

Information, Calcul, Communication (partie programmation) : VARIABLES & EXPRESSIONS

Jean-Cédric Chappelier

Laboratoire d'Intelligence Artificielle
Faculté I&C

Objectifs de la leçon d'aujourd'hui

- ▶ Résumer ce qu'il faut avoir retenu des premières leçons :
 - ▶ variables
 - ▶ types
 - ▶ expressions
- ▶ Etude de cas (très simple ici)
- ▶ Compléments de cours :
 - ▶ `auto`
 - ▶ `const/constexpr`
- ▶ Répondre à vos questions

Rappel du calendrier

	MOOC	décalage / MOOC	exercices prog. 1h45 Jeudi 8-10	cours prog. 45 min. Jeudi 10-11
1 11.09.25	--	-1	prise en main	Bienvenue/Introduction
2 18.09.25	1. variables	0	variables / expressions	variables / expressions
3 25.09.25	2. if	0	if – switch	if – switch
4 02.10.25	3. for/while	0	for / while	for / while
5 09.10.25	4. fonctions	0	fonctions (1)	fonctions (1)
6 16.10.25		1	fonctions (2)	fonctions (2)
- 23.10.25				
7 30.10.25	5. tableaux (vector)	1	vector	vector
8 06.11.25	6. string + struct	1	array / string	array / string
9 13.11.25		2	structures	structures
10 20.11.25	7. pointeurs	2	pointeurs	pointeurs
11 27.11.25		-	entrées/sorties	entrées/sorties
12 04.12.25		-	erreurs / exceptions	erreurs / exceptions
13 11.12.25		-	révisions	théorie : sécurité
14 18.12.25	8. étude de cas	-	révisions	Révisions

Le langage C++

Le langage C++ est un langage **orienté-objet compilé fortement typé**.
Schématiquement :

$$\text{C++} = \text{C} + \text{typage fort} + \text{objets}$$

Parmi les avantages de C++, on peut citer :

- ▶ un des langages **objets** les plus utilisés ;
- ▶ un langage **compilé**, ce qui permet la réalisation d'applications efficaces ;
- ▶ un **typage fort**, ce qui permet au compilateur d'effectuer de nombreuses vérifications lors de la compilation $\Rightarrow$ moins de « bugs »... ;
- ▶ un langage disponible sur pratiquement toutes les plate-formes.

Variables et types

À retenir :

- ▶ variable = représentation interne d'une « donnée » du problème traité = une *valeur*
- ▶ en C++, les valeurs (donc aussi les variables) sont **typées** :
« nature »/« ensemble d'appartenance » de la valeur
- ▶ Les principaux **types élémentaires** définis en C++ sont :
 - `int` : (une partie des) nombres entiers
 - `double` : (une partie des) nombres décimaux
 - `bool` : les valeurs logiques « *vrai* » (`true`) et
« *faux* » (`false`)
 - `char` : les caractères (`'a'`, `'!'`, ...)

Note : nous verrons plus tard d'autres types :
les types **composés**, les types **énumérés** et les types **synonymes** (alias de types).

Valeurs Littérales

- ▶ valeurs littérales de type `int` : `1`, `12`, ...
- ▶ valeurs littérales de type `double` : `1.23`, ...
Remarque :
 - `12.3e4` correspond à $12.3 \cdot 10^4$ (soit `123000`)
 - `12.3e-4` correspond à $12.3 \cdot 10^{-4}$ (soit `0.00123`)
- ▶ valeurs littérales de type `char` : `'a'`, `'!'`, ...
Remarque :
 - le caractère `'` se représente par `\'` (donc `'\''`)
 - le caractère `\` se représente par `\\` (donc `'\\'`)
- ▶ valeurs littérales de type booléen : `true`, `false`

Remarque : la valeur littérale `0` est une valeur d'initialisation qui peut être affectée à une variable de n'importe quel type.

