Algorithmique - Correction du TD5

IUT 1ère Année

20 janvier 2013

1 Les énumérations et structures

Exercice 1. Ecrire un algorithme permettant de résoudre le problème suivant :
- Données : une chaîne représentant un mot en minuscules (nom simple, verbe, adjectif) de la langue française.
- Résultat : le nombre de voyelles et le nombre de consonnes dans la chaîne.

Par exemple, le mot *élève* contient trois voyelles et deux consonnes.

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

// Retourne le nombre de voyelles d'un mot
int nombre_de_voyelles(const string& mot)
{
string voyelles = "aeiouyàâèéêëïîôùûœ";
int nbVoyelles = 0;
for(int i = 0; i < mot.size(); i++)
    {
    bool est_une_voyelle = 0;
    int j = 0;
    while(!est_une_voyelle && j < voyelles.size())
        {
        est_une_voyelle = (mot[i] == voyelles[j]);
        j++;
        }
    nbVoyelles += est_une_voyelle;
    }
return nbVoyelles;
}
```

Exercice 2. Ecrire un algorithme permettant de résoudre le problème suivant :
- Données : deux chaînes x et y représentant chacune un mot en majuscules (on n'utilise pas les accents) de la langue française.
- Résultat : *vrai* si x et y sont des anagrammes et *faux* sinon.

Rappelons que x est un *anagramme* de y si x est une permutation des lettres de y Par exemple, le mot CHIEN est un anagramme du mot NICHE. De la même manière, le mot LIERRE est un anagramme du mot RELIER. Mais LIRE n'est pas un anagramme de LIERRE puisqu'il n'utilise qu'un seul R et qu'un seul E.

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

```
// Trie les caractères d'un mot selon l'ordre lexicographique
void trier(string& mot)
{
int taille = mot.size();
for(int i = 0; i < taille - 1; i++)
    for(int j = i+1; j < taille; j++)
        if(mot[j] < mot[i]) swap(mot[i],mot[j]);
}

// Teste si deux mots sont des anagrammes
bool anagrammes(const string& mot1, const string& mot2)
{
string copie1(mot1);
string copie2(mot2);
trier(copie1);
trier(copie2);
return (copie1 == copie2);
}
```

Exercice 3. Ecrire un algorithme permettant de résoudre le problème suivant :

- Données : une chaîne de caractères x représentant un mot ou une expression de la langue française dont tous les caractères sont en majuscules.
- Résultat : *vrai* si x est un palindrome et faux sinon.

Rappelons que x est un *palindrome* s'il se lit de la même manière de gauche à droite et de droite à gauche. Par exemple, le mot RADAR et l'expression MON NOM sont des palindromes. Mais l'expression LE SEL n'est pas (strictement) un palindrome à cause du caractère 'espace'.

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

// Teste si un mot est un palindrome
bool palindrome(const string& mot)
{
int taille = mot.size();
int milieu = taille / 2;
bool est_symetrique = 1;
int i = 0;
while(est_symetrique && i < milieu)
    {
    est_symetrique = (mot[i] == mot[taille - i - 1]);
    i++;
    }
return est_symetrique;
}
```

Exercice 4. Ecrire un algorithme permettant de résoudre le problème suivant :

- Données : une phrase de la langue française ne contenant pas de symbole de ponctuation autre que le point final.
- Résultat : le nombre de mots dans la phrase.

Par exemple, la phrase "*Le chat est couché dans le canapé.*" contient 7 mots.

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

// Compte le nombre de mots dans une phrase
int nombre_de_mots(const string& phrase)
{
int i = 0, mots = 0;
while(i < phrase.size())
    {
    while(phrase[i] == ' ')
        i++;
    mots++;
    while(i < phrase.size() && phrase[i] != ' ')
        i++;
    }
return mots;
}
```

Exercice 5. Ecrire un algorithme permettant de résoudre le problème suivant :

- Données : une phrase P de la langue française et un mot m, tous deux écrits en majuscules.
- Résultat : le nombre d'occurrences de m dans P.

Par exemple, la phrase "LE CHAT S'APPELLE PACHA." contient 2 fois le mot LE et 2 fois le mot CHA.

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

// Compte le nombre d'occurrences de mot dans phrase (variante de l'algorithme vu dans le cours 7)
int nombre_de_mots(const string& phrase, const string& mot)
{
int i = 0, occurrences  = 0;
while(i < phrase.size() - mot.size())
    {
    bool trouve = 1;
    int  temp = i;
    int  j = 0;
    while(trouve && j < mot.size())
        {
        trouve = (mot[j] == phrase[i]);
        i++;
        j++;
        }
    occurrences += trouve;
    i = temp + 1;
    }
return occurrences;
}
```

