ENSTA
2024-2025

Résolution de problème algorithmique

Examen intermédiaire - 4 mars 2025

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), les IA génératives (e.g., Mistral AI, ChatGPT, Copilot, etc.).

— Votre travail sera évalué par un mécanisme automatique. Vous **devez** respecter les règles de **nommage** et autres **consignes** qui vous sont données, en particulier, le format des sorties de vos programmes.

— La connaissance de C est un pré-requis du cours CSC_3IN03_TA. Ainsi, la présence d'erreur(s) à la compilation induira une note de zéro à l'exercice considéré, par le mécanisme de correction automatique.

— 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 totalité des points d'un exercice est donnée si le programme compile sans erreur, son exécution termine sans erreur et le résultat produit pour chaque jeu de tests est conforme aux spécifications. C'est donc une notation binaire : cela fonctionne ou cela ne fonctionne pas, il n'y aura pas de correction manuelle des programmes.

— **IMPORTANT** : Le site de correction automatique ne doit pas être utilisé pour mettre au point vos programmes ! Pour le développement de vos programmes, vous devez utiliser votre machine et le mode BYOD sur lequel la bibliothèque `libin103` est installée. Une fois que vos programmes compilent et s'exécutent correctement sur les jeux de données fournis alors vous pouvez soumettre votre réponse pour évaluation. La correction automatique testera vos programmes sur un ensemble de jeux de données augmenté (au moins un de plus que ceux qui vous ont été fournis).

— Le sujet comporte 8 pages et l'examen dure **2h30**.

Version de la bibliothèque `libin103`

La version $\mathbf{1.4.2}$ est la dernière version en date de la bibliothèque `libin103`. Sauf si vous n'avez pas déjà installée cette bilbiothèqe, voici la procédure :

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-intermediaire-2425-material.tar.gz
```

Comme pour les TP, 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).

De plus, dans chaque répertoire des exercices se trouve également un répertoire `tests` qui contient des jeux de données d'entrée et de sorties attendues. Une façon simple de lancer les tests est d'utiliser la directive `test` du fichier `Makefile` ou, plus précisément, en lançant la commande

```
$ make test
```

Cette commande compile le programme et exécute les tests en les comparant la sortie du programme avec la sortie attendue. De manière plus détaillée, la commande exécutée est, par exemple,

```
$ ./mon_programme.x < tests/test1.in | diff - tests/test1.out
```

`mon_programme.x` est le programme qui vous auriez écrit, `tests/test1.in` est un jeu de données d'entrée lu sur l'entrée standard du programme, et `tests/test1.out` est la sortie attendue pour une exécution correcte du programme pour la donnée d'entrée. La commande `diff` permet de comparer deux flux d'informations (venant de l'entrée standard ou d'un fichier). Si aucune différence entre les deux flux n'est détectée, la sortie de la commande `diff` ne produit aucun affichage.

Cependant, pour clarifier le message, le lancement de la commande `make test` a été associé à un affichage joli qui permet d'afficher `OK` quand le jeu de test est conforme aux attentes et `KO` quand la sortie produite par le programme n'est pas conforme aux attendus. Par exemple,

```
exo1 $ make test
gcc -Wall -Werror -I$(HOME)/Library/libin103-1.4.2/include gain2.c -o gain2.x \
                            -L$(HOME)/Library/libin103-1.4.2/source -lin103
Test 1: OK
Test 2: OK
Test 3: OK
Test 4: OK
Test 5: OK
```

La première commande (sur 2 lignes) est associée à la compilation du programme, les suivantes sont les différents tests qui sont effectués et leur status.

À SOUMETTRE

Exercice 1 – Mutliplication par deux (2.5 pts)

Le matériel pour cet exercice est donné dans le répertoire `exo1`.

Description L'objectif de cet exercice est d'écrire un programme qui lit un entier positif n sur son entrée standard et affiche sur sa sortie standard le double de n, c'est-à-dire $2 \times n$.

Exemple On considère que le programme est un fichier nommé `gain2.x`. On considère le premier fichier de test, `test1.in` qui se situe dans le répertoire `tests`,

```
$ cat tests/test1.in
10
$ ./gain2.x < tests/test1.in
20
$
```

Spécifications des entrées L'entrée est un entier positif n compris entre zéro et `INT_MAX` qui vaut 2147483647. Cet entier est lu sur l'entrée standard.

Spécifications des sorties La sortie est un entier positif compris en zéro et `INT_MAX` qui vaut 2147483647. Si la valeur du résultat $2 \times n$ est plus grande que `INT_MAX` alors le programme renvoie `INT_MAX`. Le résultat du calcul est affiché sur la sortie standard avec un saut de ligne.

Exercice 2 – Générateur de séquences d'entiers (2.5 pts)

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

Description Sous Linux la commande `seq` permet de générer des séquences de nombres. L'objectif de cet exercice est de programmer une version simplifiée de cette commande. Dans cet exercice, nous allons considérer uniquement des générations de séquences d'entiers.

Exemple On considère que le programme est un fichier nommé `sequence.x`. Et nous considérons les deux premiers fichiers de test, `test1.in` et `test2.in` qui se situent dans le répertoire `tests`,

