# La notion de variables

Programmation de base sur PC

# Plan du cours

- Présentation de l'enseignant
- Présentation des élèves
- Présentation du plan ce cours
- Introduction
- La notion de variables
  - La notion de type de variables
  - Déclaration d'une variable
  - Instruction d'affectation
  - Opérateurs arithmétiques
  - Exercices

# Présentation de l’enseignant

# Présentation des élèves

Quels sont les éléments connus en informatique?

# Présentation du plan de cours

# En général

# Introduction

- Afin d'effectuer des tâches de la vie de tous les jours, vous utilisez des logiciels informatique :
  - Traitement de texte pour créer votre curriculum vitae
  - Logiciel de gestion des ventes pour acheter dans un magasin
  - Site web pour réserver un voyage
  - Jeu vidéo pour se divertir

- Peu importe le langage utilisé pour programmer, votre ordinateur, lui, ne comprend qu'un seul langage, le binaire
  - Suite de 0 et de 1

# Introduction

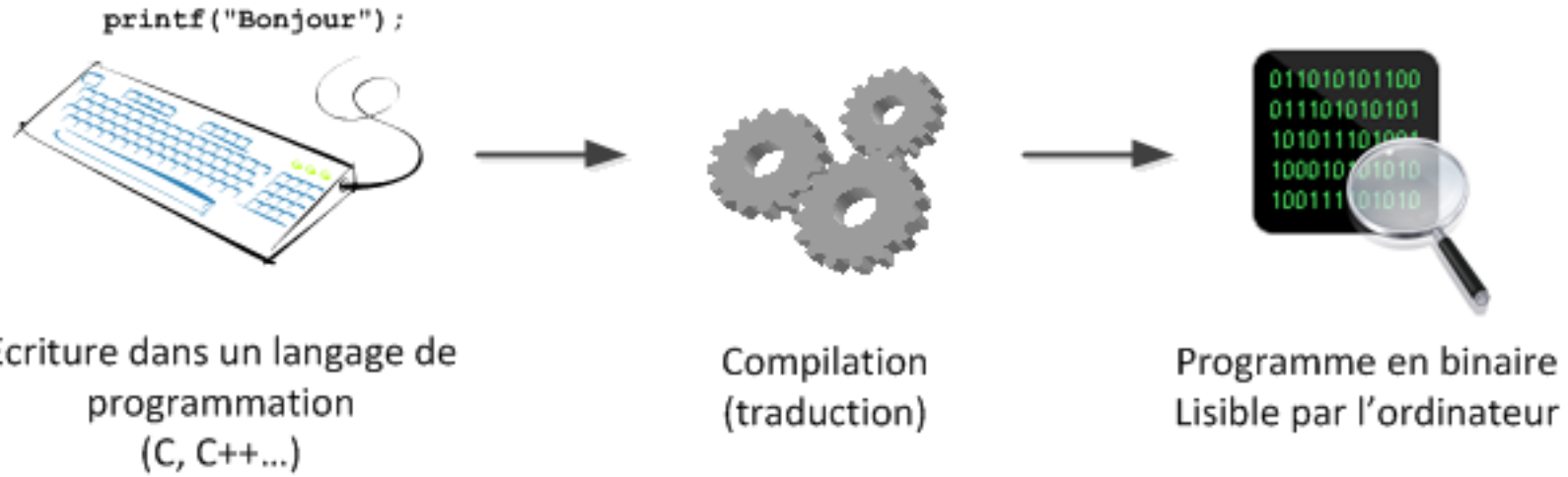
- Imaginez-vous programmer un logiciel de gestion des achats avec de simples 0 et de 1.
  - 01101011 00011011 11111010 00010101
- Impossible à comprendre me direz-vous
  - C’est pourquoi les langages de programmation sont nés; pour transformer du langage plus « humain » vers une série de 0 et de 1 compréhensible par les ordinateurs.

# Langages dit « traditionnels »

- Prenons par exemple le langage C
- Afin d'écrire « Bonjour » à l'écran nous devrons écrire ce code :

