LIFAP1 – TP7 : Tableaux à 2 dimensions

Objectifs : Manipulation des tableaux à 2 dimensions
Utilisation de tailles variables dans les tableaux

1. Tableaux 2D : la base
   a. Soit un tableau d'entiers à deux dimensions de taille maximum `TAILLE_LIGNE` et `TAILLE_COLONNE`. Écrivez une procédure qui, à partir du nombre de lignes et de colonnes données par l'utilisateur remplit les `tailleL * tailleC` cellules de ce tableau.
   b. Écrivez une procédure qui affiche (proprement) le contenu du tableau précédent.

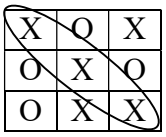
```
#include <iostream>
using namespace std;

// Dimension max d'un tableau 2D
const int TAILLE_LIGNE = 100;
const int TAILLE_COLONNE = 100;

// Afficher tous les elements d'un tableau 2D : tailleL x tailleC
void tabRemplir(int T[TAILLE_LIGNE][TAILLE_COLONNE], int tailleL, int tailleC)
{  int ligne,col;
   for (ligne=0; ligne<tailleL; ligne++)
   {   for (col=0; col<tailleC; col++) {
         cout << " Donnez la valeur " << endl;
           cin>>T[ligne][col];
      }
   }
}
void tabAff(int T[TAILLE_LIGNE][TAILLE_COLONNE], int tailleL, int tailleC)
{  int ligne,col;
   for (ligne=0; ligne<tailleL; ligne++)
   {   for (col=0; col<tailleC; col++) {
         cout << "  " << T[ligne][col];
      }
      cout << endl;
   }
}

int main(void)
{
   int T[TAILLE_LIGNE][TAILLE_COLONNE];
   int tL, tC, l, c;
   cout << "Tableaux à 2 dimensions " << endl;
   cout << "Combien de lignes ? (< TAILLE_LIGNE)";     cin >> tL;
   cout << "Combien de colonnes ? (< TAILLE_COLONNE)  ";   cin >> tC;
   tabRemplir(T, tL, tC);
   cout << "Afficher le tableau 2D" << endl;
   tabAff(T, tL, tC);
   return 0;
}
```

2. Le morpion : des caractères dans des tableaux !!!

Dans cette partie nous allons programmer le jeu du morpion. Pour cela, vous avez besoin d'une grille 3*3 et de 2 joueurs ayant des pions différents (les croix et les ronds).

X	X	O
O	X	X
X	O	O

Grille sans



Grille avec

gagnant gagnant

A tour de rôle, chaque joueur positionne un de ses pions sur la grille. Le jeu se finit quand un joueur a réalisé une ligne, une colonne ou une diagonale avec ses pions (c'est le gagnant) ou quand la grille est pleine (pas de gagnant).
La grille est représentée par un tableau à 2 dimensions de caractères dont chaque case contiendra soit '_', soit 'O', soit 'X'. Pour réaliser l'implémentation de ce jeu, écrivez les sous-programmes suivants.

