Programmation « orientée système »

INTRODUCTION

Jean-Cédric Chappelier

Laboratoire d'Intelligence Artificielle
Faculté I&C

Objectifs de la leçon d'aujourd'hui

- Présenter le cours
  - Objectifs (« Quoi ? »)
  - Administration (« Comment ? »)
- Début du cours de C
  - les variables
  - expressions et opérateurs

Jean-Cédric Chappelier
EPFL

Objectifs du cours

☞ Apprendre à programmer **plus proche du système** :
entre Java et l'assembleur

Savoir *écrire des programmes en C* avancés qui :

1. manipulent directement la mémoire (**pointeurs**) ;
2. utilisent des fichiers ;
3. et les arguments de la ligne de commande.

☞ programmer plus proche du système ? ?

Jean-Cédric Chappelier
EPFL

Rappels

Un ordinateur, c'est :

processeur(s) ☞ *traitements* (processus)
mémoire ☞ *données*
périphériques ☞ échanges/communication (systèmes de fichiers, réseaux, ...)

Programmer c'est décomposer une **tâche** à automatiser sous la forme d'une séquence d'instructions (**traitements**) et de **données** adaptées à l'automate programmable utilisé.

traitements : **algorithmes**
➨ processus / threads

données
➨ internes : mémoire
➨ externes : fichiers, réseaux, autres entrées/sorties

Jean-Cédric Chappelier
EPFL

Contenu du cours

	C	(Rappel) Java
langage	+	+
gest. mémoire	+	N.A.
Fichiers	+	+
Processus/Threads	(cours OS)	±
Réseaux	/	±

\+ : abordé
± : évoqué
/ : pas présenté
N.A. : ne s'applique pas

Jean-Cédric Chappelier

Présentation générale du cours

Public : Cours « 1+2 » obligatoire pour les IN-BA4 et SC-BA4

Connaissances préalables requises : bases de programmation (p.ex. Java), bases de connaissances système

Langue : Français

Moyens : Concepts théoriques introduits lors de **cours** magistraux pré-enregistrés (`https://go.epfl.ch/progos_videos`) complétés de séances interactives « *live* » (Lu 8^{15}–9^{00})

mis en pratique, de manière guidée, lors de **séances d'exercices** sur (vos) machines (Lu 9^{15}–11^{00})

Jean-Cédric Chappelier

Présentation générale du cours

Principes :

cours = concepts et principes généraux

exercices = mise en pratique et approfondissement personnel
☞ voir les remarques préliminaires sur le site du cours

web = détails et recherche de compléments (https://progos.epfl.ch/)

forum = demande d'information et dialogue

Horaires et Contenu :
Un planning détaillant le contenu de chaque séance est disponible sur le site internet du cours.

https://moodle.epfl.ch/course/view.php?id=6731

Encadrement :
voir également le site Moodle du cours

Jean-Cédric Chappelier
EPFL

Interaction avec les enseignants

Plusieurs moyens pour contacter l'enseignant, assistants et étudiants-assistants pour poser des questions sur le cours ou les exercices :

- Durant les séances d'exercices :
  c'est le moyen le plus direct, et généralement le plus efficace.
- Par l'intermédiaire du forum
  ☞ moyen idéal pour diffuser la connaissance
  N'hésitez pas à en faire usage !

Les contacts personnel par email, téléphone ou visites devront être **strictement réservés aux cas urgents ou personnels !**

Jean-Cédric Chappelier

Support de cours

- Transparents mis à disposition via le **site Web** (`https://progos.epfl.ch/`)
- Énoncé des exercices
  disponibles sur le site Web en début de semaine
- Corrigé des exercices
  disponibles sur le site Web en début de semaine suivante

Ces éléments devraient constituer une **documentation suffisante** pour ce cours.

Notes et examens

☞ Branche de semestre de **3 crédits**

La note finale pour ce cours sera calculée de la façon suivante :

- 2 exercices à rendre (maison) ⇨ coef. 2 et 3 respectivement
- Examen théorique (120 min.) ⇨ coef. 15 (= 75%)

La note finale N de ce cours est calculée directement sur les points obtenus (et non pas les notes intermédiaires arrondies) par :

$$N = 1 - 0.25 \left\lfloor -20 \cdot \frac{\sum_x \theta_x (p_x / t_x)}{\sum_x \theta_x} \right\rfloor$$

où θ_x est le coefficient de l'épreuve x, avec p_x le nombre de points obtenus sur un total maximal de t_x.
En complément du total de points p_x, une note intermédiaire sera également publiée *à titre indicatif* pour chaque épreuve x :

$$n_x = 1 - 0.25 \left\lfloor -20 \cdot \frac{p_x}{t_x} \right\rfloor$$

Jean-Cédric Chappelier

Notes et examens — Exercices à rendre

Objectifs :

- vérifier la maîtrise pratique des concepts exposés en cours ;
- encourager un travail régulier ;
- fournir plus de retour aux étudiants.

