INF1005C PROGRAMMATION PROCÉDURALE

Travail dirigé No. 3
Structures de décision et répétition, Lecture de fichiers,
Énumérations et Tableaux.

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 le type énumération.

Durée : Deux séances de laboratoire.

Remise du travail : Avant 23h30, le jeudi 9 octobre 2014.

Travail préparatoire : Lecture des exercices et rédaction des algorithmes.

Documents à remettre : L'ensemble des fichiers .cpp, dans un dossier compressé (.zip), dans la section « Remise des travaux » pour votre section de laboratoire sur Moodle.

Correction : Seulement 3 exercices seront corrigés, mais vous devez les faire tous.

Directives :

- 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 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 votre programme ne devrait pas afficher d'avertissements (« warnings »).**

---

1. Demander d'entrer un nombre et demander quelle opération appliquer dessus : carré (option 1), racine (option 2) ou cube (option 3). Utiliser une cascade de « if… else if… ». Afficher le résultat à l'écran.

2. Reprendre l'exercice précédent avec une structure « switch… case… ».

3. **Suite de Fibonacci.** Écrire un programme qui détermine la nième valeur (*n* étant fourni en donnée) de la « suite de Fibonacci » définie comme suit :

$$F_1 = 1, F_2 = 1$$
$$F_n = F_{n-1} + F_{n-2} \text{ pour } n > 2$$

4. Max et Min d'une série. Écrire un programme qui trouve la plus grande et la plus petite valeur d'une succession de notes (nombres entiers entre 0 et 20) fournies en données, ainsi que le nombre de fois où ce maximum et ce minimum ont été attribués. On supposera que les notes, en nombre non connu à l'avance, seront terminées par une valeur négative. On s'astreindra à ne pas utiliser de « tableau ». L'exécution du programme pourra se présenter ainsi :

```
Donnez une note (-1 pour finir) : 12
Donnez une note (-1 pour finir) : 13
Donnez une note (-1 pour finir) : 7
Donnez une note (-1 pour finir) : 11
Donnez une note (-1 pour finir) : 7
Donnez une note (-1 pour finir) : -1
Note maximale : 13 attribuée 1 fois
Note minimale : 7 attribuée 2 fois
```

5. Méthode de Newton-Raphson. Le département de mathématiques vous demande d'écrire un programme pouvant calculer la racine carrée de A (A étant un réel strictement positif) au moyen de la méthode de Newton-Raphson et avec une précision de ε fixée par l'usager.
La méthode de Newton-Raphson utilise une suite d'éléments (x_i) définie par :

$$x_0 = \frac{A}{2} \quad \text{et} \quad x_i = \frac{1}{2}(x_{i-1} + \frac{A}{x_{i-1}})$$

Le terme x_i est une valeur approchée de la racine carrée de A avec une précision de ε lorsque

$$\frac{|x_i - x_{i-1}|}{x_{i-1}} \leq \varepsilon$$

Note : le programme devra traiter la suite (x_i) sans l'aide d'un tableau.

6. Gérer une liste de données dans un fichier. Le fichier « scientifiques.txt » contient des informations sur quelques scientifiques du 19e siècle. Chaque ligne de ce fichier se présente sous le format :

Nom du scientifique; Naissance - Mort; Domaine et nationalité

Écrire un programme qui demande à l'usager d'entrer le nom d'un scientifique du 19e siècle. Si ce scientifique existe dans le fichier, le programme doit afficher toutes ses informations. S'il n'existe pas, le programme doit lire à partir du clavier toutes les informations sur ce scientifique et les ajouter au fichier « scientifiques.txt ». Avant d'enregistrer les informations dans le fichier, le programme doit également vérifier que le scientifique entré est du 19e siècle.
Le programme devrait s'exécuter tant que l'usager n'a pas décidé de quitter.
Exemple d'exécution du programme :

```
Entrer le nom d'un scientifique du 19e siecle (Q pour quitter le
programme) : Nikola Tesla
Ingenieur en electronique americain d'origine serbe
Date de naissance : 1856. Date de deces : 1943.
Entrer le nom d'un scientifique du 19e siecle (Q pour quitter le
programme) : Erwin Schrodinger
Ce nom ne figure pas dans la liste.
Entrer sa date de naissance : 1887
Entrer sa date de deces : 1961
Entrer son domaine et sa nationalite: physicien autrichien
```

```
Les informations sont enregistrees dans le fichier.
Entrer le nom d'un scientifique du 19e siecle (Q pour quitter le
programme) : Al Khawarizmi
Ce nom ne figure pas dans la liste.
Entrer sa date de naissance : 980
Entrer sa date de deces : 1037
Entrer son domaine et sa nationalite: mathématicien perse
Il ne s'agit pas d'un scientifique du 19e siecle
Entrer le nom d'un scientifique du 19e siecle (Q pour quitter le
programme) : Q
```

