INF1005C - PROGRAMMATION PROCÉDURALE
Travail dirigé No. 3
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 : Deux séances de laboratoire.
Remise du travail : dimanche 5 octobre 2025, avant 23 h 30.
Travail préparatoire : Lecture 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 2022 (ou 2019), 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 sauf si indiqué dans la question.

1. **Nombres premiers :** Écrivez un programme qui lit du clavier un nombre entier, puis détermine si ce nombre est premier ou non. S'il n'est pas premier, le plus petit entier qui est un diviseur de ce nombre est affiché. Pour trouver un diviseur, commencez par vérifier si le nombre entré est divisible par 2, sinon essayez de le diviser par les nombres impairs entre 3 et $\sqrt{n}$, et arrêtez-vous au plus petit (aide : un algorithme similaire est écrit dans l'énoncé du TD1).
   Exemple d'affichage :
   ```
   Entrer un nombre entier : 8
   Ce nombre n'est pas premier car il est divisible par 2.
   Entrer un nombre : 2
   Ce nombre est premier.
   ```

2. **Emprunt** : Écrivez un programme qui demande et lit du clavier une somme d'argent qui a été prêtée, le montant remboursé chaque mois, ainsi que le taux d'intérêt annuel (le taux mensuel est le 1/12 du taux annuel), et retourne le nombre de mois nécessaires pour rembourser la dette ainsi que la somme des intérêts perçus par le prêteur. Vous devrez vérifier la validité des données entrées par l'utilisateur (montant positifs et taux compris entre 0 et 100), et redemander ceux-ci lorsqu'ils ne sont pas valides. Vous devez utiliser une structure de répétition, et non la solution analytique de l'équation de récurrence. Écrivez une fonction pour l'entrée validée d'une valeur réelle dans un intervalle; utilisez la constante `INFINITY` de `cmath` s'il n'y a pas de borne supérieure.

3. **Hauteur de rebond** : Écrivez un programme qui détermine la hauteur atteinte par une balle en tenant compte de la hauteur initiale et du nombre de rebonds. Les données à demander et à lire du clavier sont donc : la hauteur initiale, le nombre de rebonds au bout duquel on souhaite connaître la hauteur de la balle, ainsi que le coefficient de rebond. Vous devrez vérifier la validité des données entrées par l'utilisateur (hauteur et nombre de rebonds positifs, et coefficient compris entre 0 et 1) en réutilisant la fonction écrite à la question précédente.

   Les variables sont les suivantes :

   - h_i est la hauteur avant le rebond numéro *i*, et h_{i+1} la hauteur après le rebond numéro *i*.
   - v_i est la vitesse de la balle avant le rebond numéro *i*, et v_{i+1} est la vitesse après le rebond.

   Les relations entre les variables sont les suivantes :

   - $v_i = \sqrt{2 \cdot g \cdot h_i}$, avec *g* la constante de gravité égale à 9,81 dans notre cas.
   - $v_{i+1} = c \cdot v_i$, avec *c* le coefficient de rebond.
   - $h_{i+1} = \frac{v_{i+1}^2}{2 \cdot g}$

   Utilisez une structure de répétition pour répéter le calcul autant de fois qu'il y a de rebonds souhaités. Vous ne devez pas utiliser de tableau.

4. **Approximation de pi** : Écrivez un programme qui permet de calculer une valeur approchée de pi par la méthode de Monte-Carlo basée sur les probabilités.

   L'idée est la suivante : si on insère un cercle de rayon 1 (ce qui correspond à une aire de pi) dans un carré de côté 2 (et donc d'aire 4), la probabilité qu'un point placé aléatoirement uniformément distribué dans le carré soit également dans le cercle est pi/4 (le rapport des aires).

   Voici donc ce qu'il faut faire :

   - Vous allez générer deux nombres réels « aléatoires indépendants »* *x* et *y*, tous deux « uniformément distribués »* entre -1 et 1, donc dans un carré de côté 2.
   - Si $x^2 + y^2 < 1$, le point est dans le cercle de rayon 1.

   Il faut donc demander et lire du clavier le nombre d'itérations souhaité, puis afficher la valeur approchée de pi par cette méthode puis l'écart relatif entre cette valeur approchée et la valeur précise à 10^{-6} : 3,141593. Vous veillerez à afficher votre valeur approchée avec une précision de 6 chiffres après la virgule.

   Écrivez une fonction pour générer un nombre aléatoire entre -1 et 1. Vous devez utiliser les fonctions srand() et rand() pour la génération de nombres aléatoires. Ces fonctions exigent d'inclure les fichiers cstdlib et ctime (voir le programme dans la « Question 4 » des « Exercices sur les variables (II) » sur Moodle).

   * On suppose que rand() retourne des valeurs suffisamment indépendantes et uniformément distribuées pour cette application (le standard ne donne pas de garantie sur la qualité du générateur rand()).

5. **Tableau pair/impair** : Écrire un programme qui saisit un tableau d'entiers de taille 10 et qui l'affiche de telle sorte que tous les entiers pairs se retrouvent avant les entiers impairs. Par exemple, si le tableau initial contenait {7, 4, 7, 8, 4, 6, 3, 9, 6, 11} vous devez stocker {4, 8, 4, 6, 6, 7, 7, 3, 9, 11} dans un autre tableau puis afficher ce tableau.

6. **Dictionnaire** : Écrire un programme de traitement de données pour un dictionnaire (mot, nature/genre, définition). Le programme doit créer un tableau de structures, y placer le dictionnaire en le lisant d'un fichier, et ensuite afficher le mot le plus long du dictionnaire dans le format suivant :

   mot (nature/genre) : définition

   Un fichier « dictionnaire.txt » vous est fourni, où chaque ligne contient un mot sous la forme :

   mot *tab* nature/genre *tab* définition *finDeLigne*

   Le *tab* (caractère de tabulation) est le caractère indiqué par « `\t` » en C/C++. Vous pouvez supposer que le nombre de mots dans le dictionnaire est connu lors de la compilation (une constante définie dans votre programme) et que le nom du fichier l'est aussi.
   (Les mots de notre dictionnaire ont été tirés au hasard et les définitions viennent de Wiktionnaire.)

****

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
83-84 : aligner les variables lors des déclarations ainsi que les énoncés

Points du TD2 :

3 : noms des variables et constantes en lowerCamelCase
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
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
85 : mieux écrire du code incompréhensible plutôt qu'y ajouter des commentaires
87 : préférer // pour les commentaires