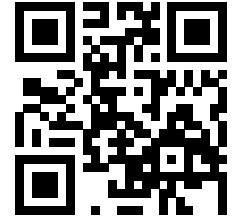


NOM : Hanon Ymous
(000000)
Place : 0

#0000



Information, Calcul et Communication (SMA/SPH) : Examen II

22 novembre 2018

INSTRUCTIONS (à lire attentivement)

IMPORTANT! Veuillez suivre les instructions suivantes à la lettre sous peine de voir votre examen annulé dans le cas contraire.

1. Vous disposez d'une heure quarante-cinq minutes pour faire cet examen (9h15 – 11h00).
2. Vous devez **écrire à l'encre noire ou bleu foncée**, pas de crayon ni d'autre couleur.
N'utilisez **pas non plus de stylo effaçable** (perte de l'information à la chaleur).
3. Vous avez droit à toute documentation papier.
En revanche, vous ne pouvez pas utiliser d'ordinateur personnel, ni de téléphone portable, ni aucun autre matériel électronique.
4. Répondez aux questions directement sur la donnée, **MAIS** ne mélangez pas les réponses des deux exercices!
Ne joignez aucune feuilles supplémentaires ; **seul ce document sera corrigé.**
5. Lisez attentivement et *complètement* les questions de façon à ne faire que ce qui vous est demandé. Si l'énoncé ne vous paraît pas clair, ou si vous avez un doute, demandez des précisions à l'un des assistants.
6. L'examen comporte deux exercices indépendants, qui peuvent être traités dans n'importe quel ordre, mais qui ne rapportent pas la même chose (les points sont indiqués, le total est de 56). Toutes les questions comptent pour la note finale.



Question 1 — Moyenne et écart-type [sur 28 points]

On s'intéresse dans cette question à écrire un programme qui calcule la dispersion statistique de certaines lettres d'un tableau de chaînes de caractères autour de leur moyenne d'apparition (on parle d'« écart-type », expliqué plus loin). Typiquement, on souhaiterait avoir un résultat similaire à cet exemple :

```
Entrez une chaîne de caractères : court
  Entrez une chaîne d'au moins 8 caractères, ou la chaîne vide pour terminer.
Entrez une chaîne de caractères : ok j'ai compris !
Entrez une chaîne de caractères : mais ce n'est pas si simple de trouver \
  des exemples pertinents.
Entrez une chaîne de caractères : surtout avec un seul y en tout !
Entrez une chaîne de caractères : et un seul c par phrase ;-)
Entrez une chaîne de caractères :
Pour a, moyenne = 1.5, ecart-type = 0.57735
Pour c, moyenne = 1, ecart-type = 0
Pour e, moyenne = 4.25, ecart-type = 4.71699
Pour i, moyenne = 1.5, ecart-type = 1.91485
Pour l, moyenne = 1, ecart-type = 0.816497
Pour m, moyenne = 1, ecart-type = 1.41421
Pour n, moyenne = 1.5, ecart-type = 1.29099
Pour o, moyenne = 1.25, ecart-type = 0.957427
Pour p, moyenne = 1.75, ecart-type = 1.70783
Pour t, moyenne = 2.25, ecart-type = 2.06155
Pour u, moyenne = 2, ecart-type = 2.16025
Pour y, moyenne = 0.25, ecart-type = 0.5
```

Le programme complet visé calculera, pour chaque caractère appartenant à la liste (a, c, e, i, l, m, n, o, p, t, u, y), la moyenne et l'écart-type de son nombre d'apparition par phrase (expliqués plus loin) sur un ensemble de phrases entrées par l'utilisateur ; puis en affichera le résultat comme dans l'exemple ci-dessus.

Ce programme est constitué de la fonction `main()` suivante :

```
int main()
{
    print_stats(get_strings(), "aceilmnoptuy");
    return 0;
}
```

On vous demande ici de compléter *certaines parties* de ce programme, en utilisant le langage C++ et une approche « procédurale » avec le moins de duplication de code possible. En plus des fonctions et types de données demandés dans cet énoncé, vous êtes libres d'ajouter vos propres fonctions ou types de données intermédiaires si nécessaire.

