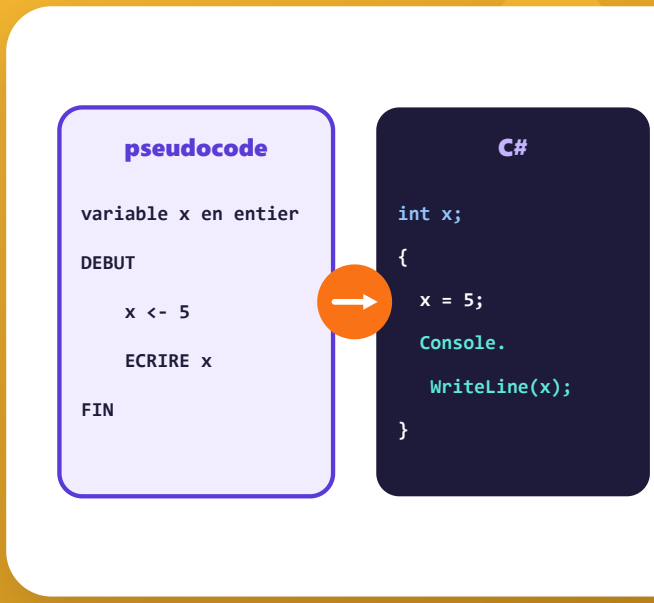
CHAPITRE 6

Du pseudocode au C# : la conversion

La même logique, une autre syntaxe



AU PROGRAMME

1. La structure d'un programme C#
2. Les types et les variables
3. Les opérateurs
4. Écrire et lire
5. Les tests et les boucles
6. Traduire un programme complet

Nom : ..... **Classe :** .....

AVANT DE COMMENCER

Ce que tu vas apprendre

Tout ce que tu as appris en pseudocode reste vrai en C# : variables, lecture et écriture, tests et boucles. Seule l'écriture change. D'ailleurs, quand tu appuies sur F5 dans VS Code, l'extension traduit déjà ton pseudocode en C# avant de l'exécuter. Ce chapitre t'apprend à faire cette traduction toi-même, instruction par instruction.

1. **Reconnaître la structure d'un programme C# : using, class, Main, accolades et points-virgules.**
2. **Traduire les types et les déclarations : int, double, string, bool, char.**
3. **Traduire les opérateurs, en évitant les pièges de = / == et de la division entière.**
4. **Afficher avec `Console.WriteLine` et lire avec `Console.ReadLine` et `Parse`.**
5. **Traduire les tests et les boucles : if, else if, else, while, for.**
6. **Traduire un algorithme complet, pas à pas, et corriger les erreurs fréquentes.**

PARTIE 1

La structure d'un programme C#

Le **C#** (on dit « C sharp ») est un langage de programmation professionnel, créé par Microsoft. On s'en sert pour des applications, des sites web et des jeux vidéo (le moteur Unity). Il est plus strict que le pseudocode : chaque symbole compte.

