ENSTA
2024-2025

Résolution de problème algorithmique

Examen rattrapage - 1 juillet 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)**.
— **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.

— Le sujet comporte 9 pages et l'examen dure **2h00**.

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-rattrapage-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.

Pour la mise au point de vos programmes, la commande

```
$ ./mon_programme.x < tests/test1.in
```

vous permettra d'analyser le comportement de votre programme avec un jeu de données précis sans comparaison avec la sortie attendue.

À SOUMETTRE

Exercice 1 – Taxe sur la valeur ajoutée (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 une liste de flottants sur son entrée standard (un par ligne) et affiche sur sa sortie standard (un par ligne) le produit du flottant par 1.2.

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

```
$ cat tests/test1.in
2.1
$ ./ttc.x < tests/test1.in
2.52
$
```

Spécifications des entrées L'entrée est un flottant en double précision (double) qui est lu sur l'entrée standard.

Spécifications des sorties La sortie est un flottant en double précision (double). Le résultat des calculs est affiché, avec seulement deux chiffres après la virgule, sur la sortie standard avec un saut de ligne en fin.

Exercice 2 – Suite arithmétique (2.5 pts)

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

Description L'objectif de cet exercice est de programmer un générateur de suite arithmétique. Ce programme doit lire sur son entrée strandard trois entiers r, u_0 et n avec r la raison de la suite, u_0 le premier terme de la suite et n le nombre d'éléments de la suite. Il affichera, un par ligne, la séquence d'entiers.

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

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

Spécifications des entrées Le programme lit sur son entrée standard trois entiers (un sur chaque ligne). De plus, $r > 0$, u_0 peut prendre des valeurs entre `INT_MIN` (-2147483648) et `INT_MAX` (2147483647) et $n > 0$.

Spécifications des sorties Affichez les nombres un par ligne suivi d'un saut de ligne.

Exercice 3 – Nombres premiers (2.5 pts)

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

Description L'objectif de cet exercice est de lire sur son entrée deux entiers n et m et doit afficher les nombres premiers compris entre n et m inclus. Pour calculer la suite des nombres premiers, vous vous appuierez sur l'algorithme décrite par le Crible d'Eratosthène dont le pseudo-code est

```
Crible d'Ératosthène
    entrée : N > 1 entier
    sortie : la liste de tous les nombres premiers <= N
    L = tableau de booléens de taille N, initialisés à Vrai
    L[1] = Faux
    Pour i de 2 à N
        Si L[i]
            Pour j allant de i*i à N par pas de i
            on peut commencer à i*i car tous les multiples de i inférieurs à i*i ont déjà été rayés
                L[j] = Faux
            Fin pour
        Fin si
    Fin pour
    Retourner La liste des i de 2 à N tels que L[i] est vrai
Fin fonction
```

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

```
$ cat tests/test1.in
2 10
$ ./prime.x < tests/test1.in
2
3
5
7
```

Spécifications des entrées Le programme lit sur son entrée standard deux entiers n et m. n représenter la borne basse de l'intervalle et m représente la borne haute de l'intervalle dans lequel les nombres premiers sont considérés pour être affchés.

Spécifications des sorties Le programme affiche sur sa sortie standard, un par ligne, la liste des nombres premiers considérés. A noter que si la valeur de $m < 1$ ou $n > m$ alors le message d'erreur "Something wrong happens : max <= 1, min > max, ..." doit être affiché suivi d'un saut de ligne.

Exercice 4 – Entier couicable (2.5 pts)

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

Description Un nombre n est dit couicable si la somme des chiffres de sa partie droite est égale à la somme des chiffres de sa partie gauche. Le nombre de chiffres composant le nombre étudié doit être pair. Le programme lira sur son entrée standard un entier $n > 0$.

Exemple On considère que le programme est un fichier nommé `couicable.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
256823
$ cat tests/test2.in
12345678
$ ./couicable.x < tests/test1.in
256823 est couicable
$ ./couicable.x < tests/test2.in
12345678 n'est pas couicable
```

Spécifications des entrées Le programme lit sur son entrée standard un entier stricement positif inférieur ou égal à `INT_MAX` (2147483647).

Spécifications des sorties Le programme affiche sur sa sortie standard un message de la forme suivante si le nombre est couicable

```
n est couicable
```

ou si le nombre n'est pas couicable

```
n n'est pas couicable
```

où n est le nombre considéré, suivi d'un saut de ligne.

Exercice 5 – Opération booléenne (2.5 pts)

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

Description L'objectif de ce programme est d'appliquer une opération booléenne ("ou" logique ou "et" logique) entre deux séquences de nombres entiers positifs compris entre 0 (représentant la valeur de vérité "Faux") et 1 (représentant la valeur de vérité "Vrai") .

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