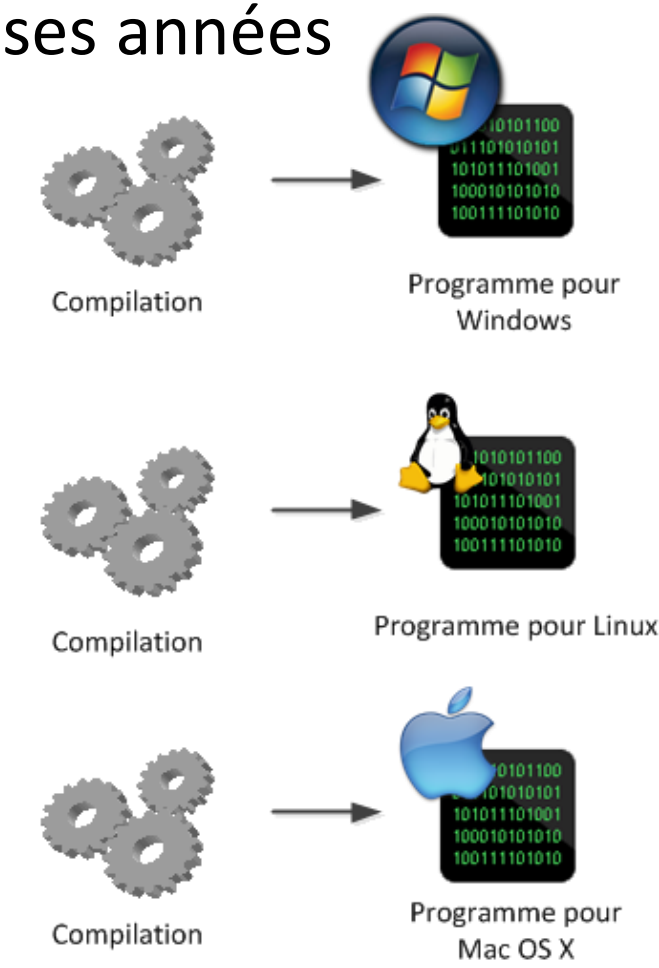
```
printf("Bonjour");
```

- On doit compiler ce code pour que l'ordinateur comprenne



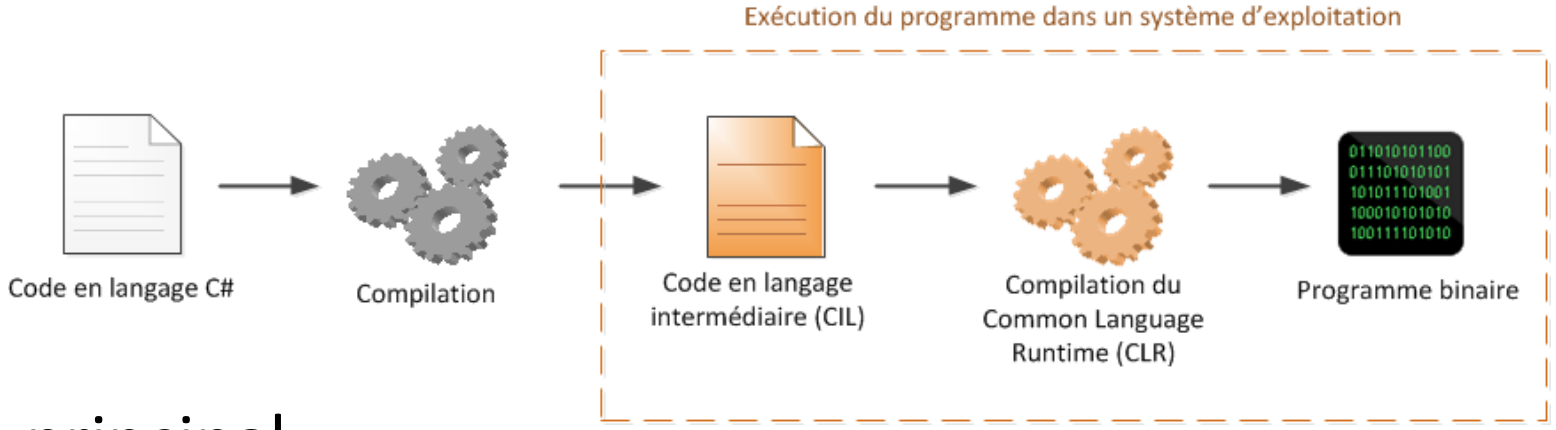
# Langages dit « traditionnels »

- Cette méthode a fait ses preuves depuis de nombreuses années
- Inconvénients
  - Impossible d'exécuter une application compilée pour Windows sur un ordinateur Mac
  - Il faut compiler pour chaque système d'exploitation
  - Fastidieux à gérer



# Langages dit « managé »

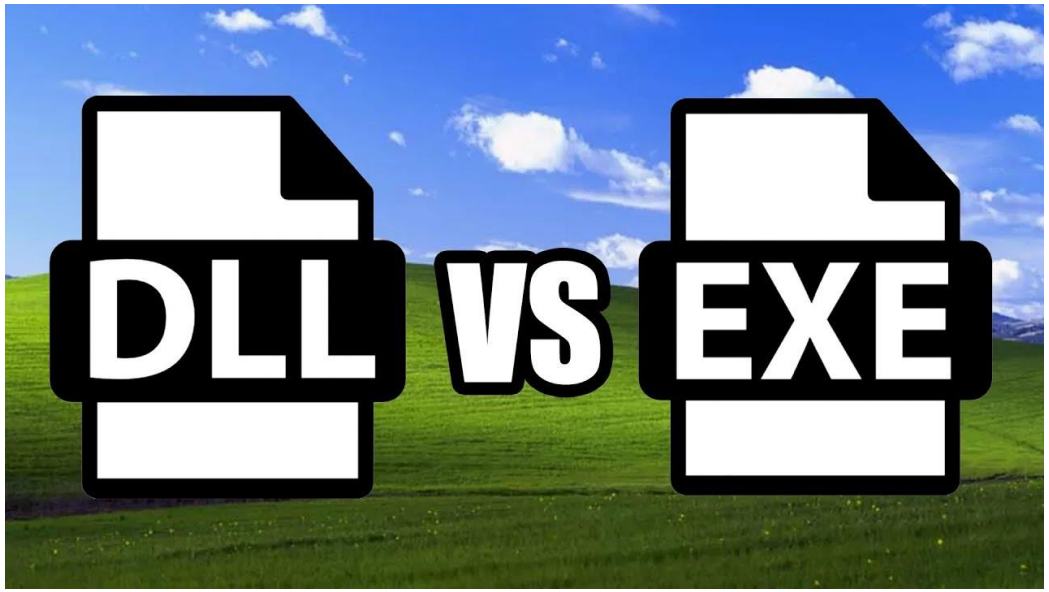
- La compilation en C# ne donnera pas du code binaire
  - Transformé en langage intermédiaire
    - CLI en français (Code Langage Intermédaire)
    - MSIL en anglais (MicroSoft Intermediate Language)




