1 Module A.4 : Les chaînes de caractères

La manipulation de textes est une tâche courante effectuée par les programmes informatiques. Le type principal associé à la gestion du texte est **string**.
Ce module a pour objectif de présenter ses principales caractéristiques ainsi qu'une liste de fonctions qui lui est associée. De plus, il précise son formatage, les subtilités associées à son système de comparaison et finit par la présentation du type des chaînes de caractères **mutables** à savoir **StringBuilder**.

1.1 Généralités

Le type **string** permet de stocker et de manipuler du texte, chaque caractère étant encodé en UTF-16. Une instance peut stocker une grande chaîne de caractères (jusqu'à 2 GB de données). Le type **string** est un type par référence, **immuable**. Cela signifie qu'une instance ne peut plus être modifiée une fois qu'elle a été initialisée. Chaque changement produit une nouvelle instance. Une déclaration d'une instance de chaîne de caractères est encadrée par des guillemets anglais `""`.

Exemple : Déclaration simple

```
1 string a = "Le corps humain ";
```

1.1.1 Concaténation

Une des particularités du type **string** est la possibilité de générer des chaînes de caractères en emboîtant des chaînes les unes dans les autres. L'opérateur + permet de l'exécuter simplement. Chacun des opérandes doit être convertible en une chaîne de caractères.

Exemple : Concaténation

```
1 string debut = "Le mot " + "début ";
2 string fin = "contient " + 5 + " caractères alphanumériques.";
3 string phrase = debut + fin;
4 // phrase : "Le mot début contient 5 caractères alphanumériques."
```

Remarque
Chaque concaténation induit la création de chaînes de caractères intermédiaires. En cas de créations répétées, il est intéressant d'utiliser une instance d'une chaîne mutable (voir section 1.5).

1.1.2 Énumération

Les chaînes de caractères peuvent être vues comme un tableau de caractères. Chaque caractère est atteignable à l'aide de son indice. On peut utiliser une instance de **string** dans des boucles foreach, for.

Exemple : Énumération du **string**

```
1 string str = "Hello World!";
2 Console.WriteLine (str[3]); // l
3
4 foreach (char ch in str)
```

```
5  {
6    Console.WriteLine (ch); // Affichage d'un caractère par ligne.
7  }
```

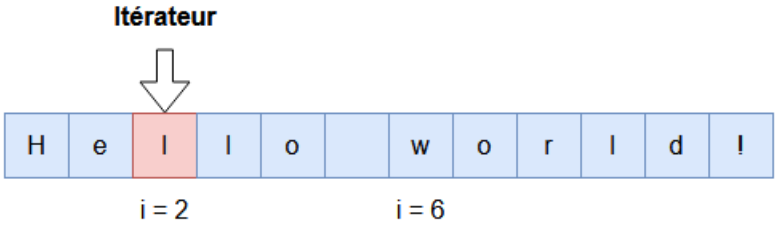

FIGURE 1 – Structure interne d'une chaîne de caractères

1.1.3 Interpolation (C# 6)

Une chaîne de caractères précédée par le caractère **$** est une chaîne de caractères interpolée pouvant inclure des expressions sous accolades dont le contenu doit être convertible est une instance de **string**.

Exemple : Interpolation simple

```
1  int x = 4; int y = 3
2  Console.WriteLine ($"La pièce a une superficie de {x*y} m carrés.");
3  // phrase : La pièce a une superficie de 12 m carrés.
```

Une expression d'interpolation peut être associée à un format de type **String.Format** afin d'affiner son contenu (présentation dans la section 1.3).

Exemple : Interpolation avec format spécifié

```
1  string s = $"255 en format hexadécimal vaut {byte.MaxValue:X2}.";
2  // X2 : format hexadécimal sur deux chiffres.
3  // s : 255 en format hexadécimal vaut FF.
```

1.2 Liste de fonctions utiles

Cette section a pour objectif de présenter les principales fonctions utiles à la manipulation de chaînes de caractères. Elle n'est pas exhaustive et ne montre pas l'ensemble des cas d'usage possible de chacune des méthodes énoncées. Dans un but didactique, les fonctions ont été divisées en trois groupes distincts. Premièrement, celles renseignant le contenu de la chaîne sur laquelle elles sont appliquées, puis celles produisant une chaîne de sortie à partir de la chaîne de caractères de départ et pour finir les fonctions produisant une chaîne de sortie à partir de l'interaction entre plusieurs chaînes de départ.