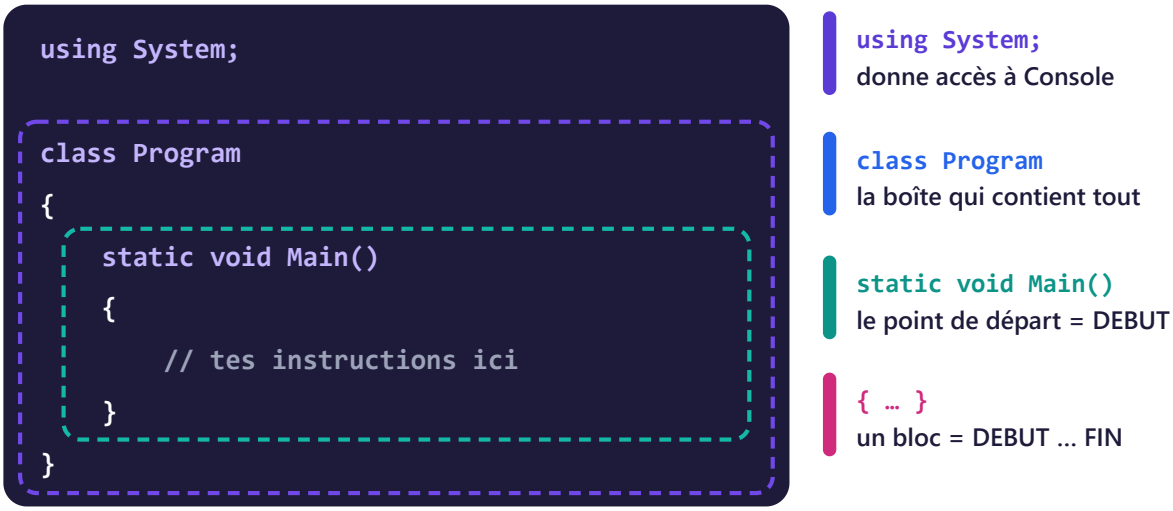


Figure 6.1 — Le squelette de tout programme C# du cours.

Le même programme

PSEUDOCODE

```
DEBUT
    ECRIRE "Bonjour !"
FIN
```

C#

```
using System;
class Program
{
    static void Main()
    {
        Console.WriteLine("Bonjour
!");
    }
}
```

À RETENIR

Les quatre règles d'écriture du C#

- Chaque instruction se termine par un **point-virgule** `;`.
- Les **accolades** `{ }` remplacent `DEBUT` et `FIN` : elles ouvrent et ferment un **bloc** d'instructions.
- Le C# est **sensible à la casse** : `Console` et `console` sont deux mots différents ; `age` et `Age`, deux variables différentes.
- Les **commentaires** commencent par `//` (une ligne) ou s'écrivent entre `/*` et `*/` (plusieurs lignes) ; ils sont ignorés à l'exécution.

POUR ALLER PLUS LOIN

`using System;` donne accès à `Console`. `class Program` est la « boîte » qui contient tout le programme. `static void Main()` est la fonction lancée en premier : tu découvriras les fonctions au chapitre 7.

PARTIE 2

Les types et les variables

Les types

Information	Pseudocode	C#	Exemple en C#
un nombre entier	`entier`	`int`	`int age = 16;`
un nombre à virgule	`numerique`	`double` (ou `float`)	`double taille = 1.68;`
un seul caractère	—	`char`	`char lettre = 'A';`
un texte	`chaine de caracteres`	`string`	`string prenom = "Lina"`
vrai ou faux	`booleen`	`bool`	`bool majeur = true;`

ATTENTION

Apostrophes ou guillemets ?

Un `char` s'écrit entre **apostrophes** et contient un seul caractère : `'A'`. Un `string` s'écrit entre **guillemets** : `"A"`. Ce ne sont pas les mêmes types.

Les types numériques du C#

Type	Taille	Valeurs possibles
`byte`	1 octet	de 0 à 255
`short`	2 octets	de -32 768 à 32 767
`int`	4 octets	d'environ -2,1 milliards à 2,1 milliards
`long`	8 octets	d'environ -9×10^{18} à 9×10^{18}
`float`	4 octets	nombres à virgule, environ 7 chiffres précis
`double`	8 octets	nombres à virgule, environ 15 chiffres précis

POUR ALLER PLUS LOIN

Dans ce cours, `int` et `double` suffisent. Les autres existent pour gagner de la place en mémoire ou stocker de très grands nombres.

Déclarer, affecter, initialiser

PSEUDOCODE

```
variable age en entier
variables a, b en entier
DEBUT
    age <- 25
FIN
```

C#

```
int age;
int a, b;
age = 25;
int note = 14;
```

La déclaration

Le **type** puis le nom : `int age;`. Une variable ne se déclare **qu'une fois** dans un même bloc.

L'affectation

La flèche `<-` devient le signe `=` : `age = 25;`.

L'initialisation

Déclarer et affecter en une ligne : `int note = 14;`. C'est la forme la plus courante en C#.

La portée des variables

En C#, une variable n'existe que dans le **bloc** `{ }` où elle est déclarée. Une variable déclarée dans les accolades d'un `if` n'existe plus après l'accolade fermante. Déclare donc tes variables au début de `Main`, comme tu le faisais avant `DEBUT`.