7. Déplacement d'un robot. Un robot effectue ses déplacements d'après les commandes qui lui sont fournies à partir d'un fichier texte « robot.txt ». Dans ce fichier chaque ligne contient une commande indiquant une direction dans laquelle le robot devra effectuer un déplacement d'une unité. Les commandes possibles sont HAUT, BAS, DROITE, GAUCHE et REPOS.
Elles correspondent aux déplacements suivants dans un plan cartésien :

HAUT : Déplacement d'une unité vers les **y positifs**
BAS : Déplacement d'une unité vers les **y négatifs**
DROITE : Déplacement d'une unité vers les **x positifs**
GAUCHE : Déplacement d'une unité vers les **x négatifs**
REPOS : **Aucun déplacement**

On suppose que le robot se trouve initialement à l'origine (0,0). Écrire un programme qui lit les commandes du déplacement a partir du fichier « robot.txt » et affiche sur l'écran les coordonnés finales du robot et le nombre de déplacements effectués en tout. Vous pouvez supposer que le fichier ne contient aucun autre mot que ceux indiqués ci-dessus.

8. Catégorisation d'une population. On veut définir un type énumération pour catégoriser la population selon les catégories suivantes : **étudiant, travailleur, chômeur, retraité**. Déclarer un tableau de 4 éléments ayant pour indice ces catégories.
Écrire un programme qui remplit ce tableau de nombres aléatoires entre [0, 10000] et permet d'afficher le contenu de ce tableau en affichant la chaîne de caractères correspondant à la valeur de l'indice.

9. Le tri de Shell. On cherche ici à implémenter une méthode de tri assez efficace en pratique, surtout pour des tableaux de taille faible à moyenne. Il s'agit du « tri de Shell » du nom de son inventeur.
Le tri de Shell est un « tri par insertion » avec un incrément : seulement un élément à chaque k est trié, puis k est réduit jusqu'à ce que tous les éléments soient triés.
Vous n'avez pas à comprendre l'algorithme, il vous faut simplement le traduire en C++ :

Pour k de la taille du tableau / 2, jusqu'à 1, en divisant k par 2 à chaque itération
    Pour i de k+1 à la taille du tableau
        $j = i - k$
        Tant que $j > 0$ et tableau[j] > tableau[$j + k$]
            Échanger tableau[j] et tableau[$j + k$]
            $j = j - k$

Le fichier « liste.txt » contient une liste de 100 entiers qui ont été générés aléatoirement.
En utilisant l'algorithme de tri de Shell, écrire un programme qui trie et stocke cette liste dans un fichier « trie.txt ».

10. **Tableau d'aléatoires**. Écrivez un programme qui génère 12 nombres aléatoires compris dans l'intervalle [2.500; 8,750] et les stocke dans un tableau à **deux** dimensions (3 lignes, 4 colonnes). Faites l'affichage du tableau ainsi que de la colonne où la somme des éléments est la plus élevée. Formatez les colonnes de façon à ce qu'elles soient correctement alignées. Affichez trois chiffres après la virgule.

Exemple :

```
2.678 3.456 3.141 6.789
7.854 2.987 3.456 4.567
3.213 8.749 2.501 7.467
La colonne avec la somme la plus élevée est :
6.789
4.567
7.467
```

Annexe 1 : Principaux points du guide de codage à respecter

3 : noms des variables en lowerCamelCase
4 : noms des constantes en MAJUSCULES
14, 52 : portée des variables (noms courts/noms)
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
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
46 : initialiser les variables à la déclaration (int x = 0;)
47 : pas plus d'une signification par variable
51 : test de 0 explicite (if (nombre!= 0))
53-54 : boucles for et while
58-61 : instructions conditionnelles
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)
67-78,88 : indentation du code et commentaires
79-81 : espacement et lignes de séparation
83-84 : aligner les variables lors des déclarations ainsi que les énoncés
85 : réécrire du code incompréhensible plutôt que de le commenter
87 : préférer // pour les commentaires

Bonne chance !