

Information, Calcul, Communication : INTRODUCTION GÉNÉRALE DU COURS

Jean-Cédric Chappelier

Faculté I&C

Objectifs du cours ICC

Ce cours « Information, Calcul et Communication » a pour buts essentiels de :

1. présenter l'Informatique en tant que discipline scientifique
 2. exposer ses principes fondamentaux (partie théorie, vendredis)
 3. développer la « **pensée algorithmique** » (« Computational Thinking »)
 4. expliquer les bases de fonctionnement du « monde numérique » (partie théorie, vendredis)
 5. sensibiliser à la sécurité dans ce « monde numérique » (partie théorie, vendredis)
 6. vous apprendre à **programmer** :
savoir les bases et connaître correctement au moins un langage (ici : le C++)
 7. Savoir comment fonctionne un ordinateur
et savoir l'utiliser (sous Linux)
- 👉 voir le document « *Présentation générale du cours* » envoyé par email
(et présent sur le site Moodle du cours)

À LIRE ABSOLUMENT !

Moyens

- ▶ Moodle : support de cours, exercices, compléments, ...
<https://moodle.epfl.ch/course/view.php?id=14023>

- ▶ partie programmation : **MOOC** (Courseware) : vidéos, quiz, exercices, devoirs notés

<https://courseware.epfl.ch/courses/course-v1:EPFL+init-prog-cpp+2024/about>

- ▶ Interactions :
 - ▶ partie programmation (jeudi) : exercices (2h), puis *compléments* de cours
 - ▶ partie théorie (vendredi) : *compléments interactifs* de cours (2h), puis exercices (1h)
- ▶ Forums (sur le MOOC et sur Ed Discussion (via Moodle))

Organisation du travail (semaines)

- ▶ **AVANT** le cours de programmation (jeudi) : voir les vidéos du MOOC
- ▶ si possible avant le cours : faire les quiz et commencer des exercices libres
- ▶ jeudi 8¹⁵–10⁰⁰ : séance d'exercices avec assistant(e)s
- ▶ en cours (jeudi 10¹⁵–11⁰⁰) : rappels, approfondissements, questions
- ▶ après le cours : encore *plus* d'exercices ; puis faire et soumettre les « exercices de programmation » du MOOC (correcteur automatique)
- ▶ **AVANT** le cours de théorie (vendredi) : voir la vidéo du cours
- ▶ compléments de cours, partie théorie (vendredi 13¹⁵–15⁰⁰)
- ▶ exercices, partie théorie (vendredi 15¹⁵–16⁰⁰)
- ▶ vendredi 16¹⁵–18⁰⁰ : séances « d'appui » optionnelles (venue libre)
- 👉 voir le document « *Présentation générale du cours* » mis à disposition.

À LIRE ABSOLUMENT !

Organisation du travail (semestre)

	MOOC	décalage / MOOC	exercices prog. 1h45 Jeudi 8-10	cours prog. 45 min. Jeudi 10-11	cours théorie 90 min. Vendredi 13-14 Vendredi 14-15		exercices théorie 45 min. Vendredi 15-16	
1 10.09.26	--	-1	prise en main	Bienvenue/Introduction	Introduction + Algo 1		Algo 1	11.09.26
2 17.09.26	1. variables	0	variables / expressions	variables / expressions	Algorithmes 1 (suite)		Algo 1 suite	18.09.26
3 24.09.26	2. if	0	if – switch	if – switch	Algo 1	Algo 2 (stratégies)	Algo 2	25.09.26
4 01.10.26	3. for/while	0	for / while	for / while	Algo 2 (stratégies)	Calculabilité	Calculabilité	02.10.26
5 08.10.26	4. fonctions	0	fonctions (1)	fonctions (1)	Calculabilité	Représentations numériques	Représentations numériques	09.10.26
6 15.10.26		1	fonctions (2)	fonctions (2)	Représentations numériques	Signaux + Filtrage	Révisions	16.10.26
- 22.10.26					Examen 1 (1h45)			30.10.26
7 29.10.26	5. tableaux (vector)	1	vector	vector	Th. d'échantillonnage		Signaux–Echantillonnage	06.11.26
8 05.11.26	6. string + struct	1	array / string	array / string	Signaux–Echantillonnage	Compression 1	Compression 1	13.11.26
9 12.11.26		2	structures	structures	Compression 1	Compression 2	Compression 2	20.11.26
10 19.11.26	7. pointeurs	2	pointeurs	pointeurs	Compression 2	Architecture des ordinateurs	Architecture des ordinateurs	27.11.26
11 26.11.26		-	entrées/sorties	entrées/sorties	Architecture des ordinateurs	Stockage/Réseaux	Stockage/Réseaux	04.12.26
12 03.12.26		-	erreurs / exceptions	erreurs / exceptions	Stockage/Réseaux	Sécurité	Révisions	11.12.26
13 10.12.26		-	révisions	théorie : sécurité	Examen final (2h45)			18.12.26
14 17.12.26	8. étude de cas	-	révisions	Révisions	(prép. examen)		(« classe inversée » : rép. questions + compléments)	
				(ne sont pas sur le MOOC)				

