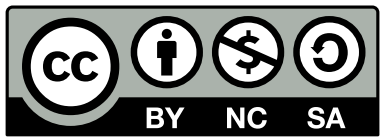
INF1135
Programmation procédurale
Chapitre 1 — Introduction

Alexandre Blondin Massé

Université du Québec à Montréal

v263



Plan

1. Plan de cours
2. Généralités
3. Développement

Plan

1 Plan de cours

2 Généralités

3 Développement

Informations générales

- **Trimestre** : Automne 2026
- **Titre du cours** : Programmation procédurale
- **Sigle** : INF1135
- **Département** : Informatique
- **Enseignant** : Alexandre Blondin Massé
- **Coordonnateur** : Alexandre Blondin Massé
- **Courriel** : blondin_masse.alexandre@uqam.ca
- **Plan de cours** : cliquer ici

Modalités d'évaluation

2 examens (55 %)

- 1 examen intra (25 %)
- 1 examen final (30 %)

8 quiz (15 %)

- En classe, ~20 minutes
- 3 % chacun (5 meilleurs)

3 travaux pratiques (30 %)

- 10 % chacun

Réussite du cours

Moyenne globale $\geq 55\,\%$ (pas de double seuil)

Commentaires additionnels sur l'évaluation

Examens

- Notions vues en **classe** ou en **laboratoire** : toutes évaluables

Quiz

- **Absence** = 0 (peu importe le motif)

Travaux pratiques

- Travaux : **individuels**
- IA générative **pas interdite**, mais **déconseillée**

Cours

- Seuil de 55 % : **ferme** (54,99 % = échec)
- **Aucun travail supplémentaire** possible pour « rattraper »

Contenu détaillé (1/2)

Chapitre 1. Introduction. Historique du C. Linux. Premier programme. Cycle de développement avec `gcc`.
Chapitre 2. Bases du C. Types de base (dont `bool`). Constantes. Flux standards et code de retour. Opérateurs. Conversions de types. `make`. Style. Documentation. Débogage.
Chapitre 3. Structures de contrôle. `if`/`else`. `switch`. Boucles `while`, `do-while`, `for`. `break` et `continue`. Tests et débogage.
Chapitre 4. Variables. Fonctions (premier contact). Portée. Durée de vie. Classes de stockage. Variables statiques et globales. Adresses mémoire.
Chapitre 5. Fonctions. Paramètres et valeur de retour. Prototypes. Passage par valeur. Documentation. Récursion. Fonctions variadiques. Tests unitaires. `gdb`.
Chapitre 6. Pointeurs. Déclaration et déréférencement. Passage par adresse. Tableaux (premier contact). Arithmétique de pointeurs. Pointeurs et `const`. Débogage.

Contenu détaillé (2/2)

Chapitre 7. Tableaux et chaînes. Tableaux 1D. Recherche et tri. Chaînes de caractères (`<string.h>`). Tableaux à plusieurs dimensions. Pointeurs de pointeurs. Tests et débogage.

Chapitre 8. Définition de types. Énumérations. Structures. Champs de bits. Unions.

Chapitre 9. Gestion de la mémoire. Organisation mémoire. Allocation dynamique (`malloc`/`free`). Fuites mémoire. Valgrind.

Chapitre 10. Entrées/sorties. Type `FILE`. Fichiers texte et binaires. Gestion des erreurs.

Chapitre 11. Modularité. Fichiers `.h`/`.c`. Gardes d'inclusion. Préprocesseur. Compilation séparée. Bibliothèques. Makefile complet (variables, règles génériques).

Chapitre 12. Introduction à la POO. Classes, attributs, méthodes, visibilité. Encapsulation, constructeurs, destructeurs. `std::string`, `std::vector`.

Plan

1 Plan de cours

2 Généralités

3 Développement

Paradigme procédural

Trois structures fondamentales :

- **Séquence** Instructions exécutées l'une après l'autre
- **Sélection** Branchement conditionnel (`if`, `switch`)
- **Itération** Répétition (`for`, `while`, `do-while`)

Théorème de Böhm-Jacopini (1966)

Tout programme calculable : exprimable avec ces trois structures seulement.

