4 \cdot 2 \cdot (-4) = 41$$

Si $\Delta > 0$ alors 2 solutions :

$$y_{1,2} = \frac{-b \pm \sqrt{\Delta}}{2a}$$

$$Y_1 = \frac{-3 + \sqrt{41}}{2 \cdot 2} = 0,851$$

$$Y_2 = \frac{-3 - \sqrt{41}}{2 \cdot 2} = -2,351$$

3^e exemple : $z^2 + 4z + 4 = 0$
 $a \ b \ c$

$$\Delta = b^2 - 4ac = 4^2 - 4 \cdot 1 \cdot 4 = 0$$

Si $\Delta = 0$ alors 1 solution :

$$z = -\frac{b}{2a} = -\frac{4}{2 \cdot 1} = -2$$

3. Analyser les étapes réalisées à la main :

J'ai déterminé les coefficients a, b et c

Ensuite j'ai calculé le discriminant selon une formule connue

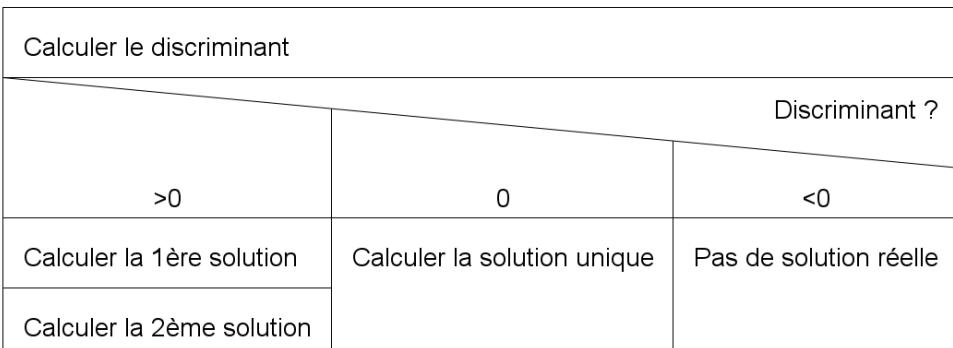
Selon que le discriminant est positif, nul ou négatif, j'ai appliqué la bonne formule pour calculer le résultat, qui peut être soit 2 racines, soit 1 racine, soit aucune racine réelle

4. Décrire en **mots-clés** une **démarche** à appliquer à n'importe quel cas pour atteindre le résultat demandé :

- 1) Demander les valeurs des coefficients a, b et c
- 2) Calculer le discriminant
- 3) Si le discriminant est positif, alors calculer les 2 racines
Si le discriminant est nul, alors calculer la racine unique
Si le discriminant est négatif, alors il n'y a pas de racine réelle

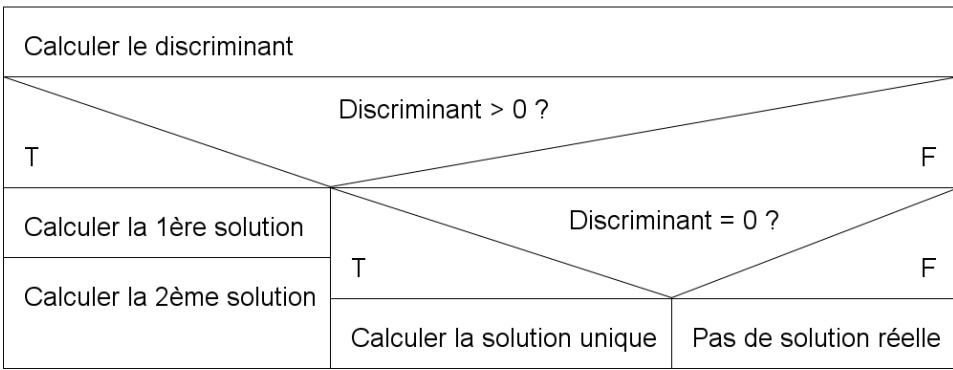
5. Concevoir une solution sous forme d'**algorithme** (structogramme [GNS](#) ou [JSP](#), [pseudo-code](#), etc.) :

Equation du 2e degré



6. On peut **affiner** ensuite le structogramme pour tenir compte des **spécificités du langage** utilisé pour le codage qui suivra (p. ex. la structure **switch case** n'étant pas très efficace en C# V7.3, il est préférable d'utiliser des **if** imbriqués) :

Equation du 2e degré



7. **Tester** le fonctionnement de l'algorithme (structogramme) **sur papier** en passant par tous les chemins possibles

8. Traduire cet algorithme sous forme d'un **programme** en un langage de programmation adéquat (**commenter** le programme pour qu'il soit compréhensible)

Bien utilisé¹, le logiciel Structorizer permet de générer le code source en C# (la structure et commentaires) qui correspond au structogramme et de copier la partie qui nous intéresse :

```
// Generated by Structorizer 3.32-34

/// <summary>
/// Equation du 2e degré
/// </summary>
public static void Equation_du_2e_degré()
{
    // TODO: Check and accomplish variable declarations:

    // Calculer le discriminant
    ?;
    // Discriminant > 0 ?
    if (?)
    {
        // Calculer la première solution
        ?;
        // Calculer la deuxième solution
        ?;
    }
    // Discriminant = 0 ?
    else if (?)
    {
        // Calculer la solution unique
        ?;
    }
    else
    {
        // Pas de solution réelle
        ?;
    }
}
```

Et la coller dans la solution Application Windows Forms (.NET Framework) écrite en VC# :

```
/* Sujet : Résoudre l'équation du 2e degré 0 = a*x^2 + b*x + c
 * Auteur : Michel Berset
 * Date : 10.09.2021
 * Dernière modif. : 09.12.2024
 * Structogrammes : Equation 2e degré.nsd
 */
using ...
namespace EquationDu2eDegree
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void btnCalculer_Click(object sender, EventArgs e)
        {
            // Demander à l'utilisateur les valeurs a, b et c
            // Calculer le discriminant
            ?;
            // Discriminant > 0 ?
            if (?)
            {
                // Calculer la première solution
                ?;
                // Calculer la deuxième solution
                ?;
            }
            // Discriminant = 0 ?
            else if (?)
            {
                // Calculer la solution unique
                ?;
            }
            else
            {
                // Pas de solution réelle
                ?;
            }
            // Afficher les solutions
        }
    }
}
```

Il reste ensuite à écrire du code correspondant aux commentaires, partout où il y a des points d'interrogation

¹ Voir document « Utiliser Structorizer pour faciliter le passage au code »

9. Faire **traduire** le programme du langage évolué au langage compréhensible par le processeur (compiler)

10. Faire **exécuter** ce programme par le processeur (déboguer)

11. **Tester** le programme en remplissant un **protocole**

Comment concevoir une solution informatique - Exemple

Ceci est un exemple illustrant quelques points de la théorie exposée dans le document « *Comment concevoir une solution informatique* ».

1. Bien **analyser et comprendre** le problème (pas la solution !) :

On me demande de créer une application pour résoudre l'équation du 2^{ème} degré. Je devrai appliquer la méthode vue en cours de math pour calculer les 2 racines.

2. Réaliser **à la main sur papier**, les étapes nécessaires à trouver le résultat demandé :

Créez une application permettant de résoudre une équation du 2^{ème} degré

Exemple : $5x^2 - 3x + 4 = 0$, combien vaut x ?
 $a \ b \ c$

$$\Delta = b^2 - 4ac = (-3)^2 - 4 \cdot 5 \cdot 4 = -71$$

Si $\Delta < 0$ alors pas de solution réelle

2^e exemple : $2y^2 + 3y - 4 = 0$
 $a \ b \ c$

$$\Delta = b^2 - 4 \cdot a \cdot c = 3^2 - 4 \cdot 2 \cdot (-4) = 41$$

Si $\Delta > 0$ alors 2 solutions :

$$y_{1,2} = \frac{-b \pm \sqrt{\Delta}}{2a}$$

$$Y_1 = \frac{-3 + \sqrt{41}}{2 \cdot 2} = 0,851$$

$$Y_2 = \frac{-3 - \sqrt{41}}{2 \cdot 2} = -2,351$$

3^e exemple : $z^2 + 4z + 4 = 0$
 $a \ b \ c$

$$\Delta = b^2 - 4ac = 4^2 - 4 \cdot 1 \cdot 4 = 0$$

Si $\Delta = 0$ alors 1 solution :

$$z = -\frac{b}{2a} = -\frac{4}{2 \cdot 1} = -2$$

3. Analyser les étapes réalisées à la main :

J'ai déterminé les coefficients a, b et c

Ensuite j'ai calculé le discriminant selon une formule connue

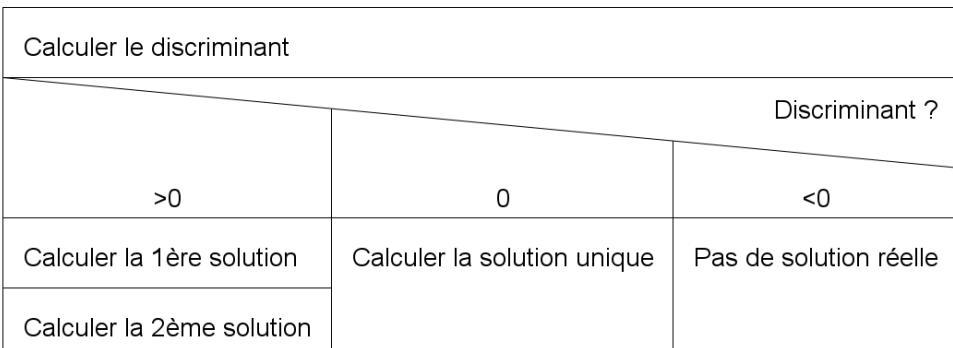
Selon que le discriminant est positif, nul ou négatif, j'ai appliqué la bonne formule pour calculer le résultat, qui peut être soit 2 racines, soit 1 racine, soit aucune racine réelle

