TP 5 : Fonctions et tableaux

WIMS.
20 minutes de contrôle sur WIMS.

Exercice 1.

(1) Télécharger l'archive `Semaine5.zip`, et l'extraire dans votre répertoire `Info111`.

(2) Configurer `Code::Blocks` pour compiler le C++ selon le standard ISO C++ de 2011 :

```
Settings -> Compiler -> Global compiler settings -> Compiler Flags:
Have g++ follow the C++11 ISO C++ language standard
```

(3) Vérifier que le fichier fourni `min-max.cpp` compile.

(4) Compléter la documentation et le code dans le fichier fourni `min-max.cpp` en cherchant toutes les indications « À faire ». Vérifier que tous les tests passent.

Correction :

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

/** Renvoie l'élément le plus grand du tableau **/
int max(vector<int> t) {
    int m = t[0];
    for ( int i = 1; i < t.size(); i++ ){
        if ( t[i] > m )
            m = t[i];
    }
    return m;
}

/** Renvoie l'élément le plus petit du tableau **/
int min(vector<int> t) {
    int m = t[0];
    for ( int i = 1; i < t.size(); i++ ){
        if ( t[i] < m )
            m = t[i];
    }
    return m;
}

int main() {
    vector<int> tab = { 5, -1, 3, 7, -4, 8, 4 };
    cout << "Le max est "<< max(tab) <<endl;
    cout << "Le min est "<< min(tab) <<endl;
```

```
    return 0;
}
```

(5) Même chose pour le fichier fourni `tableaux.cpp`.

Correction :

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

/** Affiche un tableau
 * @param tab un tableau d'entiers
 **/
void afficheTableau(vector<int> tab) {
    for(int i=0;i<tab.size();i++) {
        cout<<tab[i]<<'\t';
    }
    cout<<endl;
}

/** Teste si deux tableaux sont égaux
    (tailles égales, éléments de même indice égaux)
    * @param tab1 le premier tableau d'entiers
    * @param tab2 le deuxième tableau d'entiers
    * @return vrai si les deux tableaux sont égaux, faux sinon
    **/
bool testEgaux(vector<int> tab1, vector<int> tab2) {
    int taille = tab1.size();
    if(tab2.size()  != taille) {
        return false;
    }
    else{
        for(int i=0; i<taille; i++) {
            if(tab1[i]  != tab2[i]) {
                return false;
            }
        }
        return true;
    }
}

/** Compte le nombre d'éléments plus grands que 10 dans le tableau
 * @param tab un tableau d'entier
 * @return nombre d'éléments plus grand que 10
 */
int plusGrandQueDix(vector<int> tab) {
    int res = 0;
    for(int i=0; i<tab.size(); i++) {
        if(tab[i] > 10)
            res = res + 1;
    }
    return res;
}

/** Teste si un tableau est trié en ordre croissant
```

```
 * @param tab un tableau d'entier
 * @return true si le tableau est trié en ordre croissant, false sinon
 */
bool estTrie(vector<int> tab) {
    for(int i=0; i<tab.size()-1;i++) {
        if(tab[i] > tab[i+1])
            return false;
    }
    return true;
}


int main() {
    vector<int> tab_1 = { 5, -1, 3, 14, -4, 0,  7 };
    vector<int> tab_2 = { 8, 11, 9, 29, -3, 1, 13 };
    vector<int> tab_3 = { 1,  2, 3,  4,  5, 6,  7 };
    vector<int> tab_4 = { 10, 9, 8,  5,  2, 2, -1 };

    // Tests manuels; vous pouvez commenter les lignes dont vous n'avez pas e
    afficheTableau(tab_1);

    cout << "Les tableaux tab_1 et tab_2 sont égaux: "<< testEgaux(tab_1, tab
    cout << "Les tableaux tab_1 et tab_1 sont égaux: "<< testEgaux(tab_1, tab

    cout << "Nombre d'éléments plus grand que 10 dans tab_1: "<< plusGrandQue
    cout << "Nombre d'éléments plus grand que 10 dans tab_2: "<< plusGrandQue

    cout << "Le tableau tab_3 est trié: "<< estTrie(tab_3) << endl;
    cout << "Le tableau tab_4 est trié: "<< estTrie(tab_4) << endl;

    return 0;
}
```

(6) ♣ Transformer les tests manuels en tests automatiques (en utilisant ASSERT).