1.2.1 Recherche d'éléments

Contains :
La méthode **Contains** permet de déterminer la présence ou non d'une sous-chaîne spécifiée dans la chaîne testée. Elle prend en paramètre un **char** ou un **string** et renvoie un **bool**.

Exemple :

```
string s = "La recherche est fructueuse."
if(s.Contains("est"))
{
    Console.WriteLine("est se trouve dans l'entrée.");
}
```

StartsWith et EndsWith :

Ces deux méthodes fonctionnent sur le même principe que **Contains** mais s'astreignent à tester le début et la fin de la chaîne testée. En cas de présence d'espaces ou caractères spéciaux en début ou fin de chaîne, le résultat ne correspondra pas exactement à ce que l'on souhaite obtenir.

Exemple :

```
string s = "La recherche est fructueuse."
if(s.StartsWith("La") && s.EndsWith('.'))
{
    Console.WriteLine("L'entrée commence par La et finit par un point.");
}
```

IndexOf et LastIndexOf :

Ces deux méthodes effectuent une recherche de la sous-chaîne spécifiée en entrée et retournent respectivement la première et la dernière position où elle se trouve. Lorsque celle-ci n'est pas trouvée, la valeur retournée est -1. Deux surcharges de ces deux méthodes consistent à ajouter en paramètre l'indice de départ de la recherche et (en option) le nombre de caractères sur laquelle l'effectuer.

Exemple :

```
string s = "La recherche est fructueuse."
int premierE = s.IndexOf('e');              //  4
int secondE = s.IndexOf('e', premierE + 1); //  7
int dernierE = s.LastIndexOf('e');          // 26
Console.WriteLine($
"Le premier, second et dernier e sont à {premierE}, {secondE}, {dernierE}");
```

1.2.2 Modifications

SubString :

La méthode **SubString** retourne une sous-chaîne de la chaîne principale à partir de l'indice donné en paramètre et (en option) sur le nombre de caractères spécifiés.

Exemple :

```
string s = "La recherche est fructueuse."
string debut = s.SubString(0, 12)
Console.WriteLine(debut); // La recherche
```

TrimStart, TrimEnd, Trim :

Les trois méthodes **Trim** suppriment l'ensemble des caractères passés en paramètres (si aucun paramètre n'est spécifié, les espaces blancs sont traités) au début, à la fin ou les deux à la fois dans la chaîne retournée. Les opérations de suppression s'arrêtent lorsqu'elles rencontrent des caractères non présents dans la liste fournie. Ces méthodes sont particulièrement utiles en cas de réception de données dont on veut nettoyer le contenu indésirable.

Exemple :

```
1 string s = "     Aymeric Bonnaz!!!!";
2 string debut = s.TrimStart();
3 string fin = s.TrimEnd('!');
4 string testSup = s.Trim(' ', '!');
5 Console.WriteLine(debut);   // Aymeric Bonnaz!!!!
6 Console.WriteLine(fin);     //      Aymeric Bonnaz
7 Console.WriteLine(testSup); // Aymeric Bonnaz
```

Insert et Remove :

La méthode **Insert** permet l'insertion d'une sous-chaîne dans la chaîne d'entrée à l'indice donné en paramètre et de retourner le résultat. La méthode **Remove** supprime le contenu à partir de l'indice fourni et sur la longueur spécifiée (en option).