4. Décrire en **mots-clés** une **démarche** à appliquer à n'importe quel cas pour atteindre le résultat demandé :

- 1) Demander les valeurs des coefficients a, b et c
- 2) Calculer le discriminant
- 3) Si le discriminant est positif, alors calculer les 2 racines
Si le discriminant est nul, alors calculer la racine unique
Si le discriminant est négatif, alors il n'y a pas de racine réelle

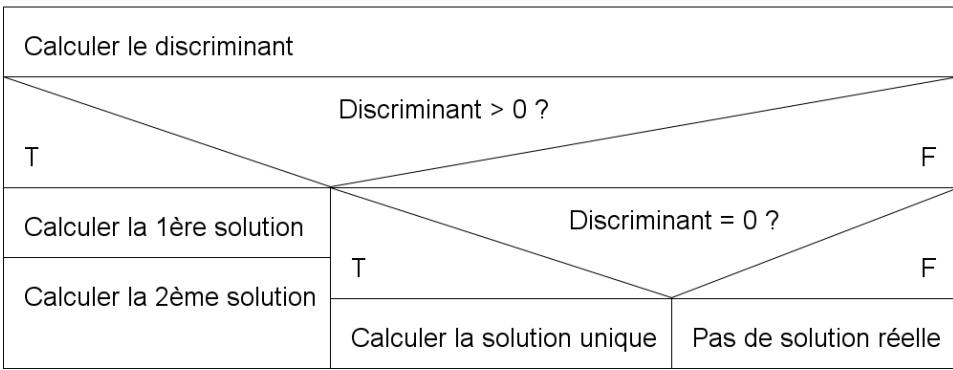
5. Concevoir une solution sous forme d'**algorithme** (structogramme [GNS](#) ou [JSP](#), [pseudo-code](#), etc.) :

Equation du 2e degré



6. On peut **affiner** ensuite le structogramme pour tenir compte des **spécificités du langage** utilisé pour le codage qui suivra (p. ex. la structure **switch case** n'étant pas très efficace en C# V7.3, il est préférable d'utiliser des **if** imbriqués) :

Equation du 2e degré



7. **Tester** le fonctionnement de l'algorithme (structogramme) **sur papier** en passant par tous les chemins possibles

8. Traduire cet algorithme sous forme d'un **programme** en un langage de programmation adéquat (**commenter** le programme pour qu'il soit compréhensible)

Bien utilisé¹, le logiciel Structorizer permet de générer le code source en C# (la structure et commentaires) qui correspond au structogramme et de copier la partie qui nous intéresse :

```
// Generated by Structorizer 3.32-34

/// <summary>
/// Equation du 2e degré
/// </summary>
public static void Equation_du_2e_degré()
{
    // TODO: Check and accomplish variable declarations:

    // Calculer le discriminant
    ?;
    // Discriminant > 0 ?
    if (?)
    {
        // Calculer la première solution
        ?;
        // Calculer la deuxième solution
        ?;
    }
    // Discriminant = 0 ?
    else if (?)
    {
        // Calculer la solution unique
        ?;
    }
    else
    {
        // Pas de solution réelle
        ?;
    }
}
```

Et la coller dans la solution Application Windows Forms (.NET Framework) écrite en VC# :

```
/* Sujet : Résoudre l'équation du 2e degré 0 = a*x^2 + b*x + c
 * Auteur : Michel Berset
 * Date : 10.09.2021
 * Dernière modif. : 09.12.2024
 * Structogrammes : Equation 2e degré.nsd
 */
using ...
namespace EquationDu2eDegree
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void btnCalculer_Click(object sender, EventArgs e)
        {
            // Demander à l'utilisateur les valeurs a, b et c
            // Calculer le discriminant
            ?;
            // Discriminant > 0 ?
            if (?)
            {
                // Calculer la première solution
                ?;
                // Calculer la deuxième solution
                ?;
            }
            // Discriminant = 0 ?
            else if (?)
            {
                // Calculer la solution unique
                ?;
            }
            else
            {
                // Pas de solution réelle
                ?;
            }
            // Afficher les solutions
        }
    }
}
```

Il reste ensuite à écrire du code correspondant aux commentaires, partout où il y a des points d'interrogation

¹ Voir document « Utiliser Structorizer pour faciliter le passage au code »

9. Faire **traduire** le programme du langage évolué au langage compréhensible par le processeur (compiler)

10. Faire **exécuter** ce programme par le processeur (déboguer)

11. **Tester** le programme en remplissant un **protocole**

Utilisation du logiciel Structorizer pour faciliter le passage au code

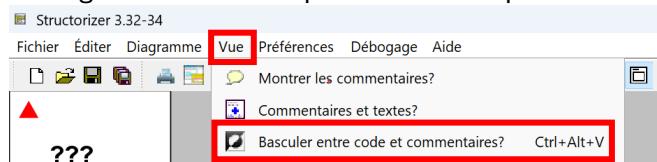
Résumé :

1. On peut utiliser le logiciel Structorizer pour générer automatiquement le code source contenant la structure et les commentaires.
2. Une utilisation plus avancée, mais plus compliquée car elle nécessite d'apprendre la syntaxe, permet de tester le structogramme et de générer le code source contenant la structure et les commentaires, mais également les instructions.

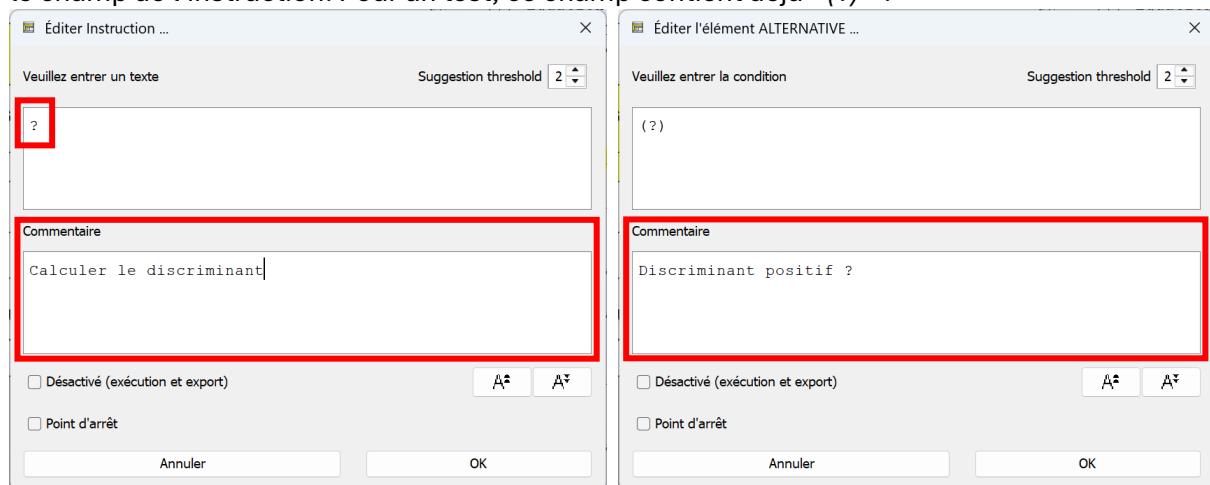
Le code source généré doit tout de même encore être adapté, il ne suffit pas de faire un copier-coller.