2 Les tableaux multi-dimensionnels

Exercice 6. Ecrire un algorithme permettant de résoudre le problème suivant :

– Données : une matrice **M** de dimension $n \times n$
– Résultat : *vrai* si **M** est la matrice identité et faux sinon.

Dans la matrice identité, tous les éléments de sa diagonale sont égaux à 1 et tous les autres éléments sont égaux à 0.

```
using namespace std;

typedef float Matrice[n][n];

// Teste si une matrice M donnée est la matrice identité
bool identite(const Matrice& M)
{
int i = 0, int j = 0;
bool est_identite = 1;
while(est_identite && i < n)
    {
    while(est_identite && j < n)
        {
        if(i == j)
            est_identite = (M[i][j] == 1);
        else
            est_identite = (M[i][j] == 0);
        j++;
        }
    i++;
    }
return est_identite;
}
```

Exercice 7. Ecrire un algorithme permettant de résoudre le problème suivant :
– Données : deux matrices **A** et **B** de dimension $m \times n$
– Résultat : la somme **A** + **B** des deux matrices.

Rappelons que si **C** est la somme de **A** et de **B**, alors pour toute rangée $1 \leq i \leq m$ et toute colonne $1 \leq j \leq n$ de **C**, nous avons $c_{ij} = a_{ij} + b_{ij}$.

```
using namespace std;

typedef float Matrice[m][n];

// Calcule la somme C de deux matrices A et B
void somme(const Matrice& A, const matrice& B, matrice& C)
{
for(int i = 0; i < m; i++)
    for(int j = 0; j < n; j++)
        C[i][j] = A[i][j] + B[i][j];
}
```

Exercice 8. Ecrire un algorithme permettant de résoudre le problème suivant :
– Données : deux matrices **A** et **B** de dimension $n \times n$
– Résultat : le produit **AB** des deux matrices.

Rappelons que si **C** est le produit de **A** et de **B**, alors pour toute rangée $1 \leq i \leq n$ et toute colonne $1 \leq j \leq n$ de **C**, nous avons $c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$.

```
using namespace std;

typedef float Matrice[n][n];

// Calcule le produit C de deux matrices A et B
void somme(const Matrice& A, const matrice& B, matrice& C)
{
for(int i = 0; i < m; i++)
    for(int j = 0; j < n; j++)
        {
        float prod = 0;
        for(int k = 0; k < n; k++)
            prod += A[i][k] * B[k][j];
        C[i][j] = prod;
        }
}
```

Exercice 9*. Nous cherchons à construire un algorithme permettant d'effectuer des rotations d'objets en 2D. Etant donné un réel positif θ, la rotation en 2D selon l'angle θ est donnée par la matrice suivante :

$$\mathbf{R} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}$$

1. Ecrire une procédure qui prend en entrée un point **P** dans $\mathbb{R}^2$ et un angle θ, et qui effectue une rotation de **P** par θ.
2. Ecrire un algorithme qui prend en entrée un tableau de points (l'objet géométrique que l'on cherche à faire tourner) et un angle θ et qui effectue une rotation par θ de tous ses points.

```
// Si Q est une matrice et P un point, l'application de Q sur P est la matrice produit QP
// dans laquelle P est vu comme un vecteur colonne

using namespace std;
typedef float Matrice[2][2];
typedef float Point[2];

// Calcule la rotation du point P par la matrice R
void rotation(const Matrice& R, Point& P)
{
Point Q;
Q[0] *= (R[0][0]P[0] + R[0][1]P[1]);
Q[1] *= (R[1][0]P[0] + R[1][1]P[1]);
P[0] = Q[0];
P[1] = Q[1];
}

// Calcule la rotation de tout un objet O constitué de points
void rotation(const Matrice& R, vector<Point>& O)
{
for(int i=0; i< O.size(); i++)
    rotation(R,O[i]);
}
```