PARTIE 3

Les opérateurs

La table de conversion des opérateurs

Rôle	Pseudocode	C#
affectation	`<-`	`=`
égal à (comparaison)	`=`	`==`
différent de	`<>`	`!=`
plus petit, plus grand…	`<` `>` `<=` `>=`	`<` `>` `<=` `>=`
concaténation	`&`	`+`
et, ou, ou exclusif	`ET` `OU` `XOR`	`&&` `||` `^`
non	`NON(…)`	`!(…)`
reste de la division	`mod`	`%`
calculs	`+` `-` `*` `/`	`+` `-` `*` `/`

ATTENTION

Le piège n° 1 : = et ==

En C#, `=` **range** une valeur et `==` **compare**. `if (age = 18)` est une erreur : on écrit `if (age == 18)`.

La division entière

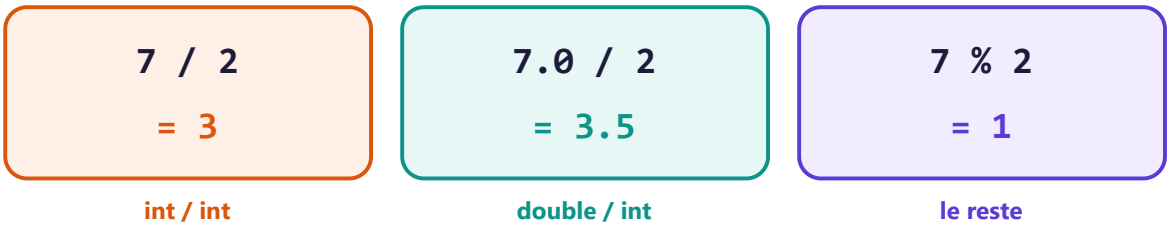


Figure 6.2 — Entre deux int, la division garde seulement la partie entière.

Quand les deux nombres sont des `int`, le C# fait une **division entière** : `7 / 2` donne 3, la partie décimale est perdue. Pour obtenir 3,5, il faut qu'au moins un des deux soit un `double` : `7.0 / 2`. C'est pour cela que `/` entre deux `entier` donnait un quotient entier dans VS Code.

ATTENTION

Jamais diviser par zéro

Une division par zéro (ou `% 0`) entre entiers **arrête le programme** sur une erreur. Vérifie avec un `if` avant de diviser par une valeur tapée par l'utilisateur.

EXEMPLE

En C#, le `+` sert à la fois à additionner des nombres et à concaténer des textes : `"Score : " + 12` donne le texte « Score : 12 ». Dès qu'un `string` est en jeu, le `+` concatène.

PARTIE 4

Écrire et lire

Pseudocode	C#	Effet
`ECRIRE "Bonjour"`	`Console.WriteLine("Bonjour");`	affiche, puis passe à la ligne
—	`Console.Write("Ton âge : ");`	affiche **sans** passer à la ligne
`ECRIRE "Salut " & nom`	`Console.WriteLine("Salut " + nom);`	concaténation avec `+`
`LIRE nom`	`nom = Console.ReadLine();`	lit un texte (`string`)
`LIRE age`	`age = int.Parse(Console.ReadLine());`	lit un texte, puis le convertit en `int`

DÉFINITION

Console.ReadLine()

Attend que l'utilisateur tape une ligne et appuie sur Entrée, puis **renvoie ce texte**. Elle renvoie **toujours** un `string`, même si l'utilisateur tape un nombre.

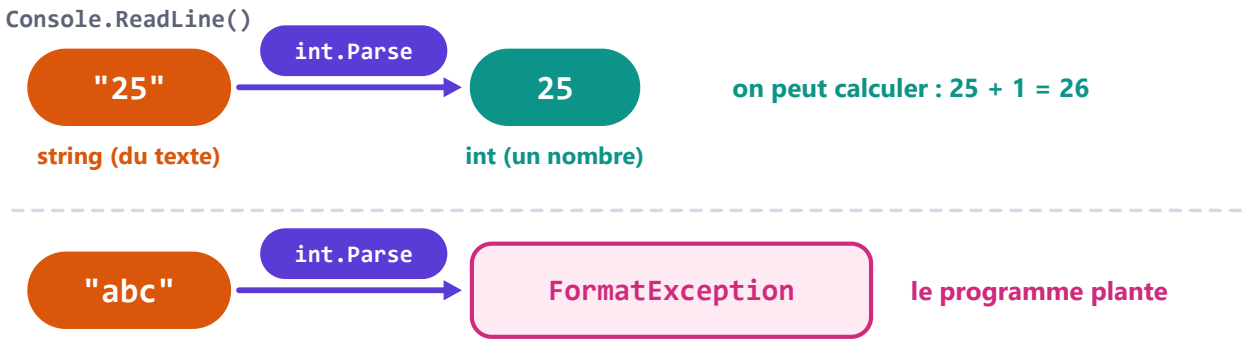


Figure 6.3 — Parse transforme un texte en nombre... s'il contient bien un nombre.