1. Utilisation simple de Structorizer pour générer la structure et les commentaires

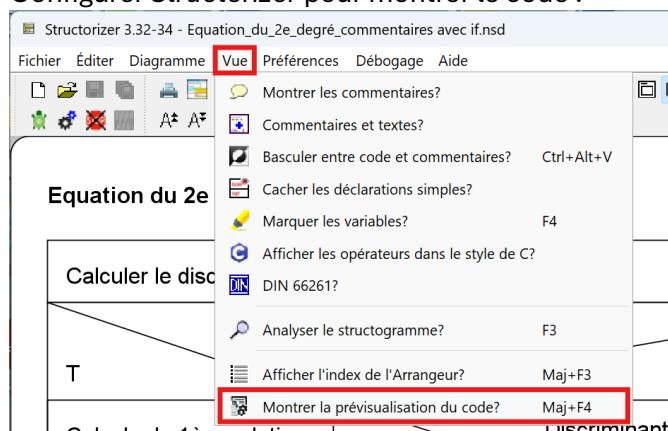
Configurer Structorizer pour n'afficher que les commentaires :



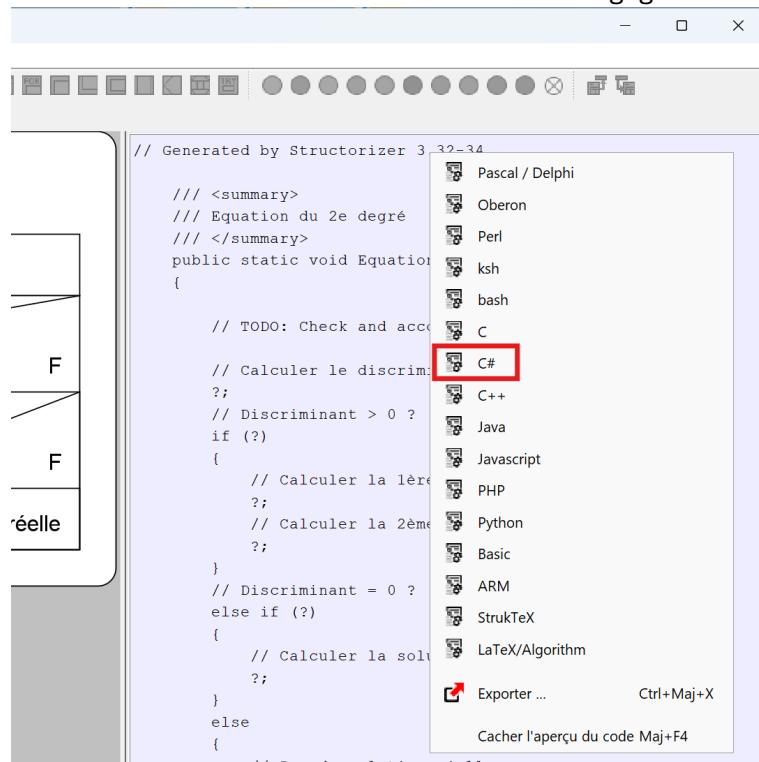
Dans les éléments, ne remplir que le champ **Commentaire**, éventuellement mettre un « ? » dans le champ de l'instruction. Pour un test, ce champ contient déjà « (?) » :



Configurer Structorizer pour montrer le code :



Clic-droit sur la fenêtre du code et choisir le langage C# :



Le code source en C# contient la structure et les commentaires correspondant au structogramme :

The screenshot shows the Structorizer interface with a structogram on the left and its generated C# code on the right. The structogram illustrates the logic for solving a quadratic equation based on the discriminant value (T for True, F for False). The C# code generated by Structorizer is as follows:

```
// Generated by Structorizer 3.32-34
/// <summary>
/// Equation du 2e degré
/// </summary>
public static void Equation_du_2e_degré()
{
    // TODO: Check and accomplish variable declarations:
    // Calculer le discriminant
    // Discriminant > 0 ?
    if (?)
    {
        // Calculer la 1ère solution
        ?;
        // Calculer la 2ème solution
        ?;
    }
    // Discriminant = 0 ?
    else if (?)
    {
        // Calculer la solution unique
        ?;
    }
    else
    {
        // Pas de solution réelle
        ?;
    }
}
```

Sélectionner la partie intéressante du code et la copier avec [ctrl]+[C] :

The diagram illustrates the process of extracting code from a Structorizer-generated summary. On the left, a flowchart titled "Equation du 2e degré" shows the logic for solving a quadratic equation based on the discriminant value. The right side shows the corresponding C# code generated by Structorizer. A red box highlights the section of code that corresponds to the flowchart's logic for calculating solutions based on the discriminant value.

```
// Generated by Structorizer 3.32-34
/// <summary>
/// Equation du 2e degré
/// </summary>
public static void Equation_du_2e_degré()
{
    // TODO: Check and accomplish variable declarations:
    // Calculer le discriminant
    // Discriminant > 0 ?
    if (?)
    {
        // Calculer la 1ère solution
        ?;
        // Calculer la 2ème solution
        ?;
    }
    // Discriminant = 0 ?
    else if (?)
    {
        // Calculer la solution unique
        ?;
    }
    else
    {
        // Pas de solution réelle
        ?;
    }
}
```

Et la coller dans la solution Application Windows Forms (.NET Framework) écrite en VC# avec [ctrl]+[V] suivi de [ctrl]+[Z] pour l'indenter correctement :

The screenshot shows the Visual Studio IDE with the "Form1.cs" code editor open. The copied code from Structorizer has been pasted into the editor. A red box highlights the same section of code as in the previous diagram, showing the correctly indented C# code intended for the "btnCalcule_Click" event handler.

```
/* Sujet : Résoudre l'équation du 2e degré 0 = a*x*x + b*x + c
 * Auteur : Michel Berset
 * Date : 10.09.2021
 * Dernière modif. : 09.12.2024
 * Structogrammes : Equation 2e degré.nsd
 */
using ...
namespace EquationDu2eDegree
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void btnCalcule_Click(object sender, EventArgs e)
        {
            // Demander à l'utilisateur les valeurs a, b et c

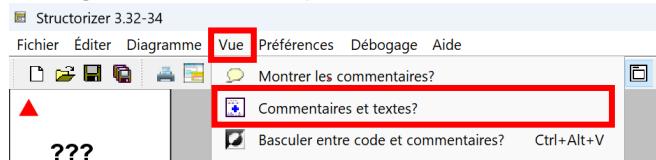
            // Calculer le discriminant
            ?;
            // Discriminant > 0 ?
            if (?)
            {
                // Calculer la 1ère solution
                ?;
                // Calculer la 2ème solution
                ?;
            }
            // Discriminant = 0 ?
            else if (?)
            {
                // Calculer la solution unique
                ?;
            }
            else
            {
                // Pas de solution réelle
                ?;
            }

            // Afficher les solutions
        }
    }
}
```

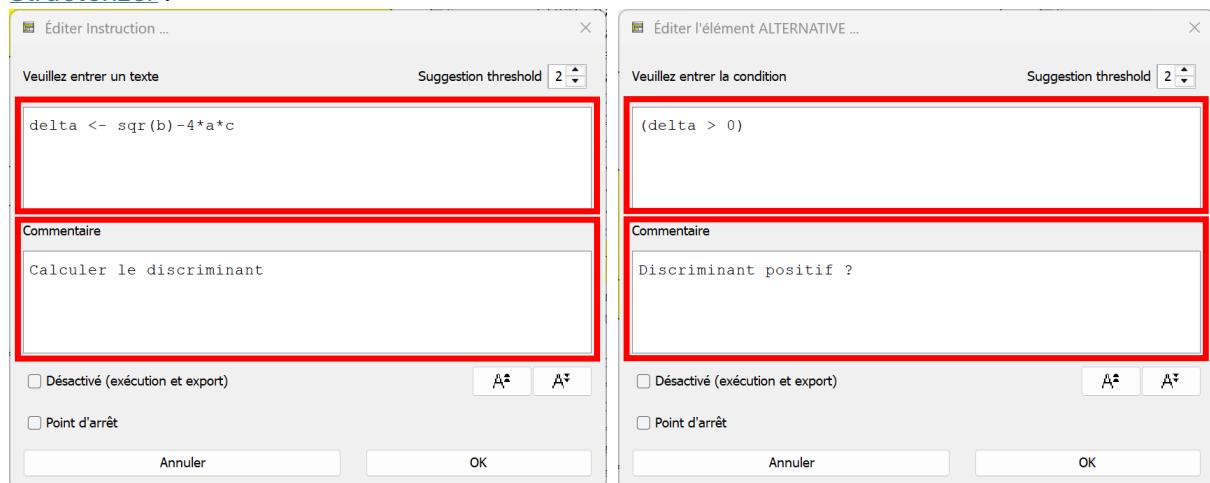
Il reste ensuite à écrire du code correspondant aux commentaires, partout où il y a des points d'interrogation.