Exercice 10*. Le *tic-tac-toe* (morpion) est un jeu de plateau à deux joueurs, le joueur × et le joueur ∘. Le plateau est un tableau 3×3 initialement vide. A chaque tour de jeu, le joueur × place une croix dans une case vide, puis le joueur ∘ place un cercle dans une autre case vide. Le jeu se termine lorsqu'un des deux cas se produit :

- L'un des joueurs a rempli une rangée, une colonne ou une diagonale. Dans ce cas le joueur gagne 1 points et son adversaire -1 point.
- Tout le plateau est rempli, sans qu'aucune rangée, colonne ou diagonale n'ait été remplie par un des joueurs. Dans ce cas, les deux joueurs ont 0 point.

Vous devez construire un algorithme qui prend en entrée un état du jeu et teste si le jeu est terminé. Si c'est bien le cas, l'algorithme calcule le nombre de points gagnés par chaque joueur.

```
using namespace std;
// Modéliser les cases du jeu
enum     Case {croix,rond,vide};

// Modéliser le jeu
typedef Case TicTacToe[3][3];

// Modéliser les points : index 0 pour la croix, index 1 pour le rond
typedef Points[2];


// Teste si un joueur a gagné
bool Gagne(const TicTacToe& jeu, const Case joueur)
{
bool aGagne = 0;
int i = 0;

// Teste rangées
while(!aGagne && i < 3)
    {
    aGagne = jeu[i][0] == jeu[i][1] && jeu[i][1] == jeu[i][2] && jeu[i][2] == joueur;
    i++;
    }
i = 0;

// Teste colonnes
while(!aGagne && i < 3)
    {
    aGagne = jeu[0][i] == jeu[1][i] && jeu[1][i] == jeu[2][i] && jeu[2][i] == joueur;
    i++;
    }

// Teste diagonales
if(!aGagne)
    aGagne = jeu[0][0] == jeu[1][1] && jeu[1][1] == jeu[2][2] && jeu[2][2] == joueur;
if(!aGagne)
    aGagne = jeu[0][2] == jeu[1][1] && jeu[1][1] == jeu[2][0] && jeu[2][0] == joueur;

return aGagne;
}

// Teste si le jeu est rempli
```

```
bool Rempli(const TicTacToe& jeu)
{
bool estRempli = 1;
int i = 0;
while(estRempli && i < 3)
    {
    int j = 0;
    while(estRempli && j < 3)
        {
        estRempli = jeu[i][j] != vide;
        j++;
        }
    i++;
    }
return estRempli;
}

// Teste l'état du jeu et retourne des points si le jeu est fini
bool Tester(const TicTacToe& jeu, Points& points)
{
if(Gagne(jeu,croix))
    {
    points[croix]++;
    points[rond]--;
    return true;
    }
if(Gagne(jeu,rond))
    {
    points[croix]--;
    points[rond]++;
    return true;
    }
return Rempli(jeu);
}
```

Exercice 11*. *Puissance 4* est un jeu à deux joueurs constitué d'un plateau vertical de 6 rangées et de 7 colonnes. Le plateau est initialement vide. A chaque tour de jeu, le joueur *rouge* pose un jeton rouge dans une colonne j. Par gravité, le jeton vient au-dessus de la dernière case remplie de la colonne j. Le joueur *jaune* réalise la même opération en posant un jeton jaune dans une colonne. Vous devez construire une procédure qui prend en entrée un état du jeu, un joueur (jaune ou rouge) et une colonne (j), et produit en sortie l'état suivant du jeu. On supposera que la colonne j choisie par le joueur contient au moins une case vide.

```
using namespace std;
// Modéliser les cases du jeu
enum     Case {jaune,rouge,vide};

// Modéliser le jeu
typedef Case Puissance4[6][7];

// Mettre à jour le jeu selon l'action du joueur (choix d'une colonne)
void MettreAJour(TicTacToe& jeu, const Case joueur, const int colonne)
```

```
{
int rangee = 5;
while(rangee >= 0 && jeu[rangee][colonne] != vide)
    rangee--;

// Cas où la colonne est pleine (on ne fait rien)
if(rangee < 0)
    return;

// Cas où la colonne n'est pas pleine (on met à joueur)
jeu[rangee][colonne] = joueur;
}
```

Exercice 12*. Le jeu *Puissance 4* se termine lorsqu'un des deux cas se produit.

