

Information, Calcul, Communication (partie programmation) : Exceptions / Synthèse

Jean-Cédric Chappelier

Faculté I&C

Rappel du calendrier

	MOOC	décalage / MOOC	exercices prog. 1h45 Jeudi 8-10	cours prog. 45 min. Jeudi 10-11
1 10.09.26	--	-1	prise en main	Bienvenue/Introduction
2 17.09.26	1. variables	0	variables / expressions	variables / expressions
3 24.09.26	2. if	0	if – switch	if – switch
4 01.10.26	3. for/while	0	for / while	for / while
5 08.10.26	4. fonctions	0	fonctions (1)	fonctions (1)
6 15.10.26		1	fonctions (2)	fonctions (2)
- 22.10.26				
7 29.10.26	5. tableaux (vector)	1	vector	vector
8 05.11.26	6. string + struct	1	array / string	array / string
9 12.11.26		2	structures	structures
10 19.11.26	7. pointeurs	2	pointeurs	pointeurs
11 26.11.26		-	entrées/sorties	entrées/sorties
12 03.12.26		-	erreurs / exceptions	erreurs / exceptions
13 10.12.26		-	révisions	théorie : sécurité
14 17.12.26	8. étude de cas	-	révisions	Révisions

Objectifs du cours d'aujourd'hui

- ▶ Rappels sur les exceptions
- ▶ Synthèse de la partie programmation du cours
- ▶ Étude(s) de cas (ou questions ?) sur les exceptions ?

Exceptions

- ▶ anticiper des erreurs d'utilisation d'une portion de programme (typiquement d'une fonction)
- ▶ principe de base :
prévoir une erreur à un endroit et de la gérer à un **autre** endroit
- ▶ `throw` d'un côté
`try/catch` de l'autre
- ▶ bien comprendre les déroulements possibles
- ▶ ne pas en abuser (utiliser à bon escient)

Qu'avons nous vu en programmation ?

Programmer, c'est **décomposer** une **tâche** à automatiser
en une **séquence d'instructions** (**traitements**) et des **données**

Algorithme	S.D.A.
Traitements	Données
Expressions & Opérateurs Structures de contrôle Fonctions	Variables Portée Chaînes de caractères Tableaux statiques Tableaux dynamiques Structures Pointeurs Entrées/Sorties Exceptions

👉 Il faut que vous **soyez au clair**
sur chacune des notions
ci-contre :

1. Qu'est-ce que c'est ?
2. À quoi ça sert ?
3. Comment ça s'utilise ?

FONDAMENTAUX

1. déclarez avant d'utiliser

- ▶ variables (pensez à les initialiser)

```
int i(0);  
vector<double> v;
```

- ▶ fonctions  **prototype**

```
double sin(double x);  
bool cherche_valeur(Listechaine l, Valeur v);
```

2. passages (par valeur, par référence[, par référence constante])

3. choisir le **type** approprié

4. **modularisez** / décomposez / pensez « atomique »

4.1 **conception** (qu'est ce qu'on veut ?)

4.2 **implémentation** (comment ça se réalise ?)

4.3 **syntaxe** (comment ça s'écrit ?)

4.4 **tests** (où sont mes fautes, comment pourrais-je les tester ?)

Proposition de méthode de révision

- ▶ prendre le tableau synthétique du [transparent 5](#)
- ▶ prendre les fiches résumé
- ▶ et pour chacun des points, se demander si on sait :
 - ▶ de quoi ça parle ?
 - ▶ ce que ça veut dire ?
 - ▶ l'utiliser ?

👉 se focaliser sur les **concepts**.

Les détails de syntaxe (comment ça s'écrit) peuvent être **ensuite** rapidement retrouvés dans la fiche résumé, si on sait ce qu'on cherche (c'est-à-dire si on a le concept)

- ▶ pratiquer, pratiquer, pratiquer

par soi-même, seul(e) (sans « tricher » = sans aide, sans corrigé, ...)

Noter ses points de blocage/ses difficultés et revoir ensuite les concepts, puis exercices correspondants

Conseils pour examen(s)

- ▶ **lisez** la donnée et faites ce qui est demandé
ni plus (perte de temps), ni moins (perte de points)
- ▶ montrez **ce que vous savez**, pas ce que vous ne savez pas :
 1. ne perdez pas du temps à essayer de répondre à une question dont vous savez que vous ne connaissez pas le sujet (perte de temps, voire de points)
 2. et cf point ci-dessus : ne pas répondre à un aspect demandé alors que vous le connaissez est une perte de points
- ▶ **gérer** correctement **votre temps**
 1. ne ratez pas des questions auxquelles vous savez répondre
 2. ne perdez pas trop de temps sur une question qui ne rapporte pas beaucoup de points

Etude de cas ?

- ▶ Reprise du dernier exemple du cours ?
- ▶ Ajout d'exceptions dans un programme existant ?