```
$ cat tests/test1.in
0 0
1 1
0 1
0 1
-1 -1
1
$ /booleen.x < tests/test1.in
0
1
1
1
```

Spécifications des entrées Le programme lit sur son entrée standard deux séquences de valeurs booléennes de même longueur suivi d'une opération à effectuer qui sera modélisé par un entier :

- 1 pour le "et" logique
- 2 pour le "ou" logique

Entre les séquences de booléens et l'opération il y aura un séparateur matérialisé par la ligne "-1 -1"

Spécifications des sorties La lise des entiers représentant le résultat de l'opération booléenne, un par ligne.

Exercice 6 – Prison (2.5 pts)

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

Description Nous sommes dans une prision dans l'espace réservée à des monstres.

Votre objectif est d'assigner un monstre parmis 4 espèces différentes (vampire, fantôme, ogre et troll) à chacun des nouvelles cellules de cette prison. Ces monstres sont très territoriaux et vous devez donc vous assurer qu'aucun monstre ne peut voir un autre monstre de sa propre espèce depuis son enclos. Le responsable de la prison vous a envoyé un fichier avec la visibilité des différentes cellules et vous devez assigner un type de monstre à chaque cellule. A la fin du processus, toutes les cellules doivent avoir un type de monstre assigné.

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

```
$ cat tests/test1.in
8
1-2
3-1
4-5
4-8
1-7
1-4
7-1
2-4
1-8
6-7
2-3
1-5
1-6
7-6
7-8
2-5
7-1
3-4
5-6
7-8
$ ./monsters.x < tests/test1.in
1 1
2 2
3 3
4 4
5 3
6 2
7 3
8 2
```

Spécifications des entrées La première ligne du fichier d'entrée (lu sur l'entrée standard) contient le nombre de cellules ($n <= 100$). Chacune des lignes suivantes contient une restriction de visibilité : 1-3 signifie que les monstres de la cellule 1 peuvent voir les monstres de la cellule 3 et que les monstres de la cellule 3 peuvent voir les monstres de la cellule 1. Notez que le gestionnaire de la prison n'est pas une personne très bien organisée et que, par conséquent, certaines données figurant dans le fichier peuvent être redondantes.

Spécifications des sorties L'affichage sur la sortie standard contiendra une par une de toutes les assignations des cellules possibles. La sortie se compose d'une ligne par cellule et chaque ligne contient le numéro de la cellule suivi du type de monstre (1= vampire, 2= fantôme, 3= orgre, 4= troll). L'assignation des cellules doit apparaître dans l'ordre croissant.

Exercice 7 – k-ieme plus petit (2.5 pts)

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

Description L'objectif de ce programme est de trouver le k-ième plus petit élément d'une liste d'entiers.

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

```
$ cat tests/test1.in
5
1
2
3
4
5
6
7
$ ./findk.x < tests/test1.in
5
```

Spécifications des entrées L'entrée suivra le format suivant :
- sur la première ligne un entier strictement positif n, et dont la valeur maximale sera 1000, donnant la valeur de k.
- une liste d'entiers non triés.

Spécifications des sorties Le programme affichera sur une seule ligne suivie d'un retour le message :
- le k-ième plus petit entier trouvé
- le message "Not found" la valeur de k est plus grande que le nombre d'entiers dans la liste.

Exercice 8 – Campus (2.5 pts)

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

Description Le campus IP Paris est en cours de renovation et des pré-fabriqués doivent être installés pour accueillir les étudiants en cours pendant que les amphithéâtres sont en travaux. Il faut relier chaque pré-fabriqués à l'électricité et on se demande comment minimiser la longeur du cablage pour réaliser cette tâche.

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

```
$ at tests/test1.in
4
1 1 2
1 2 3
2 4 4
3 3 4
```

```
$ ./campus.x < tests/test1.in
1 -{1.0}-> 2
1 -{2.0}-> 3
3 -{3.0}-> 4
```

Spécifications des entrées Le programme lira sur son entrée standard sur une ligne un nombre entier positif indiquant le nombre de pré-fabriqués à relier entre eux avec des cables électriques/ Un certain nombre de lignes suivront et auront la forme suivante :

$$x \quad y \quad z$$

x et z représente deux numéros de pré-fabriqués (entiers strictement positifs) et z représente le métrage de cable de cuivre nécessaire (flottant en double précision) pour relier x et y.

Spécifications des sorties Le programme affichera la liste des connexions électriques entre pré-fabriqués, une par ligne. Plus précisément chaque connexion entre deux pré-fabriqués aura la forme

```
x -{p}-> y
```

avec x et y des entiers et p le métrage afficher avec un chiffre après la virgule.