2. Utilisation avancée de Structorizer pour tester le structogramme et générer le code

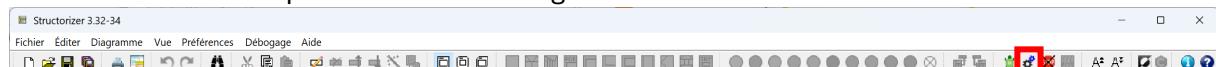
Configurer Structorizer pour afficher les commentaires et les instructions :



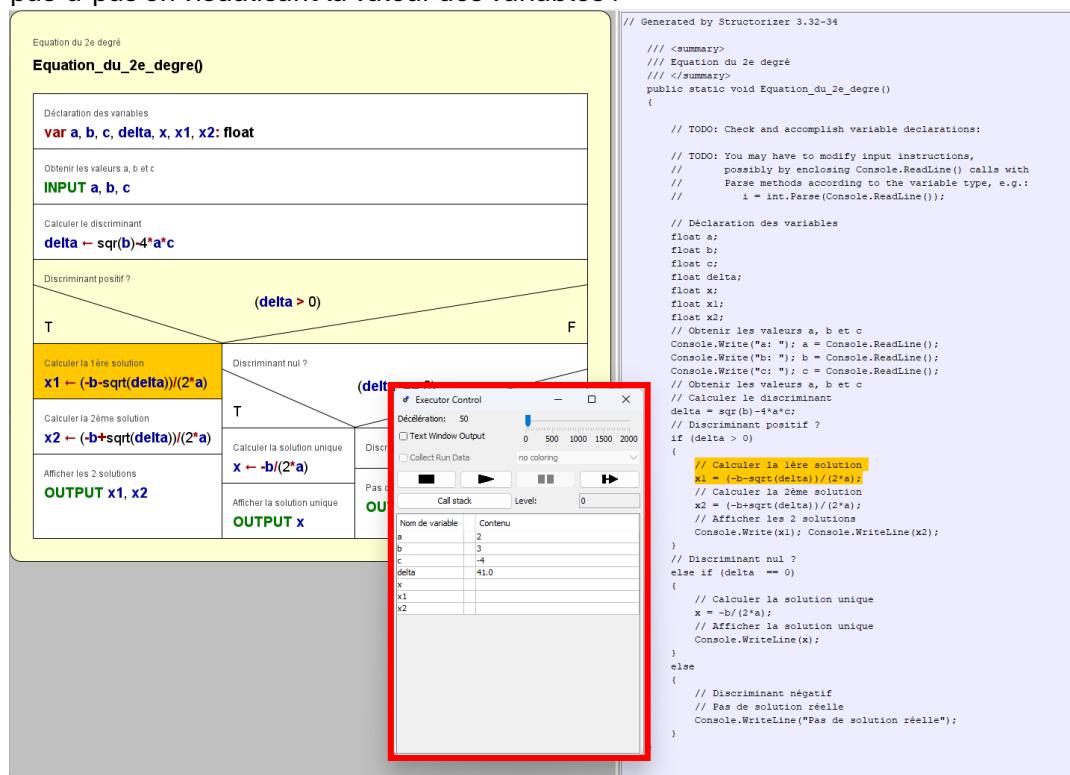
Dans les éléments, remplir le champ *Commentaire* et l'instruction en respectant la [syntaxe de Structorizer](#):



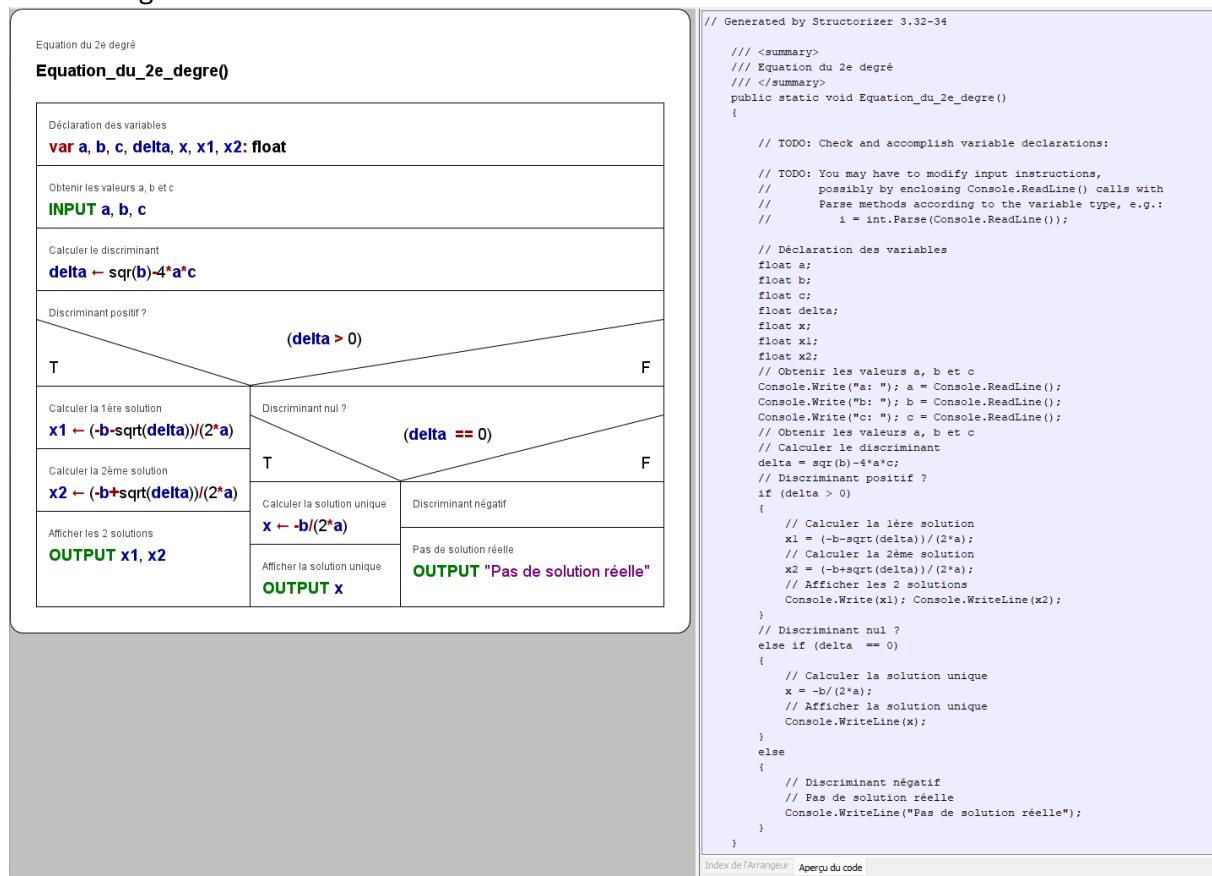
Démarrer l'*Exécuteur* pour tester le structogramme :



La fenêtre de contrôle de l'*Exécuteur* s'affiche et on peut exécuter le structogramme en mode pas-à-pas en visualisant la valeur des variables :



Le code source en **C#** contient la structure, les commentaires et les instructions correspondant au structogramme :



Sélectionner la partie intéressante du code, la copier avec [ctrl]+[C] : voir point 1. ci-dessus.

Et la coller dans la solution *Application Windows Forms (.NET Framework)* écrite en **VC#** avec [ctrl]+[V] suivi de **[ctrl]+[Z]** pour l'indenter correctement : voir point 1. ci-dessus.

Il reste ensuite à adapter le code.

Bibliographie :

- [STRUCTORIZER User Guide > Use Cases > The Purist Documentation](#)
- [STRUCTORIZER User Guide > Use Cases > Design for External Refinement](#)
- [STRUCTORIZER User Guide > Use Cases > Design with Internal Refinement](#)
- [STRUCTORIZER User Guide > Use Cases > Programming for Beginners / Prototyping](#)
- [STRUCTORIZER User Guide > Syntax \(Executable Dialect\)](#)
- [STRUCTORIZER User Guide > Features > Executor](#)

Codage de l'information

Comment stocker une information dans un système électronique ? Par exemple comment mémoriser la taille d'un habit ?

- En convenant qu'une tension de 42V représente la taille 42 d'un habit, une idée serait d'appliquer cette tension à un condensateur pour mémoriser la taille de manière analogique. Cependant, le condensateur réel présente une autodécharge significative, entraînant une diminution de la tension mémorisée au fil du temps et affectant la qualité de la mémorisation.
- Il est plus efficace d'utiliser un bistable SR capable de stocker un état logique haut ou bas, appelée « cellule mémoire ». On peut en assembler plusieurs pour mémoriser une donnée binaire composée de plusieurs bits (en général 8).

L'architecture interne d'une [PROM](#) est très explicite :

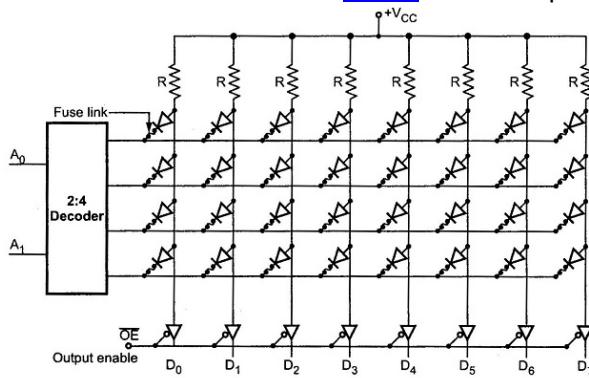


Fig. 3.71 Four byte PROM

Comment mémoriser une information quelconque dans des cases mémoires ne pouvant stocker qu'un « 1 » ou un « 0 » ?

