[redacted]

Nom, prénom, numéro d'étudiant :

Coller ou agrafer ici | Coller ou agrafer ici | Coller ou agrafer ici

Université Paris Sud, Licence MPI, Info 111 — Examen du 14 décembre 2015 (deux heures)

Exercice 1	Exercice 2	Exercice 3	Exercice 4	Exercice 5	Total	Note / 20

Calculatrices et autres gadgets électroniques interdits.
Seul document autorisé : une feuille au format A4 avec, au recto, le résumé de la syntaxe C++ de la fiche de TD 2 et, au verso, des notes manuscrites. Pour les étudiants inscrits en Français Langue Étrangère, un dictionnaire est autorisé.
Les exercices sont indépendants les uns des autres ; il n'est pas nécessaire de les faire dans l'ordre.
Les réponses sont à donner, autant que possible, sur le sujet ; sinon, mettre un renvoi.

Exercice 1 (Cours)**.**
Quelles sont les étapes pour construire un tableau à deux dimensions en C++ ?

Correction.
Un tableau à deux dimensions se construit en quatre étapes :

1. Déclaration du tableau
2. Allocation du tableau
3. Allocation de chaque ligne
4. Initialization

Illustrer votre réponse avec la construction d'un tableau à deux dimensions avec 7 lignes et 6 colonnes, initialisé de sorte que la valeur à la ligne i et colonne j soit $i + j$.

Correction.

```
vector<vector<int>> grille (6);
for(int i=0; i<grille.size(); i++) {
    grille[i] = vector<int> (7);
    for(int j=0; j<grille[i].size(); j++)
        grille[i][j] = i+j;
}
```

Exercice 2 (Triangle de Pascal)**.**

1. On rappelle que $n! = 1 \times 2 \times \cdots \times n$. Implanter la fonction factorielle dont la documentation et les tests sont donnés ci-dessous.

```
/** La fonction factorielle
 *  @param n un nombre entier positif
 *  @return n!
 **/
int factorielle(int n) {
```

```
void factorielleTest() {
    ASSERT( factorielle(0) == 1  );
    ASSERT( factorielle(1) == 1  );
    ASSERT( factorielle(2) == 2  );
    ASSERT( factorielle(3) == 6  );
    ASSERT( factorielle(4) == 24 );
}
```

Correction.

```
int factorielle(int n) {
    int resultat = 1;
    for ( int k = 1; k <= n; k++ ) {
        resultat = resultat * k;
    }
    return resultat;
}
```

2. On rappelle que le coefficient binomial est défini par $\binom{n}{k} = \frac{n!}{k!(n-k)!}$. Écrire la documentation d'une fonction binomial calculant le coefficient binomial $\binom{n}{k}$, puis implanter cette fonction.
Correction.

```
/** La fonction binomial
 *  @param n un nombre entier positif
 *  @param k un nombre entier positif
 *  @return le coefficient binomial n,k
 **/
int binomial(int n, int k) {
    return factorielle(n) / ( factorielle(k) * factorielle(n-k) );
}
```

3. Écrire quelques tests pour la fonction binomial.
Correction.

```
    ASSERT( binomial(2, 5) == 15 );
    ASSERT( binomial(1, 1) == 1  );
    ASSERT( binomial(0, 0) == 1  );
```

4. Le triangle de Pascal est une présentation des coefficients binomiaux dans un triangle. Par exemple la ligne d'indice 2 d'un triangle de Pascal correspond aux trois coefficients binomiaux $\binom{2}{0}$, $\binom{2}{1}$ et $\binom{2}{2}$ qui valent, respectivement, $1, 2$ et 1. Ainsi, la ligne d'indice n correspond aux $n+1$ coefficients binomiaux

$$\binom{n}{0}, \binom{n}{1}, \ldots, \binom{n}{n}.$$

Écrire un programme qui affiche les dix premières lignes du triangle de Pascal comme ci-dessous :

```
1
1  1
```

```
1  2  1
1  3  3  1
1  4  6  4  1
1  5  10   10   5  1
1  6  15   20   15   6  1
1  7  21   35   35   21   7  1
1  8  28   56   70   56   28   8  1
1  9  36   84   126  126  84   36   9  1
```

Pour bien aligner les nombres, on pourra les séparer par des tabulations (caractère "\t").
Correction.

```
void pascal(int nmax) {
    for (int n=0; n < nmax; n++) {
        for (int k=0; k <= n; k++) {
            cout << binomial(n, k) << "\t";
        }
        cout << endl;
    }
}
```

5. Quelles sont les complexités
   – de la fonction factorielle ,
   – de la fonction binomial,
   – du programme.

   Bien préciser le *modèle de calcul* et *justifier* les réponses.

