TP 4 : Des fonctions, des tests et de la documentation

Le but de ce TP est de vous familiariser avec la programmation de fonctions en général, avec tests et documentation, et d'acquérir de l'aisance avec les boucles `for` avec accumulateur.

Exercice 1.

(1) Télécharger l'archive `Semaine4.zip`, et l'extraire dans votre répertoire `Info111`.

(2) Un programme vous est fourni dans le fichier `Semaine4/conversion.cpp`. Malheureusement, il génère des erreurs lors de la compilation. Trouver et corriger ces erreurs.
Indications :
- reformater le programme (Menu Plugins, Source code formatter) ;
- une fonction doit être définie avant d'être appelée ;
- vérifier le nombre d'arguments.

(3) Ajouter dans le même fichier une fonction réalisant la conversion réciproque, et vérifier sur quelques exemples que les deux fonctions sont réciproques l'une de l'autre.

Exercice 2 (Suite de l'exercice 4 du TD)**.**
Vérifier sur votre machine le comportement prédit en TD pour les deux programmes `variable_locale.cpp` et `variable_globale.cpp` fournis dans `Semaine4`.

Exercice 3 (Calcul de l'exponentielle à un rang fixé)**.**
La fonction exponentielle est définie par

$$\exp(a) := \sum_{n=0}^{\infty} \frac{a^n}{n!} .$$

Comme vous pouvez le remarquer, cette formule mathématique fait intervenir une *somme infinie*, qui ne peut pas être directement calculée par un un ordinateur qui est *fini*. Dans cet exercice, nous allons utiliser cette formule pour calculer une *approximation* de la fonction exponentielle en tronquant la somme à ses r premiers termes.

La formule de l'exponentielle faisant intervenir des calculs de puissances et de factorielles ; nous allons commencer par récupérer les fonctions que nous avons déjà implantées.

(1) Écrire un programme `factorielle.cpp`, contenant la fonction `factorielle` du TD :

```
double factorielle(int n) {
        ...
}
```

ainsi que la fonction `main` suivante :

```
int main() {
    cout << factorielle(0) << endl;
    cout << factorielle(1) << endl;
    cout << factorielle(4) << endl;
    cout << factorielle(5) << endl;
    cout << factorielle(100) << endl;
    return 0;
}
```

(2) Compiler et lancer le programme, et vérifier que l'affichage obtenu est bien :

```
1
1
24
120
9.33262e+157
```

Explication : la valeur de 100! est un très grand nombre ; vous devez donc en voir une **approximation** écrite sous la forme `9.33e+157` qui signifie 9.33×10^{157}. Si ce n'est pas le cas, c'est que vous n'effectuez pas votre calcul en type `double` !

(3) Dans un nouveau fichier `"puissance.cpp"` écrire la fonction suivante :

```
/** La fonction puissance
 *  @param a un nombre à virgule flottante en double précision
 *  @param n un nombre entier positif
 *  @return la n-ième puissance a^n de a
 **/
double puissance(double a, int n) {
```

Compléter le programme avec une fonction `main` qui affiche `puissance(2,0)` (égal à 1), `puissance(2,1)`, `puissance(2,4)`, `puissance(2.5,2)`, `puissance(9,100)` dont vous vérifierez les valeurs.

(4) On a maintenant tous les ingrédients de la fonction exponentielle elle-même. Dans un nouveau fichier `exponnentielle-rang.cpp` mettre vos deux fonctions `factorielle` et `puissance` ainsi qu'une fonction

```
/** La fonction exponnentielle tronquée à un rang donné
 * @param a un nombre à virgule flottante en double précision
 * @param r un nombre entier positif
 * @return 1 + a + a^2/2 + ... + a^r/r!
 **/
double exponnentielle_rang(double a, int r) {
```

qui calcule une **approximation** de $\exp(a)$ par la somme **finie** :

$$\sum_{n=0}^{r} \frac{a^n}{n!}\,.$$

Par exemple, `exponentielleRang(5,2)` donne

$$\frac{5^0}{0!} + \frac{5^1}{1!} + \frac{5^2}{2!} = 18.5\,.$$

Indications : utiliser les fonctions `factorielle` et `puissance` et une boucle (`for` ou `while`) avec un accumulateur.

(5) Ajouter l'affichage d'exemples dans la fonction `main` : pour $a = 5$, vous devez trouver 6 pour $r = 1$, 18.5 pour $r = 2$, et 39.333 pour $r = 3$. Plus le rang r est élevé, plus la valeur se rapproche de $\exp(a)$ (approximativement 148.413 pour $a = 5$). On doit observer qu'à partir d'un certain rang, le nombre affiché reste le même : la valeur ajoutée est trop petite pour modifier l'affichage. En faisant plusieurs essais, trouver ce rang pour $a = 5$.

(6) Le programme fourni `setprecision.cpp` illustre comment configurer le nombre de chiffres significatifs affichés pour les nombres flottants. En vous en inspirant, mettez à jour votre programme `exponnentielle-rang.cpp` pour qu'il affiche les résultats avec 10 chiffres après la décimale.

Exercice 4 (Comparaison de nombres à virgule)**.**
Sur les nombres à virgule (`double`), l'opérateur `==` n'est pas très utile à cause des erreurs d'arrondis. Pire, comme le programme n'affiche pas tous les chiffres, deux nombres qui s'affichent de la même manière peuvent être en fait différents ! Par exemple, sur ma machine et avec `gcc version 4.8.2 (Ubuntu 4.8.2-19ubuntu1)`, le programme

```
double x = 15.999999;
double y = 16.0;

cout << x << " " << y << endl;
cout << (x == y) << endl;
```

affiche

```
16 16
0
```

Pour résoudre ce problème, quand on veut comparer deux nombres à virgule, on teste si la valeur absolue de la différence est négligeable devant les deux nombres :

$$|x - y| < \varepsilon|x| \qquad \text{et} \qquad |x - y| < \varepsilon|y|$$

où ε est un très petit nombre.