Que représente l'information stockée comme ceci : **10000001_b** ?

Attention : l'hexadécimal n'est pas un codage de l'information, c'est une autre manière de représenter une suite binaire qu'avec des 0 et des 1, utilisée pour faciliter la lisibilité par les humains de longues chaînes de bits.

- Le nombre entier **129** en codage binaire absolu (non-signé)
 - Le nombre entier **-127** en codage binaire signé, [complément à deux](#)
 - Le nombre entier **-1** en codage binaire signé en [complément à un](#)
 - Le nombre entier **81** en [BCD](#)
 - Le nombre entier **254** selon le [code de Gray](#), [convertisseur](#)
 - Le caractère ü en [ASCII étendu](#) (voir [ASCII](#) 7 bits, [Unicode](#) 16-32 bits)
 - La couleur **purple** (n°129) selon le [code de couleurs Xterm](#)
 - Un [nombre décimal](#) (de l'ensemble D) selon un code à [virgule flottante](#) (mais il n'existe pas de type sur 8 bits), codé comme ceci : signe, mantisse entière, exposant entier : Signe·Mantisse·Base^{Exposant}
p. ex. : $(-1)^S \cdot M \cdot 2^{(E-127)}$
- Le nombre décimal **1.80767501897901402149161116244E-43** en codage [IEEE754 Single precision 32-bit, convertisseur](#)
- L'instruction machine **AJMP** selon le [jeu d'instructions en assembleur du processeur Intel 8051](#)

Il est nécessaire d'établir une convention pour « coder » n'importe quelle information en une donnée binaire, qui permet aussi de « décoder » cette donnée binaire afin de retrouver l'information d'origine. Cette convention est appelée le « codage » ou « encodage ».

Les types de donnée en langage évolué définissent :

- La taille de l'emplacement de stockage (nombre de bits)
- Le codage utilisé pour coder l'information stockée en mémoire en donnée binaire

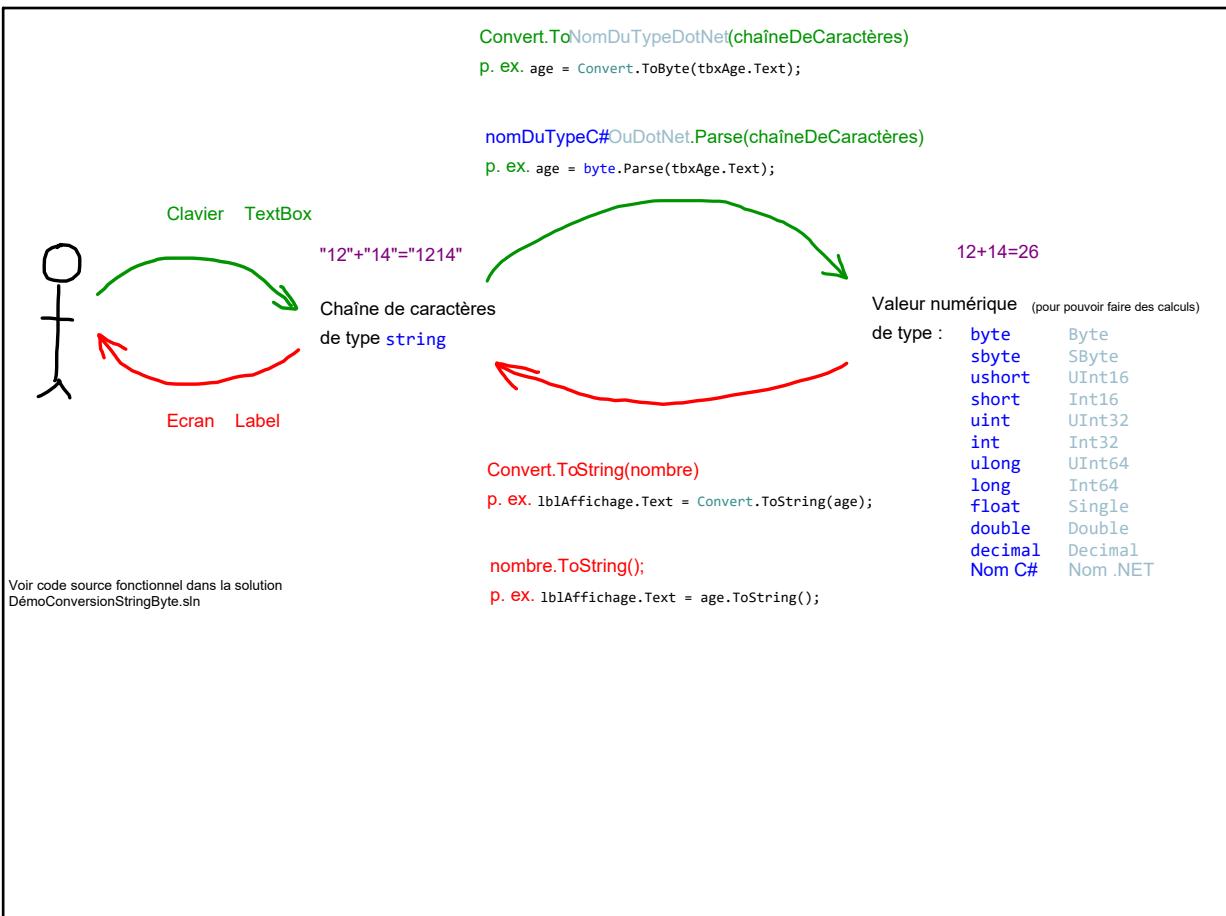
Dans Visual Studio, les propriétés des contrôles sont typées.

HID (Human Interface Devices)

Clavier	
Souris	
Ecran tactile, stylet	
Microphone (voix)	
Caméra, Kinect (reconnaissance gestuelle)	https://fr.wikipedia.org/wiki/Kinect
Manette, joystick, volant	
Scanner	
Oculométrie (eye-tracking)	https://fr.wikipedia.org/wiki/Oculom%C3%A9trie
Accéléromètre, gyroscope (motion tracking)	
Lecteur d'empreinte digitale	
Télécommande (TVBox)	
Interfaces directes neuronales (Neuralink)	https://fr.wikipedia.org/wiki/Implant_c%C3%A9ntral
Historiquement : 8 commutateurs 1/0	https://fr.wikipedia.org/wiki/Interface_neuronale_directe
	https://en.wikipedia.org/wiki/ENIAC