Exemple :

```
1 string s = "L'informatique est une matière fort compliquée.";
2 int indice = s.IndexOf("est");
3 string inter = s.Remove(indice, 4); // Suppression de est et d'un espace.
4 string final = inter.Insert(indice, "était ");
5 Console.WriteLine(final);
6 // L'informatique était une matière fort compliquée.
```

Replace :

La méthode **Replace** permet la substitution d'une série de caractères par une autre et retourne le résultat obtenu. Si la sous-chaîne spécifiée n'est pas rencontrée, aucune modification n'est opérée.

Exemple :

```
1 string s = "L'informatique est une matière fort compliquée.";
2 string final = s.Replace("est", "était");
3 Console.WriteLine(final);
4 // L'informatique était une matière fort compliquée.
```

ToUpper et ToLower :

Les méthodes **ToUpper** et **ToLower** retournent la chaîne de caractères en convertissant respectivement les lettres en majuscules et en minuscules. Les autres caractères ne sont pas impactés par ces méthodes.

Exemple :

```
string s = "inFormAtique";
string smin = s.ToLower();
Console.WriteLine(smin); // informatique
string smaj = s.ToUpper();
Console.WriteLine(smaj); // INFORMATIQUE
```

1.2.3 Interactions entre chaînes

Split et Join :
La méthode **Split** part d'une liste de caractères séparateurs et retourne un tableau contenant l'ensemble des sous-chaînes produites (sans les séparateurs). La méthode **Join** réalise l'opération inverse, elle part d'un tableau de chaînes de caractères, un séparateur et retourne la chaîne de caractères formée.

Exemple :

```
string phrase = "Le chat est sur le canapé";
string[] mots = phrase.Split (' ');
foreach (string mot in mots)
{
  Console.WriteLine (mot);
}
```

Concat :
La méthode **Concat** a pour objectif de concaténer en une seule fois toutes les instances **string**. On évite ainsi les copies intermédiaires. Préférez cette méthode en cas de nombreuses concaténations à la suite.

Exemple :

```
string s1 = "Le chat ";
string s2 = "est sur ";
string s3 = "le canapé.";
string resultat = String.Concat(s1, s2, s3);
Console.WriteLine(resultat);
// Le chat est sur le canapé.
```

1.3 Formatage en chaînes de caractères

Après avoir présenté le type **string** et les principales méthodes permettant d'interagir avec celui-ci, cette section illustre la conversion en chaîne de caractères à partir d'un autre type. Elle se concentre sur la conversion des types numériques (pour la conversion des dates, voir le module approprié).

Il existe plusieurs moyens de réaliser cette étape :

— La méthode **ToString**

— Une expression d'interpolation : {instance :**Format**}

1.3.1 Format

Afin de définir le résultat de la conversion d'un type numérique vers une chaîne de caractères, il existe deux types de formats :

— Les chaînes de format numérique standard.

— Les chaînes de format numérique personnalisées.

Le détail de ces deux types de formats est présent dans l'annexe 2.2. Par défaut, si aucun format n'est fourni, le format utilisé est le format standard général **G**.

Exemple : Format standard

```
1  double montant = 1203.359;
2  double tva = 0.20;
3  Console.WriteLine
4  ($"Le montant de la facture est de {montant:C} avec {tva:P0} de TVA.");
5  // Le montant de la facture est de $1200.36 avec 20 % de TVA.
```

Exemple : Format personnalisé

```
1  double montant = 1203.359;
2  double tva = 0.20;
3  Console.WriteLine
4  ($"Le montant de la facture est de {montant:#,###.0} avec {tva:00%} de TVA.");
5  // Le montant de la facture est de $1,200.4 avec 20 % de TVA.
```

1.3.2 Culture

Un autre paramètre pouvant influencer la conversion d'un type numérique est la culture associée à cette opération. Le type représentant la culture est **CultureInfo**. Si ce paramètre n'est pas utilisé, la culture appliquée est la culture du système utilisant le programme, **CurrentCulture**.
La création d'une instance de **CultureInfo** se réalise à l'aide d'une chaîne de caractères indiquant la culture souhaitée auquelle on ajoute éventuellement une région spécifique.
Par exemple, **en-GB** est associée à une culture anglo-saxonne (**en**) et à la région britannique (**GB**).