- Avantage principal
  - Le programme est toujours adapté à l'ordinateur sur lequel il tourne

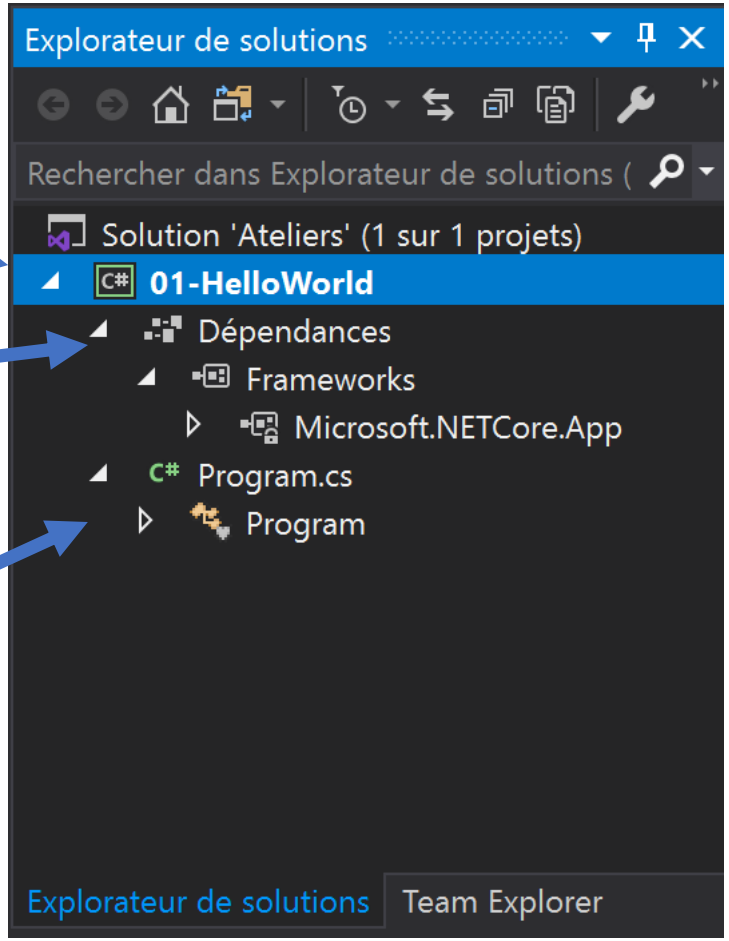
# Application vs. bibliothèque

- La compilation en langage intermédiaire se nomme « assembly »
- Il existe deux type d'assembly
  - Fichiers .exe qui représentent des applications
  - Fichiers .dll qui représentent des bibliothèques de fonctions



# La structure d'un projet

- Nom de la solution en cours
- Ce que VS a besoin pour faire fonctionner votre programme, les dépendances
- Emplacement où nous placerons le code
  - Les fichier .cs pour C Sharp (Program.cs)



# La console

- Le but du projet 01 est de créer une application console
- Pour ce faire nous aurons souvent besoin d'écrire et de lire du texte
- La fonction WriteLine de la classe Console sert à écrire du texte

```
Console.WriteLine(”Bonne journée à toi!");
```

- La fonction ReadLine permet de lire une ligne de texte du clavier

```
var line = Console.ReadLine();
Console.WriteLine(line);
```