👉 voir le document « *Présentation générale du cours* » mis à disposition.

À LIRE ABSOLUMENT !

Interaction avec l'enseignant et les assistant(e)s

Plusieurs moyens pour contacter l'enseignant, les assistants et étudiant(e)s-assistant(e)s 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 des forums du cours (dans site Moodle [spécifique] ou MOOC [général])
 - ☞ moyen idéal pour diffuser la connaissance

N'hésitez pas à en faire usage !

Les contacts personnels avec l'enseignant (email, téléphone ou visites) devront être
strictement réservés aux cas personnels et/ou urgents !

Livres ?

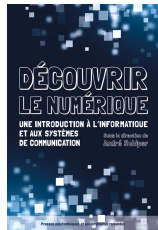
Les éléments fournis (sur le MOOC ou Moodle) devraient constituer une **documentation suffisante** pour ce cours !

Pour ceux qui souhaitent avoir un livre, les ouvrages suivants sont également recommandés.

Partie théorie :

A. Schiper (éditeur)
Découvrir le numérique, EPFL Press, 2^e édition,
2018.

Mais la première édition reste *utilisable* pour le cours
de cette année.



Disponible pour un prix avoisinant les 35 CHF.
Une version électronique est également disponible.

Livres ?

Les éléments fournis (sur le MOOC ou Moodle) devraient constituer une **documentation suffisante** pour ce cours !

Pour ceux qui souhaitent avoir un livre, les ouvrages suivants sont également recommandés.

Partie programmation :

Jean-Cédric Chappelier, Jamila Sam & Vincent Lepetit, *Initiation à la programmation en C++*, EPFL Press, 2016.



eBook téléchargeable gratuitement chez EPFL Press.
Une version électronique est également disponible.

Livres ?

Les éléments fournis (sur le MOOC ou Moodle) devraient constituer une **documentation suffisante** pour ce cours !

Pour ceux qui souhaitent avoir un livre, les ouvrages suivants sont également recommandés.

Partie programmation :

J.-C. Chappelier & F. Seydoux
*C++ par la pratique –
recueil d'exercices corrigés et aide-mémoire*,
EPFL Press, **4^e édition, 2017**.

Mais les anciennes éditions restent *utilisables* pour le
cours de cette année.



Disponible pour un prix avoisinant les 35 CHF.
Une version électronique est également disponible.

Notes et examens (1)

4 évaluations pendant le semestre :

- ▶ examen 1 : **vendredi 30 octobre, 13h15–15h00**
- ▶ « micro-projet » (homework) : du **vendredi 13 novembre** au mercredi 25 novembre,
travail de programmation « à la maison »
- ▶ examen 2 : **vendredi 18 décembre, 13h15–16h00**
- ▶ quiz sur la sécurité informatique : à terminer avant le vendredi 18 décembre 23h59

Format des deux examens : 1h45 ou 2h45, sur papier, tous documents autorisés (mais aucun matériel électronique)

Contenu :

- ▶ examen 1 : théorie module 1 + programmation jusqu'aux « *fonctions* »
- ▶ examen 2 : **tout** le cours (programmation et théorie)

Sensibilisation à la cyber-sécurité

L'École tient à sensibiliser tous les étudiant(e)s aux cyber-risques, notamment pour protéger le campus contre les cyber-attaques.