Exercice 2.

(1) Ouvrir le fichier `fibonacci.cpp` et l'enregistrer sous le nom : `fibonacci-tableau.cpp` (cela vous permet de garder la version fournie `fibonacci.cpp` dont vous aurez besoin ensuite, tout en pouvant modifier le fichier `fibonacci-tableau.cpp`. Variante ♣ : le faire avec la commande `cp`).

(2) Modifier les premières lignes du fichier `fibonacci-tableau.cpp` pour ajouter l'inclusion de la bibliothèque `vector` :

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

(3) Implanter la version "tableau" de la fonction `fibonacci` vue en TD.

(4) Vérifier (compilation ET exécution) que votre fonction renvoie bien la valeur attendue, et en particulier que les tests automatiques passent.

(5) De même, à partir du fichier `fibonacci.cpp` créer de nouveaux fichiers `fibonacci-variables.cpp` et `fibonacci-recursif.cpp`, et y implanter les deux autres fonctions fibonacci vues en TD. À chaque fois, vérifier avec les tests automatiques que l'on obtient les résultats attendus.

Correction :

```cpp
/** @file **/
#include <iostream>
#include <vector>
using namespace std;

/** Infrastructure minimale de test **/
#define ASSERT(test) if (!(test)) cout << "Test failed in file " << __FILE__ << " line "

/** Suite de Fibonacci (variante utilisant un tableau)
 *  @param n un entier strictement positif
 *  @return le n-ième terme de la suite de Fibonacci
 **/
int fibonacciTableau(int n) {
    if(n == 1 or n == 2) {
        return 1;
    } else {
        vector<int> u(n);
        u[0] = 1;
        u[1] = 1;
        for (int i = 2; i < n; i++) {
            u[i] = u[i-1] + u[i-2];
        }
        return u[n-1];
    }
}

/** Suite de Fibonacci (variante utilisant trois variables)
 *  @param n un entier strictement positif
 *  @return le n-ième terme de la suite de Fibonacci
 **/
int fibonacciVariables(int n) {
    int u1 = 1;
    int u2 = 1;
    int tmp = 0;
    if(n == 1 or n == 2) {
        return 1;
    } else {
        for (int i = 3; i <= n; i++) {
            tmp = u2;
            u2 = u1 + u2;
            u1 = tmp;
        }
        return u2;
    }
}

/** Suite de Fibonacci (variante récursive)
 *  @param n un entier strictement positif
```

```
 *  @return le n-ième terme de la suite de Fibonacci
 **/
int fibonacciRecursive(int n) {
    if(n == 1 or n == 2) {
        return 1;
    } else {
        return fibonacciRecursive(n-2) + fibonacciRecursive(n-1);
    }
}

void fibonacciTest() {
    // Remplacer par fibonacciRecursive ou fibonacciTableau pour
    // tester les autres fonctions
    ASSERT( fibonacciTableau(1) == 1 );
    ASSERT( fibonacciTableau(2) == 1 );
    ASSERT( fibonacciTableau(3) == 2 );
    ASSERT( fibonacciTableau(4) == 3 );
    ASSERT( fibonacciTableau(5) == 5 );
    ASSERT( fibonacciTableau(6) == 8 );
}

int main(){
    fibonacciTest();

    cout << "u_7 (par boucle): "  << fibonacciVariables(7) << endl;
    cout << "u_7 (recursif): "    << fibonacciRecursive(7) << endl;
    cout << "u_7 (par tableau): " << fibonacciTableau  (7) << endl;
    return 0;
}
```

Exercice ♣ 3.

(1) Implanter une fonction `bool contient(vector<int> tab, int x)` retournant vrai si le tableau `tab` contient l'élément `x` et faux sinon.

(2) Implanter une nouvelle fonction `vector<int> intersection(vector<int> tab1, vector<int> tab2)` retournant un vecteur des éléments communs de `tab1` et `tab2`.

