C# : variables, constantes et structures de contrôle

Achref El Mouelhi

Docteur de l'université d'Aix-Marseille
Chercheur en programmation par contrainte (IA)
Ingénieur en génie logiciel

elmouelhi.achref@gmail.com

Plan

Plan

Plan

C#

Une variable ?

- Zone mémoire
- Permettant de stocker une ou plusieurs données
- Pouvant avoir plusieurs valeurs différentes dans un programme

C#

Une variable ?

- Zone mémoire
- Permettant de stocker une ou plusieurs données
- Pouvant avoir plusieurs valeurs différentes dans un programme

Remarque

- **C#** : langage de programmation fortement typé.
- Une variable peut changer de valeurs mais ne peut jamais changer de type.

C#

Pourquoi typer les variables ?

Pour

- connaître l'espace de stockage nécessaire pour la variable.
- déterminer des valeurs minimale et maximale pour la variable.
- pouvoir appliquer des méthodes et des opérations sur les valeurs de ce type.

C#

Pourquoi typer les variables ?

Pour

- connaître l'espace de stockage nécessaire pour la variable.
- déterminer des valeurs minimale et maximale pour la variable.
- pouvoir appliquer des méthodes et des opérations sur les valeurs de ce type.

Deux modes de typage en **C#**

- **Statique** : type précisé à la déclaration.
- **Dynamique** : type déterminé selon la valeur affectée à la variable à la déclaration.

C#

Déclarer une variable

```
type nomVariable;
```

Principaux types simples en c# (le nom d'un type commence par une lettre en minuscule)

- `sbyte` : entier codé sur 1 octet, valeur comprise entre -128 et 127 (équivalent non signé `byte`, valeur comprise entre 0 et 255)
- `short` : entier codé sur 2 octets (entre -32 768 et 32 767, équivalent non signé `ushort`)
- `int` : entier codé sur 4 octets (entre -2 147 483 648 et 2 147 483 647, équivalent non signé `uint`)
- `long` : entier codé sur 8 octets (entre -9 223 372 036 854 775 808 et +9 223 372 036 854 775 807, équivalent non signé `ulong`)
- `float` : nombre à virgule codé sur 4 octets
- `double` : nombre à virgule codé sur 8 octets
- `decimal` : nombre à virgule codé sur 16 octets
- `bool` : variable booléenne acceptant les valeurs `true` ou `false`
- `char` : caractère codé sur 2 octet situé entre deux `'`
- `string` : une chaîne de caractère situé entre deux `"`

C#

Exemple

```
int x;
```

C#

Exemple

```
int x;
```

Déclarer et initialiser une variable

```
int x = 5;
```

C#

Exemple

```
int x;
```

Déclarer et initialiser une variable

```
int x = 5;
```

Ceci est une erreur

```
byte x = 300;
```

C#

Pour afficher le contenu d'une variable dans la console

```
Console.Write("Valeur saisie : {0}", x);
```

C#

Pour afficher le contenu d'une variable dans la console

```
Console.Write("Valeur saisie : {0}", x);
```

{0} fait référence à la première variable située après le texte du message à afficher.

C#

Pour afficher le contenu d'une variable dans la console

```
Console.Write("Valeur saisie : {0}", x);
```

{0} fait référence à la première variable située après le texte du message à afficher.

On peut aussi utiliser la syntaxe suivante pour l'affichage d'une variable

```
Console.Write($"Valeur saisie : {x}");
```

C#

Pour afficher le contenu d'une variable dans la console

```
Console.Write("Valeur saisie : {0}", x);
```

{0} fait référence à la première variable située après le texte du message à afficher.

On peut aussi utiliser la syntaxe suivante pour l'affichage d'une variable

```
Console.Write($"Valeur saisie : {x}");
```

$ permet de remplacer les variables situées entre { } par leurs valeurs respectives

C#

Pour convertir le contenu d'une variable (le `cast` pour les types compatibles)

```
int x = 100;
byte z = (byte) x;
Console.Write(z); // affiche 100
```

C#

Pour convertir le contenu d'une variable (le `cast` pour les types compatibles)

```
int x = 100;
byte z = (byte) x;
Console.Write(z); // affiche 100
```

Attention aux valeurs qui dépassent l'intervalle

```
int x = 300;
byte z = (byte) x;
Console.WriteLine(z); // affiche 44
```

C#

Pour connaître le type d'une variable

```
int x = 300;
Console.WriteLine(x.GetType());
// affiche System.Int32

Console.WriteLine(x.GetType().Name);
// affiche Int32

Console.WriteLine(x is int);
// affiche True
```

C#

Pour connaître le type d'une variable

```
int x = 300;
Console.WriteLine(x.GetType());
// affiche System.Int32

Console.WriteLine(x.GetType().Name);
// affiche Int32

Console.WriteLine(x is int);
// affiche True
```


`int` VS `Int32`

- `int` est un synonyme (raccourci) de `Int32`
- En programmation, `int` est plus familier que `Int32`

C#

Pour connaître le nom complet d'un type

```
Console.WriteLine(typeof(int));
// affiche System.Int32
```

C#

Pour connaître le nom complet d'un type

```
Console.WriteLine(typeof(int));
// affiche System.Int32
```


Remarque

`typeof` s'applique seulement sur les types (pas sur les variables).

C#

Liste des synonymes

- `Byte` pour `byte`
- `SByte` pour `sbyte`
- `Int16` pour `short`
- `Int64` pour `long`
- `Int32` pour `int`
- `Single` pour `float`
- `Double` pour `double`
- `Boolean` pour `bool`
- `Char` pour `char`
- `String` pour `string`

C#

En C#, on ne peut utiliser une variable non initialisée

```
int z;
int x = z + 1;
```

C#

En C#, on ne peut utiliser une variable non initialisée

```
int z;
int x = z + 1;
```

Depuis C# 7.1, on peut initialiser une variable avec la valeur par défaut de son type

```
int z = default;
int x = z + 1;
Console.WriteLine(x);
// affiche 1
```

C#

Valeur par défaut selon le·s type·s

- 0 pour les types numériques
- `'\0'` pour `char`
- `false` pour `bool`

C#

Le type `Nullable`

- Les variables de type numérique, les variables booléennes et les caractères n'acceptent pas la valeur `null`
- Pour chacun de ces types non `nullable`, il existe un type `nullable` qui lui est associé `type?`