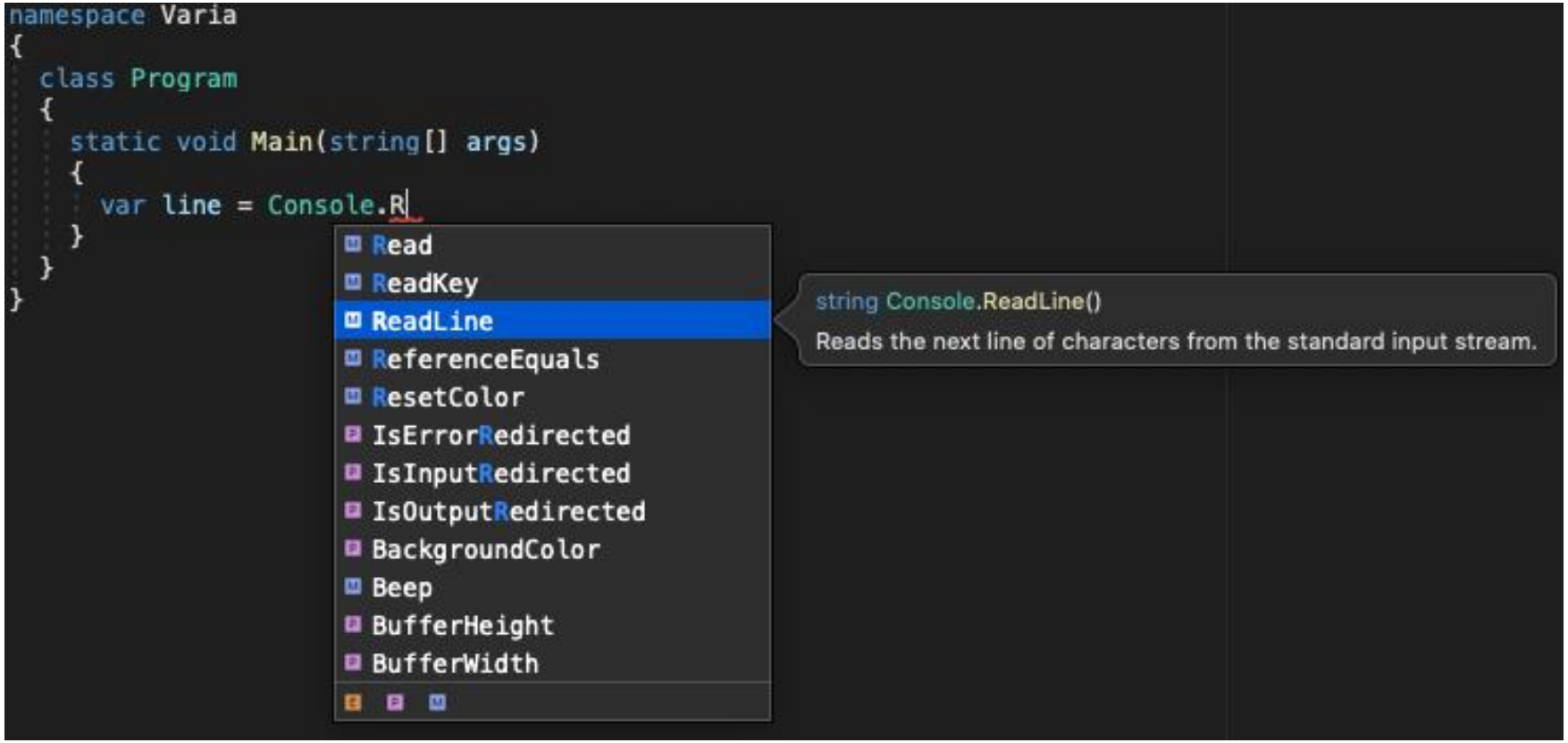
# Les commentaires

- Lors d'une approche pédagogique, il est recommandé d'ajouter des commentaire à votre code afin de se rappeler des points précis.
- Il est possible d'écrire un commentaire :
  - Par ligne en débutant la ligne avec //
  - Par bloc en intégrant le texte du commentaire entre /*  */

```
 /* permet d'afficher du texte
sur la console */
Console.WriteLine("Hello World !!"); // ne pas oublier le point virgule
```

# La complétion automatique

- Propose de compléter automatiquement ce que l'on a l'intention d'écrire comme code et propose des alternatives et de l'aide



# La notion de variables

# Les variables

- Espace mémoire accessible par un nom permettant d'emmagasiner des données
- Il existe plusieurs types de variables pour sauvegarder plusieurs types de données.
  - Je met 10 dans la variable `moisEnCours`
  - Je met « Francis Robert » dans la variable `nomDuMembre`
  - Je met 89,5 dans la variable `kilogrammes`

| Nom variable | Adresse mémoire | Valeur |
|---|---|---|
| kilogrammes | 1000 | 89,5 |

# Les variables: Règles à respecter

- Il faut donner un nom significatif aux variables
  - Éviter les nom non significatif: `abc`, `datenaiss`, `hr`, etc
- Tous les noms de variables ne sont pas automatiquement valide.
- Nous devons respecter ces règles:
  1. Seul les caractères alphanumérique , _ est permis
  2. Elle doivent commencer par une lettre, _ seulement
  3. Aucun espace ne peut figurer dans un nom de variable
  4. Sont sensibles aux majuscules/minuscules (*case-sensitive*), par exemple, `unNom` est différent de `UNNOM`
  5. Ne doivent pas représenter un mot clé C#

# Les variables: Règles à respecter

- Autorisés
  - `Compte` (majuscule ok)
  - `Num_2` (underscore permis et chiffre pas au début)
  - `Undeux` (majuscule ok et pas d'espace)
  - `VALEUR_temporaire` (underscore permis)
- Non autorisés
  - `pourquoi#pas` (caractère #)
  - `-plus` (caractère -)
  - `@adresse` (caractère @)
  - `base` (mot clé)
  - `2001espace` (chiffre en début de variable)

# La notion de type

- Un programme doit gérer des informations de **nature** diverse.
- `123` ou `2.4` sont de type numérique
- `"Spinoza"` est un mot composé de caractères
- À chaque type sont associés les éléments suivants :
  - Un **code spécifique** permettant la traduction de l'information en binaire
  - Un **ensemble d'opérations réalisables** en fonction du type de variable utilisé.
    - Par exemple, si la division est une opération cohérente pour deux valeurs numériques, elle ne l'est pas pour deux valeurs de type caractère.
  - Un **intervalle de valeurs possibles** dépendant du codage utilisé.
    - Par définition, à chaque type correspond un même nombre d'octets et, par conséquent, un nombre limité de valeurs différentes.

