

INF3135

Construction et maintenance de logiciels

Cours 3: Bases du C, partie 2

Alexandre Blondin Massé

Université du Québec à Montréal
Département d'informatique

Été 2020

Table des matières

1 Résumé des capsules

2 Présentation du TP1

3 Exercices

Résumé des capsules

Tableaux

- Collection de données **homogènes** (de même type)
- Stockées de façon **contiguë** en mémoire
- Aucune **vérification** s'il y a dépassement

```
int t1[8];           // Réserve 8 * sizeof(int) sur le tas
double t2[n];        // Réserve n * sizeof(double) sur la pile
int t3[] = {5,2,0,1,3,4,7,6};
                    // Réserve 8 * sizeof(int) sur le tas
int t4[6] = {1,1};   // Initialise les 2 premières valeurs
char s1[] = "git";    // Équivalent à s1[4] = {'g','i','t','\0'};
char s2[4] = "ACGT";  // Équivalent à s2[4] = {'A','C','G','T'};
```

Chaînes de caractères

- Cas **particulier** de tableau
- Chaînes **littérales** délimitées par des guillemets " "
- Chaîne **bien formée**: doit terminer par le caractère `\0`
- On verra la bibliothèque `string.h` plus tard

Structures et unions

- Regroupent en un même bloc des données **hétérogènes**
- **Structures**: concaténation
- **Union**: alternative
- **Alignement**: compilateur décale selon l'architecture
- Opérateur `.` pour accéder à un **champ**

```
struct choice {  
    bool is_number;  
    union {  
        float number;  
        enum {YES, NO, MAYBE} answer;  
    };  
};  
struct choice c;  
// Accès: c.is_number, c.number, c.answer
```

- Affectation en bloc possible avec **conversion explicite**
- Peuvent être **imbriquées** et **combinées**
- Peuvent être **anonymes**
- typedef (**abusif**): permet d'omettre struct, union (et enum)

Fonctions

- **Unité de base** d'un programme avec les types
- Facilite la **réutilisation** et la **factorisation**
- Doivent être **bien nommées**
- Avec une syntaxe et une logique **uniforme**
- **Terminologie**: paramètres, arguments, fonction pure, à effets de bord, récursive
- **Valeur de retour**: une seule valeur permise ou void
- Paramètres et valeur de retour passés par **copie**
- On peut passer par **adresse** (on va y revenir)
- Fonction **spéciale**: main avec paramètres argc et argv
- Fonction utile: printf, pour affichage **formaté**
- Peuvent utiliser des variables **statiques**: durée de vie, tout le programme, portée limitée à la fonction

Compilation et Makefiles

GCC = GNU Compiler Collection

- `-c`: compilation seulement
- `-o`: spécifie le nom du fichier produit
- `-std=STD`: spécifie le standard
- `-Wall` et `-Wextra`: plus d'avertissements

Makefiles

- Règle **explicite**:

```
<cible>: <prérequis>  
<tab><recette>
```

- **Variables**: définition avec `nom=valeur`, accès avec `$(nom)`
- **Cibles spéciales**: `.PHONY`
- **Invocation**: `make` ou `make <cible>`

Préprocesseur

Précompilation

- Directives interprétées par le préprocesseur **avant** la compilation
- Symbole # au **début** de la ligne
- On peut insérer des **espaces** entre # et la directive

Directives permises

- #include: **inclusion** d'un fichier externe
- #define: définition d'un **symbole** ou d'une **macro**
- #undef: **annulation** d'un symbole ou d'une macro
- #ifdef/#ifndef: **vérifie** si une macro est définie ou non définie
- #if/#else/#elif/#endif: structure **conditionnelle**
- #error: indique une **erreur fatale**
- #pragma: pour des traitements plus **spécifiques**

Présentation du TP1

Travail pratique 1

- Date de remise: **18 juin**
- **20%** de la note totale
- Doit être fait **seul**
- **Dépôt GitLab**: doit être *forké*, sa visibilité mise à *privé*, puis l'accès en mode *Developer* doit être donné à blondin_al
- **Description du travail**: le fichier source `sujet.md` sera disponible dans le dépôt cloné, ne pas le modifier
- À **compléter**: `canvascii.c`, `Makefile`, `.gitignore`, `README.md`
- À **modifier**: dans `check.bats`, supprimer `skip` pour activer le test

Objectifs

- **Initiation à C**: manipuler `stdin`, `stdout`, `argc` et `argv`
- **Makefile**: automatiser tâches, dont la compilation
- **Git**: faire des *commits* propres et atomiques, `.gitignore`
- **Documentation**: fichier `README`, documenter le code source

Exercices

Exercices

1. Écrire un programme qui
 - définit un **type** `struct square_matrix` permettant de représenter une matrice carrée de doubles (vous pouvez supposer que l'ordre de la matrice est limité à 10);
 - définit une **fonction** `initialize_matrix(m,n,v)` qui crée une matrice `m` de dimensions `n x n` et qui initialise chacune des entrées de la matrice à `v`;
 - définit une **fonction** `print_matrix(m)` permettant d'afficher le contenu de la matrice sur `stdout`.
2. Écrire un programme nommé `somme.c` qui prend exactement un argument de la forme `a,b,c`, où `a`, `b` et `c` sont des entiers, et qui affiche la somme des trois nombres sur `stdout`:

```
$ gcc -o somme somme.c
```

```
$ ./somme 1,2,3
```

```
6
```