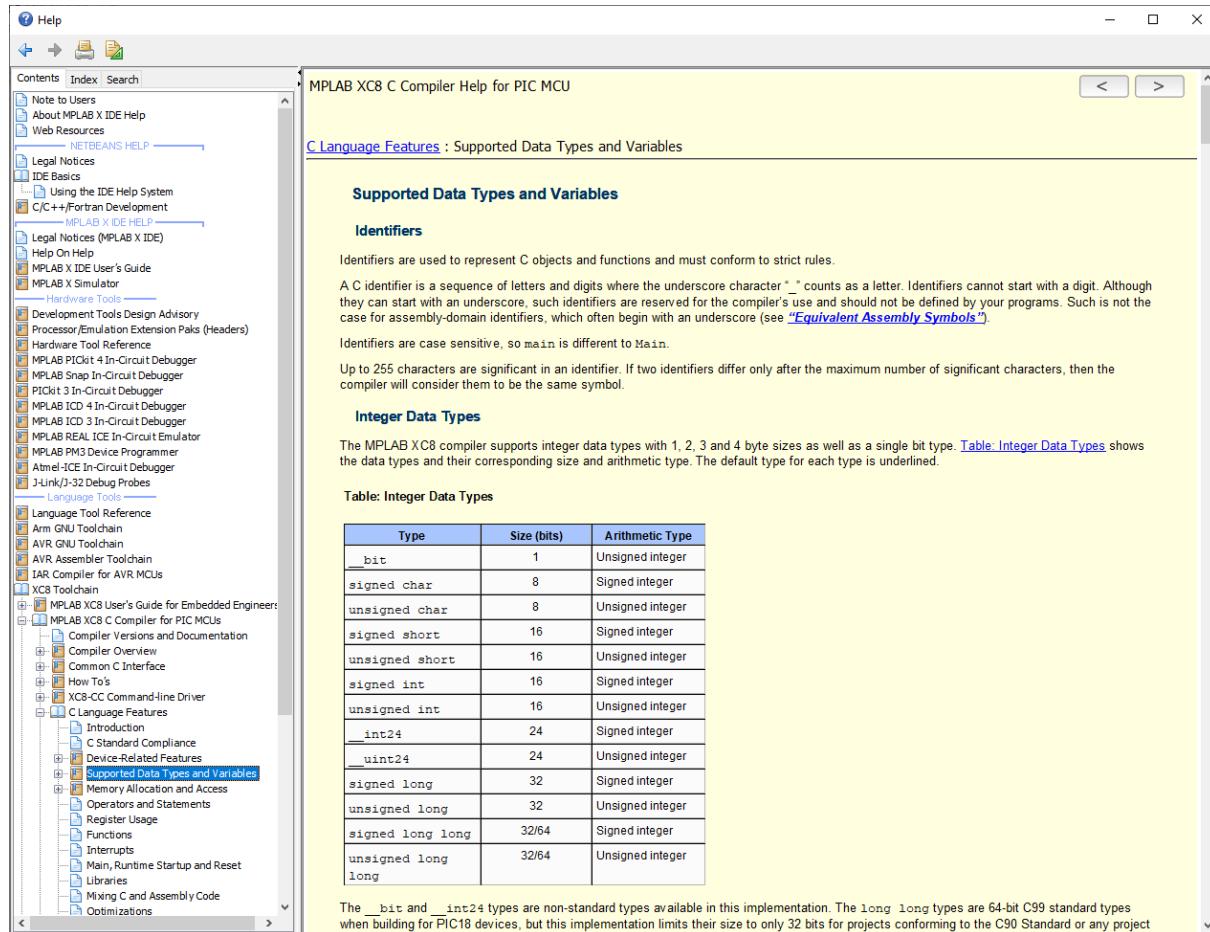


Ecran / beamer	
Imprimante	
Haut-parleur	
Vibrer (retour haptique)	https://fr.wikipedia.org/wiki/Dispositif_haptique
Casque VR	
Historiquement : 8 lampes allumée/éteinte	https://en.wikipedia.org/wiki/ENIAC



Les types en C sur MPLAB avec le compilateur XC8

- Ouvrir *MPLAB*
- Menu *Help > Help Contents*
- Section *Language Tools : XC8 Toolchain > MPLAB XC8 C Compiler for PIC MCUs > C Language Features > Supported Data Types and Variables*



Procédure pour installer **MS Visual Studio Community 2022** pour développer des applications Windows en VC#

(Environnement de développement intégré (IDE) gratuit et totalement fonctionnel pour les étudiants, les développeurs open source et les développeurs particuliers).

ATTENTION : ne pas effectuer cette procédure sur un PC de l'EMF. **MS Visual Studio Community 2022** est déjà déployé automatiquement par le SITel sur tous les PC de l'EMF.

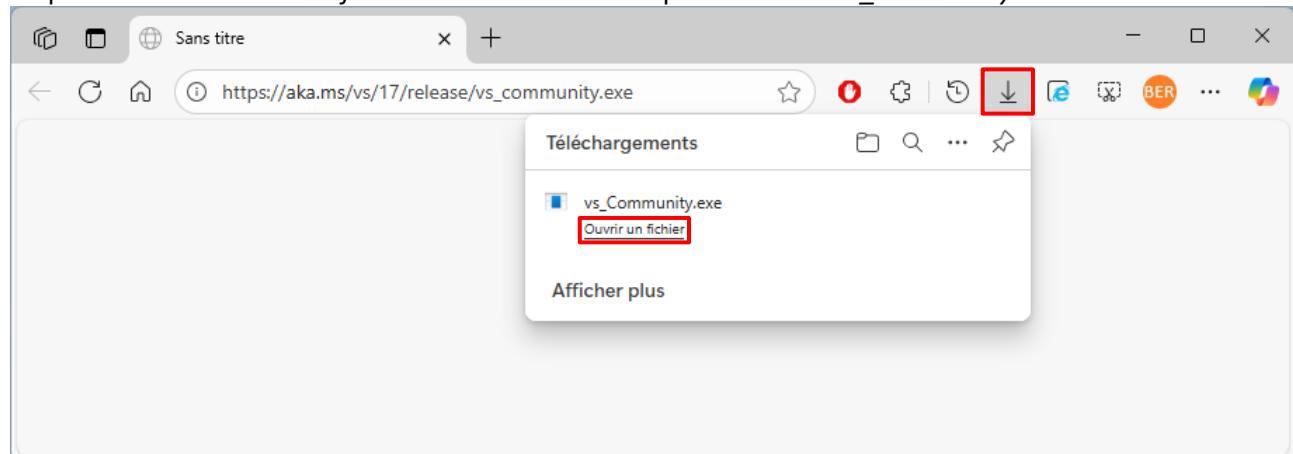
Avertissement : le téléchargement et l'installation peuvent durer plusieurs dizaines de minutes (environ 15 minutes) ! et consomment environ 8Go de place sur le disque dur.

1. Ouvrez le lien ci-dessous pour installer la version exacte nécessaire (**Community 2022**) et non pas la version la plus récente :

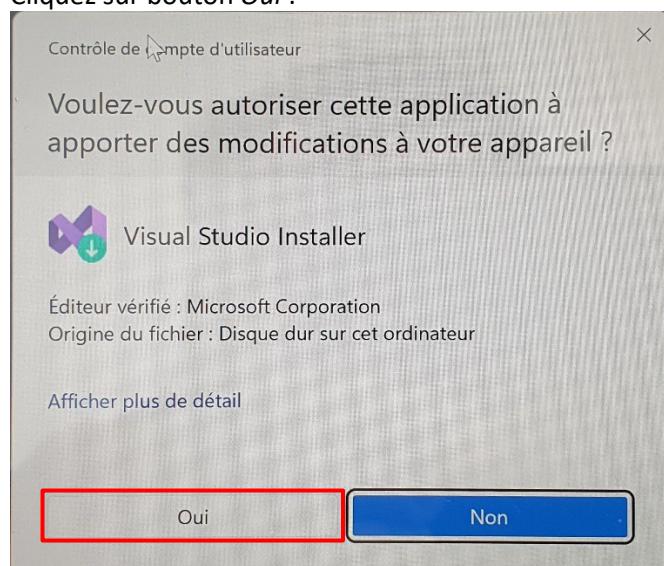
https://aka.ms/vs/17/release/vs_community.exe

Si ce lien ne fonctionne pas, contactez-moi !

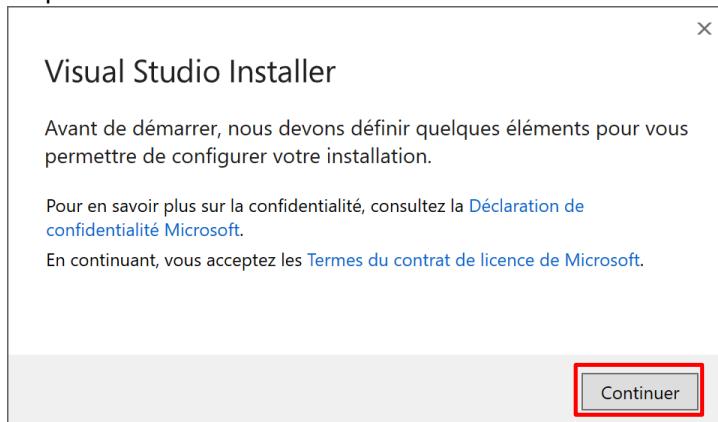
2. Cliquez sur le lien *Ouvrir un fichier* ou le bouton *Ouvrir* pour exécuter *vs_Community.exe* :



