ENSTA Paris
2023-2024

Résolution de problème algorithmique

Examen

PRÉPARATION

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 7 pages et l'examen dure **3h00**.
— 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 associée à cette séance d'examen à l'adresse :

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

Comme pour les TP, pour la plupart des exercices, un répertoire est associé à l'exercice dans lequel se trouve un fichier `Makefile` (sauf exception) 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).

Par contre, il y a également un questionnaire Microsoft Form qu'il faut remplir pour certaines questions. Les réponses aux questions associées au symbole ★ sont à entrer dans le questionnaire. L'adresse de ce questionnaire est :

`https://forms.office.com/e/f6JkG0g6mn`

À SOUMETTRE

Exercice 1 – ★ - Quelques questions de cours (5 pts)

Allez sur le formulaire, pour avoir accès aux questions et donnez vos réponses dont l'URL est

`https://forms.office.com/e/f6JkG0g6mn`

Exercice 2 – Automatisation de la compilation avec un `Makefile` ? (3 pts)

Le matériel pour cet exercice est donné dans le répertoire `exo2`.

Cet exercice le seul dont le code vous est fourni sans fichier `Makefile` pour générer le programme exécutable. L'objectif est donc d'écrire un fichier `Makefile` pour automatiser la compilation du programme `main.x` qui dépend de :

- `points2d.h` / `points2d.c` contiennent la déclaration et définition du type de données pour représenter des points dans un espace 2D.
- `convex.h` / `convex.c` contiennent la déclaration et définition d'une fonction `convex_hull` qui permet de calculer l'enveloppe convexe d'un ensemble de points 2D donné sous la forme d'un tableau.
- `main.c` qui contient la fonction principale du programme qui lit sur son entrée standard des points 2D (un par ligne) et affiche sur sa sortie standard les points 2D (un par ligne) qui composent l'enveloppe convexe.

A noter que ce programme n'utilise pas la bibliothèque `libin103`.

Dans ce répertoire, il y a également un répertoire nommé `tests` qui contient

- un fichier `hull.in` avec un ensemble de points. La première ligne de ce fichier donne le nombre de points puis les lignes suivantes sont les coordonnées des points sous la forme x, y où x et y sont des nombres flottants ;
- un fichier `hull.out` la liste des points associée à l'enveloppe convexe.

Un exemple d'exécution du programme compilé est

```
alex@MacAlex exo2 % ./main.x < tests/hull.in
(2.000000, 1.000000)
(2.500000, 4.500000)
(4.500000, 1.500000)
```

Question 1 – ★

Donnez les lignes 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 `main.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 `main.x` (en plus de supprimer les fichiers `.o`).

Exercice 3 – Détecteur de pangrammes (4 pts)

Le matériel pour cet exercice est donné dans le répertoire `exo3`.

D'après Wikipédia [1], un pangramme est une phrase comportant toutes les lettres de l'alphabet. L'objectif de cet exercice est d'écrire un programme qui permet de détecter cette propriété. Nous ne considérerons que des phrases

1. https ://fr.wikipedia.org/wiki/Pangramme

avec des lettres minuscules (sans accent), des symboles de ponctuation (., ,, ;, ′) et des caractères espaces. Quand un pangramme est détecté, on demande en plus d'afficher le nombre de lettres (sans ponctuation ni espaces) qui le compose.

Dans le répertoire `tests`, il y a 5 fichiers `*.in` de test et 5 fichiers `*.out` associés qui sont les sorties attendues. Votre programme devrait donner les comportements suivants :

```
alex@MacAlex exo3 % for i in 1 2 3 4 5; do ./is-pangram.x < tests/test$i.in; done
YES(36)
YES(47)
YES(63)
NO
NO
```

Indices :

— dans le fichier `ctype.h`, il y a une fonction `islower` qui permet de détecter si le caractère passé en paramètre est une lettre minuscule ou pas. Pour plus d'information, voir `man islower`.

— il faut juste détecter si toutes les lettres minuscules (26) sont présentes dans la phrase.

Question 1 – ★

Quelle structure de données vue en cours vous pouvez utiliser pour résoudre cet exercice ?

Question 2

Définissez le code de la fonction qui vérifie si une liste chaînée de caractères est un pangramme, son prototype est

```
bool is_pangram (character_list_t* str, int* sizePangram);
```

Le premier paramètre est la liste contenant la phrase à analyser, le second paramètre est utilisé pour renvoyer le nombre de lettres minuscules (sans accents) qui composent le pangramme. La valeur de retour booléenne indique si la liste chaînée est associée à un pangramme ou non.

Remarque : la fonction `main`, donnée dans le fichier `is-pangram.c`, crée la liste chaînée en lisant sur l'entrée standard un caractère par ligne. Vous n'avez pas à gérer cette création.

Exercice 4 – Additions (4 pts)

Le matériel pour cet exercice est dans le répertoire `exo4`.

Dans cet exercice, on considère un flot de valeurs entières qui est lu sur l'entrée standard. A chaque fois qu'on lit une nouvelle valeur, on veut afficher le résultat de la somme du produit des deux plus petites et du produit des deux plus grandes valeurs lues jusqu'à présent. Si on n'a pas lu assez de valeurs alors on affiche -1.

Dans le répertoire `tests`, il a un couple de fichiers `test1.in` et `test1.out` qui donne un jeu de données et la sortie attendue. Pour l'exemple, l'exécution du programme sur ce fichier est :

```
alex@MacAlex exo4 % cat tests/test1.in
1
6
7
8
4
3
9
10
2
5
alex@MacAlex exo4 % ./mini-sum.x < tests/test1.in
-1
12
```

```
48
62
60
59
75
93
92
92
alex@MacAlex exo4 % ./mini-sum.x < tests/test1.in | diff - tests/test1.out
```

La dernière ligne de cet exemple ne doit rien afficher si la sortie produite par le programme `mini-sum.x` comparée avec le résultat attendu donné dans le fichier `test1.out` sont les mêmes.

