

INF3135

Construction et maintenance de logiciels

Cours 9: Présentation du TP2

Alexandre Blondin Massé

Université du Québec à Montréal
Département d'informatique

Été 2020

Table des matières

- 1 Retour sur le quiz 2
- 2 Résumé du chapitre 5
- 3 Travail pratique 2

Retour sur le quiz 2

Quiz 2

- **Date:** 23 juin dernier
- **Nombre de remises:** 57
- **Moyenne:** 7,58/10
- **Plus rapide:** 11 min 52 sec (et 10/10!)
- Pas trop de problèmes **techniques** (copier-coller)

Composition

- **Chapitre 2: Outils de développement logiciel:** 1/4
- **Chapitre 3: Pointeurs:** 1/4
- **Chapitre 4: Entrées et sorties:** 1/4
- **Question longue:** 1/4

Donc **256** quiz possibles

Résumé du chapitre 5

Généralités

Structure de données

Organisation **logique** d'un ensemble de données

Plusieurs objectifs

- **Simplifier** le traitement
- Offrir des opérations **efficaces**
- Économiser de l'espace **mémoire**

Interface et implémentation

- **Interface**: opérations supportées (type abstrait)
- **Implémentation**: organisation des données en mémoire, actions effectuées pour réaliser les opérations

Invariants et opérations

Invariant

- **Propriété** qui doit être satisfaite en tout temps
- Généralement vérifiable à l'aide d'une **fonction** booléenne

Opération

- Toute fonction qui **modifie** la structure de données
- Doit toujours préserver les **invariants**

Exemples

- **Chaîne de caractères**: termine par '`\0`'
- **Liste simplement chaînée**: le dernier noeud pointe vers NULL
- **Arbre binaire de recherche**: les clés respectent l'ordre, ...

Allocation dynamique

- Jusqu'à maintenant: mémoire réservée de façon **statique**
- Or, cette information n'est **pas toujours connu** à l'avance
- **Solution**: allouer l'espace mémoire de façon **dynamique**
- Dans la bibliothèque `stdlib.h`:

```
// Réserve un bloc de taille `size`  
void *malloc(size_t size);  
// Libère l'espace mémoire pointé par `ptr`  
void free(void *ptr);  
// Réserve un bloc de taille `nmemb * size` initialisé à 0  
void *calloc(size_t nmemb, size_t size);  
// Redimensionne un bloc de taille `size` déjà alloué dynamiquement  
void *realloc(void *ptr, size_t size);  
// Redimensionne un bloc de taille `nmemb * size`  
void *reallocarray(void *ptr, size_t nmemb, size_t size);
```


Mémoire

Fuite de mémoire

- Mémoire **réservée** mais non **référéncée**
- Provoquée lorsqu'on appelle malloc ou calloc
- Et qu'on oublie de libérer avec free
- Souvent « **caché** » derrière une autre fonction (strdup)

Solutions

- Préférer un passage par **adresse**
- Et utiliser malloc/calloc/free seulement lorsqu'inévitable
- Fournir une fonction **complémentaire** qui libère l'espace alloué

Valgrind (<http://valgrind.org/>)

Permet de détecter des erreurs de **gestion de mémoire**

Structures de données

Piles (LIFO)

Implémentée avec liste simplement chaînée

File (FIFO)

Voir exercice à la fin

Tableaux dynamiques

Utilisation de `realloc`

Tableaux multidimensionnels

Utilisation de doubles pointeurs, initialisation et suppression

Arbres binaires de recherche

- Structure arborescente, avec clé, illustrée avec `treemap`
- Astuce des doubles pointeurs pour l'insertion

Travail pratique 2

Travail pratique 2

- Date de remise: **26 juillet**, à **23h59**
- 20% de la **note finale**
- Doit être fait **seul** aussi
- **Dépôt:** <https://gitlab.info.uqam.ca/inf3135-ete2020/inf3135-ete2020-tp2>
- **Sujet:** <https://gitlab.info.uqam.ca/inf3135-ete2020/inf3135-ete2020-tp2/-/blob/sujet/sujet.md>
- Vous devez faire un *fork* du dépôt
- Et organiser votre développement en **branches**

Objectifs

- Se **familiariser** avec un logiciel développé en C par quelqu'un d'autre
- Apprendre à utiliser des **bibliothèques tierces** à l'intérieur d'un programme C, en consultant la documentation disponible
- Organiser le développement des modifications à l'aide de **branches**
- Soumettre les **modifications** en utilisant des **requêtes d'intégration** (*merge requests*)
- **Documenter** convenablement des requêtes d'intégration à l'aide du format Markdown
- S'assurer que les modifications apportées sont adéquates en proposant ou en mettant à jour un **cadre de tests** qui montre que les modifications n'entraînent pas de régression

Installation des dépendances

Dépendances

- **PKG-config**: pour la compilation et l'édition des liens
- **Cairo**, pour dessiner
- **Jansson**, pour manipuler le format JSON
- **Bats**, pour les tests unitaires « externes »
- **Libtap**, pour les tests unitaires « internes »

Installation

- Avec un **gestionnaire de paquets**: Apt, Homebrew, etc.
- Lancer `ldconfig` après avoir installé Libtap

Utilisation plus avancée de Git

- Résolution de **conflits**
- Organisation en **branches**
- **Rebasements** interactifs

Démonstration

- Résoudre un conflit
- `git stash`
- `git branch`
- `git checkout -b`
- `git cherry-pick`
- `git rebase`
- `git rebase -i`