Dates :

sujet	rendu
15 mars 12h00	**30 mars 23h59**
10 mai 12h00	**25 mai 23h59**

Pour augmenter encore le retour critique sur votre code, n'hésitez pas à poser des questions, demander des analyses

- pendant les séances d'exercices ;
- sur le forum du cours.

Avertissement / Pédagogie

Ce cours s'adresse à des personnes sachant déjà programmer (typiquement en Java) et non pas à des débutants.

Plusieurs *concepts de bases* sont donc *supposés connus* et seront rapidement rappelés.

De plus, en raison de la similitude entre certaines parties de la syntaxe de C et celle de Java, plusieurs aspects du langages C seront *très rapidement* présentés (les transparents sont néanmoins présents et assez détaillés),
et nous **insisterons plutôt sur les différences** et subtilités.

Il *vous* faut **néanmoins** suffisamment pratiquer la programmation C pour vous sensibiliser aux différences par vous-même :

1. ne croyez pas que parce que vous savez programmer en Java vous savez programmer en C ;
2. et ne tombez pas dans le piège de croire que c'est parce que les syntaxes sont assez similaires pour permettre de passer rapidement dessus en cours qu'il ne faut pas travailler.

À BON ENTENDEUR...

Jean-Cédric Chappelier

Le langage C

Le langage C est un langage typé impératif compilé.

Parmi les caractéristiques de C, on peut citer :

- le(/l'un des) langage(s) de programmation le(s) plus utilisé(s)
  https://www.tiobe.com/tiobe-index/
  https://stackify.com/popular-programming-languages-2018/
- un langage **compilé**, ce qui permet la réalisation d'applications efficaces
- un langage plus proche de la machine
  (moins abstrait mais plus efficace)
- un langage disponible sur toutes les plates-formes et de façon standardisée

Jean-Cédric Chappelier

Langages compilés (rappel)

Avantages et Inconvénients :

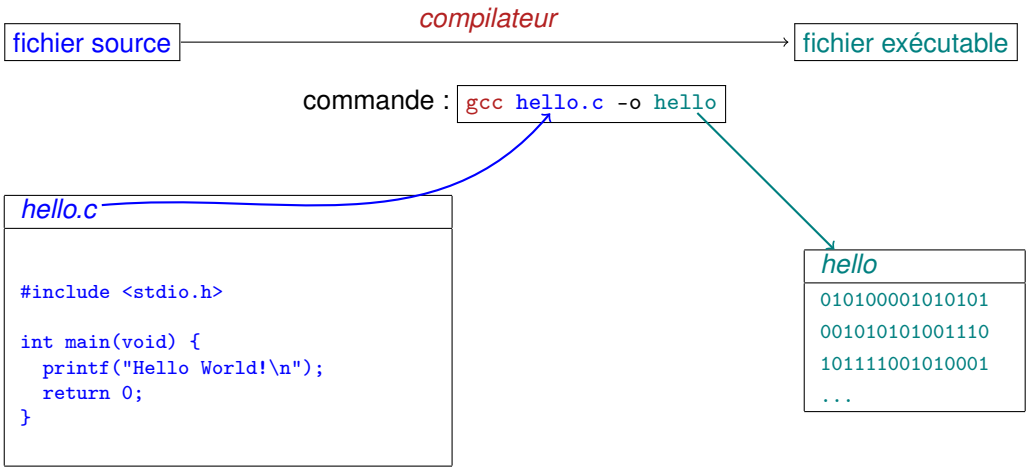
- De manière générale, un langage compilé permet la *réalisation d'applications plus efficaces ou de plus grande envergure*
  (optimisation plus globale, traduction effectuée une seule fois et non pas à chaque utilisation)

  (par opposition à un langage *interprété*, plus adapté au *développement rapide de prototypes* : on peut immédiatement tester ce que l'on est en train de réaliser)