(1) Reprendre la fonction `abs` de l'exercice 2 du TD qui calcule la valeur absolue d'un entier et l'adapter pour qu'elle calcule la valeur absolue d'un `double`. Votre programme doit aussi contenir une fonction `main` qui demande à l'utilisateur un nombre de type `double`, appelle `abs` et affiche le résultat.

(2) Écrire une fonction `egal` qui utilisera la fonction `abs` pour effectuer le test décrit ci-dessus et décider si deux nombres `double` sont assez proches pour être considérés égaux. Votre fonction prendra en paramètre trois variables `double`, `x`, `y`, et `epsilon` et retournera une variable de type `bool` (`true` ou `false`).

(3) Écrire une fonction `main` qui vérifie que pour $\varepsilon = 10^{-5}$, les deux nombres `15.999999` et `16.0` sont considérés comme égaux par cette fonction.

(4) Déterminer la précision minimale ε pour laquelle ces deux nombres sont considérés égaux par cette fonction.

Exercice 5 (Calcul de l'exponentielle, suite)**.**

(1) Plutôt que de spécifier le rang, on souhaite maintenant spécifier la précision du calcul. Pour cela, écrire une fonction `exponentiellePrec` qui prend en paramètre le nombre `a` et une précision `epsilon`. La fonction calcule la somme petit à petit. A chaque étape on vérifie grâce à la fonction `egal` de l'exercice précédent que la nouvelle valeur est différente de l'ancienne en fonction de la précision. La boucle s'arrête lorsque les valeurs sont considérées comme égales.

(2) Écrire une fonction `main` qui permette à l'utilisateur de tester les fonctions exponentielles de l'exercice sur différentes valeurs.

Exercice ♣ 6 (Documentation et tests automatiques)**.**
Un programme vous est fourni dans le fichier `Semaine4/fonction-factorielle-doctests.cpp`.

(1) Consulter la documentation HTML du programme en ouvrant `Semaine4/html/index.html`, puis en suivant les liens `Files` et `fonction-factorielle-doctests.cpp`.

(2) Lire le programme en détail et prédire son affichage.

(3) Lancer le programme.

(4) Comparer ce qu'il affiche avec ce que vous aviez prédit.

(5) Corriger le programme, si nécessaire.

(6) (Optionnel, si vous avez le logiciel `doxygen` installé sur votre machine) Regénérer la documentation HTML à l'aide du logiciel `doxygen`. Sous Linux, ouvrir un terminal, et taper les commandes suivantes :

```
> cd ~/Info111/Semaine4/
> doxygen -g
> doxygen
```

Exercice ♣ 7 (Documentation et test automatiques pour l'exponentielle)**.**
Ajouter de la documentation et des tests automatiques pour toutes les fonctions de l'exercice 3.

Exercice ♣ 8.
Compléter le programme `Semaine4/calculatrice.cpp` en vous appuyant sur la documentation et les tests fournis. Compléter les tests jusqu'à ce que vous ayez entièrement confiance en votre code.

Exercice ♣ 9.
La méthode utilisée dans l'exercice 3 n'est pas très efficace : en utilisant les fonctions `factorielle` et `puissance`, on recalcule plusieurs fois les mêmes produits. Pour aller plus vite, on peut, dans la même boucle, accumuler la factorielle, la puissance et la somme.

(1) Écrire une fonction `exponentielle2` qui utilise trois accumulateurs dans la même boucle. On gardera la même condition d'arrêt de la boucle.

(2) Chronométrer les deux fonctions pour déterminer le gain réel.

Exercice ♣ 10 (Suite de l'exercice 5 du TD)**.**

(1) Implanter dans un fichier `Semaine4/pointDeChute.cpp` l'exercice 5 du TD.

(2) Vérifier que les tests passent.

(3) (Optionnel) Générer la documentation HTML et la vérifier.

(4) Le résultat est-il correct pour un angle de -5 degrés ?

(5) Modifier la fonction `main` de sorte qu'elle demande à l'utilisateur de saisir les données en entrée et affiche le résultat à l'écran.

Exercice ♣ 11 (« O temps, suspens ton vol ... »)**.**
L'objectif de cet exercice est d'effectuer des conversions de durée entre secondes et heures/-minutes/secondes. Compléter le fichier `Semaine4/temps.cpp` en ajoutant cinq fonctions :

```
int tempsEnH(int temps): renvoie le nombre d'heures
int tempsEnM(int temps): renvoie le nombre de minutes
int tempsEnS(int temps): renvoie le nombre de seconde
string afficheHMS(int temps): renvoie une chaîne de la forme "12h 5min 3s"
int HMSEnSec(int H, int M, int S): fait l'inverse (convertit en secondes)
```

de sorte que le programme affiche exactement :

```
Le temps en Secondes: 3740
Le temps en HMS:
H:1 M:2 S:20
```

Documenter et tester chaque fonction.
(Optionnel) Ajouter une fonction qui à partir d'une heure de départ (en H/M/S), d'une durée de vol en minutes et d'un décalage horaire entre la ville de départ et d'arrivée calcule l'heure d'arrivée (en H/M/S).

Annexe : Installer doxygen

On trouvera le logiciel `doxygen` sur le site `http://www.doxygen.org/`. Sous Linux (Debian, Ubuntu), il suffit d'installer les paquetages suivants de la logithèque : `graphviz, doxygen, doxygen-gui`. Bien entendu, cela nécessite les droits d'administrateur.