3. Cliquez sur bouton *Oui* :

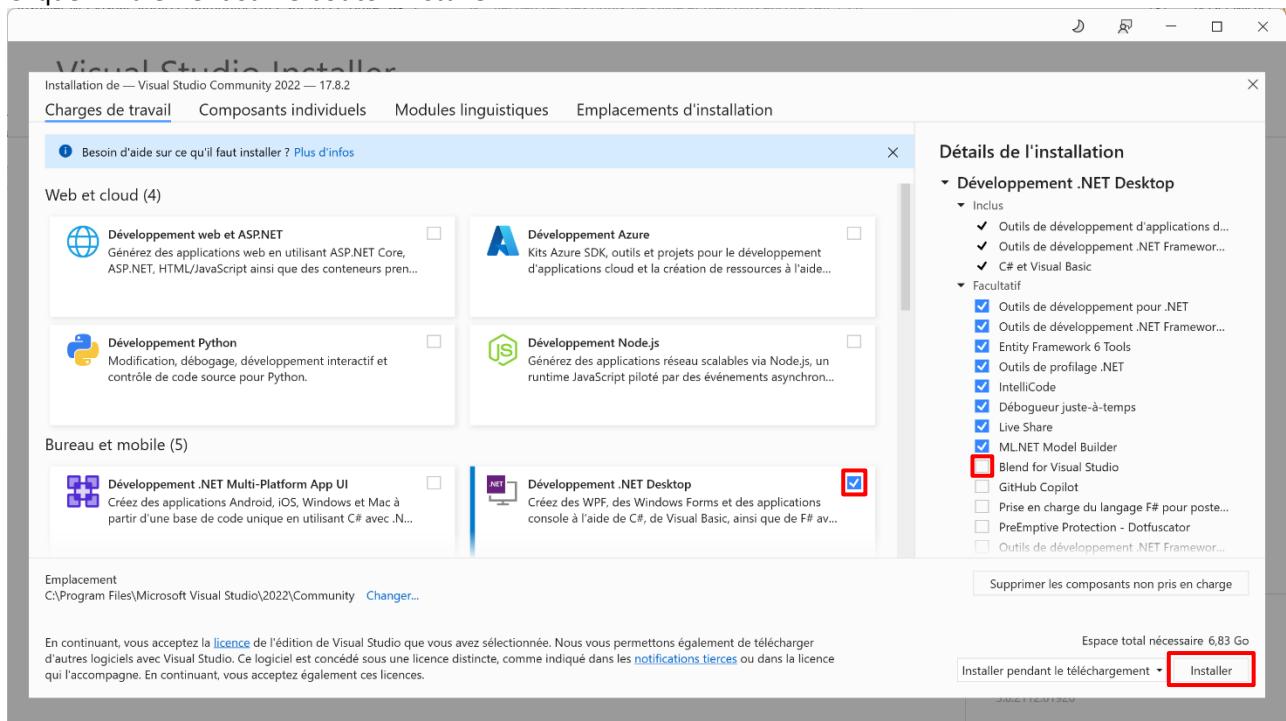


4. Cliquez sur le bouton *Continuer* :

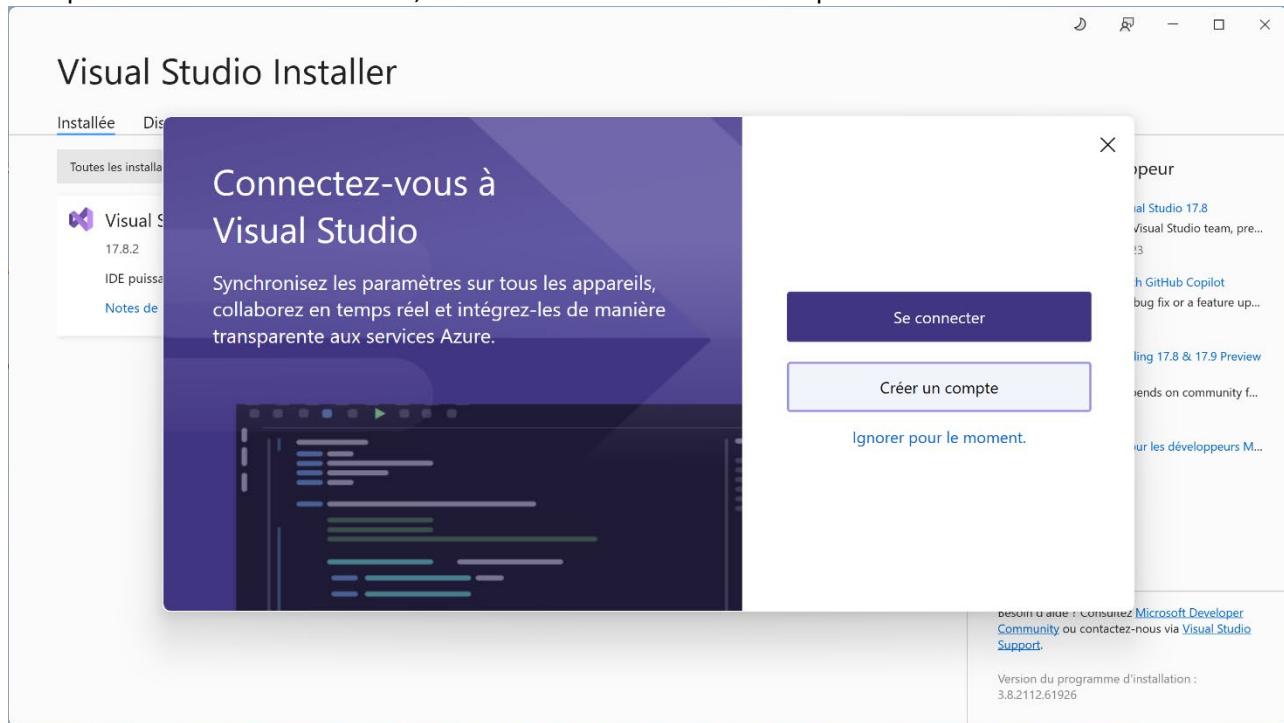


5. Cochez la case du produit *Développement .NET Desktop* et décochez la case de l'option *Blend for Visual Studio* pour installer le minimum requis pour développer des applications Windows en VC#. (Vous pouvez installer d'autres produits, mais cela va prendre plus de place sur votre disque dur et plus de temps d'installation. Il est déconseillé d'installer un produit sans savoir si vous en avez besoin, n'hésitez pas à vous renseigner avant de les installer. L'installation de produits supplémentaires peut également se faire par après.)

Cliquez finalement sur le bouton *Installer* :



6. Lorsque l'installation est terminée, *Visual Studio* démarre automatiquement :



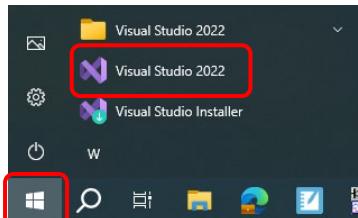
Poursuivez selon la démarche décrite dans le document *Inscription pour MS Visual Studio Community 2022* disponible sur *Teams*, à partir de l'étape 2.

Inscription pour utiliser MS Visual Studio Community 2022

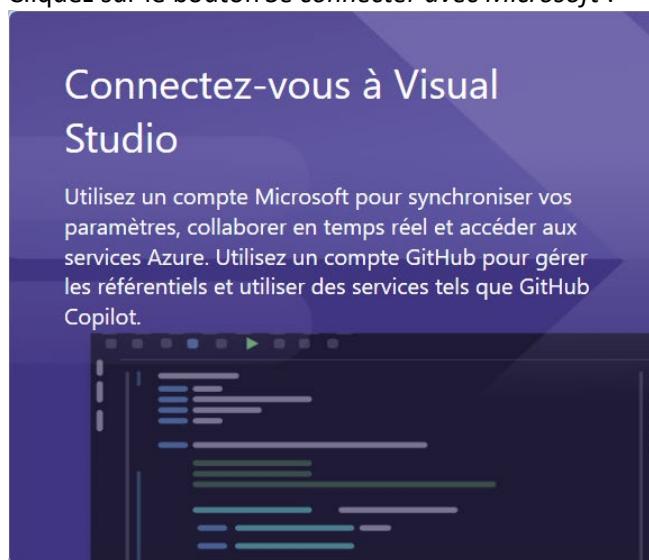
Le logiciel *Microsoft Visual Studio Community 2022* (environnement de développement intégré (IDE) gratuit et totalement fonctionnel pour les étudiants, les développeurs open source et les développeurs particuliers) demande que chaque utilisateur s'inscrive par l'intermédiaire d'un *Compte Microsoft* en guise de licence d'utilisation au-delà de la période d'évaluation de 30 jours.

Voici la procédure à suivre pour s'inscrire (c-à-d. se connecter) à *Microsoft Visual Studio Community 2022*.

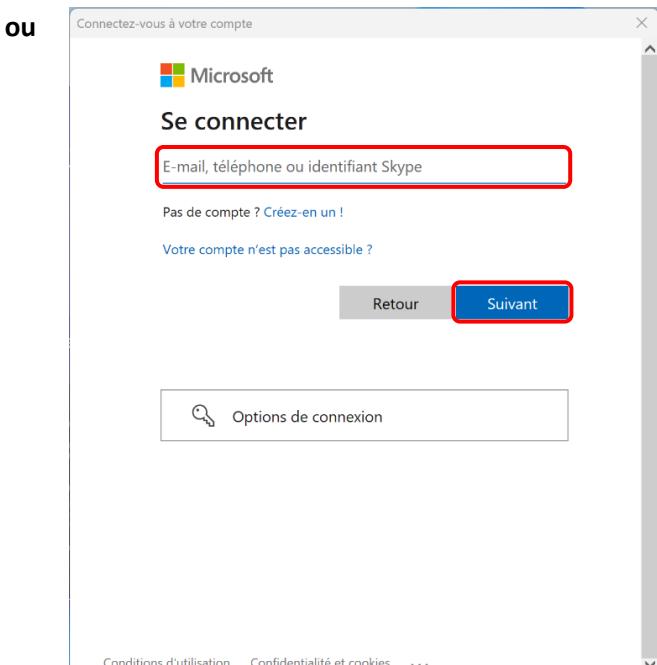
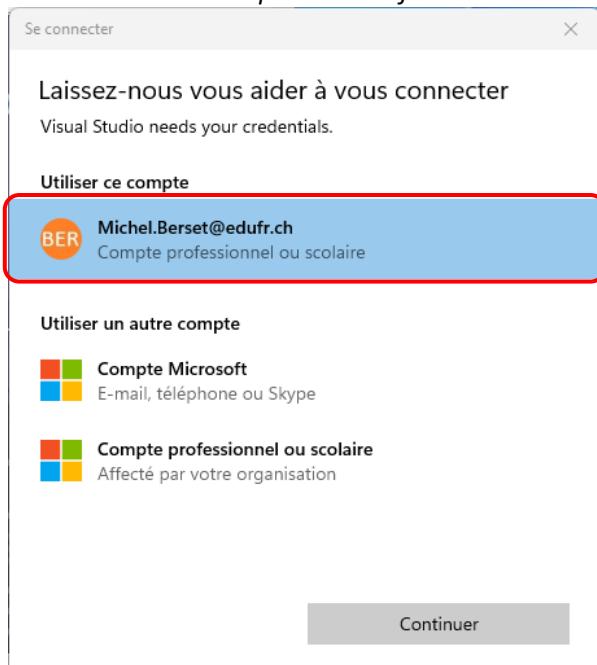
1. Menu *Démarrer > Visual Studio 2022* :