Moyens :

- ▶ 13 petites vidéos (2–3 min.) à regarder pendant le semestre
 - 👉 le lien est donné en bas de la page **Moodle**, dans la section « **Sensibilisation à la cyber-sécurité** »
- ▶ un quiz à passer (avant la fin du semestre)
 - ▶ 2 tentatives
 - ▶ 5% de la note finale

Notes et examens (2)

La note finale calculée de la façon suivante :

- ▶ examen 1 : $\theta_1 = 30\%$
- ▶ « micro-projet » (homework) : $\theta_2 = 10\%$
- ▶ examen final : $\theta_3 = 55\%$
- ▶ quiz « sécurité » : $\theta_4 = 5\%$

$$N = 1 + 0.25 \left[20 \cdot \frac{\sum_x \theta_x (p_x / t_x)}{\sum_x \theta_x} \right]$$

(p_x le nombre de points obtenus à l'examen x , 0 en cas d'absence, sur un total maximal de points pour cet examen de t_x)

👉 voir le document « *Présentation générale du cours* ».

Les **exercices de « programmation notée » du MOOC** :

- ▶ **sont obligatoires**
- ▶ sont un très bon entraînement
- ▶ n'entrent pas dans le calcul de la note EPFL.

Qu'est-ce que la programmation ?

Objectif : permettre l'**automatisation** d'un certain nombre de tâches à l'aide d'**ordinateurs**.

Un ordinateur est un exemple d'**automate programmable**.

Un **automate** est un dispositif capable d'assurer, sans intervention humaine, un enchaînement d'opérations correspondant à la réalisation d'une tâche donnée.

Exemples : **montre**, « **ramasse-quilles** », ...

Un automate est **programmable** lorsque la nature de la **tâche** qu'il est capable de réaliser peut être **modifiée** à volonté. Dans ce cas, la description de la tâche à réaliser se fait par le biais d'un **programme**, c.-à-d. une séquence d'instructions et de données susceptibles d'être traitées (i.e. « *comprises* » et « *exécutées* ») par l'automate.

Exemples : **le métier à tisser Jacquard**, **l'orgue de barbarie**, ...
... et **l'ordinateur** !

Exemple d'automate programmable



PROGRAMME :

Conception : quelles
notes enchaîner ?

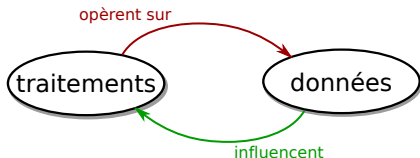
Réalisation : percer les
trous aux bons endroits

Exécution : tourner la
manivelle

Résultat : mélodie

Qu'est-ce que la programmation ? (résumé)

En résumé, programmer c'est donc **décomposer** la **tâche** à automatiser sous la forme d'une **séquence d'instructions** (**traitements**) et de **données** adaptées à l'automate utilisé.



Formalisation des **traitements** : **algorithmes**

- ☞ distinguer formellement les bons traitements des mauvais

Formalisation des **données** : **structures de données abstraites**

- ☞ distinguer formellement les bonnes structures de données des mauvaises

Les instructions de l'ordinateur

Concrètement, quelles sont les instructions et les données « adaptées » à l'ordinateur ?

Ordinateur $\simeq$

microprocesseur

détermine l'ensemble des instructions élémentaires que l'ordinateur est capable d'exécuter ;

mémoire centrale

détermine l'espace dans lequel des données peuvent être stockées en cours de traitement

périphériques

permettent l'échange ou la sauvegarde à long terme des données

Architecture de Von Neumann (1955)

Les instructions de l'ordinateur

Concrètement, quelles sont les instructions et les données « adaptées » à l'ordinateur ?

Ordinateur $\simeq$

microprocesseur

mémoire centrale

périphériques

☞ C'est donc **le microprocesseur** qui **détermine le « jeu d'instructions »** (et le type de données) à utiliser.

