A. Les variables, constantes et énumérations

1. Les variables

Les variables vont vous permettre de mémoriser, pendant l'exécution de votre application, différentes valeurs utiles pour le fonctionnement de votre application. Une variable doit obligatoirement être déclarée avant son utilisation dans le code. Lors de la déclaration d'une variable, vous fixez ses caractéristiques.

a. Nom des variables

Voyons les règles à respecter pour nommer les variables :

- le nom d'une variable commence obligatoirement par une lettre,
- il peut être constitué de lettres, de chiffres ou du caractère souligné (_),
- il peut contenir un maximum de 1023 caractères (pratiquement, il est préférable de se limiter à une taille plus raisonnable),
- il y a distinction entre minuscules et majuscules (la variable AgeDuCapitaine est différente de la variable ageducapitaine).
- les mots-clés du langage ne doivent pas être utilisés (c'est malgré tout possible mais dans ce cas, le nom de la variable doit être précédé du caractère @. Par exemple, une variable nommée `if` sera utilisée dans le code sous cette forme @if=56;).

b. Type des variables

En spécifiant un type pour une variable, nous indiquons quelles informations nous allons pouvoir stocker dans cette variable.

Deux catégories de types de variables sont disponibles :

- Les types valeur : la variable contient réellement les informations.
- Les types référence : la variable contient l'adresse mémoire où se trouvent les informations.

Les différents types de variables disponibles sont définis au niveau du Framework lui-même. Vous pouvez également utiliser les alias définis au niveau de Visual C#, peut-être plus explicites. Ainsi, le type **System.Int32** défini au niveau du framework peut être remplacé par le type **int** dans Visual C#.

Les différents types peuvent être classés en six catégories.

Les types numériques entiers

Types entiers signés			
sbyte	- 128	127	8 bits
short	-32768	32767	16 bits
int	-2 147 483 648	2 147 483 647	32 bits
long	-9223372036854775808	9223372036854775807	64 bits

Types entiers non signés			
byte	0	255	8 bits
ushort	0	65535	16 bits
uint	0	4294967295	32 bits
ulong	0	18446744073709551615	64 bits

Lorsque vous choisissez un type pour vos variables entières, vous devez prendre en compte les valeurs minimale et maximale que vous envisagez de stocker dans la variable afin d'optimiser la mémoire utilisée par la variable. Il est, en effet, inutile d'utiliser un type Long pour une variable dont la valeur n'excédera pas 50, un type byte est dans ce cas suffisant.

L'économie de mémoire semble dérisoire pour une variable unique mais devient appréciable lors de l'utilisation de tableaux de grande dimension.

Si par contre vous souhaitez optimiser la vitesse d'exécution de votre code, il est préférable d'utiliser le type int.

Les types décimaux

float	-3.40282347E+38	3.40282347E+38	4 octets
double	-1.7976931348623157E+308	1.7976931348623157E+308	8 octets
decimal	-79228162514264337593543950335	79228162514264337593543950335	16 octets

Les mêmes considérations d'optimisation que pour les variables entières doivent être prises en compte. Dans ce cas, une rapidité d'exécution maximale est obtenue avec le type double. Le type decimal est plus spécialement recommandé pour les calculs financiers pour lesquels les erreurs d'arrondis sont prohibées, mais au détriment de la rapidité d'exécution du code.

Les types caractères

Le type char (caractères) est utilisé pour stocker un caractère unique. Une variable de type char utilise deux octets pour stocker le code Unicode du caractère. Dans jeu de caractère Unicode, les 128 premiers caractères sont identiques au jeu de caractère ASCII, les caractères suivants jusqu'à 255 correspondent aux caractères spéciaux de l'alphabet latin (par exemple, les caractères accentués), le reste est utilisé pour des symboles ou pour les caractères d'autres alphabets.

L'affectation d'une valeur à une variable de type char doit être effectuée en encadrant la valeur par des caractères ''. Certains caractères ayant une signification spéciale pour le langage doivent être utilisés avec une séquence d'échappement. Cette séquence d'échappement commence toujours par le caractère \. Le tableau suivant résume les différentes séquences disponibles.

