EIDGENÖSSISCHE TECHNISCHE HOCHSCHULE – LAUSANNE
POLITECNICO FEDERALE – LOSANNA
SWISS FEDERAL INSTITUTE OF TECHNOLOGY – LAUSANNE

**Faculté Informatique et Communications**
Cours de Programmation I, sections MA et PH
Chappelier J.-C.

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

NOM : Hanon Ymous
(000000) #0000
**Place : 1**

# Programmation I (SMA/SPH) : SÉRIE NOTÉE

24 novembre 2016

## INSTRUCTIONS (à lire attentivement)

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

1. Vous disposez d'une heure quarante-cinq minutes pour faire cet examen (10h15 - 12h00).
2. Vous devez **écrire à l'encre noire ou bleu foncée**, pas de crayon ni d'autre couleur.
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 ; utilisez aussi le verso des feuilles, **MAIS** n'utilisez *que* le verso de la feuille sur laquelle se trouve la question, et non **pas** celui de la feuille précédente !
   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 65). Tous les exercices comptent pour la note finale.

## Question 1 — Chiens et chats [sur 25 points]

Le but de cet exercice est d'écrire un programme permettant de simuler l'évolution d'une population de chiens et de chats dans un environnement convivial. Chaque mois, le nombre de chats et de chiens évolue :

- si une des deux populations est supérieure à l'autre, elle est diminuée (vente) ;
- si une des deux populations est inférieure à l'autre, elle est augmentée (achat) ;
- si une population disparaît (devient nulle), des animaux sont ajoutés à cette population.

Plus précisément, on demande à l'utilisateur d'entrer :

- le nombre initial de chats (strictement positif) ;
- le nombre initial de chiens (strictement positif) ;
- le nombre de mois (strictement positif) pour faire la simulation ;
- et le nombre d'animaux (strictement positif) à ajouter si une population s'est éteinte.

Tous ces nombres doivent être traités comme des entiers.

La structure générale du programme à produire est la suivante :

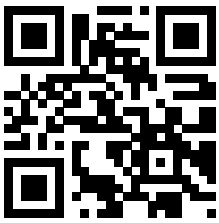
- obtenir les valeurs initiales demandées ci-dessus en utilisant une fonction `demander_nombre` (à écrire, décrite ci-dessous) ;
- afficher le nombre de chats et de chiens au départ ;
- ensuite, pour chaque mois :
  - calculer le nombre de chats et de chiens de la façon suivante :
    - si le nombre de chats est supérieur ou égal à celui des chiens, diminuer le nombre de chats en faisant appel à une fonction `diminution` (à écrire, décrite ci-dessous) et augmenter le nombre de chiens en faisant appel à une fonction `augmentation` (à écrire, décrite ci-dessous) ;
    - sinon (la population chiens est strictement supérieure à la population de chat), faire le contraire : diminuer le nombre de chiens et augmenter le nombre de chats en utilisant les deux mêmes fonctions, `diminution` et `augmentation`.

    Il faudra prêter attention à faire chaque calcul de nouvelle population à partir des populations du mois précédent.
  - afficher les populations ainsi calculées.

En plus du `main()`, on vous demande donc d'écrire trois fonctions :

- `demander_nombre` qui reçoit une chaîne de caractères à afficher comme question et attend de l'utilisateur un entier strictement positif (elle repose la question si ce n'est pas le cas) ;
- `diminution` qui prend trois nombres entiers `x`, `y` et `z` en arguments et retourne un nombre entier calculé comme suit :
  - `x` est diminué du total de « 20% de sa propre valeur » et de « `y` divisé par trois » (cette division étant effectuée au sens des nombres *réels*, pas de division entière ici) ;
  - si le résultat de cette diminution est négatif ou nul, le résultat sera alors simplement `z` (`z` représente donc le nombre « *d'animaux à ajouter lorsqu'une population disparaît* » mentionné au début).
- `augmentation` qui prend deux nombres entiers `x` et `y` en arguments et retourne simplement `x` plus 1 plus 40% de `y`.

Écrivez votre programme sur les deux pages suivantes (pages 3 et 4).



# Question 2 — Symétries de chaînes [sur 40 points]

*Palindrome* : séquence de caractères qui est la même quand on la parcourt de gauche à droite ou de droite à gauche. Par exemple, « *kayak* », « *ressasser* » ou encore « *engage le jeu que je le gagne* » sont des palindromes.

Le but du programme considéré dans cette question est détecter des palindromes dans des listes de citations en différentes langues. Nous allons donc tout d'abord modéliser des ensembles de citations, puis y rechercher des palindromes exacts ou approximatifs.

Un palindrome sera exact si la chaîne de caractères en question est parfaitement symétrique y compris les caractères non alphabétiques et les majuscules/minuscules. Par exemple, « *Madam kayak madaM* » (notez la majuscule finale) est un palindrome exact.

Un palindrome sera approximatif si la chaîne de caractères en question est symétrique lorsque l'on ignore les caractères non alphabétiques et la différence majuscule/minuscule. Par exemple, « *Engage le jeu, que je le gagne !* » est un palindrome approximatif.

## Question 2.1 – Structures de données [sur 4 points]

Pour ce programme, vous aurez besoin de certaines structures de données. Pour représenter une citation, définissez un type `Citation` qui contient : le texte de la citation, le nom de son auteur, et son année.

Par exemple, on voudra représenter la citation « *Ce reptile lit Perec.* » de Georges Perec en 1969[1].