   Correction.

   Modèle de calcul : taille du problème : n ; opérations élémentaires : multiplications et divisions

   Complexité de la fonction factorielle : $O(n)$

   Complexité de la fonction binomial : $O(n)$ (3 appels à factorielle, pour des valeurs inférieures à n).

   Complexité du programme : $O(n^3)$: n lignes de au plus n coefficient dont le calcul est de complexité n.

6. Quelle(s) ligne(s) faut-il changer dans votre programme, et comment, pour que le triangle de Pascal soit écrit dans un fichier « pascal.txt » ?
Correction.

```
void pascalFichier(int nmax) {
    ofstream fichier("pascal.txt");
    for (int n=0; n < nmax; n++) {
        for (int k=0; k <= n; k++) {
            fichier << binomial(n, k) << "\t";
        }
        fichier << endl;
    }
    fichier.close();
}
```

7. ♣ On rappelle que les coefficients binomiaux satisfont la formule récursive suivante :

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}.$$

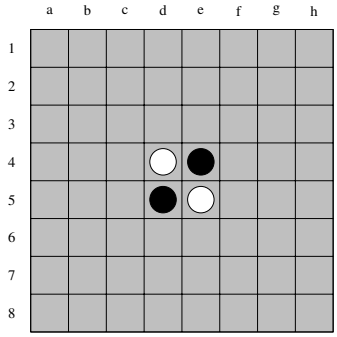
Cela se traduit, sur le triangle de Pascal, par le fait que chaque entrée est la somme de l'entrée située au-dessus à gauche et de celle située au-dessus dans la même colonne.

En utilisant cette propriété, calculer et afficher les dix premières lignes du triangle de Pascal. On pourra, au choix, modifier la fonction binomial ou le programme.
Correction.

```
int binomial_recursif(int n, int k) {
    if ( k > n ) {
        return 0;
    }
    if ( n <= 1 or k == 0 ) {
        return 1;
    } else {
        return binomial(n-1,k) + binomial(n-1,k-1);
    }
}
```

Problème

Othello est un jeu qui oppose deux joueurs, Noir et Blanc. Le jeu se joue sur un plateau de 64 cases (8x8). Chaque joueur dispose de 64 pions. En début de partie, quatre pions sont disposés au centre du plateau, comme sur la figure ci-dessous. Chaque joueur à tour de rôle pose un pion de sa couleur sur le plateau et *capture* quelques pions de la couleur opposée en les remplaçant par des pions de sa couleur, selon une règle qui sera détaillée plus tard. Le joueur Noir joue en premier. Si un joueur n'a pas de coup jouable, il passe son tour. La partie s'arrête lorsque plus aucun des deux joueurs ne peut poser de pions. Le gagnant est alors le joueur qui a le plus de pions de sa couleur sur le plateau. La partie est nulle en cas d'égalité.



Dans ce problème, on s'intéresse à l'écriture d'un programme pour jouer à Othello. Par convention, on représentera par 1 le joueur Noir ou un de ses pions, par 2 le joueur Blanc ou un de ses pions, et par 0 une case vide. On représentera le plateau d'othello par un vector<vector<**int**>>, de sorte que plateau [2][5] représente la case f3 du plateau.

Exercice 3 (Fin de partie)**.**

On suppose que l'on a à disposition la fonction suivante :

```
/** Teste si le joueur à un coup valide
 *  @param plateau une plateau d'othello
 *  @param joueur un entier 1 ou 2 désignant le joueur
 *  @return un boléen
 **/
bool peutJouer(vector<vector<int>> plateau, int joueur) {
```

1. Implanter une fonction estTerminee qui renvoie **true** si une partie est terminée, et **false** sinon.
   Correction.

```
bool estTerminee(vector<vector<int>> plateau) {
    return not (peutJouer(plateau, 1) or peutJouer(plateau, 2));
}
```

2. Implanter la fonction gagnant suivante :

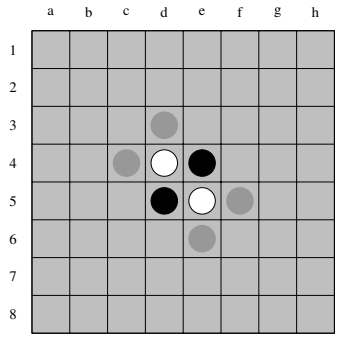
```
/** Renvoie le joueur gagnant, ou 0 en cas de partie nulle
 *  @param plateau une plateau d'othello d'une partie terminée
 *  @return le joueur gagnant, ou 0 en cas de partie nulle
 **/
int gagnant(vector<vector<int>> plateau) {
```

Correction.

