

INF3135

Construction et maintenance de logiciels

Cours 8: Structures de données

Alexandre Blondin Massé

Université du Québec à Montréal
Département d'informatique

Été 2020

Table des matières

1 Travaux pratiques

2 Résumé du chapitre 5

3 Exercices

Travaux pratiques

Travail pratique 1

- Date de remise: **19 juin**, à **23h59**
- **56 projets** remis (sur 64 inscrits)
- **Correction**: vers vendredi le **3 juillet**

Observations générales

- Attention de ne pas s'y prendre à la **dernière minute**
- Mettre en place le **Makefile** rapidement
- Mettre en place la **suite de tests** rapidement
- Vérifier **en continu** l'état du projet sur GitLab

Travail pratique 2

- Date de remise: **24 juillet**, à **23h59**
- 20% de la **note finale**
- Doit être fait **seul**

Sujet

- Disponible d'ici **vendredi**
- Sera présenté au **prochain cours**
- **Objectifs**: apporter des modifications à un programme, utilisation avancée de Git, modularité en C
- **Bibliothèques**: **Jansson** et **Cairo**

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

Exercices

Exercices

Compléter l'implémentation d'une **file** (queue):

```
// Initialise une file
void queue_initialize(queue *q);

// Indique si une file est vide
bool queue_is_empty(const queue *q);

// Ajoute un élément en fin de file
void queue_push(queue *q, unsigned int value);

// Récupère l'élément en tête de file
char queue_pop(queue *q);

// Affiche une file sur stdout
void queue_print(const queue *q);

// Détruit une file
void queue_delete(queue *q);
```