- un langage compilé permet également de diffuser les programmes sous forme binaire, **sans** pour autant imposer la **diffusion sous forme lisible** et compréhensible par un humain
  ☞ protection de la propriété intellectuelle

Jean-Cédric Chappelier

Compilation d'un programme C (version simple)



C .vs. Java

	C	Java
langage	impératif	orienté objet
code objet :	code machine	byte code
vérification à la compilation	faible	forte
allocation mémoire	statique et dynamique	dynamique (mais non explicite)
manipulation directe de la mémoire	oui (pointeurs)	non
vérification des accès mémoire	non	oui
désallocation	à la main	automatique (garbage collector)

Jean-Cédric Chappelier

Structure générale d'un programme C

La structure très générale d'un programme C est la suivante :

```
#include <des trucs utiles>
...

(déclaration d'objets globaux)                                  [à éviter]

déclarations de fonctions utiles                                [recommandé]

int main(void)
{
corps du
programme principal                                  [si possible assez court]
return un int;
}
```

Premier exemple de programme C

résoudre (dans $\mathbb{R}$) une équation du second degré de type :

$$x^2 + b\,x + c = 0$$

saisir les données b et c

$\Delta \leftarrow b^2 - 4\,c$

Si $\Delta < 0$
  afficher « pas de solution »

Sinon
  Si $\Delta = 0$
    $x \leftarrow -\frac{b}{2}$
    afficher x
  Sinon
    $x \leftarrow \frac{-b-\sqrt{\Delta}}{2}$, $y \leftarrow \frac{-b+\sqrt{\Delta}}{2}$
    afficher x et y

Jean-Cédric Chappelier
EPFL

Premier exemple de programme en C

```c
#include <stdio.h>
#include <math.h>
int main(void) {
  double b     = 0.0;
  double c     = 0.0;
  double delta = 0.0;

  printf("Entrez b : "); scanf("%lf", &b);
  printf("Entrez c : "); scanf("%lf", &c);
  delta = b*b - 4*c;
  if (delta < 0.0) {
    printf("pas de solutions reelles\n");
  } else if (delta == 0.0) {
    printf("une solution unique : %f\n", -b/2.0);
  } else {
    printf("deux solutions : %f et %f\n",
           (-b-sqrt(delta))/2.0,
           (-b+sqrt(delta))/2.0);
  }
 return 0;
}
```

données
traitements
structures de contrôle

Jean-Cédric Chappelier

normes C89, C99 et C11

Il existe essentiellement trois versions du langage C (avec quelques corrections intermédiaires) :
le « C89 », le « C99 », et le « C11 » (avec quelques corrections en « C17 »)

Au niveau de ce cours, cela ne fait pas de différence fondamentale.
(Les différences pertinentes seront indiquées.)

Les 2 principales différences par rapport à la leçon d'aujourd'hui concernent :

- la déclaration des variables :
  - en C89, il est impératif que les variables soient déclarées avant toute expression (= en début de bloc)
  - depuis C99, on peut par contre intercaler lignes de déclaration de variables et lignes d'expressions
- les commentaires.

Jean-Cédric Chappelier

normes C89, C99 et C11

Exemple :

C89 (et autres)

```c
/* declaration/init.
   des variables      */
int i = 3;
int j = 0;

/* traitements */
...
i = i + 33;
...
scanf("%d", &j);
i = i * j;
...
```

C99 ou ultérieurs

```c
/* declaration/init. des
   variables ET traitements */
int i = 3;
...
i = i + 33;
...
// ici une autre declaration
int j = 0;
scanf("%d", &j);
i = i * j;
...
```

compilation C89 : `gcc -ansi -pedantic -Wall ...`
compilation C99 : `gcc -std=c99 ...`
compilation C11 : `gcc -std=c17 ...` (oui : préférez C17 qui est une révision « bug-fix » de C11)

☞ plus d'infos sur le site Web du cours

Données et traitements

différent de Java !

Comme dans tout langage de programmation évolué, on a en C la possibilité de définir des **traitements** mis en œuvre sur des **données**.

- variables (données)
- instructions et expressions (traitements)
- structures de contrôle (ordonnancement des traitements)

Par contre, à la différence de la POO, données et traitements sont clairement séparés (et non pas regroupés par « concepts » comme en POO [*encaspulation*])

Il n'y a pas non plus de *masquaque* (ou *abstration*, *data hiding*) à proprement parler : moins claire séparation entre spécification (API) et implémentation.