Exemple : Influence de la culture

```
1  double montant = 1203.353;
2  Console.WriteLine(montant.ToString("C", new CultureInfo("fr-FR")));
3  // 1203.35 €
4  Console.WriteLine(montant.ToString("C", new CultureInfo("en-GB")));
5  // £1203.35
```

1.4 Comparaisons des chaînes de caractères

Contrairement aux types numériques, le type **string** ne permet pas l'utilisation des opérateurs $<$ et $>$. Pour comparer des séquences de caractères et savoir laquelle vient avant ou après, il est nécessaire d'utiliser la méthode **CompareTo** ou la fonction **Compare** qui renvoie un résultat entier.

Cas possibles pour a.**CompareTo**(b) ou string.**Compare**(a, b) :

- résultat > 0, b avant a.
- résultat $= 0$, b au même niveau que a.
- résultat < 0, b après a.

Exemple : Comparaison simple

```
1  string a = "Chat";
2  string b = "Chien";
3  string c = "Chien";
4  Console.WriteLine(a.CompareTo(b));        // < 0, généralement -1.
5  Console.WriteLine(string.Compare(b, c)); // 0
```

La comparaison entre deux chaînes de caractères est dépendante de la culture dans laquelle elle est réalisée. **CompareTo** ne permet pas de contrôler cette influence alors que **Compare** dispose de plusieurs versions permettant de sécuriser cette opération.

1.4.1 StringComparison

Plusieurs versions de **Compare** acceptent un paramètre supplémentaire de type **StringComparison**. **StringComparison** est une énumération dont les valeurs possibles sont présentées dans le tableau ci-dessous.

Valeur	Description
CurrentCulture	Comparaison associée à la culture **actuelle** utilisée et **sensible** à la casse.
CurrentCultureIgnoreCase	Comparaison associée à la culture **actuelle** utilisée et **insensible** à la casse.
InvariantCulture	Comparaison associée à la culture **indifférente** et **sensible** à la casse.
InvariantCultureIgnoreCase	Comparaison associée à la culture **indifférente** et **insensible** à la casse.
Ordinal	Comparaison associée aux règles de tri **ordinal** et **sensible** à la casse.
OrdinalIgnoreCase	Comparaison associée aux règles de tri **ordinal** et **insensible** à la casse.

La culture courante est la culture du système exécutant l'instruction de comparaison. La culture indifférente est proche des règles de la culture nord-américaine. Une comparaison ordinale va se baser sur la valeur numérique des différents caractères composant les deux séquences de caractères. Dans le doute, il est préférable d'utiliser la culture courante.

Exemple : Comparaison avec **StringComparison**

```
1 string a = "Ligne123_1";
2 string b = "Ligne123-1";
3 Console.WriteLine(string.Compare(a, b, StringComparison.CurrentCulture));
4 Console.WriteLine(string.Compare(a, b, StringComparison.Ordinal));
5 // cas 1 : < 0, cas 2 : > 0
```

Les deux comparaisons donnent des résultats radicalement différentes car **-** (45) a une valeur numérique inférieure à **_** (95), mais dans la culture française utilisée, **_** arrive avant **-**.

1.4.2 CultureInfo

D'autres versions de **Compare** accepte une instance de type **CultureInfo**.
L'exemple suivant met en lumière l'influence d'une culture spécifique. Dans la culture tchèque-tchèque, **ch** est un caractère unique qui est supérieur à **d**. Toutefois, dans la culture en anglais États-Unis, **ch** se compose de deux caractères et **c** est inférieur à **d**.

Exemple : Comparaison avec **CultureInfo**

```
1 string a = "changement";
2 string b = "dollar";
3 Console.WriteLine(string.Compare(a, b, false, new CultureInfo("en-US")));
4 Console.WriteLine(string.Compare(a, b, false, new CultureInfo("cs-CZ")));
5 // cas 1 : < 0, cas 2 : > 0
```

1.5 Chaînes mutables

Cette section présente succinctement le type des séquences de caractères mutables **StringBuilder**et liste les principales méthodes permettant de les manipuler.
Jusqu'à présent, des instances du type **string** représentaient les chaînes de caractères. À chaque concaténation ou modification d'une d'entre elles, une nouvelle instance était créée. Cela risque de provoquer des soucis de mémoire en cas de manipulations répétées.
Le type **StringBuilder** remédie à ce problème, car son contenu évolue sans recréation d'une instance à chaque modification. La méthode **ToString** permet d'obtenir une séquence immuable à partir d'une chaîne mutable.