# La notion de type: Valeur logique

- Il s’agit du type boolean
- Les valeurs logiques ont deux états : **”true"** (1) ou **"false”** (0)
- Elles ne peuvent prendre aucune autre valeur que ces deux états

| Type | Description |
|---|---|
| bool | Une valeur booléenne (vrai ou faux) |

```
bool estVendu = true;
bool estOuvert = false;
```

Nommez d’autre cas possibles pour l’utilisation d’une variable `bool`

# La notion de type: Les entiers

| Type | Description |
|---|---|
| byte | Entier de 0 à 255 |
| short | Entier de -32768 à 32767 |
| int | Entier de -2147483648 à 2147483647 |
| long | Entier de -92233720368547758O8 à 9223372036854775807 |

```
byte monthNumber = 10;
short currentYear = 2020;
int invoiceID = 100101;
long facebookID = 8998453928;
```

Nommez d'autre cas possibles pour l'utilisation de chacun de ces types

# La notion de type: Les réels

| Type | Description |
|---|---|
| float | Nombre simple précision de -3,402823e38 à 3,402823e38 |
| double | Nombre double précision de -1,79769313486232e308 à 1,79769313486232e308 |
| decimal | Nombre décimal convenant particulièrement aux calculs financiers |

```
float PI = 3.1416f;
double x = 12.983049;
decimal amount = 12.99m;
```

Nommez d'autre cas possibles pour l'utilisation de chacun de ces types

# La notion de type: Les caractères

- Permet de représenter les caractères isolés (‘a’, ‘Z’, ‘+’, …)
- Pour décrire une suite de caractères (mots, phrases), on utilise le type **String**
- Pour décrire une variable de type char, l’ordinateur doit utiliser une table de correspondance qui associe à un caractère donné
  - Quel que soit le jeu de caractères choisi (alphabet latin, alphabet russe, idéogrammes chinois) chaque caractère est représenté dans la mémoire de l’ordinateur par une valeur numérique unique : cette opération s’appelle **l’encodage**

http://www.asciichars.com/_site_media/ascii/ascii-chars-landscape.jpg

# La notion de type: Les caractères

- Selon la table ASCII de la page précédente :
- Quelle est la valeur décimale de la lettre ‘a’ minuscule ?
  - **97**
- Quelle est la valeur hexadécimale du signe ‘+’ plus ?
  - **2B**
- Quelle est la valeur binaire de la lettre ‘z’ miniscule?
  - **122 → 01111010**
- Que représente la valeur codée `00001101`?
  - **Un retour à la ligne (enter)**

# La notion de type: Les chaînes de caractères

- L'opérateur « + » peut également servir à concaténer des chaines de caractères.

```
String firstName = "Francis";
String lastName = "Robert";
String name = firstName + " " + lastName;
```

- La variable « name » contiendra alors la valeur « Francis Robert »

Nommez d'autre cas possibles pour l'utilisation de la concaténation

# La notion de type: Les caractères et chaîne

| Type | Description |
|---|---|
| char | Représente un caractère |
| string | Une chaine de caractère |

```
char input = 'Q';
string name = "Francis Robert"
```

- Le caractère s'inscrit entre **apostrophes**
- La chaîne de caractères s'inscrit entre **guillemets**

Nommez d'autre cas possibles pour l'utilisation de chacun de ces types

# La notion de type

| Type | Description |
|---|---|
| byte | Entier de 0 à 255 |
| short | Entier de -32768 à 32767 |
| int | Entier de -2147483648 à 2147483647 (10 chiffres) |
| long | Entier de -9223372036854775808 à 9223372036854775807 (19 chiffres) |
| float | Nombre simple précision de -3,402823e38 à 3,402823e38 |
| double | Nombre double précision de -1,7976931348623e308 à 1,7976931348623e308 |
| decimal | Nombre décimal convenant particulièrement aux calculs financiers |
| char | Représente un caractère |
| string | Une chaine de caractère |
| bool | Une valeur booléenne (vrai ou faux) |

# Déclaration d'une variable

- Peuvent être placées indifféremment au début ou en cours de programme
- Un espace mémoire de la taille de la variable est reservé par C#
  - Par exemple, déclarer une variable `int clePrimaire;` réserve 4 octets de mémoire
- Le contenu de la variable (espace mémoire) est vide
  - **0** pour les variables numériques
  - **`null`** pour les type objets (par exemple String)