Et il n'y a évidemment ni *héritage*, ni *polymorphisme* (d'aucune sorte, pas même de *surcharge*) !

Jean-Cédric Chappelier

Variables : définition

comme en Java

Une variable possède 3 caractéristiques :

- son **identificateur** qui est le *nom* par lequel la donnée est désignée.
  - ☞ n'importe quelle séquence composée de lettres, de chiffres ou du caractère '_', commençant par une lettre ou par '_', et ne correspondant pas à un mot réservé du langage.

  Exemples : `b`, `delta`, `MyWindow`, ...
- son **type** qui définit de quel « genre » est la donnée associée à la variable, en particulier, quels traitements elle peut (et ne peut pas) subir.
  Exemples : double, int, char*, struct complexe,...
- sa **valeur**.

Conseil(s) : Utilisez des noms **aussi explicites que possible**.
Et gardez les mêmes conventions (casse, soulignés) pour le choix des noms.

Déclaration et initialisation de variables

presque comme en Java

En C, une variable doit être **déclarée avant d'être utilisée**,
(en C89 : + en tête du bloc qui la concerne)

La syntaxe de la déclaration d'une variable est :

type identificateur ;

Exemples :
```
int val;
double delta;
```

Les principaux **types élémentaires** définis en C sont :

`int`	:	les nombres entiers
`double`	:	les nombres réels (approchés)
`char`	:	les caractères

Notes :

1. nous verrons plus tard d'autres types : les types composés, le type énuméré et les types synonymes.
2. en C, il **n'**y a **pas** de type « chaîne de caractères » (string).
   En C89 il n'y a pas de booléen. (depuis C99 : type `bool` dans `stdbool.h`)

Initialisation

différent de Java !

En même temps qu'elle est déclarée, une variable peut être **initialisée** (on lui donne une première valeur avant même toute utilisation)

Attention ! Il est possible d'utiliser une variable non initialisée. Ceci doit au maximum **être évité !**

Initialisez toujours vos variables... ...cela vous évitera bien des soucis par la suite (comportement imprévu).

Contrairement à Java, les variables **ne sont pas** initialisées (ni à 0, ni à rien d'autre).

La syntaxe de déclaration/initialisation d'une variable est :

type identificateur = valeur_d'initialisation ;

où *valeur_d'initialisation* est n'importe quelle constante (c'est-à-direvaleur littérale) ou expression du type indiqué.

Exemples :

```
int val = 2;
double pi = 3.1415;
char c = 'a';
int j = 2*i+5;
```

Valeurs littérales

comme en Java

- valeurs littérales de type entier : `1, 12, ...`
- valeurs littérales de type réel : `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 caractère : `'a', '!', ...`
  Remarque :
  le caractère `'` se représente par `\'`
  le caractère `\` se représente par `\\`
  le caractère nul se représente par `\0`
  le retour à la ligne se représente par `\n`

Données modifiables/non modifiables

différent de Java !

Par défaut les variables en C sont modifiables.
Mais on peut vouloir imposer que certaines « variables » ne puissent pas modifier leur contenu : définir des **accès sans modification** ($\simeq$ « constantes »).

La nature *modifiable* ou *non modifiable* de l'**accès** à une donnée peut être définie lors de la déclaration par l'indication du mot réservé `const`.

Cette donnée *ne pourra pas être modifiée via ce nom de variable* (toute tentative de modification par ce nom de variable produira un message d'erreur lors de la compilation) : `const` veut en faire simplement dire « **read only** ».

A noter que cela n'assure **pas** l'invariabilité absolue de la donnée elle-même (c'est-à-dire zone mémoire), qui pourrait être modifiée par ailleurs, p.ex. via un pointeur (ou une source extérieure, p.ex. capteur).

Exemples :

```
int const couple = 2;
double const g = 9.81;
double const pi = 3.14159265358979323846;
```

Affectation

très différent de Java !

En C, la syntaxe d'une affectation est :

identificateur = *valeur* ;

où *valeur* est une constante ou une **expression** du même type que la variable référencée par *identificateur*.

Exemple : `i = 3 ;`

Attention ! La sémantique de l'opérateur = est **TRÈS DIFFÉRENTE** entre C et Java !