C#

Le type `Nullable`

- Les variables de type numérique, les variables booléennes et les caractères n'acceptent pas la valeur `null`
- Pour chacun de ces types non `nullable`, il existe un type `nullable` qui lui est associé `type?`

Pour les entiers

```
int? x = null;
Console.WriteLine(x); // affiche une ligne vide
x = 7;
Console.WriteLine(x); // affiche 7
```

C#

Pour tester si x a une valeur

```
int? x = null;
if (x.HasValue)
{
  Console.WriteLine($"La valeur de x est {x}");
}
else
{
  Console.WriteLine("x n'a pas de valeur");
}
```

C#

Pour tester si x a une valeur

```
int? x = null;
if (x.HasValue)
{
  Console.WriteLine($"La valeur de x est {x}");
}
else
{
  Console.WriteLine("x n'a pas de valeur");
}
```

Modifier la valeur de x (2 par exemple) et tester

C#

Le type `bool?`

- peut contenir trois valeurs différentes : `true`, `false` et `null`
- Les opérateurs logiques possibles sont & (et) et | (ou)

C#

Le type `bool?`

- peut contenir trois valeurs différentes : `true`, `false` et `null`
- Les opérateurs logiques possibles sont & (et) et | (ou)

Résultat des opérations logiques quand la valeur `null` est présente

a	b	a & b	a | b
`true`	`null`	`null`	`true`
`false`	`null`	`false`	`null`
`null`	`null`	`null`	`null`

C#

Typage dynamique

- Les variables implicitement typées sont déclarées avec le mot clé `var` sans préciser le type
- Il faut initialiser la valeur de la variable implicitement typée à la déclaration
- une fois la variable initialisée, la valeur aura le type de la valeur et ne peut donc plus changer de valeur

C#

Déclaration + initialisation d'une variable avec le mot clé `var`

```
var x = 2;
```

C#

Déclaration + initialisation d'une variable avec le mot clé `var`

```
var x = 2;
```

La variable x aura comme type `int` (`Int32`)

```
x = 2;
Console.WriteLine(x.GetType());
```

C#

Déclaration + initialisation d'une variable avec le mot clé `var`

```
var x = 2;
```

La variable x aura comme type `int` (`Int32`)

```
x = 2;
Console.WriteLine(x.GetType());
```

Ceci est une erreur

```
x = 2;
Console.WriteLine(x.GetType());
x = "bonjour";
Console.WriteLine(x.GetType());
```

C#

Exemple d'affectation de type

```
var x = 2;
Console.WriteLine(x.GetType());
// affiche Int32

var y = 2L;
Console.WriteLine(y.GetType());
// affiche Int64

var z = 2f;
Console.WriteLine(z.GetType());
// affiche Single

var t = 2.0;
Console.WriteLine(t.GetType());
// affiche Double
```

C#

Pour lire une chaîne saisie dans la console et l'enregistrer dans une variable

```
string s = Console.ReadLine();
```

`string` est le type de la valeur saisie et sauvegardée dans la variable `s`

C#

Pour lire un caractère saisi dans la console

```
char c = Console.ReadKey().KeyChar;
Console.WriteLine("caractere saisi : {0}", c);
```

C#

Pour lire un caractère saisi dans la console

```
char c = Console.ReadKey().KeyChar;
Console.WriteLine("caractere saisi : {0}", c);
```

ou aussi

```
char c = (char)Console.Read();
Console.WriteLine("caractere saisi : {0}", c);
```

C#

Pour lire un caractère saisi dans la console

```
char c = Console.ReadKey().KeyChar;
Console.WriteLine("caractere saisi : {0}", c);
```

ou aussi

```
char c = (char)Console.Read();
Console.WriteLine("caractere saisi : {0}", c);
```

ou encore

```
char c = Console.ReadLine()[0];
Console.WriteLine("caractere saisi : {0}", c);
```

C#

Ceci ne permet pas de lire un chiffre

```
int j = Console.Read();
Console.WriteLine("chiffre saisi : {0}", j);
```

ça affiche son code ASCII

C#

Ceci ne permet pas de lire un chiffre

```
int j = Console.Read();
Console.WriteLine("chiffre saisi : {0}", j);
```

ça affiche son code ASCII

Pour lire un entier saisi dans la console, il faut

- lire une chaîne de caractère
- ensuite la convertir

Lire une chaîne

```
string s =  Console.ReadLine();
```

Convertir la saisie : première méthode

```
int j = int.Parse(s);
Console.WriteLine("entier saisi : {0}", j);
```

Si `s` contient du texte, une exception de type `FormatException` sera levée

Lire une chaîne

```
string s =  Console.ReadLine();
```

Convertir la saisie : première méthode

```
int j = int.Parse(s);
Console.WriteLine("entier saisi : {0}", j);
```

Si `s` contient du texte, une exception de type `FormatException` sera levée

Convertir la saisie : deuxième méthode

```
int k = Convert.ToInt16(s);
Console.WriteLine("entier saisi : {0}", k);
```

Si `s` contient du texte, une exception de type `FormatException` sera aussi levée

C#

Pour les variables numériques (`int`, `float`...)

- `=` : affectation
- `+` : addition
- `-` : soustraction
- `*` : multiplication
- `/` : division
- `%` : reste de la division

C#

Exemple

```
int x = 5;
int y = 2;
Console.WriteLine($"{ x + y }");
// affiche 7
Console.WriteLine($"{ x - y }");
// affiche 3
Console.WriteLine($"{ x * y }");
// affiche 10
Console.WriteLine($"{ x / y }");
// affiche 2
Console.WriteLine($"{ (float)x / y }");
// affiche 2,5
Console.WriteLine($"{ x % y }");
// affiche 1
```

C#

Quelques raccourcis

- `i = i + 1` ⇒ `i++;`
- `i = i - 1` ⇒ `i--;`
- `i = i + 2` ⇒ `i += 2;`
- `i = i - 2` ⇒ `i -= 2;`

C#

Exemple de post-incrémentation

```
int i = 2;
int j = i++;
Console.WriteLine(i); // affiche 3
Console.WriteLine(j); // affiche 2
```

C#

Exemple de post-incrémentation

```
int i = 2;
int j = i++;
Console.WriteLine(i); // affiche 3
Console.WriteLine(j); // affiche 2
```

Exemple de pre-incrémentation

```
int i = 2;
int j = ++i;
Console.WriteLine(i); // affiche 3
Console.WriteLine(j); // affiche 3
```

C#

Coalescence nulle (? ?)

