

**TD 4 : Des fonctions, des tests et de la documentation****Exercice 1** (Premières fonctions).

Voici une fonction qui calcule la surface d'un rectangle :

```
float surface_rectangle(float longueur, float largeur) {  
    return longueur * largeur;  
}
```

- (1) Écrire une **fonction** `surface_disque` qui calcule la surface d'un disque de rayon donné. On prendra  $\pi = 3.1415926$ .

Correction.

```
float surface_disque(float rayon) {  
    return 3.141592657 * rayon * rayon;  
}
```

- (2) Écrire une **fonction** `surface_triangle` qui calcule la surface d'un triangle de base  $b$  et de hauteur  $h$ .

Correction.

```
float surface_triangle(float b, float h) {  
    return b * h / 2;  
}
```

**Exercice 2** (Appels de fonctions).

- (1) Écrire un programme qui demande à l'utilisateur de saisir un entier et qui affiche ensuite sa valeur absolue.
- (2) Écrire un programme qui demande à l'utilisateur de saisir deux entiers et qui affiche ensuite la somme de leurs valeurs absolues.
- (3) Si vous ne l'avez pas déjà fait, réécrire le programme précédent en introduisant une fonction `abs` dont la documentation, l'entête et les tests sont donnés ci-dessous :

```
/** Valeur absolue  
 * @param n un entier  
 * @return la valeur absolue de n  
 **/  
int abs(int n) {
```

```
void absTest() {  
    ASSERT( abs( 5) == 5 );  
    ASSERT( abs(-2) == 2 );  
    ASSERT( abs( 0) == 0 );  
}
```

Correction.

```

/** Valeur absolue
 * @param n un entier
 * @return la valeur absolue de n
 */
int abs(int n) {
    if ( n < 0 ) {
        return -n;
    } else {
        return n;
    }
}

```

```

int a, b;
cout << "Entrer deux entiers" << endl;
cin >> a >> b;
cout << abs(a) + abs(b) << endl;

```

- (4) Qu'afficherait votre programme si l'on utilisait à la place la fonction suivante :

```

int abs2(int n) {
    if ( n < 0 ) {
        cout << -n << endl;
    } else {
        cout << n << endl;
    }
}

```

Correction. Le programme afficherait les valeurs absolues des deux nombres entrés (et non pas la somme voulue), puis un nombre arbitraire puisque la valeur de retour n'a pas été spécifiée. Par exemple, si l'utilisateur entre -2 et 5, l'affichage ressemblerait à

```

2
5
243523452345

```

- (5) Quel est le type de la valeur de retour des fonctions `abs` et `abs2` ?

Correction. Ces deux fonctions renvoient des entiers, qui sont de type `int`.

**Exercice 3** (En route vers l'exponentielle).

- (1) Nous avons vu en cours une fonction `factorielle(n)` qui calcule la factorielle d'un entier positif  $n$ . Pour un exercice du TP à venir, et pour éviter les problèmes de dépassement de capacité, il est souhaitable que les calculs intermédiaires et le résultat soient des `double`. Adapter en conséquence la fonction du cours.

Correction.

```
/** La fonction factorielle
 * @param n un nombre entier positif
 * @return n! comme un nombre à virgule flottante à double précision
 */
double factorielle(int n) {
    double resultat = 1;
    for ( int k = 1; k <= n; k++ ) {
        resultat = resultat * k;
    }
    return resultat;
}
```

- (2) On considère la fonction dont la documentation et l'entête sont donnés ci-dessous :

```
/** 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) {
```

Quels sont les types de ses paramètres formels et de sa valeur de retour ?

Correction. Le paramètre formel `a` est de type `double`, le paramètre formel `n` est de type `int`, la valeur de retour est de type `double`.

- (3) Implanter la fonction `puissance`.

