TD n° 6 (Correction)

Types structurés : Tableaux et struct mélangés.

Exercice 1. INTRODUCTION TABLEAU On va utiliser des tableaux simples, la différence principale avec les `vector` vus au premier semestre étant que leur taille doit être CONNUE au moment de la compilation. Voici un exemple de déclaration, pratique pour mettre directement des valeurs dans un tableau sans les saisir, ainsi que le code d'une procédure qui affiche un tableau afin que vous ayez un exemple de parcours de tableau.

```
const int TAILLE=5;
int monTableau[TAILLE] = {53, 14, 27, 2, 31};

void afficheTableau(int t[TAILLE]) {
    for (int i = 0; i < TAILLE; i++) cout << t[i] << " ";
    cout << endl;
}
```

Il existe de nombreux algorithmes pour trier des tableaux. L'un des plus simples (mais pas le plus efficace) est l'algorithme dit du *tri à bulles*, qui fonctionne de la façon suivante : on compare les éléments du tableau en position `0` et `1` et si le premier est supérieur au second, on les permute. Puis, on compare les éléments en position `1` et `2`, si l'élément en position `1` est plus grand que celui en position `2`, on les permute. On continue ainsi jusqu'à atteindre la fin du tableau. Le plus grand nombre va ainsi remonter comme si c'était une bulle.

Arrivé à la fin du tableau, on recommence le processus pour remonter la bulle d'après, etc. Si on réalise un parcours complet du tableau sans avoir eu à effectuer de permutations, cela signifie que le tableau est trié.

Exemple de tri, si on affiche le contenu du tableau après chaque permutation d'éléments.

53	14	27	2	31	état initial
14	53	27	2	31	step 1
14	27	53	2	31	step 2
14	27	2	53	31	step 3
14	27	2	31	53	step 4, la première bulle 53 est remontée !
14	2	27	31	53	on remonte la 3ème bulle (27) !
2	14	27	31	53	après ce dernier échange, plus de permutations, c'est fini

1. Écrire la procédure `remonteUneBulle` qui remonte ... une bulle. Appliquée une fois à notre tableau, elle remontera le 53, en effectuant les étapes intitulées step1 à step4.
2. Modifier la procédure `remonteUneBulle` pour qu'elle signale dans un paramètre résultat si elle a effectué ou pas une permutation, puis écrire la procédure `triBulle` qui trie tout le tableau et s'arrête quand elle a réalisé un parcours complet du tableau sans avoir eu à effectuer de permutations.
3. Tester dans le programme principal en triant `monTableau`, puis en l'affichant.

Correction :

```
void permute(int &a, int &b) {
  int temp = a; a = b; b = temp;
}

void remonteUneBulle(int t[TAILLE]) {
    for (int i = 0; i < TAILLE-1; i++) { // cela demarre a 0
        if (t[i] > t[i+1])  permute(t[i], t[i+1]);
```

```
    }
}

void remonteUneBulle2(int t[TAILLE], bool &trie) {
    trie = true;
    for (int i = 0; i < TAILLE -1; i++) {
        if (t[i]>t[i+1])  {
            trie = false;
            permute(t[i], t[i+1]);
        }
    }
}

void triBulle(int t[TAILLE]) {
    bool trie;
    do {
        remonteUneBulle2(t, trie);
    } while (not trie);
}

int main() {
    afficheTableau(monTableau);
    triBulle(monTableau);
    afficheTableau(monTableau);
    return 0;
}
```

Exercice 2. Le Bridge est un jeu de cartes qui se joue à 4 joueurs avec un jeu de 52 cartes, i.e. 13 cartes dont les valeurs appartiennent à l'ensemble énuméré :
enum **valeurCarte** {v2, v3, v4, v5, v6, v7, v8, v9, v10, Valet, Dame, Roi, As}
dans chacune des couleurs de l'ensemble énuméré :
enum **couleurCarte** {pique, cœur, carreau, trefle}

Chacun des 4 joueurs reçoit aléatoirement une **main**, c'est-à-dire 13 cartes, dont il doit évaluer la **force**.

On choisit de se donner les représentations suivantes pour une carte et une main de 13 cartes.

```
Struct carte {
    valeurCarte valeur;
    couleurCarte couleur;
};
```

using **MainJ** = carte[13];

La **force** d'une main prend en compte deux aspects : les points d'Honneurs (ptH) et les points de Distribution (ptD).

Les **points d'Honneurs** d'une main s'évaluent en sommant la valeur de chacune des cartes présentes dans la main : chaque As vaut 4 points, chaque Roi, 3 points, chaque Dame 2 points et chaque Valet 1 point, les autres cartes ne valent rien.

Le tableau *ptHCarte* est un tableau d'entiers (int[13]) qui associe à chaque valeur de carte son nombre de points d'honneurs. Il sera supposé donné en variable globale.

ptHCarte :

v2	v3	v4	v5	v6	v7	v8	v9	v10	Valet	Dame	Roi	As
0	0	0	0	0	0	0	0	0	1	2	3	4

Les **points de Distribution** s'évaluent en décomptant 3 points pour une chicane (pas de carte

dans une couleur), 2 points pour 1 singleton (1 seule carte dans une couleur) et 1 point pour 1 doubleton (2 cartes dans une couleur).

Exemple de main :

0	1	2	3	4	5	6	7	8	9	10	11	12
As ♠	Roi ♠	v10 ♠	v2 ♠	As ♡	Dame ♡	v10 ♡	v9 ♡	v8 ♡	v7 ♡	v4 ♡	Valet ♣	v6 ♣

La main présentée dans le tableau ci-dessus vaut donc : ptH = 14 et ptD = 4 (3 points pour la chicane à carreau + 1 point pour le doubleton à trèfle).