Séquence d'échappement	caractère
\'	Simple quote
\"	Double quote
\\	backSlash
\0	Caractère nul
\a	Alerte
\b	Baskspace
\f	Saut de page
\n	Saut de ligne
\r	Retour chariot
\t	Tabulation horizontale
\v	Tabulation verticale

Ces séquences d'échappement peuvent également être utilisées dans une chaîne de caractères. Chacune d'elles représente un caractère unique.

Pour pouvoir stocker des chaînes de caractères, il convient d'utiliser le type string, qui représente une suite de zéro à 2147483648 caractères. Les chaînes de caractères sont invariables car, lors de l'affectation d'une valeur à une chaîne de caractères, de l'espace est réservé en mémoire pour le stockage. Si, par la suite, cette variable reçoit une nouvelle valeur, le système lui assigne un nouvel emplacement en mémoire. Heureusement, ce mécanisme est transparent pour nous et la variable fera toujours automatiquement référence à la valeur qui lui a été assignée. Avec ce mécanisme, les chaînes de caractères peuvent avoir une taille variable. L'espace occupé en mémoire est automatiquement ajusté à la longueur de la chaîne de caractères.

Pour affecter une chaîne de caractères à une variable, le contenu de la chaîne doit être saisi entre “ ”, comme dans l'exemple ci-dessous :

Exemple

```
NomDuCapitaine = "Crochet";
```

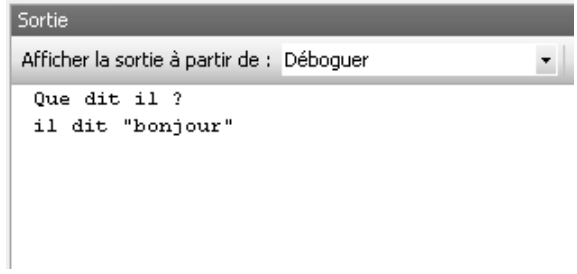
Si des caractères spéciaux doivent apparaître dans une chaîne, ils doivent être spécifiés par une séquence d'échappement. Il existe cependant une autre possibilité qui permet parfois de rendre le code plus lisible. Cette solution consiste à faire précéder la chaîne de caractères du symbole @. Le compilateur considère alors que tous les caractères contenus entre les doubles quotes doivent être utilisés tels quel, y compris les éventuels retours chariot. La seule limitation concerne le caractère " qui, s'il doit faire partie de la chaîne, doit être doublé.

Les deux déclarations de chaînes suivantes sont identiques :

```
chaine = "Que dit il ?\ril dit \"bonjour\"";

chaine = @"Que dit il ?
il dit ""bonjour""";
```

Lorsqu'elles sont affichées sur la console, elles donnent le résultat suivant :



De nombreuses fonctions de la classe string permettent la manipulation des chaînes de caractères et seront détaillées plus loin dans ce chapitre.

Le type bool

Le type bool permet d'utiliser une variable qui peut prendre deux états vrai/faux, oui/non, on/off.

L'affectation se fait directement avec les valeurs true ou false, comme dans l'exemple suivant :

```
Disponible=true;
Modifiable=false;
```

Le type Object

C'est peut-être le type le plus universel de Visual C#. Dans une variable de type Object, vous pouvez stocker n'importe quoi. En fait, ce type de variable ne stocke rien. La variable va contenir non pas la valeur elle-même, mais l'adresse, dans la mémoire de la machine, où l'on pourra trouver la valeur de la variable. Rassurez-vous, tout ce mécanisme est transparent et vous n'aurez jamais à manipuler les adresses mémoire directement.

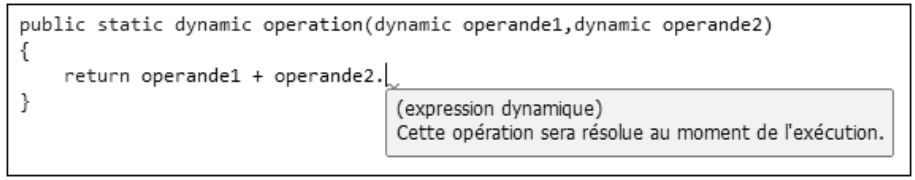
Une variable de type Object pourra donc faire référence à n'importe quel autre type de valeur y compris des types numériques simples. Cependant, le code sera moins rapide du fait de l'utilisation d'une référence.