Expression

- ▶ une expression représente un calcul à faire, à évaluer
- ▶ expression = combinaison d'expressions, de valeurs, de variables, à l'aide d'opérateurs
- ▶ toute expression a un type (et une valeur) :
en C++, **toute expression *fait* quelque chose et *vaut* quelque chose**

« Etude de cas » (une question en fait)

Qu'affiche le code suivant :

```
double x(3 / 4);  
cout << x << endl;  
  
double a(3);  
double b(4);  
double y(a / b);  
cout << y << endl;
```



C++11

auto



En **C++11**, on peut laisser le compilateur *deviner le type* d'une variable grâce au mot-clé **auto**.

Le type de la variable est déduit du *contexte*. Il faut donc qu'il y ait un contexte, c'est-à-dire une *initialisation*.

Par exemple :

```
auto val(2);  
auto j(2*i+5);  
auto x(7.2835);
```

Conseil : **N'abuser pas** de cette possibilité et explicitiez vos types autant que possibles. N'utilisez **auto** que dans les cas « techniques », par exemple (qui viendra plus tard dans le cours) :

```
for (auto p = v.begin(); p != v.end(); ++p)
```

au lieu de

```
for (vector<int>::iterator p = v.begin(); p != v.end(); ++p)
```

Données modifiables/non modifiables

Par défaut, les variables en C++ sont modifiables.

Si l'on ne souhaite pas modifier une « variable » après son initialisation : la définir comme **constante** (pour ce nom là uniquement)

La nature **modifiable** ou **non modifiable** d'une donnée *au travers de ce nom* peut être définie lors de la déclaration par l'indication du mot réservé **const**.

Elle ne pourra plus être modifiée par le programme en utilisant ce nom (toute tentative de modification *via ce nom* produira un message d'erreur lors de la compilation).

Exemples :

```
int const couple(2);
```

(mais **constexpr** serait encore mieux)

```
double const interet(3.0*taux+1.5);
```

(ici **constexpr** est impossible (sauf si **taux** est lui-même **constexpr**))

```
double const g(9.81);
```

(ici aussi **constexpr** serait encore mieux)



C++11

Expressions constantes



Depuis C++11, il existe aussi le mot clé `constexpr`.

Il est d'utilisation **plus générale**, mais est aussi **plus contraignant** que `const` : la valeur initiale doit pouvoir être calculée à la compilation.

☞ Les deux (`const` et `constexpr`) sont donc **très différents** !

- ▶ `const` indique au compilateur qu'une donnée ne changera pas de valeur au travers de ce nom ; mais
 1. le compilateur peut très bien ne pas connaître la valeur en question au moment de la compilation ; et
 2. cette valeur pourrait changer par ailleurs.
- ▶ `constexpr` indique au compilateur qu'une donnée ne changera pas du tout de valeur et qu'il doit pouvoir en calculer la valeur au moment de la compilation (c.-à-d. que cette valeur ne dépend pas de ce qu'il va se passer plus tard dans le programme).

Conseil : Si ces deux conditions sont vérifiées, on préférera utiliser `constexpr`.



Variables



En C++, une **valeur** à conserver est stockée dans une variable caractérisée par :

- ▶ son **type**
- ▶ et son **identificateur**;

(définis lors de la **déclaration**)

La **valeur** peut être définie une première fois lors de l'**initialisation**, puis éventuellement modifiée par la suite.

Rappels de syntaxe :

`type nom ;` (déclaration)

`type nom(valeur);` (initialisation)

`nom = expression ;` (affectation)

Types élémentaires :

`int`

`double`

`char`

`bool`

Exemples : `int val(2) ;`
`const double z(x+2.0*y);`
 `constexpr double pi(3.141592653);`
`i = j + 3;`



Opérateurs



Opérateurs arithmétiques

*	multiplication	
/	division	
%	modulo	
+	addition	
-	soustraction	
++	incrément	(1 opérande)
--	décrément	(1 opérande)

Opérateurs de comparaison

==	teste l'égalité logique
!=	non égalité
<	inférieur
>	supérieur
<=	inférieur ou égal
>=	supérieur ou égal

Opérateurs logiques

and	&&	« et » logique	
or		ou	
not	!	négation	(1 opérande)

Priorités (par ordre décroissant, tous les opérateurs d'un même groupe sont de priorité égale) :

not ++ --, * / %, + -, < <= > >=, == !=, and, or