Naissance du langage C

- **1969** : Ken Thompson développe le système Unix chez Bell Labs
- **1972** : Dennis Ritchie crée le langage C

→ Fortement lié à l'histoire d'Unix

→ Environ 90 % d'Unix écrit en C

- **1978** : Publication du livre *The C Programming Language* (K&R)

Normalisation

- **1983** : L'ANSI forme un comité pour normaliser le langage
- **1989** : Norme **ANSI C** (C89 / C90)
- **1999** : Révision du standard — **ISO C99**
- **2011** : Révision du standard — **ISO C11**
- **2017** : Révision mineure — **ISO C17**
- **2023** : Révision du standard — **ISO C23**

Dans ce cours

C23 avec `gcc` sous Linux : option `-std=c23` (`gcc` 14 et plus) ou `-std=c2x` (`gcc` 13).

Caractéristiques du langage

- **Bas niveau** : proche du langage machine, contrôle élevé de la mémoire
- **Typé statiquement** : le type des variables doit être déclaré
- **Structuré** : organisé en blocs (accolades), indentation
- **Modulaire** : division en fichiers, compilation séparée
- **Portable** : mais parfois avec un certain effort
- **Simple** : spécification relativement courte
- **Efficace** : adapté aux systèmes embarqués et aux bibliothèques

Linux

- **Obligatoire** pour les laboratoires et travaux pratiques
- Environnement standard pour le développement en C
- Outils natifs : `gcc`, `make`, `gdb`, `valgrind`
- Installé sur les postes des laboratoires

Sur votre ordinateur

- Double démarrage (*dual-boot*)
- Machine virtuelle (VirtualBox)
- Windows Subsystem for Linux (WSL) sous Windows

Plan

1. Plan de cours
2. Généralités
3. Développement

Un premier programme

Fichier `hello.c` :

```
1  #include <stdio.h>
2
3  int main(void) {
4    printf("Hello, world!\n");
5    return 0;
6  }
```

- `#include <stdio.h>` — inclusion de la bibliothèque d'E/S standard
- `int main(void)` — point d'entrée du programme
- `printf` — affichage sur la sortie standard
- `return 0` — code de retour (0 = succès)

Compilation avec gcc

Compiler et exécuter `hello.c` :

```
$ gcc -Wall -Wextra -std=c23 -o hello hello.c
$ ./hello
Hello, world!
```

Options importantes de `gcc` :

- `-o nom` — nom de l'exécutable produit
- `-Wall` — active les avertissements courants
- `-Wextra` — active des avertissements supplémentaires
- `-std=c23` — spécifie la norme C23 (`-std=c2x` avec `gcc` 13)

Terminal

Travail en **ligne de commande** obligatoire. Détails : **INF1070**.

Cycle de développement

1. **Édition** : écrire le code source (`.c`, `.h`)
2. **Prétraitement** : expansion de `#include`, `#define` et macros
3. **Compilation** : traduction du C en langage d'assemblage
4. **Assemblage** : traduction de l'assembleur en code machine
5. **Édition des liens** : résolution des symboles, production de l'exécutable
6. **Exécution** : lancement du programme

$$\texttt{.c} \xrightarrow{\texttt{cpp}} \texttt{.i} \xrightarrow{\texttt{cc1}} \texttt{.s} \xrightarrow{\texttt{as}} \texttt{.o} \xrightarrow{\texttt{ld}} \text{exécutable}$$

`cpp`	= C preprocessor	`as`	= assembler
`cc1`	= C compiler (pass 1)	`ld`	= link editor (linker)

Exemple : le cycle étape par étape

On peut demander à gcc de s'arrêter après chaque étape :

```
$ gcc -E hello.c -o hello.i
$ gcc -S hello.i -o hello.s
$ gcc -c hello.s -o hello.o
$ gcc hello.o -o hello
$ ls
hello  hello.c  hello.i  hello.o  hello.s
$ wc -l hello.c hello.i hello.s
    6 hello.c
  819 hello.i
   45 hello.s
  870 total
$ ./hello
Hello, world!
```

- `-E` : prétraitement ; `hello.i` contient `stdio.h` (819 lignes !)
- `-S` : code **assembleur**
- `-c` : code **objet**, pas encore exécutable
- Sans option : édition des liens, d'où l'**exécutable**
- En pratique, un seul appel à gcc enchaîne tout

Édition

Première étape du cycle : **écrire le code source**.

Éditeurs graphiques

- Par exemple, **Visual Studio Code**
- Léger, extensible, terminal intégré
- Coloration syntaxique, autocomplétion, débogueur intégré

Éditeurs en mode texte

- Par exemple, **Vim** / **Neovim**
- Disponibles partout (serveurs, terminaux distants)
- Édition au clavier, sans souris — très efficace une fois maîtrisé

Dans ce cours

L'éditeur le **plus efficace** pour vous.