# Instruction d’affectation

# Rôle et mécanisme de l'affectation

- L'affectation est le mécanisme qui permet de placer une valeur dans un emplacement mémoire. Elle a pour forme :
  - `Variable = Valeur;`
  - `Variable = Expression mathématique;`
- L'instruction d'affectation s'effectue dans l'ordre suivant :
  1. calcule la valeur de l'expression figurant à droite du signe égal ;
  2. range le résultat obtenu dans la variable mentionnée à gauche du signe égal.
- Quel serait alors la valeur de la variable **p** ici ?

```
int n, p;
n = 4;
p = 1+n*5 ;
```

→ `n*5 = 20` → `1+20 = 21`

# Rôle et mécanisme de l’affectation

- La variable placée à droite du signe égal (=) n’est jamais modifiée, alors que celle qui est à gauche l’est toujours.
- Exemple:

```
int a, b;
a = 1;
b = a + 3;
a = 3;
```

| Instruction | a | b |
|---|---|---|
| `a = 1` | 1 | - |
| `b = a + 3` | 1 | 4 |
| `a = 3` | 3 | 4 |

# Déclaration et affectation d'une variable

- Déclarer une variable revient à lui réserver une adresse mémoire

```
byte moisEnCours;
String nomDuMembre;
float kilogrammes;
```

- Affecter une valeur c'est placer une donnée dans cet espace mémoire.

```
moisEnCours = 10;
nomDuMembre = "Francis Robert";
kilogrammes = 89.5;
```

- Il est possible de déclarer et d'affecter en une seule ligne

```
byte moisEnCours = 10;
String nomDuMembre = "Francis Robert";
float kilogrammes = 89.5;
```

# Confusion à éviter

- Le symbole de l'affectation est le signe égal ( = ). Ce signe, très largement utilisé dans l'écriture d'équations mathématiques, est source de confusion lorsqu'il est employé à contre-sens.

```
int a = 8;
a = a + 1;
```

- Si cette expression est impossible à écrire d'un point de vue mathématique, elle est très largement utilisée dans le langage informatique. Elle signifie :
  - calculer l'expression a + 1 ;
  - ranger le résultat dans a.
- Ce qui revient à augmenter de 1 la valeur de a

# Mise-en-oeuvre

Afin de mieux comprendre, programmons !

```
String nomDuMembre;
float kilogrammes;
Console.WriteLine("Quel est le nom du nouveau membre ?");
nomDuMembre = Console.ReadLine();
Console.WriteLine("Quel est le poids de " + nomDuMembre + " ?");
String poids = Console.ReadLine();
kilogrammes = float.Parse(poids);
Console.WriteLine("Bienvenue à " + nomDuMembre + " pesant " + kilogrammes);
```

```
Quel est le nom du nouveau membre ?
Francis Robert
Quel est le poids de Francis Robert ?
89.5
Bienvenue à Francis Robert pesant 89.5 Kg
```

# Permuter les valeurs de deux variables

- Nous souhaitons échanger les valeurs de deux variables de même type, appelées `a` et `b`
- La pratique courante de l'écriture des expressions mathématiques fait que, dans un premier temps, nous écrivions les instructions suivantes :

```
a = b;
b = a;
```

| | a | b |
|---|---|---|
| `valeur initiale` | 2 | 8 |
| `a = b` | 8 | 8 |
| `b = a` | 8 | 8 |

- `a = b` détruit la valeur de `a` en plaçant la valeur de `b` dans la case mémoire `a`
- Mais comment faire alors ?

# Permuter les valeurs de deux variables

- Une solution consiste à utiliser une variable supplémentaire
  - contenir temporairement une copie de la valeur de `a`
  - avant que cette dernière soit écrasée par la valeur de `b`

```
tmp = a;
a = b;
b = tmp;
```

| | a | b | tmp |
|---|---|---|---|
| `valeur initiale` | 2 | 8 | _ |
| `tmp = a` | 2 | 8 | 2 |
| `a = b` | 8 | 8 | 2 |
| `b = tmp` | 8 | 2 | 2 |

# Les variables: Opérations de base

- Les opérateurs « + », « * », « / » ou encore « - » servent à faire les opérations mathématiques.

Programmons pour mieux comprendre

```
int addition = 4 + 5;
int subtraction = 1540 - 849;
int multiplication = 9 * 3;
int division = 10 / 3;

String message =
@"Addition = " + addition
+ "\nSoustraction = " + subtraction
+ "\nMultiplication = " + multiplication
+ "\nDivision = " + division;

Console.WriteLine(message);
```

https://csharpindepth.com/Articles/Strings

# Mise-en-oeuvre

Programmez un algorithme qui permet de permuter les valeurs des variables a et b tel que prescrit par la sortie suivante

```
Entrez le nombre a :
50

Entrez le nombre b :
75

a vaut maintenant 75 et b 50
```

# Les opérateurs arithmétiques

# Les opérateurs: Les bases

- Écrire un programme ne consiste pas uniquement à échanger des valeurs
- Java utilise des caractères qui symbolisent les opérateurs arithmétiques

| Symbole | Opération |
|---|---|
| + | Addition |
| – | Soustraction |
| * | Multiplication |
| / | Division |
| % | Modulo |

# Les opérateurs: Les bases

- L'opération d'addition s'écrit : `a = b + 4`
- L'opération de soustraction s'écrit : `a = b - 5`
- L'opération de division s'écrit : `a = b / 2`
- L'opération de multiplication s'écrit : `a = b * 4`
  - et non pas `a = 4b` ou `a = 4 x b`
- L'opération de modulo s'écrit : `a = b % 3`
  - Le modulo d'une valeur correspond au reste de la **division entière**.
  - Ainsi : `5 % 2 = 1`.

