ENSTA Paris
2023-2024

Résolution de problème algorithmique

Examen – rattrapage 1

PRÉAMBULE

Consignes générales (À LIRE IMPÉRATIVEMENT)

- **Lisez attentivement les consignes et tout le sujet avant de commencer.**

- Les documents (polys, transparents, TDs, livres ...) sont autorisés.
- Sont **absolument interdits** : le WEB, le courrier électronique, les messageries diverses et variées, le répertoire des camarades, le téléphone (même pour avoir l'heure puisque vous l'avez sur votre ordinateur).

- Votre travail sera (en partie) évalué par un mécanisme automatique. Vous **devez** respecter les règles de **nommage** des fichiers et autres **consignes** qui vous sont données.
- La connaissance de C est un pré-requis d'IN103. Ainsi, la présence d'avertissement(s) ou d'erreur(s) à la compilation réduira mécaniquement la note de l'exercice considéré.
- Lorsqu'il vous est demandé que votre programme réponde en affichant « **Yes** » ou « **No** », il ne doit **rien** afficher d'autre, et **pas** « Oui » ou « Yes. » ou « no » ou « La réponse est : no ». Donc, pensez à retirer vos affichages de test / debug.
- La **lisibilité** et l'efficacité / simplicité / complexité de vos programmes seront **prises en compte** dans l'évaluation.

- **À la fin de l'examen**, vous devrez créer une **archive** contenant **tous** les fichiers **sources** que vous avez écrits (`.c`, `.h`). **N'incluez pas** d'exécutables dans l'archive, le mail pourrait la considérer comme un attachement dangereux et le supprimer. Le nom de cette archive devra avoir la structure suivante :

  `nom_prenom.zip` ou `.tgz` (selon l'outil d'archivage que vous utilisez).

  Les commandes sont :

  - `tar xvzf nom_prenom.tgz in103-examen-material`
  - `zip -r nom_prenom.zip in103-examen-material`

  Remarque un fichier `Makefile` est donné dans le répertoire `in103-examen-material` qui permet de supprimer tous les fichiers exécutables en exécutant la règle `realclean` dans les répertoires des exercices. Il suffit donc d'exécuter la commande `make` dans ce répertoire pour supprimer les exécutables.

- Vous devrez **m'envoyer** cette archive par mail (`alexandre.chapoutot@ensta-paris.fr`). En cas d'envoi incorrect, il vous sera demandé de refaire l'archive et l'envoi. Par contre, vous ne devrez **surtout pas modifier** les fichiers : leurs dates de dernière modification ne devra pas être ultérieure à l'heure de fin de l'épreuve sous peine d'être considérés comme nuls.

- **N'oubliez pas** d'effectuer cet envoi sinon nous devrons considérer que vous n'avez rien rendu sur la partie programmation !
- **Il y aura également des questions dont les réponses sont à donner dans un formulaire Microsoft Form**. Il faudra également penser à soumettre ce formulaire à la fin de l'épreuve. Ce formulaire accepte l'enregistrement des réponses et la modification des réponses enregistrées.

- Le sujet comporte 5 pages et l'examen dure **1h30**.
- Le barème est donné à titre indicatif.

Nouvelle version de la bibliothèque `libin103`

La version **1.4.2** est la nouvelle version de la bibliothèque `libin103` qui corrige quelques bugs. Il faut donc mettre à jour votre environnement de travail en suivant les étapes données ci-dessous.

1. À la racine de votre compte, créez un répertoire nommé `Library` **s'il n'a pas déjà été créé**, puis placez vous dans ce répertoire.

   ```
   mkdir ~/Library; cd ~/Library
   ```

2. Téléchargez l'archive `libin103-1.4.2.tar.gz` sur le site du cours

   wget
   `https://perso.ensta-paris.fr/~chapoutot/teaching/in103/practical-work/libin103-1.4.2.tar.gz`

3. Désarchivez l'archive

   ```
   tar -xvzf libin103-1.4.2.tar.gz
   ```

4. Allez dans le répertoire `libin103-1.4.2` et compilez la bibliothèque.
   - Il faut utiliser la commande `make`. À la fin de la compilation vérifiez la présence du fichier `libin103.a` dans le répertoire `source`.

Pour rappel, la documentation de la bibliothèque est accessible sur le site web du cours

`https://perso.ensta-paris.fr/~chapoutot/teaching/in103/refman/index.html`

L'onglet `fichier` regroupe la documentation de tous les fichiers `.h`.

Matériel pour l'examen

Récupérez l'archive associé à cette séance de TP à l'adresse :

`https://perso.ensta-paris.fr/~chapoutot/teaching/in103/practical-work/in103-examen-rat-material.tar.gz`

Comme pour les TP, pour la plus part des exercices, un répertoire est associé à chaque exercice dans lequel se trouve un fichier Makefile et un ou plusieurs fichiers code source. Ce fichier Makefile a été configuré pour utiliser la dernière version de la bibliothèque `libin103` (version 1.4.2).

PARTIE À RENDRE

Exercice 1 – Somme d'éléments d'une file (3 pts)

Le matériel pour cet exercice est donné dans le répertoire `exo1`. L'objectif de cet exercice est de définir le corps de la fonction

```
int sum_queue_elements (integer_queue_t *queue);
```

Cette fonction prend en argument une file d'entiers et doit renvoyer la somme de ses éléments **sans modifier le contenu de la file** (c'est-à-dire qu'à la fin de la fonction la file est toujours pleine). Par exemple, une utilisation de ce programme serait :

```
alex@MacAlex exo1 % ./sum-queue.x 3 4 5
Queue before: 3, 4, 5
The sum of elements of the queue is "12"
Queue after: 3, 4, 5
```

Question 1

Programmez cette fonction **en utilisant uniquement que les fonctions de l'API des Files (queues)**, c'est-à-dire, les fonctions données dans le fichier `integer_queue.h`. Il ne faut pas convertir la file en liste ou en tableau.