Lire un nombre

PSEUDOCODE

```
variable age en entier
DEBUT
    ECRIRE "Ton âge ?"
    LIRE age
    ECRIRE "Tu as " & age & " ans."
FIN
```

C#

```
int age;
Console.Write("Ton âge ? ");
age = int.Parse(Console.ReadLine());
Console.WriteLine("Tu as " + age + "
ans.");
```

`int.Parse(…)` convertit le texte en `int` ; `double.Parse(…)` en `double` ; `bool.Parse(…)` en `bool`. Pour un `string`, pas de conversion : `nom = Console.ReadLine();`.

Cast ou Parse ?

	Le cast	Le Parse
Sert à	convertir entre types **compatibles** (nombre vers nombre)	convertir un **texte** en nombre
Syntaxe	`(int)3.99` donne 3	`int.Parse("42")` donne 42
Risque	perte de la partie décimale	erreur si le texte n'est pas un nombre

Certaines conversions sont **implicites** (automatiques) car sans risque : un `int` se range dans un `double` (`double d = 10;`). L'inverse demande un **cast explicite** : `int n = (int)3.14;` donne 3.

PARTIE 5

Les tests et les boucles

SI, SINONSI, SINON

PSEUDOCODE

```
SI age < 12 ALORS
    ECRIRE "Enfant"
SINONSI age < 18 ALORS
    ECRIRE "Adolescent"
SINON
    ECRIRE "Adulte"
FINSI
```

C#

```
if (age < 12)
{
    Console.WriteLine("Enfant");
}
else if (age < 18)
{
    Console.WriteLine("Adolescent");
}
else
{
    Console.WriteLine("Adulte");
}
```

À RETENIR

Les tests en C#

- La condition est **toujours entre parenthèses** : `if (age < 12)`. Pas de `ALORS`, pas de `;` après.
- Chaque bloc a ses accolades ; il n'y a pas de `FINSI`.
- `SINONSI` devient `else if` (deux mots), `SINON` devient `else`.

TANTQUE devient while

PSEUDOCODE

```
LIRE reponse
TANTQUE reponse <> "O" ET reponse <>
"N"
    ECRIRE "O ou N ?"
    LIRE reponse
FINTANTQUE
```

C#

```
reponse = Console.ReadLine();
while (reponse != "O" && reponse !=
"N")
{
    Console.WriteLine("O ou N ?");
    reponse = Console.ReadLine();
}
```

POUR devient for

PSEUDOCODE

```
POUR i <- 1 à 10
    ECRIRE i
POUR SUIVANT
FINPOUR
```

C#

```
for (int i = 1; i <= 10; i++)
{
    Console.WriteLine(i);
}
```

Morceau du for	Rôle
`int i = 1`	le compteur est créé et part de 1 (avant le premier tour)
`i <= 10`	la condition, testée avant chaque tour
`i++`	après chaque tour, i augmente de 1 (`i++` veut dire `i = i + 1`)

PARTIE 6

Traduire un programme complet

La méthode

1. **Le squelette**
   Écris `using System;`, `class Program`, `static void Main()` et leurs accolades.

2. **Les déclarations**
   Traduis chaque `variable` : le type C# puis le nom, avec `;`.

3. **Ligne par ligne**
   Traduis chaque instruction avec la fiche de conversion, sans oublier les `;`.

4. **Les lectures**
   Ajoute `int.Parse` ou `double.Parse` pour chaque nombre lu.

5. **Vérifie**
   Accolades par paires, `==` dans les conditions, `+` pour concaténer. Puis lance le programme.