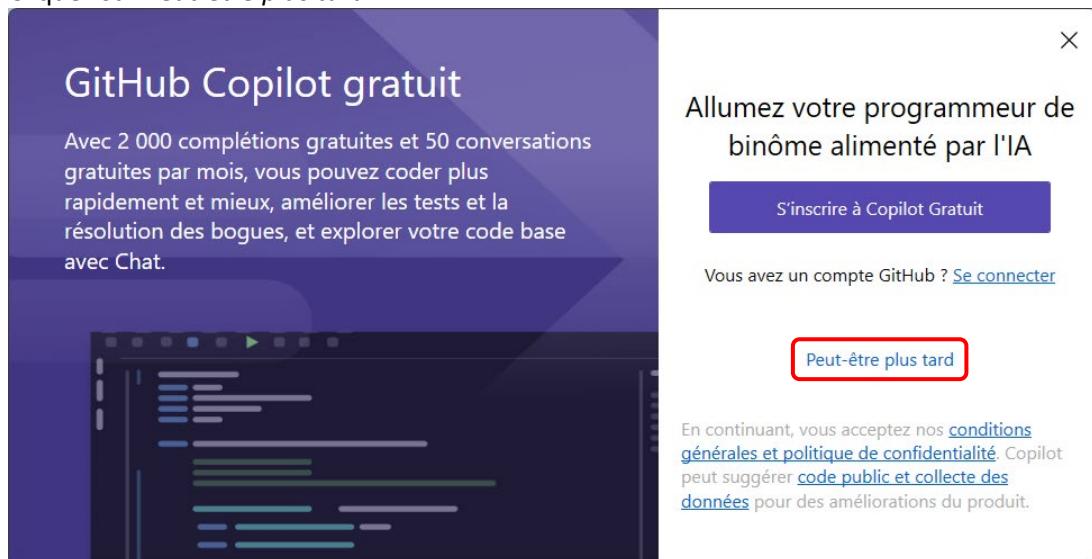
2. Cliquez sur le bouton *Se connecter avec Microsoft* :



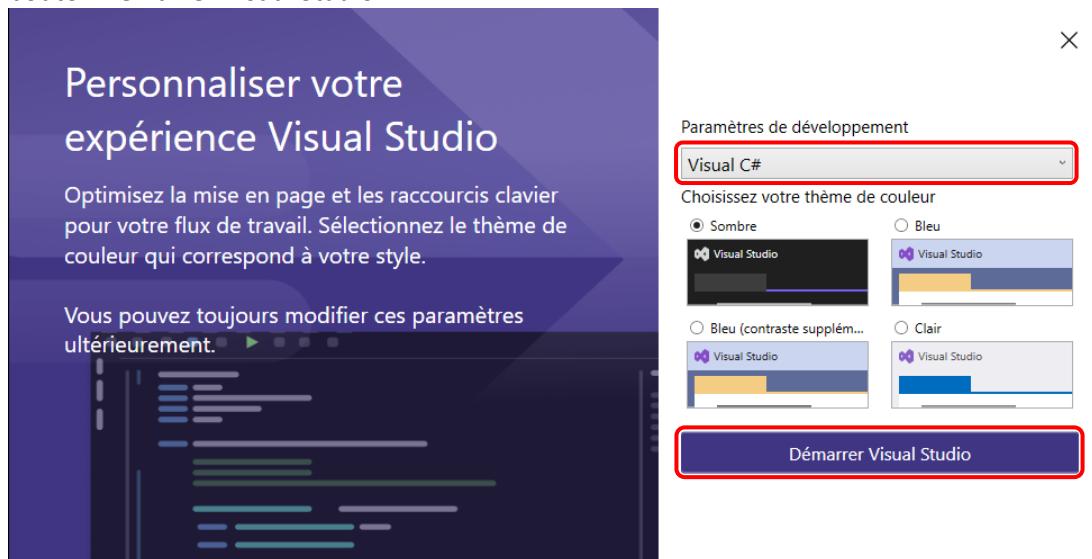
3. Choisissez votre *Compte Microsoft de l'EMF* :



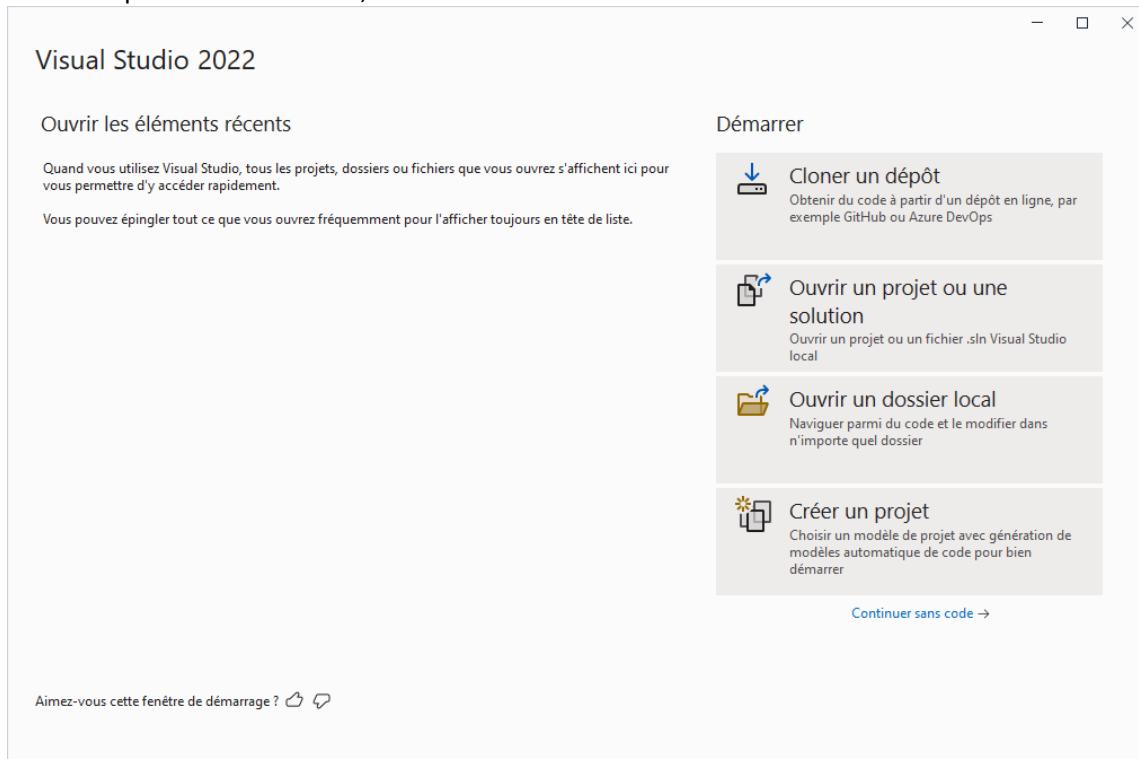
4. Cliquez sur *Peut-être plus tard* :



5. Si la fenêtre ci-dessous s'affiche, choisissez **Visual C#** dans le menu déroulant *Paramètres de développement*. Vous pouvez choisir votre thème préféré, pour terminer, cliquez sur le bouton *Démarrer Visual Studio* :



6. Si tout se passe correctement, vous devriez arriver à cet écran d'accueil de *Visual Studio* :



Vous pouvez maintenant utiliser *Visual Studio 2022*.

**Comprenez le code source que vous écrivez et
n'écrivez que du code source que vous comprenez !**

Ne faites pas simplement un *copier-coller* d'un code source que vous trouvez sur Internet ou que vous faites générer par une IA sans le comprendre ! ou alors, trouvez une autre solution !