- Introduit dans **C# 7.3**
- Permet d'éviter d'affecter la valeur `null` à une variable

C#

Coalescence nulle (??)

- Introduit dans **C# 7.3**
- Permet d'éviter d'affecter la valeur `null` à une variable

Exemple

```
int? x = null;
int y = x ?? 5;

Console.WriteLine(y);
// affiche 5

x = 2;
y = x ?? 5;

Console.WriteLine(y);
// affiche 2
```

C#

Coalescence nulle et affectation (? ?=)

- Introduit dans **C# 8.0**
- Permet d'affecter une valeur à la même variable si sa valeur actuelle est `null`

C#

Coalescence nulle et affectation (? ?=)

- Introduit dans **C# 8.0**
- Permet d'affecter une valeur à la même variable si sa valeur actuelle est `null`

Exemple

```
int? x = null;
x ??= 5;

Console.WriteLine(x);
// affiche 5

x ??= 2;
Console.WriteLine(x);
// affiche 5
```

C#

L'opérateur + pour concaténer deux chaînes de caractères

```
string str1 = "bon";
string str2 = "jour";

Console.WriteLine(str1 + str2);
// affiche bonjour
```

Quelques méthodes pour les chaînes de caractères

- `Length` : retourne le nombre de caractère de la chaîne.
- `IndexOf(x)` : retourne l'indice de la première occurrence de la valeur de `x` dans la chaîne, -1 sinon.
- `Contains(x)` : retourne `true` si la chaîne contient la sous-chaîne `x`, `false` sinon.
- `Substring(i, j)` : permet d'extraire une sous-chaîne de taille `j` de la chaîne à partir de l'indice `i`
- `Equals(str)` : permet de comparer la chaîne à `str` et retourne `true` en cas d'égalité, `false` sinon.
- `Replace(old, new)` : permet de remplacer toute occurrence de la chaîne `old` dans la chaîne courante par `new` et retourne la nouvelle chaîne
- ...

Quelques méthodes pour les chaînes de caractères

- `Length` : retourne le nombre de caractère de la chaîne.
- `IndexOf(x)` : retourne l'indice de la première occurrence de la valeur de `x` dans la chaîne, -1 sinon.
- `Contains(x)` : retourne `true` si la chaîne contient la sous-chaîne `x`, `false` sinon.
- `Substring(i, j)` : permet d'extraire une sous-chaîne de taille `j` de la chaîne à partir de l'indice `i`
- `Equals(str)` : permet de comparer la chaîne à `str` et retourne `true` en cas d'égalité, `false` sinon.
- `Replace(old, new)` : permet de remplacer toute occurrence de la chaîne `old` dans la chaîne courante par `new` et retourne la nouvelle chaîne
- ...

Pour accéder à un caractère d'indice `i` d'une chaîne `str`, il faut écrire `str[i]`. Le premier caractère est d'indice 0.

C#

Exemple : `IndexOf(str)`

```
string str = "bonjour les bons jours";
int pos = str.IndexOf("bon");
Console.WriteLine(pos);
// affiche 0
```

C#

Exemple : `IndexOf(str)`

```
string str = "bonjour les bons jours";
int pos = str.IndexOf("bon");
Console.WriteLine(pos);
// affiche 0
```

Exemple : `IndexOf(str, startIndex)`

```
string str = "bonjour les bons jours";
int pos = str.IndexOf("bon", 5);
Console.WriteLine(pos);
// affiche 12
```

C#

Exemple : `Replace(old, new)`

```
string str = "bonjour les bons jours";
string newStr = str.Replace("jour", "soir");

Console.Write(str);
// affiche bonjour les bons jours

Console.Write(newStr);
// affiche bonsoir les bons soirs
```

C#

Exemple : `Replace(old, new)`

```
string str = "bonjour les bons jours";
string newStr = str.Replace("jour", "soir");

Console.Write(str);
// affiche bonjour les bons jours

Console.Write(newStr);
// affiche bonsoir les bons soirs
```


Remarques

- Les méthodes qui s'appliquent sur les `string` ne modifient pas l'objet appelant mais retournent un nouveau.
- On dit donc qu'un objet de type `string` est **immuable**.

C#

Quelques méthodes statiques de la classe Math

- `Math.Abs(x)` : retourne la valeur absolue de `x`.
- `Math.Pow(x, y)` : retourne la `x` puissance `y`.
- `Math.Max(x, y)` : retourne le max de `x` et `y`.
- `Math.Min(x, y)` : retourne le min de `x` et `y`.
- `Math.Sqrt(x)` : retourne la racine carré de `x`.
- `Math.Floor(x)`, `Math.Ceiling(x)` et `Math.Round(x)` : retournent l'arrondi de `x`.
- ...

C#

Exemples

```
Console.WriteLine(Math.Pow(2, 3));
// affiche 8

Console.WriteLine(Math.Sqrt(4));
// affiche 2

Console.WriteLine(Math.Abs(-2));
// affiche 2

Console.WriteLine(Math.Min(0, 1, 4, 2, -4, -5));
// affiche -5

Console.WriteLine(Math.Max(0, 1, 4, 2, -4, -5));
// affiche 4
```

C#

Exemple avec `Math.Floor(x)`, `Math.Ceiling(x)` **et** `Math.Round(x)`

```
Console.WriteLine(Math.Round(1.9)); // affiche 2
Console.WriteLine(Math.Round(1.5)); // affiche 2
Console.WriteLine(Math.Round(1.4)); // affiche 1
Console.WriteLine(Math.Ceiling(1.9)); // affiche 2
Console.WriteLine(Math.Ceiling(1.5)); // affiche 2
Console.WriteLine(Math.Ceiling(1.4)); // affiche 2
Console.WriteLine(Math.Floor(1.9)); // affiche 1
Console.WriteLine(Math.Floor(1.5)); // affiche 1
Console.WriteLine(Math.Floor(1.4)); // affiche 1
```

C#

Exécuter si une condition est vraie

```
if (condition)
{
    ...
}
```

C#

Exécuter si une condition est vraie

```
if (condition)
{
    ...
}
```

Exemple

```
var x = 3;
if (x >= 0)
{
   Console.WriteLine($"{ x } est positif");
}
```

C#

Exécuter si une condition est vraie

```
if (condition)
{
    ...
}
```

Exemple

```
var x = 3;
if (x >= 0)
{
   Console.WriteLine($"{ x } est positif");
}
```

Pour les conditions, on utilise les opérateurs de comparaison.