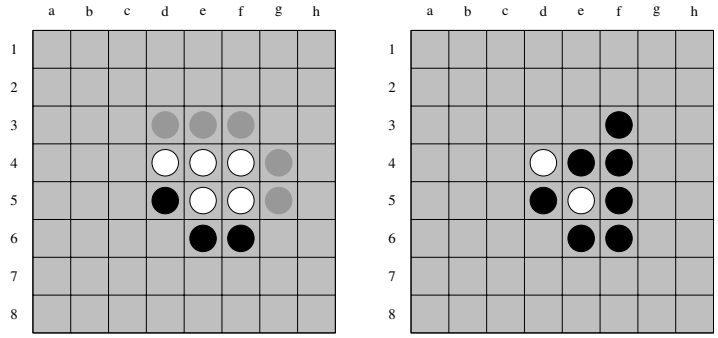
```
int gagnant(vector<vector<int>> plateau) {
    vector<int> compteur(3);
    for (int i = 0; i < 8; i++)
        for (int j = 0; j < 8; j++)
            compteur[plateau[i][j]]++;
    if (compteur[1] == compteur[2])
        return 0;
    else if (compteur[1] > compteur[2])
        return 1;
    else
        return 2;
}
```

Nous donnons maintenant les détails de la règle pour chaque coup de la partie.

Un joueur peut poser un pion sur une case donnée du plateau, *i.e.* le coup est jouable, si et seulement si la case est vide et le coup joué permet au joueur de capturer des pions adverses. Un joueur capture des pions adverses lorsqu'il pose un pion de sa couleur qui permet de fermer une ligne constituée de pions adverses et dont l'autre extrémité est un pion de sa propre couleur. La ligne ainsi capturée peut être horizontale, verticale ou diagonale. La figure suivante montre les coups valides du joueur Noir au début du jeu :



Lorsque un joueur capture des pions adverses, ils sont remplacés par des pions de sa propre couleur. Il n'y a pas de capture en cascade : on ne prend en compte que les lignes fermées par le pion qui vient d'être posé. La figure suivante montre un exemple de capture de pions, lorsque Noir choisit de jouer en f3 :



Exercice 4 (Coups jouables sur un plateau 1x8)**.**
Dans la suite de cet exercice, on considère un plateau d'une seule ligne et huit colonnes (1x8), représenté par un vector**<int>**.

1. On a la fonction estJouableDroite suivante :

```
1  bool estJouableDroite(vector<int> plateau, int joueur, int colonne) {
2      int adversaire = joueur % 2 + 1;
3      int j = colonne+1;
4      while (j < plateau.size() and plateau[j] == adversaire) {
5          j++;
6      }
7      if (j != colonne +1 and j < plateau.size() and plateau[j] == joueur) {
8          return true;
9      }
10     return false;
11 }
```

Que fait cette fonction ?

La documenter et expliquer son fonctionnement en commentant les lignes les plus importantes.
Correction.

```
/** Vérifie si des pions sont retournables à droite de colonne
 * @param plateau un plateau
 * @param joueur le joueur
 * @param colonne un numéro de colonne
 * @return true si des pions sont retournables à droite, false sinon
 **/
bool estJouableDroiteCorrection(vector<int> plateau, int joueur, int colonne) {
    int adversaire = joueur % 2 + 1; // Le numéro de joueur de l'adversaire
    int j = colonne+1; // La première case à droite de colonne
    while (j < plateau.size() and plateau[j] == adversaire)
        // Tant que les cases à droite appartiennent à l'adversaire
        // et qu'on ne dépasse pas les limites de la plateau
        j++; // on va une case à droite
    if (j != colonne +1 and j < plateau.size() and plateau[j] == joueur)
        return true; // Si la dernière case visitée appartient à joueur, ds pions sont ret
    return false;
}
```

2. Compléter les tests ci-dessous avec au moins deux autres appels pertinents à ASSERT.

```
void estJouableDroiteTest() {
    ASSERT(     estJouableDroite({0,0,0,1,1,2,0,0}, 2, 2));
    ASSERT(not estJouableDroite({0,0,0,0,0,2,2,2}, 1, 4));




}
```