Exercice 2 – Et si on automatisait la compilation avec un `Makefile` ? (2 pts)

Le matériel pour cet exercice est donné dans le répertoire `exo2`. Nous considérons dans cet exercice le seul exercice dont le code vous est fourni sans fichier `Makefile` pour générer le programme exécutable. L'objectif donc est décrire un fichier `Makefile` pour automatiser la compilation du programme `principal.x` qui dépend de :

- `interval.h` / `interval.c` contiennent la déclaration et définition du type de données pour représenter des intervalles fermés et les opérations arithmétiques d'addition, soustraction et multiplication sur les intervalles fermés ainsi qu'une fonction d'affichage.
- `principal.c` qui contient la fonction principale du programme.

Un exemple d'exécution du programme compilé est

```
alex@MacAlex exo2 % ./principal.x
x = [1.00; 2.00]
y = [-2.00; -1.00]
add(x, y) = [-1.00; 1.00]
sub(x, y) = [2.00; 4.00]
mul(x, y) = [-4.00; -1.00]
```

Question 1

Donnez les lignes (sur la feuille) de compilation pour générer les fichiers `.o` à partir des fichiers `.c` donnés dans le répertoire `exo2`. Nous utiliserons les options `-Wall` et `-Werror` de la commande `gcc`.

Question 2

Écrivez le fichier `Makefile` pour automatiser la génération du fichier exécutable `principal.x`. On ne cherchera pas forcément à simplifier les règles.

Question 3

Ajoutez les règles `clean` et `realclean` qui permettent respectivement de supprimer les fichiers `.o` générés et de supprimer le fichier exécutable `principal.x` (en plus de supprimer les fichiers `.o`).

Exercice 3 – Expressions bien parenthésées (5 pts)

Le matériel pour cet exercice est donné dans le répertoire `exo3`. Cet exercice considère le problème de détecter des expressions bien parenthésées, c'est-à-dire, ayant pour chaque parenthèse ouvrante une parenthèse fermante associée, dans des listes chaînées de caractères.

Le code source fourni dans le répertoire `exo3` permet de lire une chaîne de caractères sur la ligne de commande et la transforme en une liste chaînée de caractères. Ci-dessous, quelques exemples de fonctionnement du programme :

```
alex@MacAlex exo3 % ./is-well-bracketed.x
Usage ./is-well-bracketed.x string
alex@MacAlex exo3 % ./is-well-bracketed.x "("
The string "(" is not well bracketed
alex@MacAlex exo3 % ./is-well-bracketed.x "()"
The string "()" is well bracketed
alex@MacAlex exo3 % ./is-well-bracketed.x "())"
The string "())" is not well bracketed
alex@MacAlex exo3 % ./is-well-bracketed.x "(a()g)"
The string "(a()g)" is well bracketed
alex@MacAlex exo3 % ./is-well-bracketed.x "(a(x+c)g)"
The string "(a(x+c)g)" is well bracketed
```

Question 1

Définissez le code de la fonction qui vérifie si la liste chaînée de caractères est bien parenthésée, son prototype est

```
bool is_well_bracketed (character_list_t* str);
```

Exercice 4 – Optimisation d'urbanisme (8 pts)

Dans un quartier, il y a n chocolatiers, chacun d'entre eux a besoin d'un approvisionnement en électricité. La construction d'un champs de panneaux solaire pour le chocolatier u coûte $w[u]$ milliers d'euros, et la construction d'un câble électrique entre le chocolatier i et le chocolatier j coûte $c[i][j]$ milliers d'euros. Un chocolatier peut recevoir de l'électricité si un champs de panneaux solaires y a été construit ou s'il existe un câble électrique menant à un chocolatier doté d'un champs de panneaux solaires. Modélisez ce problème et concevez un algorithme pour trouver le montant minimum d'argent nécessaire pour approvisionner chaque chocolatier en électricité.

Question 1

Quel algorithmique vu en cours allez-vous utiliser pour répondre à ce problème ?

Question 2

Décrivez la modélisation du problème à l'aide d'un schéma et/ou d'un court texte.

Question 3

Programmez votre proposition d'algorithme pour un problème avec les valeurs numériques suivantes :

- $n = 4$
- $w[0] = 120$, $w[1] = 150$, $w[2] = 172$, $w[3] = 125$
- La matrice des coûts de câblage est

$$\begin{pmatrix} 0 & 25 & 28 & 33 \\ 25 & 0 & 12 & 67 \\ 28 & 12 & 0 & 23 \\ 33 & 67 & 23 & 0 \end{pmatrix}$$

Le matériel pour cet exercice est donné dans le répertoire `exo4` mais est très minimal.

Exercice 5 – Union-find (2 pts)

Pas de programmation dans cet exercice. Vous réponderez dans le fichier texte “la-reponse.txt” présent dans le répertoire `exo5`.

Question 1

Décrivez brièvement le fonctionnement de la structure de données union-find pour les opérations `find` et `union` quand ces opérations utilisent l'union par rang et la compression de chemin.