C#

Opérateurs de comparaison

- == : pour tester l'égalité
- != : pour tester l'inégalité
- > : supérieur à
- < : inférieur à
- >= : supérieur ou égal à
- <= : inférieur ou égal à

C#

Opérateurs de comparaison

- == : pour tester l'égalité
- != : pour tester l'inégalité
- > : supérieur à
- < : inférieur à
- >= : supérieur ou égal à
- <= : inférieur ou égal à

En **C#**, on ne peut comparer deux valeurs de type incompatible.

C#

Exercice

Écrire un code **C#** qui demande à l'utilisateur de saisir un entier positif et qui affiche ensuite sa parité (sans `else`).

C#

Exécuter un premier bloc si une condition est vraie, un deuxième sinon (le bloc `else`)

```
if (condition1)
{
   ...
}
else
{
   ...
}
```

C#

Exécuter un premier bloc si une condition est vraie, un deuxième sinon (le bloc `else`)

```
if (condition1)
{
   ...
}
else
{
   ...
}
```

Exemple

```
var x = 3;
if (x > 0)
{
    Console.WriteLine($"{ x } est positif");
}
else
{
    Console.WriteLine($"{ x } est négatif");
}
```

C#

Exercice

Écrire un programme **C#** qui permet de déterminer si une chaîne de caractères saisie par l'utilisateur contient un nombre pair ou impair de caractères.

C#

On peut enchaîner les conditions avec `else if` (et avoir un troisième bloc voire ... un nième)

```
if (condition1)
{
    ...
}
else if (condition2)
{
    ...
}
...
else
{
    ...
}
```

C#

Exemple

```
var x = -3;
if (x > 0)
{
    Console.WriteLine($"{ x } est positif");
}
else if (x < 0)
{
    Console.WriteLine($"{ x } est négatif");
}
else
{
    Console.WriteLine($"{ x } est nul");
}
```

C#

Exercice 1

Écrire un programme qui

- demande à l'utilisateur de saisir deux nombres et un opérateur (+, *, - ou /) de type caractère.
- affiche le résultat de l'opération arithmétique en fonction de l'opérateur saisi.

C#

Exercice 2

Écrire un code **C#** qui

- demande à l'utilisateur de saisir trois entiers `a`, `b` et `c`
- affiche le résultat de l'équation $\texttt{a}x^2 \texttt{ + b}x \texttt{ + c = 0}$.

C#

Opérateurs logiques

- `&&` : et
- `||` : ou
- `!` : non
- `^` : xor

C#

Opérateurs logiques

- && : et
- || : ou
- ! : non
- ^ : xor

Tester plusieurs conditions (en utilisant des opérateurs logiques)

```
if (condition1 && !condition2 || condition3)
{
    ...
}
[else ...]
```

C#

Exercice 1

Écrire un code **C#** qui

- demande à l'utilisateur de saisir une année (un entier),
- affiche si l'année saisie est bissextile (voir `https://fr.wikipedia.org/wiki/Ann%C3%A9e_bissextile`).

C#

Exercice 2

Écrire un code **C#** qui

- demande à l'utilisateur de saisir deux entiers `a` et `b` différents de zéro,
- affiche le signe du résultat de la multiplication sans calculer le produit.

C#

Exercice 3

Écrire un code **C#** qui

- demande à l'utilisateur de saisir deux entiers `a` et `b`,
- détermine et affiche si le résultat de l'addition (sans calculer la somme) est pair ou impair.

C#

Structure conditionnelle avec `switch`

```
int x = 5;
switch (x)
{
    case 1:
      Console.WriteLine("un");
      break;
    case 2:
      Console.WriteLine("deux");
      break;
    case 3:
      Console.WriteLine("trois");
      break;
    default:
      Console.WriteLine("autre");
      break;
}
```

C#

La variable dans `switch` peut être de type

- numérique : `int`, `long`...
- booléen : `bool`
- text : `char` ou `string`
- énumération

Le bloc `default` dans `switch`

- Le bloc `default` peut apparaître à n'importe quelle position dans `switch`. Quelle que soit sa position, il est toujours évalué en dernier, une fois que tous les blocs `case` ont été évaluées.
- En l'absence d'un bloc `default` et si aucun bloc `case` n'est exécuté, le bloc `switch` sera traversé sans être exécuté.
- `break` permet de quitter `switch`
- Même dans bloc `default`, il faut placer un `break`.

C#

On peut aussi regrouper des `case`

```
int x = 1;
switch (x)
{
    case 1:
    case 2:
      Console.WriteLine("un ou deux");
      break;
    case 3:
      Console.WriteLine("trois");
      break;
    default:
      Console.WriteLine("autre");
      break;
}
```

C#

On peut faire autres tests que l'égalité dans un `switch` [C# 9]

```
int x = 5;
switch (x)
{
    case < 0:
        Console.WriteLine("négatif");
        break;
    case > 0:
        Console.WriteLine("positif");
        break;
    default:
        Console.WriteLine("nul");
        break;
}
```

C#

On peut aussi tester deux variables

```
int x = 3, y = -5;
switch (x, y)
{
    case ( > 0, > 0):
        Console.WriteLine($"{x} et {y} sont positifs");
        break;
    case ( < 0, < 0):
        Console.WriteLine($"{x} et {y} sont négatifs");
        break;
    default:
        Console.WriteLine($"{x} et {y} n'ont pas le même signe");
        break;
}
```