```
$ cat tests/test1.in
10 -1 0
$ ./sequence.x < tests/test1.in
10
9
8
7
6
5
4
3
2
1
0
$ cat tests/test2.in
```

```
10 1 0
$ ./sequence.x < tests/test2.in
needs negative increment
```

Spécifications des entrées Le programme lit sur son entrée standard trois entiers sous la forme

$$x \quad y \quad z$$

où x est la borne de début de séquence et z est la borne de fin de la séquence. La quantité y représente la valeur de l'incrément entre chaque élément de la séquence. De plus, x, y, et z peuvent prendre des valeurs entre `INT_MIN` (-2147483648) et `INT_MAX` (2147483647).

Spécifications des sorties Dans le mode nominal, chaque élément de la séquence est affiché seul sur une ligne. La fin de la séquence se termine également par un saut de ligne. Il y a plusieurs cas d'erreur à gérer :

- si $x > z$ et $y > 0$ alors le programme affichera `needs negative increment` avec un saut de ligne ;
- si $x < z$ et $y < 0$ alors le programme affichera `needs positive increment` avec un saut de ligne ;
- si $y = 0$ et $x \neq z$ alors le programme affichera `incorrect increment value` avec un saut de ligne.

Exercice 3 – Détecteur d'anagrammes (2.5 pts)

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

Description Une anagramme est « est un mot ou une phrase obtenu en permutant les lettres d'un mot ou d'une phrase de départ ». L'objectif d'écrire un programme qui permet de détecter si deux mots sont des anagrammes. Pour simplifier, nous ne considérerons que des mots avec des lettres minuscules (sans accent).

Exemple On considère que le programme est un fichier nommé `is-anagram.x`. Et nous considérons le premier fichier de test, `test1.in` qui se situe dans le répertoire `tests`,

```
$ cat tests/test1.in
s
a
l
u
t
*
t
u
l
a
s
$ ./is-anagram.x < tests/test1.in
OUI
```

Spécifications des entrées Le programme lit sur son entrée standard deux séquences de caractères à comparer. Chaque caractère des séquences occupe une ligne de l'entrée standard. Les deux séquences de caractères sont séparées par le caractère spécial `*`. Les caractères considérés sont uniquement les lettres minuscules sans accent.

Spécifications des sorties Le programme affiche un message sur une ligne suivi d'un saut de ligne. Le message possible est

- `OUI` si les deux chaînes sont des anagrammes ;
- `NON` si les deux chaînes ne sont pas des anagrammes.

Exercice 4 – Tri d'entiers dans l'ordre décroissant (2.5 pts)

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

Description Dans cet exercice, on considère un flot de valeurs entières qui est lu sur l'entrée standard. L'objectif est de trier ces valeurs dans l'ordre décroissant et de l'afficher sur la sortie standard.

Exemple On considère que le programme est un fichier nommé `decreasing-order.x`. Et nous considérons le premier fichier de test, `test1.in` qui se situe dans le répertoire `tests`,

```
$ cat tests/test1.in
1
6
7
8
4
3
9
10
2
5
$ ./decreasing-order.x < tests/test1.in
10
9
8
7
6
5
4
3
2
1
```

Spécifications des entrées Le programme lit sur son entrée standard des nombres qui sont positionnés un par ligne. Chaque nombre est dans les bornes de valeurs entre `INT_MIN` (-2147483648) et `INT_MAX` (2147483647). Au maximum, il y aura 1000 de données à traiter.

Spécifications des sorties Le programme affiche sur sa sortie standard la suite de nombre dans l'ordre décroissant en placant un nombre sur chaque ligne. La fin de la séquence est suivie par un saut de ligne.

Exercice 5 – Détection de monotonie (2.5 pts)

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

Description L'objectif de ce programme est d'analyser la monotonie d'une séquence de valeurs entières lue sur son entrée standard.

Exemple On considère que le programme est un fichier nommé `monotony.x`. Et nous considérons deux fichiers de test, `test1.in` et `test3.in` qui se situent dans le répertoire `tests`,