On les appelle « Instructions Machine », « **Langage Machine** », $\simeq$ « Assembleur »

On peut programmer directement le microprocesseur en langage machine...
...mais c'est un peu fastidieux et de plus, chaque processeur utilise ses propres instructions



Exemple d'instructions-machine



Programme en Assembleur

```

1: LOAD 10 5
2: CMP 10 0
3: JUMP +3
4: DECR 10
5: JUMP -3
6: END
  
```

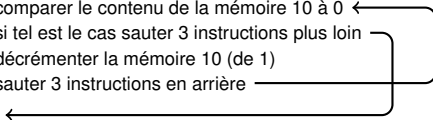
mettre 5 dans la mémoire 10

comparer le contenu de la mémoire 10 à 0

si tel est le cas sauter 3 instructions plus loin

décrémenter la mémoire 10 (de 1)

sauter 3 instructions en arrière



Instructions machine :

Instructions	Code Machine	données	Code Machine
CMP	00000000	-3	10000011
DECR	00000001	0	00000000
END	00000010	2	00000010
JUMP	00000011	3	00000011
LOAD	00000100	5	00000101
		6	00000110
		10	00001010

Le programme ci-dessus correspond donc physiquement en machine à la séquence :

00000100000010100000001010000000000000101000000000000001100000011

0000000100001010000000111000001100000010

La notion de langage de programmation

Cependant, ces instructions-machine sont **trop élémentaires** pour pouvoir être efficacement utilisées (par les humains) pour l'écriture de programmes...

... il faut donc fournir au programmeur la possibilité d'**utiliser des instructions de plus haut niveau**, plus proches de notre manière de penser et de conceptualiser les problèmes ...

Exemples de langage de programmation de haut niveau :

« <i>vieux</i> » BASIC	C
<pre>1 N=5 2 IF (N>0) THEN PRINT N; N=N-1; GOTO 2 3 END</pre>	<pre>int main() { for (int n=5; n>0; --n) printf("%d\n", n); return 0; }</pre>

La notion de langage de programmation (2)

Comment rendre les instructions plus sophistiquées compréhensibles par l'ordinateur ?

👉 **traduire** les séquences d'instructions de haut niveau en instructions-machine directement exécutables par le microprocesseur

Selon ses caractéristiques, un tel traducteur est appelé **compilateur** ou **interpréteur**

L'ensemble des instructions de plus haut niveau qu'un compilateur ou un interpréteur est capable de traiter constitue un **langage de programmation**.

Interpréteur / Compilateur

Compilateur et **Interpréteur** sont des traducteurs de langage de programmation de haut niveau en séries d'instructions-machine directement exécutables par l'ordinateur.

La différence réside dans la manière dont la traduction est réalisée :

- ▶ le **compilateur** traduit les programmes *dans leur ensemble* : tout le programme doit être fourni en bloc au compilateur pour la traduction. Il est traduit *une seule fois*.
- ▶ l'**interpréteur** traduit les programmes *instruction par instruction* dans le cadre d'une interaction continue avec l'utilisateur. Un programme est traduit à *chaque exécution*.

Remarques : Certains langages peuvent indifféremment être interprétés ou compilés

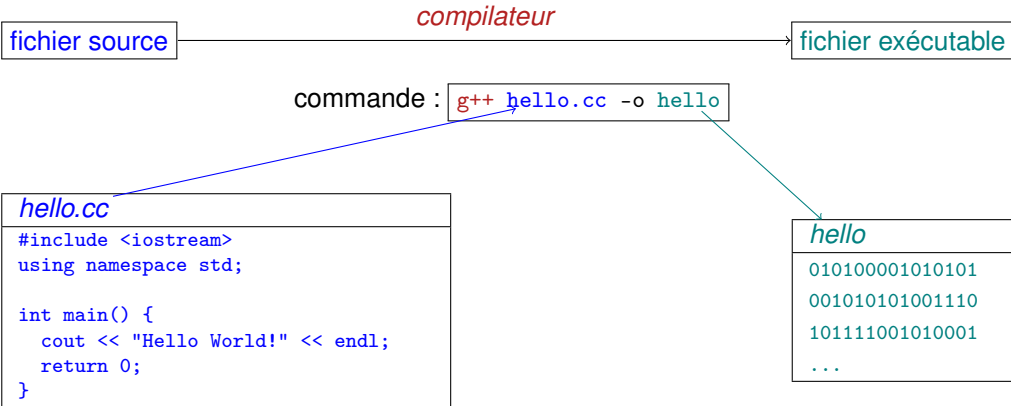
C++ est un langage compilé, jamais interprété

Interpréteur / Compilateur (2)

Avantages et Inconvénients :