C#

On peut aussi aussi ajouter une spécification avec `when`

```
int x = 3, y = -5;
switch (x, y)
{
    case ( > 0, > 0) when x == y:
        Console.WriteLine($"{x} et {y} sont positifs et égaux");
        break;
    case ( > 0, > 0):
        Console.WriteLine($"{x} et {y} sont positifs");
        break;
    case ( < 0, < 0):
        Console.WriteLine($"{x} et {y} sont négatifs");
        break;
    default:
        Console.WriteLine($"{x} et {y} n'ont pas le même signe");
        break;
}
```

C#

Exercice

Écrire un programme qui demande à l'utilisateur de saisir l'indice d'un mois (entier compris entre 1 et 12) et qui retourne le nombre de jours de ce mois

- si l'entier est égal à 1, 3, 5, 7, 8, 10 ou 12 le programme affiche 31
- sinon si l'entier est égal à 4, 6, 9 ou 11 le programme affiche 30
- sinon si l'entier est égal à 2, le programme demande à l'utilisateur de saisir l'année et lui retourne 29 si l'année est bissextile, 28 sinon (voir `https://fr.wikipedia.org/wiki/Ann%C3%A9e_bissextile`)
- pour toute autre valeur, le programme affiche une erreur

C#

Depuis la version 8 de C#, on peut simplifier l'écriture d'un bloc switch

```
int x = 1;

var result = x switch
{
    1 => "un",
    2 => "deux",
    3 => "trois",
    _ => "autre",
};

Console.WriteLine(result);
```

C#

Depuis la version 8 de C#, on peut simplifier l'écriture d'un bloc switch

```
int x = 1;

var result = x switch
{
    1 => "un",
    2 => "deux",
    3 => "trois",
    _ => "autre",
};

Console.WriteLine(result);
```

_ remplace `default`.

C#

On peut aussi utiliser des `goto` pour renvoyer vers un autre `case`

```
int x = 4;
switch (x)
{
    case 1:
        Console.WriteLine($"{x} est divisible par 1");
        break;
    case 2:
        Console.WriteLine($"{x} est divisible par 2");
        goto case 1;
    case 3:
        Console.WriteLine($"{x} est divisible par 3");
        goto case 1;
    case 4:
        Console.WriteLine($"{x} est divisible par 4");
        goto case 2;
    default:
        Console.WriteLine("autre");
        break;
}
```

C#

On peut aussi simplifier les tests en utilisant les expressions ternaires (Elvis Operator)

```
int x = 2;
String  type = ( x % 2 == 0 ) ? "pair" : "impair";
Console.WriteLine(type); // affiche pair
```

C#

Une constante ?

- élément qui ne peut changer de valeur
- déclaré avec le mot-clé `const`

C#

Une constante ?

- élément qui ne peut changer de valeur
- déclaré avec le mot-clé `const`

Déclaration d'une constante

```
const double Pi = 3.1415;
```

C#

Une constante ?

- élément qui ne peut changer de valeur
- déclaré avec le mot-clé `const`

Déclaration d'une constante

```
const double Pi = 3.1415;
```

L'instruction suivante ne peut être acceptée

```
Pi = 5;
```

C#

Boucle `while` : à chaque itération on teste si la condition est vraie avant d'accéder aux traitements

```
while (condition[s]) {
   ...
}
```

C#

Boucle `while` : à chaque itération on teste si la condition est vraie avant d'accéder aux traitements

```
while (condition[s]) {
   ...
}
```


Attention aux boucles infinies, vérifier que la condition d'arrêt sera bien atteinte après un certain nombre d'itérations.

C#

Exemple

```
var i = 0;
while (i < 5) {
  Console.WriteLine(i);
  i++;
}
```

C#

Exemple

```
var i = 0;
while (i < 5) {
  Console.WriteLine(i);
  i++;
}
```

Le résultat est

```
0
1
2
3
4
```

C#

Exercice 1

Écrire un programme qui permet d'afficher les nombres pairs inférieurs à 10.

C#

Exercice 2

Écrire un programme qui

- demande à l'utilisateur de saisir deux entiers `a` et `b` (avec `a < b`),
- afficher les nombres pairs compris entre `a` et `b`.

C#

Exercice 3

Écrire un programme qui demande à l'utilisateur de saisir un entier positif `n` et qui calcule puis affiche la somme de tous les entiers positifs inférieurs à `n`.

C#

La Boucle `do` ... `while` exécute le bloc au moins une fois ensuite elle vérifie la condition

```
do {
   ...
}
while (condition[s]);
```

C#

La Boucle `do` ... `while` exécute le bloc au moins une fois ensuite elle vérifie la condition

```
do {
   ...
}
while (condition[s]);
```


Attention aux boucles infinies, vérifier que la condition d'arrêt sera bien atteinte après un certain nombre d'itérations.

C#

Exemple

```
var i = 0;
do {
  Console.WriteLine(i);
  i++;
} while (i < 5);
```

C#

Exemple

```
var i = 0;
do {
  Console.WriteLine(i);
  i++;
} while (i < 5);
```


Le résultat est

```
0
1
2
3
4
```

C#

Exercice

Écrire un programme qui demande en boucle à l'utilisateur de saisir un entier jusqu'à l'obtention de 0 puis affiche la somme de tous ces entiers saisis.

C#

Boucle `for`

```
for (initialisation; condition[s]; incrémentation) {
   ...
}
```

C#

Boucle `for`

```
for (initialisation; condition[s]; incrémentation) {
   ...
}
```


Attention aux boucles infinies si vous modifiez la valeur du compteur à l'intérieur de la boucle.

C#

Exemple

```
for (var i = 0; i < 5; i++) {
    Console.WriteLine(i);
}
```

C#

Exemple

```
for (var i = 0; i < 5; i++) {
    Console.WriteLine(i);
}
```

Le résultat est

```
0
1
2
3
4
```

C#

Exercice 1

Écrire un code **C#** qui permet d'afficher les nombres pairs compris entre 0 et 10.