Le type dynamic

Depuis sa première version, le langage C# est un langage statiquement typé. Chaque variable utilisée doit être déclarée avec un type défini. Cette contrainte permet au compilateur de vérifier que vous ne réalisez avec cette variable que des opérations compatibles avec son type. Ceci impose bien sûr de connaître le type de la variable au moment de la conception de l'application. Il arrive cependant parfois que le type de la variable ne soit connu qu'au moment de l'exécution de l'application. Il est, dans ce cas, possible d'utiliser le mot-clé **dynamic** comme type pour la variable concernée. Pour les variables déclarées avec ce type, le compilateur ne fait aucune vérification de compatibilité concernant les opérations exécutées avec cette variable. Ces opérations de vérification sont effectuées seulement au moment de l'exécution de l'application. Si ces opérations ne sont pas compatibles avec le type de la variable, une exception est déclenchée. La fonction ci-dessous attend deux paramètres dont le véritable type n'est pas connu lors de la conception de la fonction, c'est pourquoi ils sont déclarés avec le mot-clé **dynamic**. Le type retourné par la fonction, dépendant du type des paramètres qui lui sont passés au moment de l'appel, est également déclaré avec le mot-clé **dynamic**. Cette fonction utilise l'opérateur + sur les deux paramètres qui lui sont passés.

```
public static dynamic operation(dynamic operande1,dynamic operande2)
  {
      return operande1 + operande2;
  }
```

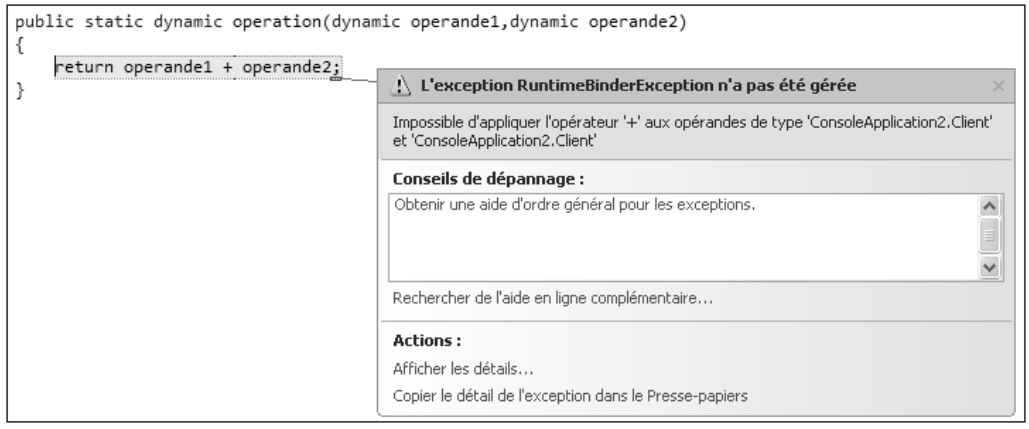
Le type des paramètres `operande1` et `operande2` étant inconnu au moment de la conception, Visual Studio est incapable de faire la moindre proposition sur les différentes méthodes pouvant être utilisées sur ces variables.



De la même façon il accepte sans problème que la fonction soit utilisée dans les différents cas de figure ci-dessous :

```
int i1;
int i2;
i1 = 2;
i2 = 4;
Console.WriteLine(operation(i1, i2));
String s1;
String s2;
s1 = "2";
s2 = "4";
Console.WriteLine(operation(s1, s2));
Client c1;
c1 = new Client();
Client c2;
c2 = new Client();
Console.WriteLine(operation(c1, c2));
```

Au moment de l'exécution, les deux premiers appels de la fonction se passent sans problème. Le premier effectue une addition des deux entiers, le second effectue une concaténation des deux chaînes de caractères. Par contre, le troisième appel qui utilise des instances de la classe `Client` déclenche une exception car l'opérateur `+` n'est pas applicable sur ce type de données.



Cet exemple montre bien qu'il faut être prudent avec l'utilisation du type **dynamic** et toujours prévoir la récupération de l'exception qui peut se produire en cas d'utilisation inadaptée du type réel de données.

Cette fonctionnalité est principalement utilisée pour manipuler des éléments obtenus à partir d'un langage dynamique (IronRuby ou IronPython) ou d'une API COM.