Jean-Cédric Chappelier

Sémantique de l'opérateur =

très différent de Java !

	Java	C

	`Objet a = new Objet();`
	`Objet b;`
	`b = a;`
	`b.modification();`
		`Type a = une_valeur;`
		`Type b;`
		`b = a;`
		`modification(&b);`

`a` est-il modifié ?		

En C, l'opérateur = **modifie le contenu** de son premier opérande (à gauche) : **sémantique de valeur**
En Java, cela ne fait que créer une référence de plus sur son second opérande (celui de droite) : sémantique de référence

La sémantique est donc *très* différente !

Pour faire simple, « `a=b;` » en C correspond plutôt à « `a=b.clone();` » en Java (je ne parle pas ici des types natifs).

Jean-Cédric Chappelier

Variables

En C, une donnée est stockée dans une variable caractérisée par :

- son **type** et son **identificateur** (définis lors de la **déclaration**) ;
- sa **valeur**, définie la première fois lors de l'**initialisation** puis éventuellement modifiée par la suite.

Rappels de syntaxe :

```
type id ;
type id = valeur;

id = expression ;
```

Types élémentaires :

```
int
double
char
```

Exemples :

```
int val = 2 ;
double const pi = 3.141592653;
i=j+3;
```

Les variables non modifiables se déclarent avec le mot réservé `const` :

```
double const g = 9.81;
```

Opérateurs et expressions

comme en Java

Tout langage de programmation fournit des **opérateurs** permettant de manipuler les objets prédéfinis.

Exemple : nous avons déjà précédemment rencontré un opérateur : `=`, l'opérateur d'affectation (qui est universel : s'applique à tout type).

Les **expressions** sont des séquences (« *bien formées* » au sens de la syntaxe) combinant des opérateurs et des arguments (variables ou valeurs).

Exemple d'expression numérique : `(2*(13-i)/(1+4))`

L'évaluation d'une expression conduit (naturellement) à sa valeur.

Exemple : l'évaluation de l'expression `(2*(13-3)/(1+4))` correspond à la valeur `4`

Jean-Cédric Chappelier

Opérateurs arithmétiques

presque comme en Java

Les opérateurs arithmétiques sont :

`*`	multiplication
`/`	division
`%`	modulo
`+`	addition
`-`	soustraction
`-`	opposé

(Remarque : le modulo est le reste de la division entière. Il est du signe de son premier opérande.)

En C, on **ne peut pas** concaténer des chaînes de caractères avec `+`

(Remarque : opérateur unaire ici)

Exemples :

```
z = (x + 3) % y;
z = (3 * x + y) / 10;
```

C fournit un certain nombre de **notations abrégées** pour des affectations particulières :

`x = x + y` peut aussi s'écrire `x += y`
(idem pour `-`, `*`, `/` et `%`)

`x = x + 1` peut aussi s'écrire `++x`
(idem pour `-` : `--x`)

Jean-Cédric Chappelier

ATTENTION PIÈGE !

comme en Java

Remarque sur l'opérateur de division en C :

- si `a` et `b` sont des *entiers*, `a/b` est le quotient de la division **entière** de `a` par `b`
  Exemple : `5/2 = 2`
  (et `a%b` est le reste de la division entière de `a` par `b`
  Exemple : `5%2 = 1`)

- si `a` ou `b` sont des *réels*, `a/b` est le résultat de la division réelle de `a` par `b`
  Exemple : `5.0/2.0 = 2.5`

Note : dans une expression constante, on distingue un réel d'un entier en lui ajoutant `.` à la fin. En général pour la lisibilité on préfère ajouter `.0` :

`5.0` (réel) $\longleftrightarrow$ `5` (entier)

Ici, il y a un point !..

Jean-Cédric Chappelier

Remarque sur `++x` et `x++`

Il existe deux opérateurs `++` : l'un préfixé et l'autre sufixé :

expression	FAIT :	VAUT :
`++x`	incrémente `x`	la valeur de `x` après évaluation
`x++`	incrémente `x`	la valeur de `x` avant évaluation

En C, la seule différence a donc lieu si l'on utilise la valeur de ces expressions...
...ce que je **déconseille fortement** !
(Écrivez du code simple, facilement compréhensible par tous.
☞ mettez plutôt les incrémentations sur une ligne séparée).