```
$ cat tests/test1.in
1
2
```

```
3
4
5
$ ./monotony.x < tests/test1.in
strictly increasing
$ cat tests/test3.in
7
7
7
7
7
$ ./monotony.x < tests/test3.in
constant
```

Spécifications des entrées Le programme lit sur son entrée standard des nombres qui sont positionnés un par ligne. Chaque nombre est dans les bornes de valeurs entre `INT_MIN` (-2147483648) et `INT_MAX` (2147483647). Au maximum, il y aura 1000 de données à traiter.

Spécifications des sorties Le programme affiche sur sa sortie standard différents messages suivis d'un saut de ligne. Les messages sont :

- `constant` quand la séquence est constante ;
- `increasing` quand la séquence est croissante mais pas strictement (des portions de séquence sont constantes) ;
- `strictly increasing` quand la séquence est strictement croissante ;
- `decreasing` quand la séquence est décroissante mais pas strictement (des portions de séquence sont constantes) ;
- `strictly decreasing` quand la séquence est strictement décroissante ;
- `not monotone` quand la séquence n'est pas monotone.

Exercice 6 – Entrelacement (2.5 pts)

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

Description Des chaînes de caractères ont été entrelacées et il faut pouvoir les isoler. L'objectif de ce programme est de lire des chaînes de caractères entrelacées, lues sur l'entrée standard, et d'afficher les chaînes reconstituées sur la sortie standard.

Exemple On considère que le programme est un fichier nommé `interplay.x`. Et nous considérons le fichier de test, `test2.in` qui se situe dans le répertoire `tests`,

```
$ at tests/test2.in
2
r
p
e
a
m
u
i
l
$ /interplay.x < tests/test2.in
remi
paul
```

Spécifications des entrées Le format de l'entrée sera le suivant. Sur une ligne un entier n qui donne le nombre de chaînes qui sont entrelacées. Les lignes suivantes seront composée d'un caractère appartenant à chaque chaîne de caractères. Remarque, pour un problème donné, les chaînes auront forcément la même longueur et seront constituées uniquement de caractères en minuscule sans accent.

Spécifications des sorties Le programme devra afficher les chaînes de caractères reconstituées, une par ligne et terminera l'affichage par un saut de ligne.

Exercice 7 – Disjonction d'ensembles (2.5 pts)

Le matériel pour cet exercice est donné dans le répertoire `exo7`.

Description L'objectif de ce programme est de vérifier qu'un certain nombre n d'ensemble d'entiers sont disjoints deux à deux.

Exemple On considère que le programme est un fichier nommé `set-packing.x`. Et nous considérons le fichier de test, `test3.in` qui se situe dans le répertoire `tests`,

```
$ cat tests/test3.in
3
1
2
3
0
3
4
5
0
6
7
8
9
$ ./set-packing.x < tests/test3.in
OVERLAP
```

Spécifications des entrées L'entrée suivra le format suivant :
- Un entier n strictement positif dont la valeur est comprise entre 1 et `INT_MAX` (2147483647).
- Une répétition de
  - une séquence de nombres (un par ligne) représentant les éléments d'un ensemble. Les nombres auront une valeur comprise entre 1 et `INT_MAX` (2147483647) ;
  - un séparateur d'ensembles représenté par le caractère spécial 0.

Spécifications des sorties Le programme affichera un message suivi d'un saut de ligne. Les messages seront :
- `OVERLAP` si au moins deux ensembles dans la liste des ensembles d'intersectent ;
- `DISJOINT` si tous les ensembles sont disjoints deux à deux.

Exercice 8 – Arbres binaires complets (2.5 pts)

Le matériel pour cet exercice est donné dans le répertoire `exo8`.

Description Le programme doit construire un arbre binaire complet dont la hauteur est donnée en entrée. Les valeurs contenues dans les nœuds sont également données en lisant l'entrée standard. Le programme affichera la valeur des nœuds en parcourant l'arbre dans l'ordre infixe.

Exemple On considère que le programme est un fichier nommé `building-tree.x`. Et nous considérons le fichier de test, `test2.in` qui se situe dans le répertoire `tests`,

```
$ cat tests/test2.in
2
a
l
e
x
a
n
d
$ ./building-bitree.x < tests/test2.in
xlaaned
```

Spécifications des entrées L'entrée suivra le format suivant :

— sur la première ligne un entier strictement positif n, et dont la valeur maximale sera 20, donnant la hauteur de l'arbre binaire.

— $2^n + 1$ caractères, un par ligne, donnant la valeur des nœuds de l'arbre à construire. Le premier caractère représentant la valeur de la racine, le second caractère le fils gauche de la racine, la troisième valeur le fils droit de la racine, la quatrième le fils gauche du fils gauche de la racine, la cinquième valeur le fils droit du fils gauche de la racine, ...

On ne considérera que des caractères en minuscule sans accent.

Spécifications des sorties Le programme affichera sur une seule ligne suivie d'un retour à la ligne, la chaîne de caractères dont les caractères sont donnés en parcours l'arbre binaire dans un parcours en ordre infixe.