Append et AppendLine :
La méthode **Append** permet d'ajouter le paramètre à la suite de l'instance courante. Ce paramètre peut être une séquence de caractères, une autre chaîne mutable ou autre type convertible en **string**.
La méthode **AppendLine** effectue la même opération et ajoute un retour à la ligne.

Exemple :

```
1  StringBuilder s = new StringBuilder();
2  s.Append("L'informatique était fort compliquée en ").Append(1994).Append('.');
3  Console.WriteLine(s);
4  // L'informatique est fort compliquée en 1994.
```

Insert et Remove :
La méthode **Insert** permet l'insertion du paramètre dans l'instance courante à l'indice donné en paramètre. La méthode **Remove** supprime le contenu à partir de l'indice fourni et sur la longueur spécifiée dans l'instance utilisée.

Exemple :

```
1  StringBuilder s = new StringBuilder();
2  s.Append("L'informatique est une matière fort compliquée.");
3  s.Remove(15, 4); // Suppression de est et d'un espace.
4  s.Insert(15, "était ");
5  Console.WriteLine(s);
6  // L'informatique était une matière fort compliquée.
```

Replace :
La méthode **Replace** permet la substitution d'une série de caractères par une autre dans l'instance courante. Si la sous-chaîne spécifiée n'est pas rencontrée, aucune modification n'est opérée.

Exemple :

```
1  StringBuilder s = new StringBuilder();
2  s.Append("L'informatique est une matière fort compliquée.");
3  s.Replace("est", "était");
4  Console.WriteLine(s);
5  // L'informatique était une matière fort compliquée.
```

2 Annexes

2.1 Liste des méthodes

2.1.1 Chaînes de caractères

Le tableau suivant présente une série de méthodes fortes utiles pour manipuler des chaînes de caractères.

Liste non exhaustive des méthodes associées à **String** :

Méthodes	**Explications**
bool Contains (**string** value)	Est ce que la chaîne passée en paramètre est présente dans la chaîne principale ?
bool StartsWith (**string** value) bool EndsWith (**string** value)	Est-ce-que la chaîne passée en paramètre correspond **au début/à la fin** de la chaîne principale ?
int IndexOf (**string** value, int startIndex) int LastIndexOf (**string** value, int startIndex)	Signalement de l'indice de la **première/dernière** occurrence de la chaîne spécifiée dans la chaîne principale. Si non trouvée, renvoie -1. Si value est null, renvoie startIndex.
string Substring (int startIndex)	Récupération d'une sous-chaîne de la chaîne principale à partir du caractère passé.
string TrimStart (params char[] trimChars) **string** TrimEnd (params char[] trimChars) **string** Trim (params char[] trimChars)	Suppression de toutes les occurrences du jeu de caractères spécifiés en paramètres au début **ou/et** fin de la chaîne de caractères. Si pas de paramètres, suppression des espaces blancs.
string Insert (int startIndex, **string** value)	Insertion de **value** à l'indice startIndex.
string Remove (int startIndex, int count)	Suppression de la sous-chaîne commençant à l'indice **startIndex** de longueur **count**.
string Replace (char oldChar, char newChar) **string** Replace (**string** oldValue, **string** newValue)	Remplacement de l'ancienne **sous-chaîne/caractère** par **le nouveau/la nouvelle**.
string ToUpper () **string** ToLower ()	Transformation des lettres **minuscules/majuscules** en **majuscules/minuscules**.
string[] Split (params char[] separator)	Divise la chaîne en sous-chaînes d'après les caractères du tableau des séparateurs.
string Join (char separator, **string**[] values)	Concaténation des sous-chaînes avec le séparateur comme liant pour obtenir une chaîne.
string Concat (**string** str0, ... , **string** str3)	Concaténation des chaînes de caractères passés en paramètres.