C#

Première solution

```
for (var i = 0; i < 10; i++) {
  if (i % 2 == 0) {
    Console.WriteLine(i);
  }
}
```

C#

Première solution

```
for (var i = 0; i < 10; i++) {
  if (i % 2 == 0) {
    Console.WriteLine(i);
  }
}
```


Deuxième solution

```
for (var i = 0; i < 10; i += 2) {
    Console.WriteLine(i);
}
```

C#

Étant donnée la chaîne de caractères suivante

```
string maChaine = "Une première phrase. Et voici une deuxième."
   ;
```

C#

Étant donnée la chaîne de caractères suivante

```
string maChaine = "Une première phrase. Et voici une deuxième."
   ;
```

Exercice 2

Écrire un programme qui permet de compter le nombre de mots et de phrases dans `maChaine`.

Exercice 3

Écrire un programme qui demande à l'utilisateur de saisir deux entiers positifs et qui calcule puis affiche leur plus grand diviseur commun.

Remarques

Dans ces structures itératives, on peut utiliser :

- `break` : pour quitter la boucle
- `continue` : pour ignorer l'itération courante

C#

Exemple avec `break`

```
int j = 5;
do
{
    Console.WriteLine(j);
    if (j == 3)
        break;
    j--;
}
while (j > 0);
```

C#

Exemple avec `break`

```
int j = 5;
do
{
    Console.WriteLine(j);
    if (j == 3)
        break;
    j--;
}
while (j > 0);
```

Résultat

5 4 3

C#

Exemple avec `continue`

```
int j = 5;
while (j > 0)
{
    if (j == 3)
    {
        j--;
        continue;
    }
    j--;
    Console.WriteLine(j);
}
```

C#

Exemple avec `continue`

```
int j = 5;
while (j > 0)
{
    if (j == 3)
    {
        j--;
        continue;
    }
    j--;
    Console.WriteLine(j);
}
```

Résultat

4 3 1 0

C#

Variables multi-valeurs

- Les variables (vues dans les sections précédentes) permettent de stocker une seule valeur à la fois.
- Mais il existe plusieurs structures de données en **C#** qui permettent de stocker simultanément plusieurs valeurs telles que
  - Les tableaux
  - Les collections (à voir dans un prochain chapitre)
  - Les énumérations (à voir dans un prochain chapitre)
  - Les tuples (à voir dans un prochain chapitre)

C#

Tableaux ?

- une variable
- contenant un ensemble de valeurs
  - de même type
  - et dont le nombre (de valeurs) est fixé à la déclaration

C#

Déclaration

```
type[] nomTableau = new type[nbrElement];
```

C#

Déclaration

```
type[] nomTableau = new type[nbrElement];
```

Exemple

```
int[] tab = new int[3];
```

C#

Déclaration

```
type[] nomTableau = new type[nbrElement];
```

Exemple

```
int[] tab = new int[3];
```

Remarques

- Tous les éléments du tableau sont initialisés à 0.
- `tab[i]` : permet d'accéder à l'élément d'indice `i` du tableau
- Le premier élément d'un tableau est d'indice 0.
- On ne peut dépasser la taille initiale d'un tableau ni changer le type déclaré.

C#

Déclaration + initialisation

```
int[] tab = new int[] { 3, 5, 4 };
```

C#

Déclaration + initialisation

```
int[] tab = new int[] { 3, 5, 4 };
```

On peut aussi utiliser le raccourci suivant

```
int[] tab =  { 3, 5, 4 };
```

C#

Déclaration + initialisation

```
int[] tab = new int[] { 3, 5, 4 };
```

On peut aussi utiliser le raccourci suivant

```
int[] tab =  { 3, 5, 4 };
```

Cette écriture déclenche un `IndexOutOfRangeException`

```
tab[3] = 2;
```

C#

Parcourir un tableau avec un `for`

```
for (int i = 0; i < tab.Length; i++)
{
    Console.WriteLine(tab[i]);
}
```

C#

Parcourir un tableau avec un `for`

```
for (int i = 0; i < tab.Length; i++)
{
    Console.WriteLine(tab[i]);
}
```

Parcourir un tableau avec un `foreach`

```
foreach (int n in tab)
{
    Console.WriteLine(n);
}
```

C#

Trier un tableau (unidimensionnel)

```
Array.sort(tab);
```

C#

Trier un tableau (unidimensionnel)

```
Array.sort(tab);
```

Autres opérations sur les tableaux

- `Array.Clear(tab, n, m)` : supprime les `m` valeurs (et non pas les éléments) du tableau en commençant par l'élément d'indice `n`. (il existe aussi `reverse` pour inverser l'ordre,...)
- `Array.IndexOf(tab, n))` : retourne l'indice de la première apparition de la valeur `n` dans le tableau `tab`. (il existe aussi `LastIndex`, `Exists`...)
- `Array.Resize(ref tab, n)` : réduit le nombre d'élément de `tab` au `n` premier élément
- ...

C#

Étant donné le tableau suivant

```
string [] voitures = { "peugeot", "ford", "fiat", "mercedes", "seat" };
```

C#

Étant donné le tableau suivant

```
string [] voitures = { "peugeot", "ford", "fiat", "mercedes", "seat" };
```

Exercice 1

Écrire un programme qui permet de calculer le nombre total de caractères de toutes les chaînes présentes dans le tableau `voitures`.

C#

Étant données les deux variables suivantes

```
string voiture = "ford";
string [] voitures = { "peugeot", "ford", "fiat", "mercedes", "seat", "
   ford", "fiat", "ford" };
```

C#

Étant données les deux variables suivantes

```
string voiture = "ford";
string [] voitures = { "peugeot", "ford", "fiat", "mercedes", "seat", "
   ford", "fiat", "ford" };
```

Exercice 2

Écrire un programme qui permet de calculer et afficher le nombre d'occurrence de `voiture` dans `voitures`.

