INF1015 – Programmation orientée objet avancée
Travail dirigé No. 1
Intro au C++ : Types, Structures de décision et répétition, lecture de fichiers, tableaux et structures.

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 (`struct`).
Durée : Une semaine de laboratoire.
Remise du travail : dimanche 28 janvier 2024, avant 23 h 30.
Travail préparatoire : « Leçon: introduction à Visual Studio » sur Moodle et lecture des exercices.
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. Vous pouvez ajouter des fonctions autres que celles décrites dans l'énoncé, dans le but d'améliorer la lisibilité.
• 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`).
• Utilisez les structures de données appropriées (tableaux, « struct »).
• 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 2022 (ou 2019), votre programme ne devrait pas afficher d'avertissements (« warnings ») de « build ».
• Les entrées/sorties au clavier/écran doivent être faites avec messages significatifs et vous n'avez pas à valider les entrées sauf si indiqué dans la question.

1. **Longueur des mots :** Écrivez un programme qui demande à l'utilisateur de saisir une phrase et qui retourne le mot le plus long et le mot le plus court de cette phrase ainsi que la longueur moyenne des mots dans cette phrase. (On suppose que les mots ne sont séparés que par un seul espace et qu'ils ne contiennent pas de ponctuation.)
   Exemple :

```
Saisissez une phrase : Bonjour les etudiants de INF1015
Le mot le plus court est : de
Le mot le plus long est : etudiants
La longueur moyenne est : 5.6 lettres
```

2. **Suite réelle :** Déterminer la limite de cette suite récurrente définie comme suit avec une précision de 5 chiffres après la virgule (la précision doit être un paramètre facilement modifiable) :

$$U_0 = 1$$
$$U_{n+1} = \sqrt{2 + U_n}$$

   On suppose que l'erreur entre un terme de la suite et la limite est inférieure à la différence entre ce terme et le précédent. Le calcul devrait être dans une fonction séparée de l'affichage.

3. **Insertion dans un tableau :** Écrire un programme qui demande à l'utilisateur un entier et l'insère au bon endroit dans un tableau trié pour que le tableau reste trié. Écrire une fonction pour la lecture de l'entier, qui redemande si l'entrée est invalide, par exemple si l'usager entre des lettres (vous pouvez lire comme un texte et utiliser la fonction *stoi*, voir la documentation sur cppreference.com; si vous voulez traiter les exceptions, regardez la syntaxe dans « Exceptions » de « De Python à C++ » sur le site Moodle). Utiliser un tableau C ou un std::array, de taille suffisante pour contenir toutes les valeurs après l'insertion (pas un std::vector, qui a une méthode pour insérer).
   Exemple :
   entier=5 tableau avant insertion={ 1, 3, 4, 7, 9 } tableau après insertion={ 1, 3, 4, 5, 7, 9 }
   Pour le tableau trié vous pouvez déclarer un tableau contenant des valeurs déjà triées et utiliser une variable qui indique le nombre d'éléments dans le tableau.

4. **Multiplication Russe :** La multiplication russe est une méthode pour multiplier deux entiers **a** et **b** en utilisant seulement des multiplications et des divisions par 2. Afin d'expliquer le principe de cette méthode on se réfère à l'exemple du tableau suivant pour multiplier 37 par 129.

37	129
18	~~258~~
9	516
4	~~1032~~
2	~~2064~~
1	4128

   a. Dans la première colonne, on divise chaque fois par 2 en prenant la partie entière.
   b. Dans la deuxième colonne, on multiplie chaque fois le nombre par 2.
   c. On barre dans la deuxième colonne tous les nombres face à un nombre pair dans la première.
   d. Le produit des deux nombre est la somme de tous les nombres non barrés de la deuxième colonne.

   Il faut donc une fonction qui calcule le produit, et un programme principale qui vérifie les multiplications de 3 paires de valeurs. On veut simplement afficher le nombre de tests passés, p.ex. « 3/3 tests passent. ». Utiliser un tableau de « struct » pour définir les tests à effectuer. (On n'utilise pas de bibliothèque de test dans cette question, ce sera pour un autre TD.)

5. **Devinette** : Écrivez un programme qui permet de deviner la valeur d'un nombre entier choisi aléatoirement entre 0 et 1000. L'utilisateur doit entrer des essais et le programme affiche si l'essai est inférieur, supérieur ou égal à la valeur choisie aléatoirement. À la fin, le programme indique le nombre de tentatives utilisées pour deviner la valeur. Écrivez une **fonction** permettant de lire une valeur réelle dans un intervalle. La fonction doit à la fois afficher pour demander, vérifier que la valeur est valide et redemander une entrée tant qu'elle ne l'est pas. Ses paramètres sont le texte à afficher, le min, le max. L'entrée de l'entier de l'utilisateur sera par cette fonction, où le réel résultant sera converti en entier.
   Exemple d'affichage (entrées de l'usager soulignées) : *(le nombre choisi aléatoirement est 45)*

```
Entrez un nombre entier : -1  //(à noter que 1001 va produire le même résultat)
Entrez un nombre entier : 5
Trop bas.
Entrez un nombre entier : 45
Bravo! Vous avez reussi en 2 tentatives!
```

6. **Tic-Tac-Toe :** Le Tic-Tac-Toe est un jeu de réflexion qui se joue par deux joueurs. Chaque joueur place dans une grille 3×3 son marqueur spécifique. Un joueur gagne lorsqu'il forme une ligne de trois cases successives par la largeur, la hauteur ou par la diagonale. Pour plus de détails sur le principe du jeu vous pouvez regarder https://fr.wikipedia.org/wiki/Tic-tac-toe.
   On propose dans cet exercice de charger une grille dont le nom du fichier est saisi par l'utilisateur (exemple : grille1.txt) et d'afficher suivant la situation de la grille l'un des messages suivant :

   ```
   Le joueur 1 gagne.
   Le joueur 2 gagne.
   Egalite
   ```

   On prend pour convention que le joueur 1 utilise le marqueur **x** et le joueur 2 utilise plutôt le marqueur **o**. On suppose que les grilles sont pleines (toutes les cases contiennent soit un **x** soit un **o**).

****

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)

3 : noms des variables et constantes en lowerCamelCase
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 (pour des indices) 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
42 : include au début
46 : initialiser à la déclaration
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 : mieux écrire du code incompréhensible plutôt qu'y ajouter des commentaires
87 : préférer // pour les commentaires