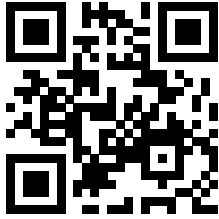


Question 1.1 – Comptes [sur 9.5 points]

Pour commencer, on vous demande d'écrire ici :

- une fonction `char_count()` qui, pour un caractère et une chaîne de caractères passés en paramètres, calcule le nombre de fois que ce caractère est contenu dans la chaîne ;
p.ex. `char_count('a', "blabla")` retourne 2 ;
- une *autre* fonction `char_count()` qui, pour un caractère et *un tableau de chaînes* de caractères passés en paramètres, calcule le nombre de fois que ce caractère est contenu dans chacune des chaînes du tableau et retourne le tableau d'entiers correspondant (dans le même ordre) ;
p.ex. `char_count('a', { "blabla", "toc-toc", "tic-tac" })` retourne (2,0,1).

suite au dos ➞



Question 1.2 – Moyenne [sur 4.5 points]

On s'intéresse maintenant à calculer la moyenne d'une collection de n valeurs x_i ($1 \leq i \leq n$) suivant la formule

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i.$$

Écrivez ici une fonction `mean()` qui calcule la moyenne d'un tableau de nombres entiers reçu en argument.

Note : la moyenne d'une collection de nombres entiers n'est pas nécessairement un entier.

Question 1.3 – Écart-type [sur 8 points]

La formule pour calculer l'écart-type σ_x d'une collection de n valeurs x_i ($1 \leq i \leq n$) est la suivante :

$$\sigma_x = \sqrt{\frac{1}{n-1} \left(\sum_{i=1}^n x_i^2 \right) - \frac{n}{n-1} \bar{x}^2}$$

où \bar{x} représente la moyenne calculée précédemment.

Écrivez sur la page suivante une fonction `std_dev()` qui calcule l'écart-type et la moyenne d'un tableau de nombres reçu en argument. Cette fonction devra (d'une façon ou d'une autre) « retourner » l'écart-type et la moyenne. Vous devez de plus utiliser *exactement* la formule donnée ci-dessus (et non pas une autre formule équivalente).

Question 1

Anonymisation : #0000
p. 5



Code de la fonction `std_dev()` :

suite au dos 



Question 1.4 – Affichage des résultats [sur 6 points]

Pour finir, nous vous demandons d'écrire ici

- une fonction `print_stats()` qui prend en paramètres un caractère et un tableau de chaînes de caractères et qui affiche sur la sortie standard la moyenne et l'écart-type du nombre de fois que ce caractère est présent dans les chaînes du tableau;

p.ex. `print_stats('a', { "blabla", "toc-toc", "tic-tac" });` affiche

Pour `a`, `moyenne = 1`, `ecart-type = 1`

puisque la moyenne du tableau $(2, 0, 1)$ vaut 1 et son écart-type vaut 1 aussi;

- une *autre* fonction `print_stats()` qui généralise la fonction précédente à une chaîne de caractères en appliquant cette dernière tour à tour à chacun des caractères de la chaîne de caractères fournie; voir la fonction `main()` et l'exemple de déroulement fournis en début d'exercice.



Question 2 — Un peu de musique [sur 28 points]

Dans cet exercice, on aimerait réaliser une application permettant de gérer des playlists, listes de fichiers de musique. Chaque fichier de musique possède plusieurs caractéristiques :

- le « nom » du fichier : chemin absolu où est il stocké sur le disque dur ;
- la durée de la chanson (en nombre entier de secondes, p.ex. 321) ;
- la taille du fichier (nombre entier de Ko, p.ex. 65432) ;
- son taux de compression (pourcentage, p.ex. 0.33).