# Les opérateurs: Priorité

- L'ordinateur doit pouvoir déterminer quel est l'ordre des opérations à effectuer
- Le calcul de l'expression a – b / c * d  peut, a priori, signifier plusieurs opérations :
  - Calculer la soustraction puis la division et pour finir la multiplication, soit le calcul :
    - `((a - b) / c) * d`
  - Calculer la multiplication puis la division et pour finir la soustraction, c'est-à-dire l'expression :
    - `a – (b / (c * d))`

# Les opérateurs: Priorité

- Afin d'éviter toute ambiguïté, il existe des règles de priorité entre les opérateurs
- Basées sur la définition de deux groupes d'opérateurs

| Groupe 1 | Groupe 2 |
|---|---|
| + - | * / % |

- Les groupes étant ainsi définis, les opérations sont réalisées sachant que:
  - Dans un même groupe, l'opération se fait dans l'ordre d'apparition des opérateurs (sens de lecture).
  - Le deuxième groupe a priorité sur le premier.

# Les opérateurs: Priorité

- L'expression `a - b / c * d` est calculée de la façon suivante :

| Priorité | Opérateur | |
|---|---|---|
| Groupe 2 | / | Le groupe 2 a priorité sur le groupe 1, et la division apparaît dans le sens de la lecture avant la multiplication. |
| Groupe 2 | * | Le groupe 2 a priorité sur le groupe 1, et la multiplication suit la division. |
| Groupe 1 | - | La soustraction est la dernière opération à exécuter, car elle est du groupe 1. |

- Cela signifie que l'expression est calculée de la façon suivante :
  - `a - (b / c * d)`

# Le type d'une expression arithmétique

- Peut être déterminé à partir du type de variables qui composent l'expression.

| Terme | Opération | Terme | Résultat |
|---|---|---|---|
| Entier | + – * / % | Entier | Entier |
| Réel | + – * / | Réel | Réel |

- De ce fait, pour un même calcul, le résultat diffère selon qu'il est effectué à l'aide de variables de type réel ou de type entier.

# Le type d'une expression arithmétique

- Exemple : diviser deux **entiers:**

```
int x = 5, y = 2, z;
z = x / y;
```

- Quel sera le résultat ?

| | x | y | z |
|---|---|---|---|
| valeur initiale | 5 | 2 | - |
| `z = x / y` | 5 | 2 | 2 |

# Le type d'une expression arithmétique

- Peut être déterminé à partir du type de variables qui composent l'expression.

| Terme | Opération | Terme | Résultat |
|---|---|---|---|
| Entier | + – * / % | Entier | Entier |
| Réel | + – * / | Réel | Réel |

- De ce fait, pour un même calcul, le résultat diffère selon qu'il est effectué à l'aide de variables de type réel ou de type entier.

# Le type d'une expression arithmétique

**Question** Que vaut la variable `résultat` après exécution des instructions suivantes :

```
int résultat, premier = 5, second = 3, coefficient = 2 ;
résultat = coefficient * premier / second ;
```

**Réponse** La multiplication et la division appartiennent au même groupe, les opérations sont donc réalisées dans le sens de la lecture. La multiplication `coefficient * premier` donne pour résultat `10`, puis la division de `10` par `second` donne `3`, puisque les deux valeurs sont entières. La variable `résultat` a donc pour valeur `3`.

# Le type d'une expression arithmétique

**Question** Que vaut la variable `résultat` après exécution des instructions suivantes :

```
int résultat, premier = 5, second = 3, coefficient = 2 ;
résultat = premier / second * coefficient;
```

**Réponse** La multiplication et la division appartiennent au même groupe, les opérations sont donc réalisées dans le sens de la lecture. La division `premier / second` a pour résultat `1`, puisque les deux valeurs sont entières. Puis la multiplication de `1` par `coefficient` donne `2`. La variable `résultat` a donc pour valeur `2`.

**Remarque** À travers ces deux questions-réponses, nous observons que l'ordre des opérations et le type des données utilisées à une forte influence sur le résultat du calcul effectué.

# Le type d'une expression arithmétique

- Exemple : diviser deux **réels:**

```
double x = 5 , y = 2, z;
z = x / y;
```

- Quel sera le résultat ?

| | x | y | z |
|---|---|---|---|
| valeur initiale | 5 | 2 | - |
| z = x / y | 5 | 2 | 2.5 |

# Le type d'une expression arithmétique

**Question** Que vaut la variable `résultat` après exécution des instructions suivantes :

```
double résultat, premier = 5, second = 3, coefficient = 2 ;
résultat = coefficient * premier / second ;
```

**Réponse** La multiplication et la division appartiennent au même groupe, les opérations sont donc réalisées dans le sens de la lecture. La multiplication `coefficient * premier` donne pour résultat `10.0`, puis la division de `10.0` par `second` donne `3.3333333`, puisque les deux valeurs sont réelles. La variable `résultat` a donc pour valeur `3.3333333`.

