TD 5 : Fonctions et tableaux

Exercice 1 (Échauffement)**.**
Quelles sont les valeurs `v[0]`, `v[1]`, ... contenues dans le tableau `v` à la fin de l'exécution du programme suivant :

```
    vector<int> v;                          // Declaration
    v = vector<int>(5);                     // Allocation
    for ( int i = 0; i < v.size(); i++ ) // Initialisation
        v[i] = i*i;
```

Exercice 2 (Mystère)**.**
On considère la fonction mystère suivante :

```
int mystere(vector<int> t) {
    int m = t[0];
    for ( int i = 1; i < t.size(); i++ ){
        if ( m < t[i] ) {
            m = t[i];
        }
    }
    return m;
}
```

(1) Exécuter pas à pas le programme suivant :

```
    vector<int> t = { 5, -1, 3, 7, -4, 8, 4 };
    int resultat = mystere(t);
```

Quelle est la valeur de la variable `resultat` à la fin de l'exécution ? Que fait la fonction mystère ?

(2) Modifier la fonction pour qu'elle calcule le minimum d'un tableau.

Exercice 3 (Quelques fonctions sur les tableaux)**.**

(1) Spécifier et écrire une fonction qui renvoie vrai si tous les éléments d'un tableau d'entiers sont positifs, et faux sinon.

(2) Spécifier et écrire une fonction qui incrémente de 1 tous les éléments d'un tableau d'entiers et renvoie le tableau.

(3) Spécifier et écrire une fonction qui teste si deux tableaux d'entiers sont égaux.

(4) Spécifier et écrire une fonction qui compte le nombre d'éléments plus grands que 10 dans un tableau d'entiers.

(5) Spécifier et écrire une fonction qui renvoie vrai si un tableau d'entiers est trié dans l'ordre croissant, et faux sinon.

Exercice 4.
On rappelle que la suite de Fibonacci est définie récursivement par la relation : $U_n = U_{n-1} + U_{n-2}$. Cette définition doit être complétée par une condition d'initialisation, en l'occurrence : $U_1 = U_2 = 1$. Introduite par Léonard de Pise (surnommé Fibonacci), cette suite décrit un modèle simplifié de l'évolution d'une population de lapins. Les premiers termes sont donnés par : 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, ... Voir aussi `http://oeis.org/A000045`.

L'objectif est de donner trois implantations de la fonction dont la documentation et les tests sont donnés ci-dessous :

```
/** Suite de Fibonacci
 *  @param n un entier strictement positif
 *  @return le n-ième terme de la suite de Fibonacci
 **/
```

```
    ASSERT( fibonacci(1) == 1 );
    ASSERT( fibonacci(2) == 1 );
    ASSERT( fibonacci(3) == 2 );
    ASSERT( fibonacci(4) == 4 );
    ASSERT( fibonacci(5) == 5 );
    ASSERT( fibonacci(6) == 8 );
```

(1) Vérifier que les tests sont cohérents avec la documentation. Corriger si nécessaire.

(2) Implanter la fonction fibonacci avec une boucle `for` et un tableau de taille n pour stocker tous les éléments de la suite $U_1, \ldots, U_n$.

(3) Implanter la fonction fibonacci sans tableau, en utilisant une boucle `for` et trois variables dont deux qui, au début de chaque tour de boucle, contiennent les valeurs de U_{k-1} et U_{k-2}.

(4) Implanter une version récursive de la fonction fibonacci, c'est-à-dire qui s'appelle elle même.

(5) Laquelle de ces trois implantations est la plus expressive ?

(6) ♣ Comparer l'efficacité des trois implantations. Quel phénomène a lieu pour la fonction récursive ? Comment pourrait-on le corriger ?

Exercice ♣ 5.

(1) Que fait la fonction suivante ?

```
vector<int> mystere(vector<int> t) {
    int n = t.size();
    vector<int> r(n);
    for (int i=0; i<n; i++) {
        r[i] = t[n-1-i];
    }
    return r;
}
```

(2) Un *palindrome* est un tableau comme $\{2, 8, -1, 6, -1, 8, 2\}$ qui peut se lire de la même façon dans les deux sens. Écrire une fonction qui teste si un tableau est un palindrome en utilisant les fonctions déjà vues dans le TD.

(3) Réécrire une fonction `palindrome`, testant si un tableau est un palindrome, sans utiliser les fonctions déjà vus dans les exercices précédents.

(4) Commenter les avantages et inconvénients des deux implantations.