- L'un des joueurs a construit une rangée, colonne ou diagonale avec 4 pions consécutifs de sa couleur. Dans ce cas le joueur gagne 1 points et son adversaire -1 point.
- Tout le plateau est rempli, sans qu'aucune rangée, colonne ou diagonale de 4 pions consécutifs n'ait été construite par un des joueurs. Dans ce cas, les deux joueurs ont 0 point.

Comme pour l'exercice 10, vous devez construire un algorithme qui prend en entrée un état du jeu et teste si le jeu est terminé. Si c'est bien le cas, l'algorithme calcule le nombre de points gagnés par chaque joueur.

```
using namespace std;
// Modéliser les cases du jeu
enum Case {jaune,rouge,vide};

// Modéliser le jeu
typedef Case Puissance4[6][7];

// Modéliser les points : index 0 pour le jaune, index 1 pour le rouge
typedef Points[2];

// Teste si une rangée de 4 a été remplie par le joueur
bool RangeeDe4(const Puissance4& jeu, const Case joueur)
{
bool motif = 0;
int i = 0;
while(!motif && i < 6)
    {
    int j = 0;
    motif = 1;
    while(motif && j < 4)
        {
        motif = (jeu[i][j] == jeu[i][j+1]) && (jeu[i][j+1] == jeu[i][j+2]) && (jeu[i][j+2] == joueur);
        j++;
        }
    i++;
    }
return motif;
}

// Teste si une colonne de 4 a été remplie par le joueur
```

```
bool ColonneDe4(const Puissance4& jeu, const Case joueur)
{
bool motif = 0;
int j = 0;
while(!motif && j < 7)
    {
    int i = 5;
    motif = 1;
    while(motif && i > 2)
        {
        motif = (jeu[i][j] == jeu[i-1][j]) && (jeu[i-1][j] == jeu[i-1][j]) && (jeu[i-1][j] == joueur);
        i--;
        }
    j++;
    }
return motif;
}

// Teste si une diagonale de 4 a été remplie par le joueur
bool DiagonaleDe4(const Puissance4& jeu, const Case joueur)
{
bool motif = 0;
int j = 0;
while(!motif && j < 7)
    {
    int i = 5;
    // On teste les diagonales vers la droite
    if(j <= 3)
        {
        motif = 1;
        while(motif && i > 2)
            {
            motif = (jeu[i][j] == jeu[i-1][j+1]) && (jeu[i-2][j+2] == jeu[i-3][j+3])
                    && (jeu[i-3][j+3] == joueur);
            i--;
            }
        }
    i = 5;
    // On teste les diagonales vers la gauche
    if(!motif && j >= 3)
        {
        motif = 1;
        while(motif && i > 2)
            {
            motif = (jeu[i][j] == jeu[i-1][j-1]) && (jeu[i-2][j-2] == jeu[i-3][j-3])
                    && (jeu[i-3][j-3] == joueur);
            i--;
            }
        }
    j++;
    }
return motif;
}
```

```
// Teste si un joueur a gagné
bool Gagne(const Puissance4& jeu, const Case joueur)
{
return (RangeeDe4(jeu,joueur) || ColonneDe4(jeu,joueur) || DiagonaleDe4(jeu,joueur));
}

// Teste si le jeu est rempli
bool Rempli(const Puissance4& jeu)
{
bool estRempli = 1;
int i = 0;
while(estRempli && i < 6)
    {
    int j = 0;
    while(estRempli && j < 7)
        {
        estRempli = jeu[i][j] != vide;
        j++;
        }
    i++;
    }
return estRempli;
}

// Teste l'état du jeu et retourne des points si le jeu est fini
bool Tester(const Puissance4& jeu, Points& points)
{
if(Gagne(jeu,jaune))
    {
    points[0]++;
    points[1]--;
    return true;
    }
if(Gagne(jeu,rouge))
    {
    points[0]--;
    points[1]++;
    return true;
    }
return Rempli(jeu);
}
```