c. Initialisation de la grille du morpion à vide (caractère '_')
   `void initialiseGrille(char grille[3][3])`
d. Affichage de la grille du morpion : _ indique case vide, O pion joueur 1 et X pion joueur 2 :
   `void afficheGrille(char grille[3][3])`
e. Saisie des coordonnées du nouveau pion à mettre sur la grille. Si les coordonnées sont en dehors de la grille ou si la case possède déjà un pion, la saisie est refusée, un message d'erreur est affiché, et le joueur doit rejouer. Dans le cas où les coordonnées sont correctes, placer le pion sur la grille à cet emplacement.
   `void metUnPionSurLaGrille(char grille[3][3], char &joueur)`
f. Teste si l'un des joueurs a gagné (ligne, colonne ou diagonale remplie de pions semblables). Dans ce cas, affiche un message pour indiquer le joueur qui a gagné. S'il n'y a pas de gagnant, teste que la grille n'est pas pleine. Si elle est pleine, affiche un message indiquant qu'aucun des joueurs n'a gagné. Retourne TRUE si la grille est pleine ou si un joueur a gagné, FALSE sinon.
   `bool testeFinJeu(char grille[3][3], char joueur)`
g. Écrivez ensuite le programme principal permettant de dérouler la partie. En voici son algorithme :

```
Algorithme principal :
   Initialisation de la grille à vide
   Tant que (pas de gagnant ou pas grille pleine)
    Afficher grille
    Mettre un pion sur la grille
   Fin Tant que
```

```
#include <iostream.h>
#define NB_LIG 3
#define NB_COL 3

/*
 * Initialise la grille du morpion a vide
 */
void initialiseGrille(char grille[NB_LIG][NB_COL])
{
  int i, j;
  for (i=0; i<NB_LIG; i++) {
    for (j=0; j<NB_COL; j++) {
      grille[i][j] ='_';
    }
  }
}

/*
  Affiche la grille du morpion
  _ indique case vide, O pion joueur 1 et X pion jour 2
 */
void afficheGrille(char grille[NB_LIG][NB_COL])
{
  int i, j;
```

```
  for (i=0; i<NB_LIG; i++)
  {
    for (j=0; j<NB_COL; j++)
      cout<<grille[i][j] ;
    cout<<endl; /* fin de la ligne */
  }
}

/*
  Saisie les coordonnées du nouveau pion a mettre sur la grille
  Si les coordonnées sont en dehors de la grille ou si la case possède
  déjà un pion, la saisie est refusée, un message d'erreur est affichée,
  et le joueur doit rejoue
 */
void metUnPionSurLaGrille(char grille[NB_LIG][NB_COL],char &Joueur )
{
  int ligne, col;
  bool saisieCorrecte = false;

  cout<<"Numeros de ligne et de colonne: ";

  do {
    cin>>ligne>>col;
    cout<<endl;

    if ((ligne > 0) && (ligne <= NB_LIG) && (col > 0) && (col <= NB_COL)) {
      ligne--; /* enleve 1 pour etre compatible avec le tableau ayant des
        indices de 0 a NB_LIG-1 */
      col--;
      if (grille[ligne][col] != '_')
   cout<<"Cette case a deja ete remplie. Veuillez recommencer: "<<endl;
      else {
   saisieCorrecte = true;
   grille[ligne][col] = Joueur;
   if (Joueur == 'O')
     Joueur = 'X';
   else
     Joueur = 'O';
      }
    } else
      cout<<"Indice de ligne ou de colonne incorrect. Veuillez recommencer:"<<endl;
  } while (!saisieCorrecte);
}

/* Teste si l'un des joueurs a gagne (ligne, colonne ou diagonale remplit
   de pions semblables). Dans ce cas affiche un message pour indiquer le
   joueur qui a gagne.
   S'il n'y a pas de gagnant, teste que la grille n'est pas pleine. Si elle
   est pleine, affiche un message indiquant qu'aucun des joueurs a gagne
   Retourne TRUE si la grille est pleine ou si un joueur a gagne
            FALSE sinon
*/
bool testeFinJeu(char grille[NB_LIG][NB_COL],char Joueur)
 {
  int i,j;
  int joueurGagnant; /* pour connaitre quel est le gagnant ie soit CROIX soit ROND */
  bool estFini = false;

  /* Teste s'il y a un gagnant */
  /* L'algorithme utilise est le plus facile mais n'est pas le plus efficace
     car on n'utilise pas la position du dernier pion ajoute sur la grille. Cette information
     permettrait de reduire le temps de la recherche.
     De plus, cet algo suppose que la taille de la matrice est de 3 par 3
```

```
  */
  /* si la case 1,1 est VIDE, cela signifie que les diagonales, la ligne 1 et la colonne 1
ne sont
     pas gagnantes
  */
  if (grille[1][1] != '_') {
    if (/* colonne 1 */ ((grille[0][1] == grille[1][1]) && (grille[1][1] == grille[2][1])) ||
    /* ligne 1 */ ((grille[1][0] == grille[1][1]) && (grille[1][1] == grille[1][2])) ||
    /* diagonale */ ((grille[0][0] == grille[1][1]) && (grille[1][1] == grille[2][2])) ||
    /* autre diag */ ((grille[0][2] == grille[1][1]) && (grille[1][1] == grille[2][0]))) {
      joueurGagnant = grille[1][1]; /* ie ROND ou CROIX */
      estFini = true;
    }
  }

  /* si la case 0,0 est vide, cela signifie que la ligne 0 et le colonne 0 ne sont pas
gagnantes */
  if ((!estFini) && (grille[0][0] != '_')) {
    if ( /* ligne 0 */ ((grille[0][0] == grille[0][1]) && (grille[0][1] == grille[0][2])) ||
    /* colonne 0*/ ((grille[0][0] == grille[1][0]) && (grille[1][0] == grille[2][0]))) {
      joueurGagnant = grille[0][0];
      estFini = true;
    }
  }

  /* si la case 2,2 est vide, cela signifie que la ligne 2 et la colonne 2 ne sont gagnantes
*/
  if ((!estFini) && (grille[2][2] != '_')) {
    if ( /* ligne 2 */ ((grille[2][0] == grille[2][1]) && (grille[2][1] == grille[2][2])) ||
    /* colonne 2 */ ((grille[0][2] == grille[1][2]) && (grille[1][2] == grille[2][2]))) {
      joueurGagnant = grille[2][2];
      estFini = true;
    }
  }

  if (estFini) {
    cout<<"Felicitations au joueur ayant les ";
    if (joueurGagnant == 'O')
      cout<<"ronds ";
    else
      cout<<"croix ";
    cout<<"qui a gagne."<<endl;
    return true;
  }

  /* teste si la grille n'est pas pleine */
  for (i=0; i<NB_LIG; i++) {
    for (j=0; j<NB_COL; j++) {
      if (grille[i][j] == '_') /* Au moins une case est vide donc le jeu n'est pas fini */
   return false;
    }
  }

  cout<<"Egalite !"<<endl ;
  return true;

}

/*
  Initialise la grille a vide puis tant que la grille n'est pas pleine ou
  qu'il n'y a pas un gagnant, saisie les pions des joueurs et affiche la grille
 */
void main()
```

```
{
   char grille_morpion[NB_LIG][NB_COL];
   char Joueur='O';
  initialiseGrille(grille_morpion);
  do {
     metUnPionSurLaGrille(grille_morpion,Joueur);
     afficheGrille(grille_morpion);
  }while(!testeFinJeu(grille_morpion,Joueur));
}
```