PAR CONTRE dans des langages où ces opérateurs peuvent s'appliquer à des *objets* (p.ex. **en C++**) il y a également une autre différence majeure :
l'opérateur suffixé (`x++`) nécessite de faire plus de choses (soit une copie, soit une soustraction) que l'opérateur préfixé.

Il est donc, dans ces cas, recommandé de préférer l'opérateur préfixé (`++x`).

Jean-Cédric Chappelier

Priorité entre Opérateurs

comme en Java

Il est largement préférable de parenthéser ses expressions (ne serait-ce que pour la lisibilité !).
Par exemple écrire `(a * b) % c` plutôt que `a * b % c`

En l'absence de parenthésage, l'évaluation se fait dans l'ordre suivant des opérateurs :

`*` ou `/` ou `%`
puis `+` ou `-`

Tous ces opérateurs sont associatifs à gauche : `a+b+c=(a+b)+c`

En cas d'ambiguïté entre opérateurs du même ordre de priorité, c'est la règle d'associativité qui s'applique.

Exemples :

```
a * b % c  =  (a * b) % c
a % b * c  =  (a % b) * c
a + b * c % d  = a + ((b * c) % d)
```


EPFL

Opérateurs de comparaison

comme en Java

Les opérateurs de comparaison sont :

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

Leur résultat est du même type que les arguments (avec priorité au `double` en cas de mélange `int`/`double`).

Exemples :
```
x >= y
x+y == 4
```

ATTENTION PIÈGE !

différent de Java !

Ne pas confondre l'opérateur de test d'égalité `==` et l'opérateur d'affectation `=` !

`x = 3` : affecte la valeur 3 à la variable `x`
(et donc modifie cette dernière)

`x == 3` : teste la valeur de la variable `x`, renvoie « vrai » si elle vaut 3 et « faux » sinon
(et donc ne modifie pas la valeur de `x`)

Cependant avec les convention de représentation des valeurs logique,
« `if (x = 3)` » est **tout à fait accepté par le compilateur** !!!
(☞ question : s'interprète comment ?)

FAITES TRÈS ATTENTION À CETTE ERREUR CLASSIQUE !

Conseil : écrivez `3 == x` plutôt que `x == 3`.

Jean-Cédric Chappelier

Expressions logiques

différent de Java !

Une expression logique est une expression représentant les valeurs de vérité logique « vraie » ou « faux ».

Cependant, en C, n'importe quelle expression de **n'importe quel type** peut être considéré comme une expression logique.

Il n'y a pas de type « valeur logique »

C utilise par contre la convention suivante :
Si l'évaluation de l'expression conditionnelle est une *valeur nulle*, alors la condition sera dite **fausse**, sinon elle sera dite **vraie**.

Exemples d'expressions vraies	Exemples d'expressions fausses
`1 || 0`	`1 && 0`
`2`	`0.0`
`0.5 + 0.33`	`16 % 2`

Conseil : évitez d'utiliser cette possibilité du langage et préférez écrire explicitement vos expressions logiques. Par exemple, écrivez `if (x != 0)` plutôt que `if (x)`.

Opérateurs logiques

comme en Java

Les opérateurs logiques sont :

`&&`	« et »
`||`	« ou »
`!`	négation

*(Remarque : cet opérateur n'a qu'**un seul** opérande)*

Exemples :

```
((z != 0.0) && (2*(x-y)/z < 3.0))

((i >= 0) || ((x*y > 0.0) && !(j == 2)))
```

Opérateurs logiques (2)

comme en Java

Les opérateurs logiques `&&`, `||` et `!` sont définis par les tables de vérité usuelles :

`x`	`y`	`!x`	`x && y`	`x || y`
vrai	vrai	faux	vrai	vrai
vrai	faux	faux	faux	vrai
faux	vrai	vrai	faux	vrai
faux	faux	vrai	faux	faux

Jean-Cédric Chappelier

Évaluation « paresseuse »

comme en Java

Les opérateurs logiques `&&` et `||` effectuent une **évaluation « paresseuse »** (« *lazy evaluation* ») de leur arguments :

l'évaluation des arguments se fait de la gauche vers la droite et seuls les arguments strictement nécessaires à la détermination de la valeur logique sont évalués.

Ainsi, dans `X1 && X2 && ... && Xn`, les arguments `Xi` ne sont évalués que *jusqu'au 1er argument faux* (s'il existe, auquel cas l'expression est fausse, sinon l'expression est vraie) ;

Exemple : dans `(x != 0.0) && (3.0/x > 12.0)` le second terme ne sera effectivement évalué uniquement si `x` est non nul. La division par `x` ne sera donc jamais erronée.

