1 Bronze 4 : Chaînes de caractères

La manipulation de textes est une tâche fréquemment accomplie par les programmes informatiques. Le type fondamental dédié à la gestion du texte est la chaîne de caractères, souvent représentée par le terme **string**. Ce module a pour but de mettre en avant les principales caractéristiques de ce type, en plus de fournir une liste exhaustive des fonctions qui lui sont associées. De plus, il aborde les considérations liées à son formatage, les subtilités inhérentes à son système de comparaison, et conclut en présentant un autre type, les chaînes de caractères **mutables**, représentées par le concept de **StringBuilder**.

1.1 Généralités

Le type **string** permet le stockage et la manipulation de texte, chaque caractère étant encodé en UTF-16. Une instance de ce type peut contenir une vaste chaîne de caractères (pouvant aller jusqu'à 2 Go de données). Il s'agit d'un type par référence, **immuable**, ce qui signifie qu'une fois initialisée, une instance ne peut plus être modifiée. Tout changement entraîne la création d'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 particularité notable du type **string** réside dans sa capacité à créer des chaînes de caractères en imbriquant plusieurs chaînes les unes dans les autres. Cette fonctionnalité est aisément exécutable à l'aide de l'opérateur +. Il convient cependant de noter que chaque opérande 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 opération de concaténation entraîne la création de chaînes de caractères intermédiaires. En cas de concaténations répétées, il devient avantageux d'utiliser une instance de chaîne mutable (voir section 1.5).

1.1.2 Énumération

Les chaînes de caractères peuvent être vues comme un tableau de caractères, où chaque caractère est accessible via son indice. Cela ouvre, par exemple, la possibilité d'utiliser une instance de type **string** au sein de boucles *foreach* et *for*.

Exemple : Énumération sur une instance de **string**

```
1  string str = "Hello World!";
2  Console.WriteLine (str[3]); // Affiche 'l'
3
4  foreach (char ch in str)
5  {
6    Console.WriteLine (ch); // Affichage d'un caractère par ligne.
7  }
```

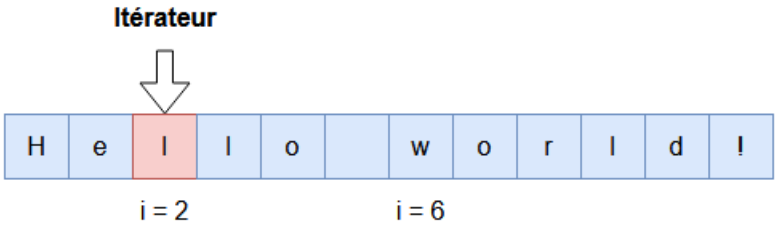

FIGURE 1 – Structure interne d'une chaîne de caractères

1.1.3 Interpolation (C# 6)

Une chaîne de caractères précédée par le symbole **$** est appelée chaîne de caractères *interpolée*. Elle est capable d'incorporer des expressions contenues entre accolades. Il est important de noter que ces expressions doivent être convertibles en instances du type **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 vise à présenter les fonctions principales, essentielles à la manipulation des chaînes de caractères. Bien qu'elle ne prétende pas à l'exhaustivité, elle offre un aperçu des méthodes clés pour la manipulation textuelle. À des fins didactiques, les fonctions sont regroupées en trois catégories distinctes. Tout d'abord, celles qui permettent d'extraire des informations de la chaîne cible, suivies par celles générant une nouvelle chaîne en fonction de la chaîne d'origine, et enfin celles produisant une sortie basée sur l'interaction entre plusieurs chaînes sources.

1.2.1 Recherche d'éléments

Contains :

La méthode **Contains** permet d'établir si une sous-chaîne spécifiée est présente dans la chaîne en question. Elle reçoit en entrée un argument de type **char** ou **string** et renvoie un résultat de type **bool**.

Exemple :

```
1  string s = "La recherche est fructueuse."
2  if(s.Contains("est"))
3  {
4      Console.WriteLine("est se trouve dans l'entrée.");
5  }
```

StartsWith et EndsWith :

Les deux méthodes suivantes opèrent selon le même principe que **Contains**, mais elles sont contraintes à vérifier le début et la fin de la chaîne considérée. Il est important de noter que si la chaîne commence ou finit par des espaces ou des caractères spéciaux, le résultat peut différer des attentes initiales.

Exemple :

```
1  string s = "La recherche est fructueuse."
2  if(s.StartsWith("La") && s.EndsWith('.'))
3  {
4      Console.WriteLine("L'entrée commence par La et finit par un point.");
5  }
```

IndexOf et LastIndexOf :

En recherchant la sous-chaîne spécifiée en entrée, ces deux méthodes fournissent respectivement la première et la dernière position où cette sous-chaîne est localisée. Lorsque la sous-chaîne n'est pas trouvée, la valeur de retour est -1. De plus, deux surcharges de ces méthodes permettent de spécifier un point de départ pour la recherche ainsi qu'une plage optionnelle de caractères sur laquelle effectuer cette recherche.

Exemple :

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

1.2.2 Modifications

SubString :
La méthode **SubString** renvoie une sous-chaîne de la chaîne principale à partir de l'indice spécifié en paramètre, avec la possibilité de limiter la longueur en caractères.