1. Réalisez la fonction **nbreCarteCouleur** qui prend en *Données* une **main** et une **couleurCarte** et qui retourne le nombre de cartes de la main qui ont la couleur donnée.
2. Réalisez la fonction **evaluePtD** qui prend en *Donnée* une **main** et retourne son nombre de points de Distribution (ptD).
3. Réalisez la fonction **evaluePtH** qui prend en *Donnée* une **main** et qui retourne son nombre de points d'honneurs (ptH).
4. On souhaite supprimer le tableau *ptHCarte*. Proposez une fonction permettant de le remplacer.

Correction : *On ne peut pas mélanger les entiers et les hautes cartes dans un type énuméré, d'où les "v" dans "v2, v3..."*

```
enum couleurCarte {pique, coeur,carreau, trefle};
enum valeurCarte {v2, v3, v4, v5, v6, v7, v8, v9, v10, Valet, Dame, Roi, As};

int ptHCarte[13] = {0,0,0,0,0,0,0,0,0,1,2,3,4};

Les valeurs d'un type enumere (les constantes d'enumeration) etant codees par des entiers,
on retrouve bien la correspondance,  ptHCarte[Roi] = 3 et ptHCarte[As] = 4

int nbreCarteCouleur(MainJ m, couleurCarte c) {
     // retourne le nbre de cartes de la main qui ont la couleur donnee//
     int nb = 0;
     for (int i = 0; i < 13; i++) {
          if ( m[i].couleur == c) nb++ ;
     }
     return nb;
}

int evaluePtD(MainJ  m) {
    int nb, pt, som = 0;
    for (couleurCarte c = pique; c <= trefle; c++ ) {
              nb = NbreCarteCouleur(m, c);
              if  (nb >= 3)  pt = 0;
              else  pt = (3 - nb);
              som = som + pt;
    }
    return som;
}

int evaluePtH(MainJ m) {
    int som = 0;
    for (int i = 0; i < 13; i++) {
         som = som + ptHCarte[m[i].valeur]  ;
    }
    return som;
}
```

```
// la fonction qui remplace le tableau, avec vc de type valeurCarte
int ptHCarte(valeurCarte vc){
    int res;
    switch (vc) {
    case Valet : res = 1; break;
    case Dame : res = 2; break;
    case Roi : res = 3; break;
    case As : res = 4; break;
    default : res = 0;
    }
    return res;
}

L'objectif est de leur faire manipuler un switch
mais en considerant que vc est code par un entier, on pourrait aussi ecrire  ceci,

int ptHCarte(int vc){
   if (vc < 9) return 0;
   else return vc - 8;
}
```

Exercice 3. On rappelle que le type `string` est un type prédéfini pour les chaînes de caractères.

On suppose que l'on dispose d'une procédure `lectureChaine`(string &Ch) qui lit un texte au clavier et nous transmet en résultat une chaîne ne contenant que des caractères alphabétiques minuscules non accentués ou des _ pour représenter des espaces blancs.

Etant donnée une chaine transmise par cette procédure, on souhaite identifier la lettre apparaissant le plus souvent dans la chaîne et afficher son nombre d'occurrences. On vous demande d'écrire :

1. la procédure **histogramme** qui étant donnée une chaîne de caractères, calcule et transmet en résultat le nombre d'occurrences de chaque lettre dans la chaîne sous la forme d'un histogramme (i.e. un tableau dont chaque indice est une lettre de l'alphabet en minuscule) ;
2. la fonction **maxOcc** qui prend en paramètre un histogramme et retourne la lettre d'occurrence maximale.
3. le programme principal utilisant les procédure et fonction précédentes.

Correction : *Exemple, pour la chaine ' le_ciel_est_bleu', le caractère 'e' a 4 occurences et ' l', 3*

En fait
- les blancs sont supposés être remplacés par des _ car sinon la lecture se fait mal
- on va utiliser le fait que les char sont concus pour représenter des caractères ou des petits entiers (l'équivalent ascii du caractère) et qu'on peut les utiliser dans une boucle for pour expliciter un parcours de 'a' à 'z'
- on va donc pourvoir utiliser les char comme indice de tableau, mais comme le 'a' par exemple se traduit par 97 en ascii ou autre chose avec une autre norme, pour éviter d'avoir des indices trop grand, on rangera le nbre occurrences d'un caractère c dans la case du tableau d'indice c -'a'

```
#include <iostream>
#include <cmath>
#include <string>

using namespace std;

const int taille=26;
```

```
using histo = int[taille];

void lectureChaine(string &ch) {
// Ne respecte pas tout a fait la specification de l'enonce
    cin >> ch;
}

void histogramme(string Ch, histo tab) {
    //initialisation du tableau tab de type histogramme
    for (char c = 'a'; c <= 'z'; c++) tab[c-'a'] = 0;
    // parcours de la chaine Ch
    for (int i = 0; i < Ch.size(); i++) {
        if (Ch[i] != '_') tab[Ch[i]-'a']= tab[Ch[i]-'a'] +1;
    }
}

char maxOcc(histo tab) {
    int max = 0;
    char c, ind_max;
    for (char c = 'a'; c <= 'z'; c++) {
        if (tab[c-'a'] > max) {
            max = tab[c-'a'];
            ind_max = c;
        }
    }
    return ind_max;
}

int main () {
    string ChaineLue;
    histo Tocc;
    char lmax;
    cout << "Entrez votre chaine" << endl;
    lectureChaine(ChaineLue);
    histogramme(ChaineLue, Tocc);
    lmax=maxOcc(Tocc);
    cout << "la lettre la + frequente est " << lmax << " avec "
        << Tocc[lmax-'a'] << " occurrences" << endl;
}
```