C#

Déclaration d'un tableau à deux dimensions

```
type[,] nomTableau = new type[nbLignes, nbColonnes];
```

C#

Déclaration d'un tableau à deux dimensions

```
type[,] nomTableau = new type[nbLignes, nbColonnes];
```

Déclaration + initialisation

```
int[,] tab2dim = new int[,]
{
    {1, 2},
    {3, 4}
};
```

C#

Déclaration d'un tableau à deux dimensions

```
type[,] nomTableau = new type[nbLignes, nbColonnes];
```

Déclaration + initialisation

```
int[,] tab2dim = new int[,]
{
    {1, 2},
    {3, 4}
};
```

Ou

```
int[,] tab2dim =
{
    {1, 2},
    {3, 4}
};
```

C#

Parcourir un tableau à deux dimensions

```
foreach (int n in tab2dim)
{
    Console.Write(n);
}
```

C#

Parcourir un tableau à deux dimensions

```
foreach (int n in tab2dim)
{
    Console.Write(n);
}
```

Ou

```
for (int i = 0; i <2; i++)
{
    for (int j = 0; j < 2; j++)
    {
        Console.WriteLine(tab2dim[i,j]);
    }
}
```

C#

Parcourir un tableau à deux dimensions

```
foreach (int n in tab2dim)
{
    Console.Write(n);
}
```

Ou

```
for (int i = 0; i <2; i++)
{
    for (int j = 0; j < 2; j++)
    {
        Console.WriteLine(tab2dim[i,j]);
    }
}
```

Ne pas confondre `tab[,]` avec `tab[][]` qui veut dire un tableau de tableaux.

Étant données les variables suivantes

```
int valeur = 5;
int[,] matrice =
{
    { 2, 3, 5, 7 },
    { 5, 1, 5, 5 },
    { 4, 2, 9, 5 },
    { 8, 5, 6, 5 }
};
```

Étant données les variables suivantes

```
int valeur = 5;
int[,] matrice =
{
    { 2, 3, 5, 7 },
    { 5, 1, 5, 5 },
    { 4, 2, 9, 5 },
    { 8, 5, 6, 5 }
};
```

Exercice

Écrire un programme qui

- parcourt la matrice précédente,
- calcule le nombre d'occurrence de `valeur` dans chaque ligne de `matrice`,
- stocke le résultat dans un tableau `occurrences`.

C#

Étant données les variables suivantes

```
int valeur = 5;
int[,] matrice =
{
    { 2, 3, 5, 7 },
    { 5, 1, 5, 5 },
    { 4, 2, 9, 5 },
    { 8, 5, 6, 5 }
};
```

C#

Étant données les variables suivantes

```
int valeur = 5;
int[,] matrice =
{
    { 2, 3, 5, 7 },
    { 5, 1, 5, 5 },
    { 4, 2, 9, 5 },
    { 8, 5, 6, 5 }
};
```


Exercice

Écrire un programme qui permet de déterminer si `valeur` est présente dans chaque ligne de `matrice`.

C#

Exercice

Écrire un programme qui calcule

1. la somme de deux matrices carrées,
2. le produit de deux matrices carrées.

C#

Une méthode

visibilité [+ `static`] + type de retour + nomMéthode([les paramètres]) + son implémentation

C#

Une méthode

visibilité [+ `static`] + type de retour + nomMéthode([les paramètres]) + son implémentation

Exemple de déclaration d'une méthode

```
public static int Somme (int a, int b)
{
    return a + b;
}
```

C#

Une méthode

visibilité [+ `static`] + type de retour + nomMéthode([les paramètres]) + son implémentation

Exemple de déclaration d'une méthode

```
public static int Somme (int a, int b)
{
    return a + b;
}
```

Exemple d'appel d'une méthode

```
Console.Writeline(Somme(2, 3));
// affiche 5
```

C#

Exercice

Écrire une méthode `Maximum2(a, b)` qui retourne le maximum entre `a` et `b`.

C#

Exercice

Écrire une méthode `Maximum2(a, b)` qui retourne le maximum entre `a` et `b`.

Exercice

Écrire une méthode `Maximum3(a, b, c)` qui retourne le maximum entre `a`, `b` et `c` (vous pouvez utiliser la méthode `Maximum2`) .

C#

Exercice

Écrire une méthode qui permet de déterminer si un nombre passé en paramètre est premier. La méthode retourne un booléen.

C#

`ref` VS `out` VS `in`

- `ref` : permet à une méthode d'utiliser et modifier la copie originelle de la valeur d'une variable passée en paramètre (modification facultative).
- `out` : oblige une méthode à modifier la valeur d'une variable passée en paramètre (modification obligatoire).
- `in` (depuis **C# 7.2**) : permet à une méthode d'utiliser la copie originelle de la valeur d'une variable passée en paramètre mais sans pouvoir la modifier (modification interdite).

C#

La modification d'un paramètre précédé par `in` n'est pas autorisée (Erreur CS8331).

```
public static void NoModification(in int a)
{
    a++;
}
```

C#

La modification d'un paramètre précédé par `in` n'est pas autorisée (Erreur CS8331).

```
public static void NoModification(in int a)
{
    a++;
}
```


Liste de toutes les erreurs

https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/compiler-messages/

C#

Considérons la méthode suivante qui permet d'échanger les valeurs de deux variables entières

```
public static void Permutation(int i, int j)
{
    int aux = i;
    i = j;
    j = aux;
}
```

C#

Considérons la méthode suivante qui permet d'échanger les valeurs de deux variables entières

```
public static void Permutation(int i, int j)
{
    int aux = i;
    i = j;
    j = aux;
}
```

Et si on teste cette méthode en ajoutant le code suivant dans le `Main`

```
int n = 2;
int m = 5;
Permutation(n, m);
Console.WriteLine($"Après permutation, n = { n }");
// affiche Après permutation, n = 2
Console.WriteLine($"Après permutation, m = { m }");
// affiche Après permutation, m = 5
```

C#

Que s'est-il passé ?

- Par défaut, les types simples sont passés par valeur
- C'est à dire la méthode appelée (ici `Permutation`) travaille seulement sur une copie de la variable
- La méthode appelante (ici `Main`) conserve donc la valeur originelle de la variable

C#

Que s'est-il passé ?

- Par défaut, les types simples sont passés par valeur
- C'est à dire la méthode appelée (ici `Permutation`) travaille seulement sur une copie de la variable
- La méthode appelante (ici `Main`) conserve donc la valeur originelle de la variable

Comment faire pour travailler sur la valeur originelle ?