Correction :

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

/** Vérifie si un tableau contient un élément
 * @param tab un tableau d'entiers
 * @param x un entier qu'on cherche
 * @return vrai si x est present dans le tableau tab
 */
bool contient(vector<int> tab, int x) {
    for (int i = 0; i < tab.size(); i++) {
        if (x == tab[i]) {
            return true;
        }
    }
    return false;
}

/** Donne les éléments communs de deux tableaux
 * @param tab1 un tableau d'entiers
 * @param tab2 un autre tableau d'entiers
 * @return un nouveau tableau qui contient les éléments communs aux deux tableaux
 */
vector<int> intersection(vector<int> tab1, vector<int> tab2) {
    vector<int> v;
    for (int i = 0; i < tab1.size(); i++) {
        if (contient(tab2, tab1[i])) {
            v.push_back(tab1[i]);
        }
    }
    return v;
}

int main() {
    vector<int> tab_1 = { 5, -1, 3, 14, -4, 0,  7 };
    vector<int> tab_2 = { 8, 11, 9,  2, -3, 1,  6 };
    vector<int> tab_3 = { 3, 11, 9, 12, -3, 1,  6 };

    cout << "Le tableau contient x: "<< contient(tab_1, -4) <<endl;
    cout << "Le tableau ne contient pas x: "<< contient(tab_1, 10) <<endl;

    vector<int> v1 = intersection(tab_1, tab_2);
    vector<int> v2 = intersection(tab_2, tab_3);
    cout << "Elements communs tab_1 et tab_2: "<<endl;
```

```
    for (int i = 0; i < v1.size(); i++) {
        cout << v1[i] << ' ' << endl;
    }
    cout << endl << "Elements communs tab_2 et tab_3: " << endl;
    for (int i = 0; i < v2.size(); i++) {
        cout << v2[i] << ' ';
    }
    cout << endl;

    return 0;
}
```

Exercice ♣ 4 (Code morse).
Rappel : les chaînes de caractères se manipulent comme les tableaux.

(1) Consulter le code de `morse.cpp`.

(2) La fonction `string morseCaractere(char a)` utilise une instruction `switch` que nous n'avons pas encore croisé. À quoi aurait ressemblé cette fonction si elle avait été écrite avec des if ?

(3) Ajouter une fonction `string morseChaine(string a)`, avec documentation et tests.

(4) Que fait la fonction `main` ?

(5) Renommer chacune des deux fonctions `morseCaractere` et `morseChaine` en `morse`. Que constate-t-on ?

Correction :

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

/** Infrastructure minimale de test **/
#define ASSERT(test) if (!(test)) cout << "Test failed in file " << __FILE__ << " l

/** Converti un caractère en code morse
 *  @param a un caractère en majuscule
 *  @return une chaîne de caractères correspondant au code morse de a
 **/
string morseCaractere(char a) {
    switch(a) {
    case 'A':
        return ".-";
    case 'B':
        return "-...";
    case 'C':
        return "-.-.";
    case 'D':
        return "-..";
    case 'E':
        return ".";
    case 'F':
        return "..-.";
    case 'G':
```

```
        return "--.";
    case 'H':
        return "....";
    case 'I':
        return "..";
    case 'J':
        return ".---";
    case 'K':
        return "-.-";
    case 'L':
        return ".-..";
    case 'M':
        return "--";
    case 'N':
        return "-.";
    case 'O':
        return "---";
    case 'P':
        return ".--.";
    case 'Q':
        return "--.-";
    case 'R':
        return ".-.";
    case 'S':
        return "...";
    case 'T':
        return "-";
    case 'U':
        return "..-";
    case 'V':
        return "...-";
    case 'W':
        return ".--";
    case 'X':
        return "-..-";
    case 'Y':
        return "-.--";
    case 'Z':
        return "--..";
    default:
        return "???";
    }
}

/** Converti une chaine de caractères en code morse
 *  @param a une chaine de caractères
 *  @return une chaîne de caractères correspondant au code morse de a
 **/
string morseChaine(string a) {
    string res = "";
    for (int i = 0; i < a.length(); i++) {
        res.append(morseCaractere(a[i]));
    }
    return res;
```

```
}

void morseChaineTest() {
    ASSERT( morseChaine("HELLO") == "......-...-..---");
    ASSERT( morseChaine("GOOD LUCK") == "--.-------.. ? ? ?.-....--.-.-.-");
}

int main() {
    morseChaineTest();

    string chaine;
    while (cin >> chaine) {
        cout << morseChaine(chaine) << endl;
    }
    return 0;
}
```

Exercice ♣ 5 (Python, Euler !)**.**
Pour des explications, voir les deux derniers exercices de la feuille de TP 2.