INF1005C - PROGRAMMATION PROCÉDURALE
Travail dirigé No. 3
Structures de décision et répétition, lecture de fichiers, tableaux et enregistrements.

Objectifs : Permettre à l'étudiant de se familiariser avec les différentes structures du langage C++, les fichiers textes, la manipulation des tableaux statiques ainsi que les enregistrements (structures).
Durée : Deux séances de laboratoire.
Remise du travail : dimanche 26 mai, avant 23 h 30.
Travail préparatoire : Lecture du guide de codage, des exercices et rédaction des algorithmes.
Documents à remettre : Sur le site Moodle des travaux pratiques, vous remettrez l'ensemble des fichiers .cpp compressés dans un fichier .zip en suivant **la procédure de remise des TDs.**
Seulement 3 exercices seront corrigés (mais vous devez les faire tous). Le barème de correction est présenté sur Moodle.

Directives particulières

- Utilisez le principe DRY (Don't Repeat Yourself). Considérez utiliser une boucle si vous devez exécuter plusieurs fois la même instruction. À chaque endroit où vous remarquez une duplication de code (vous avez écrit les mêmes opérations plus d'une fois) et qu'il n'est pas possible de l'éliminer avec les structures vues, indiquez-le en commentaire.
- Respecter le guide de codage, les points pertinents pour ce travail sont donnés en annexe à la fin.
- Faites attention à votre choix de structures de répétition. Rappelez-vous qu'une boucle `for` est utilisée lorsque le nombre d'itérations est connu au moment où elle commence, au contraire de la boucle `while` (ou `do`/`while`).
- Vos programmes doivent fonctionner avec les fichiers fournis, mais également avec tout autre fichier qui respecterait le format requis.
- Compiler avec /W4. Lors de la compilation avec Visual Studio 2017, votre programme ne devrait pas afficher d'avertissements (« warnings »).
- Comme dans le TD2, les entrées/sorties doivent être faites avec messages significatifs et vous n'avez pas à valider les entrées si la question ne le demande pas.

---

1. **Pair/impair** : Écrivez un programme qui lit du clavier un nombre, puis détermine si ce nombre est pair ou impair.

   Exemples d'affichage :

```
Entrez un nombre : 3
Ce nombre est impair.
```

```
Entrez un nombre : 6
Ce nombre est pair.
```

2. **Taxe :** Écrivez un programme qui demande et lit du clavier le prix d'un article avant taxes et qui calcule son prix avec TPS et TVQ; tel que sur une facture standard, affichez séparément le montant de la TPS, de la TVQ et le prix incluant les taxes, le tout avec 2 chiffres après la virgule. Vous devrez vérifier la validité du prix de l'article entré par l'utilisateur, c'est-à-dire que c'est bien un nombre et qu'il est positif. Redemandez d'entrer le prix lorsqu'il n'est pas valide.

3. **Nombres parfaits :** Écrire un programme pour trouver tous les nombres parfaits dans l'intervalle [1,10000]. Un nombre parfait est un entier naturel qui est égal à la somme de ses diviseurs stricts (incluant 1, mais différent du nombre lui-même). Par exemple 6 a comme diviseurs stricts 1, 2 et 3, et 6=1+2+3, donc 6 est un nombre parfait; même chose pour 28 (=1+2+4+7+14).

4. **Remplacement :** Écrivez un programme qui lit du clavier une chaîne de caractères ainsi que deux caractères. Il affiche ensuite la chaine de caractères en remplaçant les occurrences du 1er caractère par le 2e. Exemple : « programmation », « a » et « i », donne le résultat « progrimmition ». Vous ne devez pas utiliser de fonctions qui n'ont pas été vues et qui pourrait faire automatiquement ce remplacement (dont `std::replace()`).

5. **Mots contenant une lettre :** Écrivez un programme qui lit un nom de fichier et un caractère et affiche tous les mots du fichier qui contiennent ce caractère. Utilisez *LoremIpsum.txt*.

6. **Statistiques d'un texte :** Écrivez un programme qui lit un fichier, puis afficher un histogramme des longueurs de mots (nombre de mots ayant 1 caractère, 2 caractères, etc.). Afficher le nombre d'occurrence de chaque longueur de mot.
    - Vous pouvez supposer qu'aucun mot n'aura plus de 30 lettres.
    - Les mots sont séparés par des espaces ou des retours à la ligne (les traits d'union et apostrophes ne sont pas considérés comme séparant des mots).
    - Vous ne devez pas compter la ponctuation comme étant une lettre du mot. Donc si un mot contient des caractères de ponctuation (trait d'union, apostrophe, point, virgule, etc.), vous devez y soustraire ceux-ci dans le nombre de lettre du mot. Par exemple, le mot «`est?`» contient trois lettres et le mot «`[reus]`» en contient quatre. Utilisez les fonctions présentes dans `<cctype>` pour déterminer si un caractère est alphanumérique ou non.
    - Vous devez passer le texte du fichier une seule fois. L'usager rentre le nom du fichier, le programme valide l'existence du fichier et affiche un simple message si le fichier n'existe pas.
7. **Somme :** Soit la somme $S_n = \sum_{k=1}^{n} \frac{1}{k}$. On admet que S_n tend vers $+\infty$. Écrivez un programme qui permet de déterminer pour un entier n donné par l'utilisateur le plus petit **entier** a tel que $S_n \leq a$. Vous n'avez pas à considérer les problèmes liés à la précision des calculs.

****

Annexe 1 : Points du guide de codage à respecter

Les points du guide de codage à respecter impérativement pour ce TD sont les suivants :
(Voir le guide de codage sur le site Moodle du cours pour la description détaillée de chacun de ces points)

Nouveaux points depuis le TD2 :

14, 52 : portée des variables (noms courts/longs)
21 : pluriel pour les tableaux (int nombres[];)
22 : préfixe n pour désigner un nombre d'objets (int nElements;)
24 : variables d'itération i, j, k mais jamais l
51 : test de 0 explicite (if (nombre != 0))
53-54 : boucles for et while
58-61 : instructions conditionnelles
67-78, 88 : indentation du code et commentaires
83-84 : aligner les variables lors des déclarations ainsi que les énoncés

Points du TD2 :

3 : noms des variables en lowerCamelCase
4 : noms des constantes en MAJUSCULES
25 : is/est pour booléen
27 : éviter les abréviations (les acronymes communs doivent être gardés en acronymes)
29 : éviter la négation dans les noms
33 : entête de fichier avec vos noms et matricules
42 : #include au début (mais après l'entête)
46 : initialiser à la déclaration
47 : pas plus d'une signification par variable
62 : pas de nombres magiques dans le code
63-64 : « double » toujours avec au moins un chiffre de chaque côté du point (i.e. 1.0, 0.5)
79,81 : espacement et lignes de séparation
85 : mieux écrire du code incompréhensible plutôt qu'y ajouter des commentaires
87 : préférer // pour les commentaires