Et dans `X1 || X2 || ... || Xn`, les arguments ne sont évalués que *jusqu'au 1er argument vrai* (s'il existe, auquel cas l'expression est vraie, sinon l'expression est fausse).

Exemple : dans `(x == 0.0) || (3.0/x < 12.0)` le second terme ne sera effectivement évalué uniquement si `x` est non nul.

Opérateur « , »

différent de Java !

Il existe en C l'opérateur binaire « `,` » qui :

- évalue ses deux opérandes ;
- vaut la valeur de l'opérande de droite.

Ainsi `E1,E2` (pour deux expressions `E1` et `E2`), évalue d'abord `E1` puis `E2`, et vaut `E2`.

Exemples :

`x=(3,4);` ☞ `x` vaut 4

`x=(a=b,3*a);` ☞ `a=b; x=3*a;`

Conseil : Ne pas l'utiliser !

(Attention ! « `x=3,4;` » est interprété comme «`(x=3),4;`», c'est-à-dire `x` vaut `3`.)

Opérateurs bit-à-bit

comme en Java

Opérateurs de manipulation du contenu binaire :

`~`	inversion bit-à-bit
`&`	« et » bit-à-bit
`|`	« ou » bit-à-bit
`^`	« ou exclusif » bit-à-bit
`<<`	décalage à gauche
`>>`	décalage à droite

*(Remarque : cet opérateur n'a qu'**un seul** opérande)*

Attention ! Ne pas confondre les opérateurs logiques et les opérateurs bit-à-bit. Par exemple : `&&` et `&` :
`1 && 2` s'évalue comme « vrai »,
`1 & 2` *vaut* `0`, qui s'évalue comme « faux » !

Exemple :

```
char a = 6;       // 00000110 en binaire
char b = a << 2; // 00011000, soit 24
char c = a >> 1; // 00000011, soit  3
char d = a | c ; // 00000111, soit  7
char e = a ^ c ; // 00000101, soit  5
char f = a & c ; // 00000010, soit  2
unsigned char g = ~a ; // 11111001, soit 249
```


Jean-Cédric Chappelier

Opérateurs

Operateurs arithmétiques

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

Operateurs de comparaison

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

Operateurs logiques

`&&`	"et" logique	
`||`	ou	
`!`	négation	*(1 opérande)*

Priorités (par ordre décroissant, tous les opérateurs d'un même groupe sont de priorité égale) :
`() [] -> .`, `! ++ --`, `* / %`, `+ -`, `< <= > >=`, `== !=`, `&&`, `||`, `= += -=` etc., `,`

Jean-Cédric Chappelier
EPFL

[hors cours] Monsieur, et C++... ? [hors cours]

① Tout d'abord **C++ n'est pas au programme de ce cours** (ce n'est pas l'objectif). Je ne parlerais donc pas de C++.
Néanmoins, j'ajouterai des transparents (hors cours) pour attirer l'attention sur divers pièges, à l'attention de ceux qui souhaitent apprendre par eux-mêmes ce langage.

② Il y a beaucoup de différences, certaines subtiles, entre C++ et Java.

> *While Java borrows a lot of terminology and even syntax from C++, the analogies between Java and C++ are not nearly as strong as those between Java and C. C++ programmers should be careful not to be lulled into a false sense of familiarity with Java just because the languages share a number of keywords !*
>
> [D. Flanagan, *Java in a Nutshell*, O'Reilly, 1990]

③ C++, qui est un langage riche et puissant (plusieurs paradigmes de programmation, en fait !), devrait faire l'objet d'un apprentissage sérieux, à part entière.

Ce que j'ai appris aujourd'hui

- Administration du cours :
  - de quoi est constitué ce cours
  - comment il va se dérouler
  - et comment je vais être évalué(e)
- Les bases du langage C
  - variables
  - opérateurs et expressions

  et ses différences avec Java.

A NE PAS RATER cette semaine :

- infos administratives (**30 mars**, **25 mai**, `https://moodle.epfl.ch/course/view.php?id=6731`, `https://progos.epfl.ch`)
- bien travailler, DÈS AUJOURD'HUI, les bases de C en dépit de leur apparente similitude avec Java...
  ☞ slide le plus important : 13/47