Pour représenter un ensemble de citations dans une langue donnée, définissez un type `Liste_citations` comprenant une langue (`string`) et un ensemble de citations (dans une langue donnée).

Écrivez ici la définition des *types* de données qui vous semblent appropriés pour représenter les données décrites ci-dessus :

1. Je vous ai épargné son « grand palindrome » de plus de 5000 lettres (1247 mots).

## Fonctions-outils fournies

On supposera fournies les fonctions-outils suivantes :

- `est_alphabetique` qui décide si un caractère reçu est un caractère alphabétique ou non (majuscule ou minuscule) ; ainsi, `est_alphabetique('F')` et `est_alphabetique('f')` retournent `true`, alors que `est_alphabetique('!')` et `est_alphabetique(' ')` retournent `false` ;
- `to_lower` qui pour un caractère en majuscule retourne son équivalent en minuscule ; ainsi, `to_lower('F')` retourne `'f'` ; elle laisse inchangés les caractères minuscules ; ainsi, `to_lower('f')` retourne également `'f'` ;
- `void print(Citation const& a_afficher);` permettant d'afficher une `Citation` ;
- `void print(string const& chaine, vector<size_t> const& positions_longueurs);` permettant d'afficher des morceaux d'une chaîne tels qu'indiqués plus tard dans la donnée sous forme de liste de positions de départ et longueurs ; ce format sera détaillé au moment utile.

## Question 2.2 – Palindrome [sur 11 points]

Pour pouvoir chercher des citations palindromes, il nous faut commencer par avoir une fonction qui nous dit si une chaîne reçue en paramètre est un palindrome ou non.

Pour cela, définissez une fonction `est_palindrome` qui reçoit une chaîne de caractères et un paramètre booléen indiquant si l'on teste des palindromes exacts (lorsqu'il est `true`) ou approximatifs. La valeur par défaut de ce second paramètre est `true`.

La fonction `est_palindrome` retournera donc

- `true` si la chaîne reçue est un palindrome exact ;
- `true` si la chaîne reçue est un palindrome approximatif (ou exact) et que le second paramètre reçu est `false` ;
- `false` dans tous les autres cas.

Pour l'implémentation, vous êtes libres de choisir un algorithme itératif ou un algorithme récursif. Pensez cependant à utiliser (entres autres) la fonction-outil `est_alphabetique` pour sauter les caractères non alphabétiques dans le cas d'une recherche approximative.

Si vous n'arrivez pas à écrire la version pour palindromes approximatifs, vous pouvez néanmoins écrire la version pour palindromes exacts.

Écrivez ici et sur la page suivante votre définition de la fonction `est_palindrome` :

## Question 2.3 – Tous les mots palindromes [sur 12 points]

On s'intéresse maintenant à trouver tous les *mots* palindromes exacts dans une chaîne de caractères ; un « mot » étant défini comme une séquence de caractères alphabétiques.

Par exemple, pour la chaîne « *Ce kayak n'a ni rotor ni radar.* », on voudra ressortir les mots « *kayak* », « *n* », « *a* », « *rotor* » et « *radar* ».

La réponse sera fournie en valeur de retour sous forme d'une série de nombres indiquant respectivement la position (à partir de 0) et la longueur de chaque mot palindrome trouvé.

Par exemple, pour l'exemple précédent, la sortie fournie sera la liste $(3, 5, 9, 1, 11, 1, 16, 5, 25, 5)$ pour signifier :

- le premier mot trouvé (« *kayak* ») commence à la position 3 et est de longueur 5 ;
- le second mot trouvé (« *n* ») commence à la position 9 et est de longueur 1 ;
- etc.

Pour trouver de tels mots dans une chaîne reçue en paramètre, nous vous proposons l'algorithme suivant :

- parcourir tous les caractères ;
- dès que l'on a un caractère alphabétique : rechercher le prochain caractère non alphabétique (ou la fin de chaîne) à partir de ce caractère ci ; cela donne un mot ;
- tester si ce mot est un palindrome exact ;
- si c'est le cas, ajouter sa position de départ et sa longueur aux résultats à retourner.

Définissez ici (et sur la page suivante si nécessaire) une fonction `mots_palindromes` correspondant à la description donnée ci-dessus.

Pour rappel, les `string` de C++ ont une « fonction spécifique » `substr` qui prend deux paramètres, une position `p` et une longueur `L`, et retourne la sous-chaîne de longueur `L` à partir de la position `p`.

## Question 2.4 – Recherche de citations particulières [sur 13 points]

Pour finir, nous voulons écrire une fonction `affiche_citations_palindromiques` qui à partir d'un ensemble de `Liste_citations` (dans langues différentes) et de la donnée d'une langue (`string`) et deux dates (départ et fin) affiche toutes les citations dans langue donnée et comprises entre les deux dates données qui sont des palindromes ou qui contiennent au moins un mot palindrome exact.

Cette fonction `affiche_citations_palindromiques` devra donc :

- rechercher toutes les citations de la langue donnée,
- qui en plus sont comprises entre les deux dates données ;
- et, pour de telles citations :
  - vérifier si c'est un palindrome exact ou approximatif et l'afficher si c'est le cas ;
  - vérifier si elle contient des mots palindromes exacts et les afficher tous si c'est le cas.

Pour l'affichage des mots palindromes exacts, la deuxième fonction-outil `print` mentionnée en page 6 est conçue pour fonctionner directement avec comme second argument la sortie de la fonction `mots_palindromes`.

Écrivez ici (et sur les pages suivantes si nécessaire) votre définition de la fonction `affiche_citations_palindromiques` :