2.1.2 Chaînes de caractères mutables

Le tableau suivant présente une série de méthodes fortes utiles pour manipuler des chaînes de caractères **mutables**.

Liste non exhaustive des méthodes associées à **StringBuilder** :

Méthodes	Explications
StringBuilder Append (**string** value) **StringBuilder** AppendLine (**string** value)	Ajout de **value** à la fin de l'instance et retour d'une référence sur l'instance.
StringBuilder Insert (int startIndex, **string** value)	Insertion de **value** à l'indice startIndex et retour d'une référence sur l'instance.
StringBuilder Remove (int startIndex, int length)	Suppression de la sous-chaîne commençant à l'indice **startIndex** de longueur **length** et retour d'une référence sur l'instance.
StringBuilder Replace (**string** oldValue, **string** newValue)	Remplacement de l'ancienne **sous-chaîne/caractère** par **le nouveau/la nouvelle** et retour d'une référence sur l'instance.
StringBuilder Clear()	Supprime l'ensemble des caractères présents dans l'instance et retour d'une référence sur celle-ci.

2.2 Chaînes de format numérique

Chaînes de format numérique standard :

Lettre	Nom	Entrées	Résultats	Commentaires
G ou g	Général	1.2345, "G" 0.00001, "G" 0.00001, "g" 1.2345, "G3" 12345, "G3"	1.2345 1E-05 1e-05 1.23 1.23E04	Format le plus compact (notation à virgule fixe ou scientifique).
F ou f	Virgule fixe	2345.678, "F2" 2345.6, "F2"	2345.68 2345.60	F**2** arrondi à **deux** décimales après la virgule.
N ou n	Nombre	2345.678, "N2" 2345.6, "N2"	2,345.68 2,345.60	N**2** arrondi à **deux** décimales après la virgule avec un **séparateur**.
D ou d	Décimal	123, "D5" 123, "D1"	00123 123	Nombre minimal de chiffres, pas de troncature du nombre.
E ou e	Notation exponentielle	56789, "E" 56789, "e" 56789, "E2"	5.678900E+004 5.678900e+004 5.68e+004	Six chiffres par défaut.
C ou c	Devise	1.2, "C" 1.2, "C4"	$1.20 $1.2000	Montant avec spécification du nombre de décimales.
P ou p	Pourcentage	.503, "P" .503, "P0"	50.30 % 50 %	Pourcentage avec spécification du nombre de décimales.
X ou x	Hexadécimal	47, "X" 47, "x" 47, "X4"	2F 2f 002F	Chaîne hexadécimale.

Chaînes de format numérique personnalisées :

Indice	Nom	Entrées	Résultats	Commentaires
#	Espace réservé à un chiffre	12.345, ".##" 12.345, ".####"	12.35 12.345	Si chiffre existe, il remplace le #, sinon **rien**.
0	Espace réservé du zéro	12.345, "**.00**" 12.345, "**.0000**" 99, "**.000.00**"	12.35 12.3450 099.00	Si chiffre existe, il remplace le #, sinon **0**.
.	Virgule décimale			Détermine l'emplacement du séparateur décimal.
,	Séparateur de groupes	1234, "#,###,###" 12.345, "**0,000,000**"	1,234 0,001,234	Sert à la fois de séparateur de groupes et de spécificateur de mise à l'échelle des nombres.
%	Espace réservé de pourcentage	0.6, "**00%**"	60%	Nombre multiplié par 100 et pourcentage.
E ou e	Notation exponentielle	1234, "**.0E0**" 1234, "0**E**+0" 1234, "0.00**E**00" 1234, "0.00**e**00"	1E3 1E+3 1.23E03 1.23e03	
\	Caractère d'échappement	50, @"\#0"	#50	Entraîne l'interprétation du caractère suivant comme un littéral.