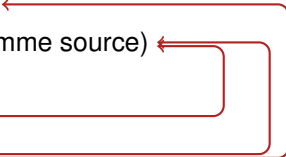
- ▶ De manière générale un langage **interprété** est donc adapté au *développement rapide de prototypes*
(on peut immédiatement tester ce que l'on est en train de réaliser)
- ▶ un langage **compilé** permet la *réalisation d'applications plus efficaces ou de plus grande envergure*
(optimisation plus globale, traduction effectuée une seule fois et non pas à chaque utilisation)
- ▶ 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

Compilation d'un programme C++



Pour résumer

Concrètement : *Cycle de développement*

- ① réfléchir au problème ; **concevoir l'algorithme** ←
 - ② traduire cette réflexion en un *texte* C++ (programme source) ←
 - ③ traduire ce texte C++ en langage machine
(compilation, programme exécutable) —
 - ④ exécution du programme (exécutable) ==
- 

En pratique :

- ▶ erreurs de compilation (mal écrit)
- ▶ erreurs d'exécution (mal pensé)

⇒ **correction(s)**

☞ d'où le(s) **cycle(s)** !

Ce que j'ai appris aujourd'hui

- ▶ Rappels sur l'organisation du cours :
lire le document de présentation
- ▶ Ce qu'est et à quoi sert un **langage de programmation**
- ▶ La différence entre langage **interprété** et **compilé**
- ▶ que **traitements** et **données** sont les deux facettes complémentaires de la programmation

La suite

- ▶ Demain : cours et exercices de la partie théorie
 - 👉 pensez à regarder les vidéos avant : Introduction (4 vidéos) et I.1.1 à I.1.4
- ▶ **vous inscrire et commencer sur le MOOC** (Courseware) pour le prochain cours de programmation
- ▶ Le prochain cours de programmation : révision et compléments sur
 - ▶ variables
 - ▶ expressions

	MOOC	décalage / MOOC	exercices prog. 1h45 Jeudi 8-10	cours prog. 45 min. Jeudi 10-11	cours théorie 90 min.		exercices théorie 45 min. Vendredi 15-16	
					Vendredi 13-14	Vendredi 14-15		
1 10.09.26	--	-1	prise en main	Bienvenue/Introduction	Introduction + Algo 1		Algo 1	11.09.26
2 17.09.26	1. variables	0	variables / expressions	variables / expressions	Algorithmes 1 (suite)		Algo 1 suite	18.09.26
3 24.09.26	2. if	0	if – switch	if – switch	Algo 1	Algo 2 (stratégies)	Algo 2	25.09.26
4 01.10.26	3. for/while	0	for / while	for / while	Algo 2 (stratégies)	Calculabilité	Calculabilité	02.10.26
5 08.10.26	4. fonctions	0	fonctions (1)	fonctions (1)	Calculabilité	Représentations numériques	Représentations numériques	09.10.26
6 15.10.26		1	fonctions (2)	fonctions (2)	Représentations numériques	Signaux + Filtrage	Révisions	16.10.26
- 22.10.26					Examen 1 (1h45)			30.10.26
7 29.10.26	5. tableaux (vector)	1	vector	vector	Th. d'échantillonnage		Signaux–Echantillonnage	06.11.26
8 05.11.26	6. string + struct	1	array / string	array / string	Signaux–Echantillonnage	Compression 1	Compression 1	13.11.26
9 12.11.26		2	structures	structures	Compression 1	Compression 2	Compression 2	20.11.26
10 19.11.26	7. pointeurs	2	pointeurs	pointeurs	Compression 2	Architecture des ordinateurs	Architecture des ordinateurs	27.11.26
11 26.11.26		-	entrées/sorties	entrées/sorties	Architecture des ordinateurs	Stockage/Réseaux	Stockage/Réseaux	04.12.26
12 03.12.26		-	erreurs / exceptions	erreurs / exceptions	Stockage/Réseaux	Sécurité	Révisions	11.12.26
13 10.12.26		-	révisions	théorie : sécurité	Examen final (2h45)			18.12.26
14 17.12.26	8. étude de cas	-	révisions	Révisions	(prép. examen)		(« classe inversée » : rép. questions + compléments)	
				(ne sont pas sur le MOOC)				