On vous demande ici de compléter *certaines parties* d'un tel programme, en utilisant le langage C++ et une approche « procédurale » avec le moins de duplication de code possible. En plus des fonctions et types de données demandées dans cet énoncé, vous êtes libres d'ajouter vos propres fonctions ou types de données intermédiaires si nécessaire.

Question 2.1 – Structures de données [sur 3 points]

Commencez par définir (ici) les types suivants :

- `Morceau`, pour représenter un fichier de musique tel que présenté ci-dessus ;
- `collectionMusique`, pour stocker un ensemble de fichiers de musique (playlist).



Question 2.2 – Maxima [sur 3.5 points]

On souhaite avoir une fonction `max()` qui, pour une playlist donnée en paramètre, sort un `Morceau` dont le champ « nom du fichier » est vide et dont chacun des autres champs contient le maximum de la valeur correspondante dans la playlist reçue en paramètre (p.ex. la durée de ce « morceau de musique » artificiel est la plus grande des durées de la playlist, etc.).

Écrivez sur la page suivante (au dos de cette feuille) le code d'une telle fonction.



Anonymisation : #0000

p. 8

Question 2

Code de la fonction `max()` :

Question 2.3 – Fréquence d'échantillonnage maximale [sur 4.5 points]

Codez une fonction `ratio_max()` qui, pour chaque fichier de musique d'une playlist passée en paramètre, calcule le ratio défini mathématiquement par $(\text{taille}/\text{durée})/\text{taux_compression}$ et retourne son maximum sur toute la playlist.

On adoptera la convention que ce ratio est nul si le taux de compression ou la durée est nul(le).



Question 2.4 – Affichage [sur 3.5 points]

Écrivez une fonction d’affichage pour les playlists. Cette fonction affichera (voir exemple ci-dessous) :

- le nombre de morceaux de la playlist ;
- les durée, taille, taux de compression et ratio maximaux ;
- la liste des morceaux avec leurs caractéristiques propres.

Par *exemple* (il n’est pas demandé de reproduire exactement ce format, seules les *informations* affichées nous importent) :

```
Playlist de 5 morceaux :
durée max. : 189 s
taille max. : 1500 Ko
taux compr. max. : 0.48
ratio max. : 35.8878
chemin          duree taille  taux
- /home/chaps/Musique/a.mp3    121  1433  0.33
- /home/chaps/Musique/b.ogg    189  1500  0.42
- /home/chaps/Musique/c.wav     73   512  0.21
- /home/chaps/Musique/d.flac   104  1000  0.48
- /home/chaps/Musique/e.soX    176   999  0.19
```

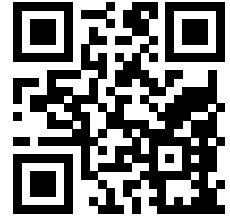
suite au dos ➡



Question 2.5 – Suppression de gros fichiers [sur 8.5 points]

Créez une fonction `nettoyer()` qui supprime d'une playlist passée en paramètre tous les fichiers de musique dont la taille est strictement supérieure à une valeur donnée en paramètre.

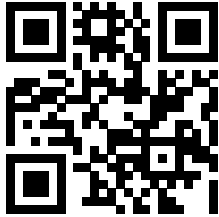
Par exemple, sur la playlist affichée dans la sous-question précédente, l'appel `nettoyer(playlist, 1000)` supprimerait les deux premiers fichiers (`a.mp3` et `b.ogg`) et garderait les autres, pas nécessairement dans le même ordre (nos playlist ne sont pas ordonnées).



Question 2.6 – Intersection de playlists [sur 5 points]

On souhaite enfin créer l'intersection de deux playlists. On considérera que deux fichiers de musique sont égaux s'ils ont simplement le même nom de fichier.

Coder ici une fonction `intersection()` réalisant cette fonctionnalité.



Anonymisation : #0000
p. 12

Question 2

(si nécessaire, place supplémentaire pour répondre ; mais
veuillez clairement indiquer le numéro de la question à laquelle vous répondez ici)