Un algorithme et sa traduction (le contenu de Main)

PSEUDOCODE

```
variable note en entier
DEBUT
    ECRIRE "Note sur 20 ?"
    LIRE note
    TANTQUE note < 0 OU note > 20
        ECRIRE "Entre 0 et 20 !"
        LIRE note
    FINTANTQUE
    SI note >= 10 ALORS
        ECRIRE "Réussi avec " & note
    SINON
        ECRIRE "Échec avec " & note
    FINSI
FIN
```

C#

```
int note;
Console.WriteLine("Note sur 20 ?");
note = int.Parse(Console.ReadLine());
while (note < 0 || note > 20)
{
    Console.WriteLine("Entre 0 et 20
!");
    note =
int.Parse(Console.ReadLine());
}
if (note >= 10)
{
    Console.WriteLine("Réussi avec "
+ note);
}
else
{
    Console.WriteLine("Échec avec " +
note);
}
```

Les erreurs les plus fréquentes

Erreur fréquente	Ce qu'il faut écrire
`Console.WriteLine("Salut")` **sans point-virgule**	`Console.WriteLine("Salut");`
`if (note = 20)`	`if (note == 20)`
`age = Console.ReadLine();` **avec** `int age`	`age = int.Parse(Console.ReadLine());`
`"Tu as " & age`	`"Tu as " + age`
`console.writeline(...)`	`Console.WriteLine(...)` : majuscules exactes
`if (age > 20 ET age < 30)`	`if (age > 20 && age < 30)`
une accolade `{` **sans sa** `}`	compter les accolades : chaque bloc ouvert est fermé

La fiche de conversion

Pseudocode	C#
`variable x en entier`	`int x;`
`x <- valeur`	`x = valeur;`
`ECRIRE "texte" & x`	`Console.WriteLine("texte" + x);`
`LIRE x` **(un entier)**	`x = int.Parse(Console.ReadLine());`
`SI … ALORS` / `SINONSI` / `SINON` / `FINSI`	`if (…) { }` / `else if (…) { }` / `else { }`
`TANTQUE … FINTANTQUE`	`while (…) { }`
`POUR i <- 1 à n … FINPOUR`	`for (int i = 1; i <= n; i++) { }`

BILAN DU CHAPITRE 6

L'essentiel à retenir

1. Un programme C# : `using System;`, `class Program`, `static void Main()` ; les **accolades** remplacent `DEBUT` et `FIN`.

2. Chaque instruction finit par `;`. Le C# distingue majuscules et minuscules.

3. Types : `entier` → `int`, `numerique` → `double`, `chaine de caracteres` → `string`, `booleen` → `bool` ; `char` pour un seul caractère entre apostrophes.

4. `<-` devient `=`, et le `=` qui compare devient `==` ; `<>` → `!=`, `&` → `+`, `ET` → `&&`, `OU` → `||`, `mod` → `%`.

5. Entre deux `int`, `/` fait une **division entière** : `7 / 2` vaut 3.

6. `ECRIRE` → `Console.WriteLine(…);` ; `LIRE` → `Console.ReadLine()`, qui renvoie un `string` : on le convertit avec `int.Parse(…)`.

7. `SI` → `if (…) { }`, `SINONSI` → `else if`, `TANTQUE` → `while`, `POUR` → `for (int i = 1; i <= n; i++)`.

Le lexique

C#
Langage de programmation de Microsoft ; VS Code y traduit ton pseudocode.

Main
La fonction lancée en premier : le point d'entrée du programme C#.

Bloc
Instructions entre accolades `{ }` ; une variable n'existe que dans son bloc.

Initialisation
Déclarer et affecter en une ligne : `int note = 14;`.

Portée
La partie du code où une variable existe.

Division entière
Division entre deux int : seule la partie entière est gardée.

Parse
Conversion d'un texte en nombre : `int.Parse("42")`.

Cast
Conversion entre types numériques : `(int)3.99` donne 3.

Sensible à la casse
Majuscules et minuscules comptent : `Age` et `age` sont différents.

Je fais le point

Coche une case par ligne : où en es-tu avant l'interro ?