Question 1 – ★

Quelle structure de données vue en cours, pouvez-vous utiliser pour résoudre cet exercice ?

Question 2

Implémentez votre algorithme solution à ce problème dans la fonction principale du fichier `mini-sum.c`.

Remarque : dans la fonction `main`, donnée dans le fichier `mini-sum.c`, les constructions pour faire la lecture sur l'entrée standard sont données.

Exercice 5 – Inventaire (4 pts)

Le matériel pour cet exercice est dans le répertoire `exo5`.

Thibault est responsable de l'équipement électronique, informatique du laboratoire de robotique. Chaque année, il fait un inventaire du matériel afin de préparer les commandes des pièces manquantes. Pour chaque tiroir de l'atelier, Thibault a la liste des pièces, un entier positif identifie une pièce. Généralement, Thibault note les pièces manquantes au fur et à mesure mais suite à une panne informatique cette liste a été altérée en mélangeant des numéros de pièces manquantes avec des numéros de pièces factices. Au final, Thibault a la liste complète des pièces et une liste de pièces restantes erronées.

Votre mission est d'aider Thibault à trouver la liste des pièces factices. Les fichiers pour chaque tiroir ont la forme suivante :

- Sur une ligne, un entier positif n indiquant un nombre de pièces dans le tiroir ;
- Les n lignes suivantes sont composées d'un entier positif p_i représentant le numéro d'une pièce ;
- Sur une ligne, un entier positif $m < n$ indiquant un nombre de pièces restantes ;
- Les m lignes suivantes sont composées d'un entier positif q_j représentant le numéro d'une pièce potentiellement factices.

Un exemple de fichier est

```
alex@MacAlex exo5 % cat tests/test1.in
10
3612
7131
8392
8401
8667
6798
2948
356
3521
5317
4
4106
2790
```

```
3478
982
```

Il y a 10 pièces dans le tiroir. Il y a 4 pièces manquantes dont certaines sont factices.

Le résultat attendu est la liste des numéros de pièces factices. Un exemple d'exécution sur le fichier exemple est

```
alex@MacAlex exo5 % ./inventory.x < tests/test1.in
4106
2790
3478
982
```

Les 4 pièces sont factices.

Question 1 – ★

Quelle structure de données vue en cours, pouvez-vous utiliser pour résoudre cet exercice ?

Question 2

Dans cette question, il faut implémenter un algorithme, dans le fichier `inventory.c`, qui permet de répondre au problème en essayant d'être le plus efficace possible si on suppose que la liste des pièces peut être assez grande (le fichier `test6.in` contient 6000 pièces).

Remarque : dans la fonction `main`, donnée dans le fichier `inventory.c`, les constructions pour faire la lecture sur l'entrée standard sont données.

Question 3

Dans cette question, on améliore l'algorithme précédent en lui faisant fournir une liste des numéros de pièces factices dans l'ordre croissant. Il faut coder cette nouvelle version dans le fichier `inventory-v2.c`. On veillera à garder une complexité basse.

Remarque : dans la fonction `main`, donnée dans le fichier `inventory-v2.c`, les constructions pour faire la lecture sur l'entrée standard sont données.

Exercice 6 – Dark City (6 pts)

Le matériel pour cet exercice est donné dans le répertoire `exo6`.

John Murdoch [2] vie dans une ville où chaque nuit les rues et les immeubles se transforment. Chaque matin il faut qu'il réapprenne la topologie de la ville et qu'il découvre comment sont connectés les quartiers. La nuit dernière John Murdoch a découvert un livre dont chaque page décrit l'évolution des connexions entre les quartiers de la ville. En feuilletant ce livre, il apprend que une fois par an tous les quartiers sont connectés lui laissant sa seule chance de s'échapper en atteignant un portail caché. Le problème est de savoir quel jour ce phénomène se produira pour l'année courante. Dans le répertoire `livre` des fichiers, représentant les pages du livre, sont donnés. On suppose que le numéro de page est associé au numéro du jour.

Les pages de cet ouvrage ont la forme suivante :

— une ligne avec un entier positif n donnant le nombre de quartiers de la ville dans sa configuration du jour ;

— les $m > 0$ lignes suivantes (et jusqu'à la fin du fichier) représentent les passages (dans les deux sens) entre deux quartiers sous la forme $x - y$ où x et y sont des entiers positifs. A noter qu'il y a potentiellement des redondances dans la description des connexions.

Pour aider, John Murdoch, vous devez écrire un programme qui permet de calculer en combien d'îlots séparés la ville a été découpée. Et en appliquant, votre programme sur toutes les pages du livre vous trouverez le jour où John Murdoch pourra s'échapper.

Par exemple, pour la page du livre suivante

2. Personnage principal du film *Dark City* 1998.

```
alex@MacAlex exo6 % cat tests/page1.in
6
0-3
1-3
1-2
3-2
4-5
```

votre programme devrait fournir la sortie

```
alex@MacAlex exo6 % ./island.x < tests/test1.in
2
```

indiquant que la ville est séparée en deux îlots disjoints.

Remarque : dans la fonction `main`, donnée dans le fichier `island.c`, les constructions pour faire la lecture sur l'entrée standard sont données.

Question 1
Codez l'algorithme qui permet de répondre à cette question s'il est appliqué à toutes les pages du livre.

Question 2 – ★
Donnez le numéro du jour où John Murdoch pourra s'échapper.