

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`



1 Variables

- Opérations de lecture et écriture
- Typage statique
- Typage dynamique

2 Opérations sur les variables

3 Structures conditionnelles

- `if`
- `if ... else`
- `if ... else if ... else`
- `switch`
- Opérateur ternaire

Plan

4 Constantes

5 Structures itératives

- while
- do ... while
- for

6 Tableaux

7 Paramètres de méthode

- in
- ref
- out
- params
- Arguments nommés
- Paramètres avec valeurs par défaut

8 Conversion

9 Méthodes de la classe Math

Une variable ?

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

© Achref EL

Une variable ?

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

Remarque

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

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.

© Achret L

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.

En C#, il y a deux types de typage

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

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`

© Achref EL MOUELHI ©

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`

Pour afficher le contenu d'une variable dans la console

```
Console.Write("chaîne saisie : {0}", s);
```

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

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`

Pour afficher le contenu d'une variable dans la console

```
Console.Write("chaine saisie : {0}", s);
```

`{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($"chaine saisie : {s}");
```

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

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;
```

© Achref EL MOUELHI ©

C#

Exemple

```
int x;
```

Déclarer et initialiser une variable

```
int x = 5;
```

© Achref EL MOUADJID

C#

Exemple

```
int x;
```

Déclarer et initialiser une variable

```
int x = 5;
```

Ceci est une erreur

```
byte x = 300;
```

C#

Exemple

```
int x;
```

Déclarer et initialiser une variable

```
int x = 5;
```

Ceci est une erreur

```
byte x = 300;
```

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#

Attention aux valeurs qui dépassent l'intervalle

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

© Achref EL MOUELHI

C#

Attention aux valeurs qui dépassent l'intervalle

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

Pour connaître le type d'une variable

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

C#

Attention aux valeurs qui dépassent l'intervalle

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

Pour connaître le type d'une variable

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

int VS Int32

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

C#

La 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

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?

© Achref L.

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
```

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");
}
```

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

Conversion d'un type `Nullable` (Nullish Coalescing)

```
int? x = null;  
int y = x ?? 5;  
Console.WriteLine($"La valeur de y est { y }");
```

Le type `bool`?

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

© Achref EL MOU

C#

Le type `bool`?

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

Le 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

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;
```

© Achref EL MOUELHI ©

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
```


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
```

Quelques raccourcis

- `i = i + 1` \Rightarrow `i++;`
- `i = i - 1` \Rightarrow `i--;`
- `i = i + 2` \Rightarrow `i += 2;`
- `i = i - 2` \Rightarrow `i -= 2;`

Exemple de post-incrémentation

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

© Achref EL MOU

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
```

L'opérateur + pour concaténer deux chaînes de caractère

```
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
- ...

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

Exemple : Replace(old, new)

```
String str = "bonjour";  
Console.Write($"{ str.Replace("jour", "soir") }");  
// affiche bonsoir
```

© Achref EL MOUL

Exemple : Replace(old, new)

```
String str = "bonjour";  
Console.Write($"{ str.Replace("jour", "soir") }");  
// affiche bonsoir
```

Exemple : IndexOf(str, startIndex)

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

C#

Exécuter si une condition est vraie

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

© Achref EL MOUËL

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.

Opérateurs de comparaison

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

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#

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

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

© Achref EL M...

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#

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");
}
```

Opérateurs logiques

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

© Achref EL

Opérateurs logiques

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

Tester plusieurs conditions (en utilisant des opérateurs logiques)

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

Exercice

- Écrire un code **C#** qui retourne le signe du résultat de la multiplication de deux nombres sans calculer leur produits.
- Les deux nombres sont des entiers différents de zéro.

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;
}
```

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 évalués.
- 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;
}
```

Exercice avec `switch`

Écrire un code **C#** qui

- demande à l'utilisateur de saisir l'indice d'un mois (une chaîne de caractères dont la valeur est comprise entre 1 et 12).
- affiche le nombre de jours de ce mois (28, 29, 30 ou 31).

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
```

Une constante ?

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

© Achref EL MOUËL

Une constante ?

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

Déclaration d'une constante

```
const double pi = 3.1415;
```

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;
```

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

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

© Achret

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++;
}
```

© Achref EL MICHAËL

C#

Exemple

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

Le résultat est

0
1
2
3
4

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

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

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);
```

© Achref EL MICHAËL

C#

Exemple

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

Le résultat est

0
1
2
3
4

C#

Boucle for

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

© Achref EL M...

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);  
}
```

© Achref EL MOUËL

C#

Exemple

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

Le résultat est

```
0  
1  
2  
3  
4
```

C#

Exercice

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

© Achref EL MOUELHI ©

C#

Exercice

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

Première solution

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

C#

Exercice

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

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);  
}
```

Remarques

Dans ces structures itératives, on peut utiliser :

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

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);
```

Affichage : 5 4 3

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);
}
```

Affichage : 4 3 1 0

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)

Tableaux ?

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

Déclaration

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

© Achref EL MOUELHI ©

C#

Déclaration

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

Exemple

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

© Achref EL MOUADJID

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.

Déclaration + initialisation

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

© Achref EL MOUELHI ©

Déclaration + initialisation

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

On peut aussi utiliser le raccourci suivant

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

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;
```

Parcourir un tableau avec un `for`

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

© Achref EL MOUËL

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#

Déclaration d'un tableau à deux dimensions

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

© Achref EL MOUELHI ©

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}  
};
```

Parcourir un tableau à deux dimensions

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

© Achref EL MOUL

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.

C#

Trier un tableau (unidimensionnel)

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

© Achref EL MOUELHI ©

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
- ...

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).

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

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

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;
}
```

© Achref EL M.

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
```


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

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;
}
```

© Achref EL ME

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
```

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;
}
```

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;
}
```

© Achref EL M...

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
```


Remarque

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

Hypothèse

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

© Achref EL W.

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;
}
```

© Achref

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#

Considérons la méthode suivante qui retourne le résultat de la division

```
public static int Division(int numérateur, int dénominateur)
{
    return numérateur / dénominateur;
}
```

© Achref EL MOUELHI ©

C#

Considérons la méthode suivante qui retourne le résultat de la division

```
public static int Division(int numérateur, int dénominateur)
{
    return numérateur / dénominateur;
}
```

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 numérateur, int dénominateur)
{
    return numérateur / dénominateur;
}
```

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(dénominateur: 2, numérateur: 5));
// affiche 2
```


C#

Nous pouvons définir des valeurs par défaut pour les différents paramètres

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

© Achref EL MOUËL

C#

Nous pouvons définir des valeurs par défaut pour les différents paramètres

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

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

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

C#

Pour lire un caractère saisi dans la console

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

© Achref EL MOUELHI ©

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#

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);
```

Pour attendre la saisie d'un caractère sans le récupérer

```
Console.ReadKey();
```

Ceci ne permet pas de lire un chiffre

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

ça affiche son code ASCII

© Achref EL MOUETI

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

C#

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

C#

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

Convertir la saisie : troisième méthode

```
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.

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.

Exercice

Écrire un code **C#** 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.

On peut aussi utiliser les 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
```

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
```