3. En s'inspirant de la fonction estJouableDroite , implanter une fonction estJouableGauche.
Correction.

```
bool estJouableGauche(vector<int> plateau, int joueur, int colonne) {
    int adversaire = joueur % 2 + 1; // Le numéro de joueur de l'adversaire
    int j = colonne-1; // La première case à droite de colonne
    while (j >= 0 and plateau[j] == adversaire) { // Tant que les cases à gauche appartiennent à
l'adversaire et qu'on ne dépasse pas les limites de la plateau
        j--; // on va une case à droite
    }
    if (j != colonne - 1 and j >= 0  and plateau[j] == joueur) {
        return true; // Si la dernière case visitée appartient à joueur, ds pions sont retournab
    }
    return false;
}
```

4. En utilisant les fonctions estJouableDroite et estJouableGauche, implanter une fonction

```
bool estJouable(vector<int> plateau, int joueur, int colonne) {
```

qui renvoie **true** si joueur peut jouer dans colonne et **false** sinon.
Correction.

```
bool estJouable(vector<int> plateau, int joueur, int colonne) {
    if (colonne >= plateau.size() or colonne < 0 or plateau[colonne] != 0) {
        return false;
    }
    return (estJouableDroite(plateau, joueur, colonne) or estJouableGauche(plateau, joueur, colo
}
```

Exercice 5 (Jeu sur un vrai plateau 8x8)**.**
Dans cet exercice, on considère à nouveau un vrai plateau d'othello, de dimensions 8x8.

On suppose que l'on a à disposition les fonctions suivantes :

```
/** Initialise la plateau
 *  @return une plateau d'othello initiale
 **/
vector<vector<int>> plateauInitiale() {
```

```
/** Fait jouer un des joueurs
 *  @param plateau une plateau d'othello
 *  @param joueur un entier 1 ou 2 désignant le joueur
 *  @return la plateau mise à jour après le coup
 **/
vector<vector<int>> unCoup(vector<vector<int>> plateau, int joueur) {
```

1. La partie principale du jeu est implantée par la fonction ci-dessous :

```
1  void othelloBogguee() {
2      vector<vector<int>> plateau = plateauInitiale();
3      int joueur = 0;
4      affiche(plateau);
5      do {
6          if (peutJouer(plateau, joueur))
7              plateau = unCoup(plateau, joueur);
8          joueur = joueur % 2 +1;
9          affiche(plateau);
10     } while ( estTerminee(plateau) );
11     bool aGagne = gagnant(plateau);
12     if ( aGagne == 0 )
13         cout << "Égalité !";
14     else
15         cout << "Joueur " << aGagne << " a gagné !";
16 }
```

La fonction othelloBogguee contient des bogues à corriger l'un après l'autre. Dans chaque cas, **identifier le numéro de la ligne fautive et comment la corriger**.

(a) La première exécution de la boucle n'a aucun effet.

Correction.

Ligne 3 : `int joueur = 1`

(b) Dès la fin du premier coup, la partie s'arrête.

Correction.

Ligne 10 : `while (not estTerminee(plateau));`

(c) S'il y a un gagnant, c'est toujours le joueur 1 qui est déclaré gagnant.

Correction.

Ligne 11 : `int aGagne = gagnant(plateau);`

2. En s'inspirant de la fonction estJouableDroite de l'exercice précédent, implanter une fonction :

```
estJouableHautDroite(vector<vector<int>> plateau, int joueur,
                     int ligne, int colonne) {
```

qui teste si le joueur peut, en posant un pion dans la case indiquée, capturer des pions adverses sur la diagonale partant vers le haut et la droite en partant de cette case.

3. ♣ Implanter une fonction

```
bool estJouableXY(vector<vector<int>> plateau, int joueur,
                  int ligne, int colonne, int dX, int dY) {
```

où le couple (dY, dX) $\in \{-1,0,1\} \times \{-1,0,1\} \setminus \{(0,0)\}$ représente une direction ; par exemple (0,1) pour droite, (-1,0) pour bas ou (-1,-1) pour bas gauche.

Correction.

```
bool estJouableXY(vector<vector<int>> plateau, int joueur,
                  int ligne, int colonne, int dX, int dY) {
    int i = ligne;
    int j = colonne;
    if (i + 2 * dY < 0 or i + 2 * dY > 7 or  j + 2 * dX < 0 or j + 2 * dX > 7)
        return false;
    if (plateau[i+dY][j+dX] != joueur % 2 +1 )
        return false;
    do {
        i += dY;
        j += dX;
    } while (i + dY >= 0 and i + dY <= 7 and
             j + dX >= 0 and j + dX < 7 and
             plateau[i][j] == joueur % 2 +1);
    return plateau[i][j] == joueur;
}
```