Il faut effectuer un passage par référence

C#

Il faut ajouter le mot-clé `ref` dans la signature de la méthode appelée

```
public static void Permutation(ref int i, ref int j)
{
    int aux = i;
    i = j;
    j = aux;
}
```

C#

Il faut ajouter le mot-clé `ref` dans la signature de la méthode appelée

```
public static void Permutation(ref int i, ref int j)
{
    int aux = i;
    i = j;
    j = aux;
}
```

Et aussi lors de l'appel de cette méthode dans `Main`

```
int n = 2;
int m = 5;
Permutation(ref n, ref m);
Console.WriteLine($"Après permutation, n = { n }");
// affiche Après permutation, n = 5
Console.WriteLine($"Après permutation, m = { m }");
// affiche Après permutation, m = 2
```

C#

Considérons deux méthodes qui permettent de retourner la valeur `min` ou `max`

```
public static int FindMax(int i, int j)
{
    return i > j ? i : j;
}

public static int FindMin(int i, int j)
{
    return i < j ? i : j;
}
```

C#

Considérons deux méthodes qui permettent de retourner la valeur `min` ou `max`

```
public static int FindMax(int i, int j)
{
    return i > j ? i : j;
}

public static int FindMin(int i, int j)
{
    return i < j ? i : j;
}
```


Comment faire pour fusionner les deux méthodes ?

Sachant qu'une méthode (tout comme une fonction) ne peut retourner qu'une seule valeur

C#

Il faut ajouter les valeurs à retourner comme paramètres de la méthode et les précéder par `out`

```
public static void FindMinMax(int i, int j, out int max, out
   int min)
{
    max = i > j ? i : j;
    min = i < j ? i : j;
}
```

C#

Il faut ajouter les valeurs à retourner comme paramètres de la méthode et les précéder par `out`

```
public static void FindMinMax(int i, int j, out int max, out
   int min)
{
    max = i > j ? i : j;
    min = i < j ? i : j;
}
```

Pour appeler cette méthode dans le `Main`

```
int x;
int y;
FindMinMax(2, 3, out x, out y);
Console.WriteLine($"Le max de 2 et 3 est : { x }");
// affiche 3
Console.WriteLine($"Le min de 2 et 3 est : { y }");
// affiche 2
```

C#

Remarque

Contrairement à `ref`, les paramètres précédés par `out` peuvent ne pas être initialisés.

C#

Convertir la saisie : troisième méthode avec `TryParse` utilisant `out`

```
int m;
int.TryParse(s, out m);
Console.WriteLine("entier saisi : {0}", m);
```

C#

Convertir la saisie : troisième méthode avec `TryParse` utilisant `out`

```
int m;
int.TryParse(s, out m);
Console.WriteLine("entier saisi : {0}", m);
```


Si `s` contient du texte, aucune exception ne sera levée et `m` contiendra 0.

C#

Remarques

- `int.TryParse` retourne `true` si la conversion a eu lieu, `false` sinon.
- En utilisant `Convert`, il faut préciser le nombre de bits pour coder l'entier.

C#

Hypothèse

Si on veut modifier la méthode `FindMax` pour qu'elle retourne la valeur maximale quel que soit le nombre de paramètres passés.

C#

Hypothèse

Si on veut modifier la méthode `FindMax` pour qu'elle retourne la valeur maximale quel que soit le nombre de paramètres passés.

Solution

utiliser l'argument `params`

C#

La méthode `FindMaxFromNValue`

```
public static int FindMaxFromNValue(params int[] list)
{
    int max = list[0];
    for (int i = 1; i < list.Length; i++)
    {
        if (list[i] > max)
            max = list[i];
    }
    return max;
}
```

C#

La méthode `FindMaxFromNValue`

```
public static int FindMaxFromNValue(params int[] list)
{
    int max = list[0];
    for (int i = 1; i < list.Length; i++)
    {
        if (list[i] > max)
            max = list[i];
    }
    return max;
}
```


Pour appeler cette méthode dans le `Main` **avec un nombre de paramètres différents**

```
int h = FindMaxFromNValue(2, 8, 5);
Console.WriteLine($"Le max est : { h }");

int g = FindMaxFromNValue(1, 7, 8, 2);
Console.WriteLine($"Le max est : { g }");
```

C#

Exercice 1

Écrire une méthode `TotalCaracteres` qui accepte un nombre variable de paramètre de type `String` et qui retourne le nombre total de caractères de toutes les chaînes reçues.

C#

Exercice 2

Écrire une méthode `EstDivisiblePar` qui détermine si le premier paramètre est divisible par tous les autres paramètres. Le nombre de paramètres est variable et la méthode retourne un booléen.

Exemple 2

```
Console.WriteLine(EstDivisiblePar(10, 2, 3, 5))
// false

Console.WriteLine(EstDivisiblePar(10, 2, 5));
// true
```

C#

Considérons la méthode suivante qui retourne le résultat de la division

```
public static int Division(int numerateur, int denominateur)
{
    return numerateur / denominateur;
}
```

C#

Considérons la méthode suivante qui retourne le résultat de la division

```
public static int Division(int numerateur, int denominateur)
{
    return numerateur / denominateur;
}
```


Le résultat peut changer selon l'ordre des arguments

```
Console.WriteLine(Division(5, 2));
// affiche 2

Console.WriteLine(Division(2, 5));
// affiche 0
```

C#

Considérons la méthode suivante qui retourne le résultat de la division

```
public static int Division(int numerateur, int denominateur)
{
    return numerateur / denominateur;
}
```


Le résultat peut changer selon l'ordre des arguments

```
Console.WriteLine(Division(5, 2));
// affiche 2

Console.WriteLine(Division(2, 5));
// affiche 0
```

En nommant les arguments, l'ordre n'a plus d'importance

```
Console.WriteLine(Division(denominateur: 2, numerateur: 5));
// affiche 2
```

C#

Nous pouvons définir des valeurs par défaut pour les différents paramètres

```
public static int Division(int numerateur = 0, int denominateur
    = 1)
{
    return numerateur / denominateur;
}
```

C#

Nous pouvons définir des valeurs par défaut pour les différents paramètres

```
public static int Division(int numerateur = 0, int denominateur
    = 1)
{
    return numerateur / denominateur;
}
```

Ainsi, nous pouvons effectuer des appels avec 0, 1 ou 2 arguments

```
Console.WriteLine(Division(5, 2));
// affiche 2

Console.WriteLine(Division(denominateur: 2));
// affiche 0

Console.WriteLine(Division());
// affiche 0
```