Je suis capable de…	Oui	Presque	Pas encore
Écrire le squelette d'un programme C#.	☐	☐	☐
Traduire une déclaration et une affectation.	☐	☐	☐
Traduire les opérateurs sans confondre `=` et `==`.	☐	☐	☐
Lire un nombre avec `Console.ReadLine()` et `int.Parse`.	☐	☐	☐
Traduire un `SI`, un `TANTQUE` et un `POUR`.	☐	☐	☐
Repérer les erreurs fréquentes dans un programme C#.	☐	☐	☐

CHAPITRE 6 · EXERCICES

Teste tes connaissances

avant l'interro

COMMENT T'Y PRENDRE

1. **Relis la synthèse** : l'essentiel, le lexique et « Je fais le point ».
2. **Fais les exercices sans le cours**, comme à l'interro.
3. **Corrige-toi** avec la correction vue en classe ou sur le site.
4. **Reviens sur tes erreurs** : relis la partie du cours concernée.

LES 6 EXERCICES

1. **Les opérateurs**
2. **Traduis les déclarations**
3. **Le C# en questions**
4. **Les règles du C#**
5. **Corrige le programme C#**
6. **Traduis en C#**

Exercice 1 Les opérateurs

- **Relie chaque opérateur du pseudocode à sa traduction en C# (écris la lettre).**

1	`<-` →	**A**	`==`
2	`=` (comparer) →	**B**	`%`
3	`<>` →	**C**	`&&`
4	`&` →	**D**	`!=`
5	`ET` →	**E**	`+`
6	`mod` →	**F**	`=`

Exercice 2 Traduis les déclarations

- **Écris la ligne C# qui correspond.**

Pseudocode	C#
`variable age en entier`	
`variable moyenne en numerique`	
`variable ville en chaine de caracteres`	
`variable inscrit en booleen`	
`ville <- "Huy"`	
`inscrit <- age >= 18`	

Exercice 3 Le C# en questions

▸ **Coche la bonne réponse.**

1. Que vaut `17 / 5` en C#, si ce sont deux int ?

☐ 3.4 ☐ 3 ☐ 2

2. Que vaut `17 % 5` ?

☐ 3 ☐ 2 ☐ 3.4

3. Que renvoie `Console.ReadLine()` ?

☐ un int ☐ un string ☐ rien

4. Quelle ligne est correcte ?

☐ `if (age = 18)` ☐ `if (age == 18)` ☐ `if age == 18`

5. Que fait `Console.Write("Nom : ");` ?

☐ affiche sans passer à la ligne ☐ lit le nom ☐ affiche et passe à la ligne

Exercice 4 Les règles du C#

▸ **Coche vrai ou faux. Si c'est faux, corrige la phrase sur la ligne.**

Affirmation	Vrai	Faux
Chaque instruction se termine par un point-virgule.	☐	☐
`Console.WriteLine` et `console.writeline` sont la même chose.	☐	☐
`'A'` est un `char` et `"A"` est un `string`.	☐	☐
`int.Parse("douze")` donne 12.	☐	☐
Une variable déclarée dans un bloc `if` existe encore après ce bloc.	☐	☐

Exercice 5 Corrige le programme C#

▸ **Ce programme contient quatre erreurs.**

```c#
1  using System;
2  class Program
3  {
4      static void Main()
5      {
6          int age;
7          age = Console.ReadLine();
8          if (age = 18)
9          {
10             Console.WriteLine("Tu as " & age)
11         }
12     }
13 }
```

1. Ligne 7 : quelle est l'erreur ?

2. Ligne 8 : quelle est l'erreur ?

3. Ligne 10 : quelles sont les deux erreurs ?

Exercice 6 Traduis en C#

▸ **Écris le programme C# complet : using, class, Main, accolades et points-virgules.**

PSEUDOCODE

```
variables a, b en entier
DEBUT
    ECRIRE "Premier nombre ?"
    LIRE a
    ECRIRE "Deuxième nombre ?"
    LIRE b
    SI a > b ALORS
        ECRIRE "Le plus grand est " & a
    SINON
        ECRIRE "Le plus grand est " & b
    FINSI
FIN
```

1. Traduis l'algorithme ci-dessus en un programme C# complet.

EXEMPLE D'ÉCRAN

```
Premier nombre ?
8
Deuxième nombre ?
13
Le plus grand est 13
```

2. Écris en C# un programme qui affiche les nombres de 10 à 1 (compte à rebours), puis « Décollage ! ».