Exemple :

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

TrimStart, TrimEnd, Trim :
Les trois méthodes **Trim** permettent d'éliminer des caractères spécifiques (ou les espaces blancs par défaut) soit au début, à la fin ou des deux extrémités de la chaîne résultante. Le processus de suppression cesse dès qu'un caractère hors de la liste de suppression est rencontré. Ces méthodes sont particulièrement utiles pour nettoyer des données indésirables lors de leur réception.

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** facilite l'ajout d'une sous-chaîne à une position donnée de la chaîne d'origine, et renvoie le résultat obtenu. En parallèle, la méthode **Remove** efface un segment à partir de l'indice spécifié et sur une longueur déterminée (si précisée).

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 le remplacement d'une séquence de caractères par une autre et fournit la nouvelle chaîne résultante. Si la sous-chaîne à remplacer n'est pas trouvée, aucun changement n'est effectué.

Exemple :

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

ToUpper et ToLower :
Les méthodes **ToUpper** et **ToLower** transforment respectivement la chaîne en majuscules ou en minuscules, en laissant les autres caractères inchangés.

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** divise une chaîne en plusieurs sous-chaînes en utilisant une liste de caractères séparateurs. Elle retourne un tableau contenant les sous-chaînes, sans les séparateurs. En revanche, la méthode **Join** réalise l'inverse : elle fusionne un tableau de chaînes en utilisant un séparateur spécifié.

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** est conçue pour concaténer efficacement plusieurs instances de **string** en une seule opération, minimisant ainsi les copies intermédiaires. Elle est recommandée lorsque vous devez effectuer de nombreuses concaténations successives.

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 exposé le type **string** ainsi que les méthodes fondamentales pour interagir avec celle-ci, cette section illustre la procédure de conversion vers une chaîne de caractères à partir d'autres types de données. Une attention particulière est portée à la conversion des types numériques (pour la conversion des dates, voir le module dédié).

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.

Les spécificités détaillées de ces deux formats de conversion sont exposées dans l'annexe 2.2. Par défaut, en l'absence de spécification de format, le format utilisé est le 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 facteur pouvant exercer une influence sur la conversion d'un type numérique est la culture associée à cette opération. La représentation de la culture est définie par le type **CultureInfo**. En cas de non-utilisation de ce paramètre, la culture appliquée correspond à celle du système en cours d'exécution du programme, soit **CurrentCulture**. La création d'une instance de **CultureInfo** s'effectue en fournissant une chaîne de caractères indiquant la culture souhaitée, à laquelle il est éventuellement possible d'ajouter une région spécifique. À titre d'exemple, la culture **en-GB** est associée à la culture anglo-saxonne (**en**) ainsi qu'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, l'utilisation des opérateurs $<$ et $>$ n'est pas possible avec le type **string**. Afin de comparer des séquences de caractères et de déterminer leur ordre relatif, il est impératif de recourir à la méthode **CompareTo** ou à la fonction **Compare**, qui fournissent un résultat sous forme d'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 de deux chaînes de caractères dépend de la culture dans laquelle elle est effectuée. Contrairement à **CompareTo**, **Compare** offre plusieurs variantes permettant de maîtriser cette influence et d'assurer une exécution sécurisée de cette opération.

1.4.1 StringComparison

Diverses variantes de la fonction **Compare** intègrent un paramètre additionnel de type **StringComparison**. Cette dernière est une énumération comprenant différentes valeurs, comme exposé 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.

Le contexte de comparaison est influencé par la culture en vigueur sur le système exécutant l'instruction de comparaison. La culture *indifférente*, similaire aux conventions nord-américaines, est également une option. Une comparaison *ordonnée* repose sur la valeur numérique attribuée aux caractères individuels constituant les deux séquences de caractères. En cas d'incertitude, opter pour la culture *courante* est recommandé.

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
```

Dans l'exemple ci-dessus, les deux méthodes de comparaison produisent des résultats radicalement différents. Le caractère - (45) possède une valeur numérique inférieure à celle de _ (95). Toutefois, dans le contexte de la culture française, _ est situé avant -, ce qui donne lieu à une différence.

1.4.2 CultureInfo

D'autres variantes de **Compare** permettent de fournir une instance de type **CultureInfo**. L'exemple suivant met en évidence l'impact d'une culture spécifique : en tchèque, **ch** est un caractère distinct et supérieur à **d**, tandis qu'en anglais américain, **ch** est constitué de deux caractères et **c** précède **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 propose une brève présentation du type des séquences de caractères *mutables*, **StringBuilder**, tout en énumérant les principales méthodes qui facilitent leur manipulation. Jusqu'à présent, les chaînes de caractères étaient représentées par des instances du type **string**. Cependant, chaque fois qu'une concaténation ou une modification était effectuée, une nouvelle instance était créée. Ce processus pouvait engendrer une sur-utilisation de la mémoire lors de manipulations répétées.
Le type **StringBuilder** remédie à ce problème en permettant l'évolution du contenu sans nécessiter la création d'une nouvelle instance à chaque modification. En outre, la méthode **ToString** offre la possibilité 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 prendre la forme d'une séquence de caractères, d'une autre chaîne *mutable*, ou d'un autre type convertible en **string**. La méthode **AppendLine** réalise une tâche similaire avec un saut de 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 :
Pour insérer du contenu au sein de l'instance courante à une position donnée, la méthode **Insert** est employée. D'autre part, la méthode **Remove** est utilisée pour effacer du contenu à partir de l'indice spécifié, avec une longueur déterminée à l'intérieur de l'instance en question.

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 séquence de caractères par une autre au sein de l'instance courante. Si la sous-chaîne spécifiée n'est pas présente, 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.