INF1005C - PROGRAMMATION PROCÉDURALE

Travail dirigé No. 2
Programmes simples
Entrées et sorties

Objectifs : Permettre à l'étudiant de faire ses premiers pas de programmation en langage C++.
Il apprendra à manipuler la structure de base d'un programme, les types de base ainsi que les entrées et les sorties du C++.
Durée : Une séance de laboratoire.
Remise du travail : Dimanche 21 septembre 2025, avant 23 h 30.
Travail préparatoire : Leçon 5 sur Moodle, lecture des exercices et rédaction des algorithmes.
Directives : N'oubliez pas de mettre les entêtes de fichiers et de respecter le guide de codage (voir la dernière page de ce document pour les points à respecter).
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 tous les faire)

Directives particulières

- Affichez toujours un message d'invite avant chaque saisie.
- Vous n'avez pas à valider les entrées.
- Vous n'avez pas à afficher les caractères accentués.
- Vous pouvez déclarer toutes les variables désirées.
- Pour chacun des exercices, utilisez des messages appropriés lors de l'affichage. Des chiffres seuls ne suffisent pas.

Pour traduire les algorithmes du TD1 :

S'assurer de déclarer les types des variables .
Afficher devient cout .
Lire devient cin .
SI *a* ALORS *b* SINON *c* devient if (*a*) { *b* } else { *c* } .
TANT QUE *a* FAIRE *b* devient while (*a*) { *b* } .

Conversion de TANT QUE en **for**, à effectuer si une même variable est initialisée, testée dans la condition d'un TANT QUE et incrémentée inconditionnellement (pas dans un SI) dans le corps du TANT QUE :

i = valeur initiale
TANT QUE *i* < valeur finale FAIRE
    a
    i = *i* + incrément
devient :
for (*i* = valeur initiale; *i* < valeur finale; *i* += incrément) {
    a
}

Une fonction du TD1 :

FONCTION *nom_fonction* **(** *nom_param1***,** *nom_param2* **)**
…
RÉSULTAT …

Devient :

```
type_du_résultat nom_fonction ( type_param1 nom_param1, type_param2 nom_param2 ) {
    …
    return … ;
}
```

1. **Orthogonal :** Implémenter le numéro 1 du TD1 en C++ : Écrire un algorithme qui détermine si deux vecteurs à deux dimensions sont orthogonaux ou non. Note : utiliser un produit scalaire fait avec des opérations de base du langage (ne cherchez pas une fonction de bibliothèque pour faire ce produit). Attention aux possibles imprécisions de calcul.

   Exemple : L'utilisateur entre les composantes des vecteurs (1 ; 0.2) et (-1.12 ; 5.6)
   L'affichage attendu est : `Les vecteurs sont orthogonaux.`

2. **Jouer au prof :** Pour ce numéro, vous devez corriger et faire compiler le programme *exo2.cpp* dans les fichiers fournis tout en gardant son utilité principale, c'est-à-dire demander un prix à payer, une somme d'argent donnée pour payer (tout en entier) et affiche la monnaie à rendre en billets de 100$, 20$, 10$, 5$ et en pièces de 1$. Vous devez aussi le corriger au niveau du style en suivant le guide de codage.

3. **Logarithme :** Implémenter le calcul du logarithme du numéro 6 du TD1 en C++, en ajoutant ce qu'il faut pour tester qu'il fonctionne : Écrire une fonction pour calculer le logarithme naturel d'un nombre réel x situé dans l'intervalle]0, 2[. La méthode pour calculer le logarithme sera d'utiliser la série définie comme :

$$\ln(x) = (x-1) - \frac{(x-1)^2}{2} + \frac{(x-1)^3}{3} - \frac{(x-1)^4}{4} + \frac{(x-1)^5}{5} - \cdots$$

   Lorsque le nombre de termes tend vers l'infini, cette série converge vers le logarithme naturel de *x*, si *x* est dans l'intervalle]0, 2[(notez que numériquement ça ne converge pas très bien pour *x* très près de 0 ou de 2). L'estimation de l'erreur pour la série ayant *n* termes est la valeur absolue du *n*ième terme (sans le signe, les termes sont décroissant sur l'intervalle de convergence). L'algorithme doit arrêter lorsque cette estimation de l'erreur est inférieure à la précision voulue. La fonction doit donc uniquement retourner le logarithme naturel en fonction de *x* et de la *précision* voulue.

   Appeler cette fonction avec 3 paires de valeurs différentes entre 1 et 2 (exclu) en vérifiant si le résultat retourné par la fonction est correct et en affichant « true » ou « false » pour chaque résultat (vous ne devez pas utiliser de « if » pour faire cette vérification et affichage). Un résultat est considéré correct s'il est à moins de *précision* du vrai logarithme (notre estimation de l'erreur n'est pas assez bonne pour que ce test soit correct pour les valeurs de *x* entre 0 et 1). Si la fonction est bonne, le programme devrait afficher 3 fois « true ».

4. **Utilisation de fichiers :** Le fichier *ventes.txt* contient 4 lignes (il a toujours le même nombre de lignes). La première est le nom complet d'un client et les trois autres contiennent le nom (sans espaces) d'un produit, la quantité vendu et le prix unitaire. Il faut écrire un programme qui lit le contenu de *ventes.txt* et écrit la facture dans un fichier *facture.txt* avec le nom du client, le sous-total avant taxe, le montant des taxes et total avec taxes, supposant un taux de 15% de taxes.

   Les montants dans la facture doivent être écrit sous le nom du client, être alignés à droite sur 10 colonnes et avoir deux chiffres décimaux suivi d'un signe de dollar. Il faut suivre le format de l'exemple.

   Vous devez utiliser une boucle pour ne pas écrire 3 fois les mêmes instructions dans votre programme.

   Exemple

ventes.txt
Joe Untel chaussures 1 249.99 chandail 3 49.99 casquette 1 34.99
facture.txt
Joe Untel SOUS-TOTAL 434.95 $ TAXES 65.24 $ TOTAL 500.19 $

Les points du **guide de codage** à respecter **impérativement** pour ce TD sont les suivants :
(lire 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 (attention que les constantes étaient avant en MAJUSCULES, voir le point 4)
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
68-78: indentation du code, voir ci-dessous
79,81: espacement et lignes de séparation
85: mieux écrire le programme plutôt qu'ajouter des commentaires
87: préférer //

Plusieurs points du guide montrent comment bien indenter, la forme du programme devrait être :

```
#include ...
Tous les débuts de lignes
sont alignés à gauche
...
bool estErreur()
{
    return false;
}

int main()
{
    // On indente (avec tabulation) similairement au TD1 :
    while (!estFini) {
        faireQuelqueChose();

        if (estErreur()) {
            afficherMessageRouge();
            nErreurs++;
        }
        else if (estAvertissement()) {
            afficherMessageJaune();
            nAvertissements++;
        }
        else
            traitementNormal();

        for (int i = 0; i < nElements; i++)
            traiterElement(i);

        estFini = !aEncoreAFaire();
    }
}
```