# Le type d'une expression arithmétique

**Question** Que vaut la variable `résultat` après exécution des instructions suivantes :

```
double résultat, premier = 5, second = 3, coefficient = 2 ;
résultat = premier / second * coefficient;
```

**Réponse** La multiplication et la division appartiennent au même groupe, les opérations sont donc réalisées dans le sens de la lecture. La division `premier / second` a pour résultat `1.6666666`, puisque les deux valeurs sont réelles. Puis la multiplication de `1.6666666` par `coefficient` donne `3.3333333`. La variable `résultat` a donc pour valeur `3.3333333`.

# La transformation de types

- Les termes d'une opération ne sont pas nécessairement tous du même type.
- Pour écrire une opération, toutes les combinaisons entre les différentes catégories de types peuvent se présenter.

| Terme | Opération | Terme | Résultat |
|---|---|---|---|
| `byte` | `+ - * /` | `int` | `int` |
| `int` | `+ - * /` | `double` | `double` |

# La transformation de types

- L'ordinateur ne sait calculer une expression mathématique que lorsque toutes les variables de l'expression sont du **même type**.
- Lorsque tel n'est pas le cas, c'est-à-dire si l'expression est mixte :
  - l'ordinateur doit transformer le type de certaines variables pour que tous les membres de l'expression deviennent de **même type**.
- Cette transformation est appelée **conversion d'ajustement de type**
  - La conversion d'un nombre réel en nombre entier, par exemple, ne peut se réaliser qu'en supprimant les nombres situés après la virgule et en ne gardant que la partie entière du nombre.
  - Il y a perte de données.

# La transformation de types

- Le compilateur effectue <u>automatiquement</u> la conversion des données selon l'ordre suivant :
  - `byte -> short -> int -> long -> float -> double`
- Quelles seront les valeurs de `result_float` et `result_int` ?

```
int a = 4;
int result_int;
float x = 2.0f;
float result_float;
result_float = a / x;
result_int = a / x;
```

# La transformation de types

- Quelles seront les valeurs de `result_float` et `result_int` ?

```
int a = 4;
int result_int;
float x = 2.0f;
float result_float;
result_float = a / x;
result_int = a / x;
```

| | a | x | result_float | result_int |
|---|---|---|---|---|
| `a = 4` | 4 | _ | _ | _ |
| `x = 2.0f` | 4 | 2.0f | _ | _ |
| `result_float = a/x` | 4 | 2.0f | 2.0f | _ |
| `result_int = a/x` | 4 | 2.0f | _ | Impossible dès la compilation |

# La transformation de types: le cast

- La conversion avec perte d'information est autorisée dans certains cas grâce au mécanisme du **cast**
- Il peut être utile de transformer un nombre réel en entier par exemple pour calculer sa partie entière (525.44 en entier = 525)
- Le compilateur demande de convertir explicitement les termes de l'opération
- Ainsi, pour transformer un float en int, il suffit de placer le terme (int) devant la variable ou l'opération de type float

# La transformation de types: le cast

- Exemple de cast :

```
int a = 5, result1;
float x = 2.0f;
result1 = (int)(a / x);
```

| | a | x | result1 |
|---|---|---|---|
| a = 5 | 5 | _ | _ |
| x = 2.0f | 5 | 2.0f | _ |
| result1 = (int) a / x | 5 | 2.0f | 2 |

- Quel est le résultat si l'on retire le cast (int) ?

# Le type d'une expression arithmétique

**Question** Que vaut la variable `résultat` après exécution des instructions suivantes :

```
int premier = 5, second = 3, coefficient = 2 ;
double résultat ;
résultat = (double) coefficient * premier / second ;
```

**Réponse** Le mécanisme de `cast` transforme la variable `coefficient` en `double`. Ensuite, la multiplication et la division appartenant au même groupe, les opérations sont réalisées dans le sens de la lecture. La multiplication `coefficient * premier` donne pour résultat `10.0` (de type `double`), puis la division de `10.0` par `second` donne `3.3333333`, puisque `10.0` est une valeur réelle. La variable `résultat` a donc pour valeur `3.3333333`.

# Le type d'une expression arithmétique

**Question** Que vaut la variable `résultat` après exécution des instructions suivantes :

```
int premier = 5, second = 3, coefficient = 2 ;
double résultat ;
résultat = (double) (coefficient * premier / second);
```

**Réponse** Les parenthèses entourant l'expression `(coefficient * premier / second)` font que cette expression est calculée avant d'être transformée en `double`. La multiplication et la division appartenant au même groupe, les opérations sont réalisées dans le sens de la lecture. La multiplication `coefficient * premier` donne pour résultat `10`, puis la division de `10` par `second` donne `3`, puisque les deux valeurs sont entières. Ce résultat est ensuite transformé en `double` grâce au mécanisme du `cast`. La variable `résultat` a donc pour valeur finale `3.0`.

# Exercices

# Exercice

```
 double a;          a = (10 / 3f + 3) / 5d;          double b;
b = (10 / 3 + 3) / 5;          decimal c;          c = 10.99m +
11.11m;          int d = 10f * 2;
```