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 : Deux séances de laboratoire.
Remise du travail : Dimanche 7 février 2021, 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. **Premier prénom :** Soit un prénom composé (avec un trait d'union) entré par l'utilisateur (exemple « jean-luc »). Retrouvez le premier prénom (exemple « jean » pour « jean-luc ») puis affichez le message « Bonjour PremierPrénom » (exemple « Bonjour Jean »). La première lettre du prénom affiché doit être en majuscule (même si l'utilisateur l'a mis en minuscule). Ne pas utiliser de condition ou de répétition. Indice : le type `string` possède des fonctions de recherche.

2. **Orthogonal :** Implémenter le numéro 1 du TD1 en C++. Le programme détermine si deux vecteurs à deux dimensions entrés par l'usager sont orthogonaux ou non. Note : utiliser un produit scalaire.

   Exemple : L'utilisateur entre les composantes des vecteurs (1 ; 0,5) et (-1 ; 2)
   L'affichage attendu est : `Les vecteurs sont orthogonaux.`

3. **Sin :** Écrivez un programme qui génère aléatoirement un nombre réel entre 0 et 1 (utilisez les fonctions `srand()` et `rand()`, et la constante `RAND_MAX` ; voir l'exemple ci-dessous), et qui calcule directement son sinus (fonction sin(x)), puis en utilisant les **3 premiers termes** de la série (il n'est pas demandé de faire une boucle pour calculer les termes) :

$$\sin(x) \cong x - \frac{x^3}{3!} + \frac{x^5}{5!} + \frac{x^7}{7!} + \frac{x^9}{9!} + \cdots$$

   Le programme affichera ensuite *x* et la différence entre l'approximation et la valeur réelle (telle que retournée par le `sin()` de `<cmath>`.

   Inspirez-vous de l'exemple suivant pour générer des nombres aléatoires entiers :

```
#include <iostream>
#include <cstdlib> // Pour srand et rand.
#include <ctime> // Pour time.
using namespace std;
int main () {
    // Initialise le générateur aléatoire. (Une seule fois au début du programme)
    srand(unsigned(time(nullptr)));
    // Affiche un entier aléatoire entre 0 et la constante RAND_MAX.
    cout << rand();
}
```

4. **Distance à l'origine en 3D :** Implémenter le numéro 5 du TD1 en C++. Demander et lire plusieurs points de coordonnées (*x*, *y* , *z*). À chaque nouveau point entré, indiquer s'il est plus près de l'origine (0, 0, 0) que tous les points précédemment entrés. Définissez une fonction pour obtenir la distance du point à l'origine. Votre programme doit calculer une seule fois la distance de chaque point entré par l'usager.

5. **Pokémon :** Dans un fichier *machamp.txt*, vous avez les statistiques d'un Pokémon, ainsi que celles de deux de ses attaques. Écrivez un programme qui lit le fichier et crée un nouveau fichier *moves.txt* qui donne le dommage causé par chaque attaque selon un niveau de défense donné par l'utilisateur. Les fichiers doivent avoir la forme suivante (on vous fournit *machamp.txt* avec des valeurs différentes) :

machamp.txt	*moves.txt*

`Machamp`
`LVL 42`
`ATK 175`
 	
`Strength  80`
`Superpower 120`
	`Machamp contre DEF=100`
	
	`Strength 52`
	`Superpower 77`

Le nom du Pokémon est sur la première ligne du fichier *machamp.txt* et peut avoir plusieurs mots, les lignes LVL et ATK seront toujours dans cet ordre, les noms d'attaques (ici, « Strength » et « Superpower ») sont en un seul mot et il y en aura toujours deux, et les valeurs doivent être écrites en entiers arrondis vers le bas.

Voici le pseudo-code pour la fonction de calcul de dommage :

```
FONCTION calculerDommage (lvl, atk, pwr, def) :
    RESULTAT ((2*lvl/5 + 2) * pwr * atk/def)/50 + 2
```

Dans l'exemple ci-dessus, où l'utilisateur aurait donné un niveau de défense de 100, les valeurs seraient : lvl=42, atk=175, def=100, ces valeurs étant communes aux deux attaques, et pwr=80 pour l'attaque *Strength* et pwr=120 pour l'attaque *Superpower*. Effectuez tous les calculs avec des entiers (pas de réels).

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();
    }
}
```