Correction.

```
double puissance(double a, int n) {
    double resultat = 1;
    for ( int k = 1; k <= n; k++ ) {
        resultat = resultat * a;
    }
    return resultat;
}
```

**Exercice 4** (Variables locales / globales, pile et exécution pas à pas).

On considère les deux programmes suivants :

```
#include <iostream>
using namespace std;

int i = 0;
```

```

int f(int j) {
    i = i + j;
    return i;
}

int main() {
    cout << i << endl;
    cout << f(1) << endl;
    cout << f(2) << endl;
    cout << f(3) << endl;
}

```

```

#include <iostream>
using namespace std;

int f(int j) {
    int i = 0;
    i = i + j;
    return i;
}

int main() {
    cout << i << endl;
    cout << f(1) << endl;
    cout << f(2) << endl;
    cout << f(3) << endl;
}

```

- (1) Quelle est la différence entre les deux programmes ?
- (2) Une ligne du deuxième programme est incorrecte : le compilateur déclencherait une erreur. Laquelle ? La supprimer.
- (3) Chercher dans le poly de cours la sémantique de l'appel d'une fonction.
- (4) Exécuter pas à pas les deux programmes en décrivant au fur et à mesure l'état de la mémoire (pile) et ce qui est affiché à l'écran.
- (5) Décrire la différence de comportement et retrouver dans les notes de cours le commentaire à ce propos.

**Correction.**

### Exercice 5.

Analyser la fonction `volumePiscine` suivante :

```

/**Calcule le volume d'une piscine parallélépipédique
 * @param profondeur la profondeur de la piscine (en mètres)
 * @param largeur la largeur de la piscine (en mètres)
 * @param longueur la longueur de la piscine (en mètres)
 * @return le volume de la piscine (en litres)
 **/
double volumePiscine(double profondeur, double largeur, double longueur) {
    return 100 * profondeur * largeur * longueur;
}

```

```
void volumePiscineTest(){
    ASSERT( volumePiscine(5, 12, 5) == 30000 );
    ASSERT( volumePiscine(1, 1, 5) == 500 );
}
```

(1) Est-ce que les tests passent ?

**Correction.** Oui, les tests passent.

(2) Est-ce que la documentation, le code et les tests sont cohérents ?

**Correction.** Les tests et l'implantation sont incohérents avec la documentation car il y est spécifié que le résultat doit être exprimé en litres.

(3) Corriger les anomalies éventuelles.

**Correction.** Le facteur de conversion devrait être de  $10^3 = 1000$  et pas 100.

### Exercice ♣ 6.

Analyser la fonction `mystere` suivante :

```
string mystere(int blop) {
    string schtroumpf = "";
    for ( int hip=1; hip <= blop; hip++ ) {
        for ( int hop=1; hop <= hip; hop++ ) {
            schtroumpf += "*";
        }
        schtroumpf += "\n";
    }
    return schtroumpf;
}

void mystereTest() {
    ASSERT( mystere(0) == "" );
    ASSERT( mystere(1) == "*\n" );
    ASSERT( mystere(2) == "*\n**\n" );
    ASSERT( mystere(3) == "*\n**\n***\n" );
}
```

(1) Comment appelle-t-on cette fonction (quelle est sa *syntaxe*) ?

(2) Que fait cette fonction (quelle est sa *sémantique*) ?

**Indications :** la notation `x += expression` est un raccourci pour `x = x + expression` ; dans une chaîne de caractères, « `\n` » représente un saut de ligne.

(3) Choisir un bon nom pour cette fonction et ses variables et en écrire la documentation.

**Exercice ♣ 7.**

Le but de cet exercice est de coder une fonction `pointDeChute` qui calcule l'abscisse  $x_c$  à laquelle retombe un projectile lancé en  $x = 0$  avec une vitesse  $v$  suivant un angle  $\alpha$  (exprimé en degrés par rapport à l'horizontale). On rappelle que l'abscisse est donnée par la formule :  $x_c = (2v_x v_y)/g$  où  $v_x = v \cos(\alpha)$ ,  $v_y = v \sin(\alpha)$  et  $g$  est l'accélération gravitationnelle (environ  $9,8m/s^2$  sur la planète Terre).

Implanter la fonction `pointDeChute`. On commencera par écrire sa documentation et des tests (voir TD 1).

Indication : en C++, les fonctions mathématiques sinus et cosinus sont implantées par les fonctions prédéfinies `sin(arg)` et `cos(arg)` dans `<cmath>`, où l'angle `arg` est exprimé en radians.