```
#include <iostream.h>

const int m_max = 50;
const int n_max = 50;
typedef bool MatrMN[m_max][n_max];

void lecture (MatrMN laby, int m, int n);
int gauche (MatrMN laby, int m, int n);
int droite (MatrMN laby, int m, int n);
bool meilleur (MatrMN laby, int m, int n);

/* Deux fonctions, l'une suit le mur de droite, l'autre le mur de gauche
 *  Celle-ci suit le mur de droite */
int
droite (MatrMN laby, int m, int n)
{
        // dir, entier deà 1  4 contenant la direction, 0 vers le bas, 1 vers la gauche
        // etc... i numero de la ligne, j numero de colonne. cpt compte le nombre de pas
        // parcourus
        int dir = 0, i = 0, j = 1, cpt = 1;
        do
        {
                // Serie de test destines a tester la direction a prendre
                if (dir == 0)   // Dans le cas d'un deplacement vers le bas:
                {
                        if (laby[i][j - 1] == 0)        // On verifie si le passage est
                                                        // libre a droite
                                dir = (dir + 3) % 4;    // Tourner d'un quart de tour
                                                        // vers la droite revient a
                                                        // tourner de 3/4 vers la gauche
                                                        // (car -1%4 = -1, nous voudrions
                                                        // 3)
                        else if (laby[i + 1][j] != 0)   // Si le passage est bloque
                                                        // devant:
                                if (laby[i][j + 1] == 0)        // On verifie s'il est
                                                                // libre a droite
                                        dir = (dir + 1) % 4;    // et on y tourne
                                else
                                        dir = (dir + 2) % 4;    // Sinon on fait demi
                                                                // tour.
                }
                else if (dir == 1)      // Test du cas d'un deplacement vers la droite.
                {
                        if (laby[i + 1][j] == 0)
                                dir = (dir + 3) % 4;
                        else if (laby[i][j + 1] != 0)
                                if (laby[i - 1][j] == 0)
                                        dir = (dir + 1) % 4;
                                else
                                        dir = (dir + 2) % 4;
                }
                else if (dir == 2)      // Cas d'un deplacement vers le haut.
                {
                        if (laby[i][j + 1] == 0)
                                dir = (dir + 3) % 4;
                        else if (laby[i - 1][j] != 0)
```

```
                        if (laby[i][j - 1] == 0)
                                dir = (dir + 1) % 4;
                        else
                                dir = (dir + 2) % 4;
                }
                else if (dir == 3)      // Cas d'un deplacement vers la droite;
                {
                        if (laby[i - 1][j] == 0)
                                dir = (dir + 3) % 4;
                        else if (laby[i][j - 1] != 0)
                                if (laby[i + 1][j] == 0)
                                        dir = (dir + 1) % 4;
                                else
                                        dir = (dir + 2) % 4;
                }
                cpt++;                  // Compte le nombre de pas franchi. Modifie les
                                        // coordonees du "curseur" en fonction de la direction
                                        // choisie precedement.
                switch (dir)
                {
                case 0:
                        i++;
                        break;
                case 1:
                        j++;
                        break;
                case 2:
                        i--;
                        break;
                case 3:
                        j--;
                        break;
                }
        }
        while (i > 0 && j > 0 && i < n - 1 && j < m - 1);        // La boucle s'execute
                                                                 // tant que l'on
                                                                 // n'atteint pas un bord.
        return cpt;
}

// Le principe est le meme pour la fonction gauche, mais on procede dans l'autre sens
int
gauche (MatrMN laby, int m, int n)
{
        int dir = 0, i = 0, j = 1, cpt = 1;
        do
        {
                if (dir == 0)
                {
                        if (laby[i][j + 1] == 0)         // On verifie si on peut tourner
                                                         // a gauche
                                dir = (dir + 1) % 4;
                        else if (laby[i + 1][j] != 0)    // Si le chemin est bloque devant
                                if (laby[i][j - 1] == 0)         // On verifie s'il est
                                                                 // libre a droite
                                        dir = (dir + 3) % 4;
```

```
                        else
                                dir = (dir + 2) % 4;
                }
                else if (dir == 1)
                {
                        if (laby[i - 1][j] == 0)
                                dir = (dir + 1) % 4;
                        else if (laby[i][j + 1] != 0)
                                if (laby[i + 1][j] == 0)
                                        dir = (dir + 3) % 4;
                                else
                                        dir = (dir + 2) % 4;
                }
                else if (dir == 2)
                {
                        if (laby[i][j - 1] == 0)
                                dir = (dir + 1) % 4;
                        else if (laby[i - 1][j] != 0)
                                if (laby[i][j + 1] == 0)
                                        dir = (dir + 3) % 4;
                                else
                                        dir = (dir + 2) % 4;
                }
                else if (dir == 3)
                {
                        if (laby[i + 1][j] == 0)
                                dir = (dir + 1) % 4;
                        else if (laby[i][j - 1] != 0)
                                if (laby[i - 1][j] == 0)
                                        dir = (dir + 3) % 4;
                                else
                                        dir = (dir + 2) % 4;
                }
                cpt++;
                switch (dir)
                {
                case 0:
                        i++;
                        break;
                case 1:
                        j++;
                        break;
                case 2:
                        i--;
                        break;
                case 3:
                        j--;
                        break;
                }
        }
        while (i > 0 && j > 0 && i < n - 1 && j < m - 1);
        return cpt;
}

bool
meilleur (MatrMN laby, int m, int n)
```

```
{
        // La fonction meilleur renvoie vrai si le chemin de droite est plus court.
        return (droite (laby, m, n) < gauche (laby, m, n));
}

// Simple fonction destinee a afficher la matrice a l'ecran.
void
lecture (MatrMN laby, int m, int n)
{
        for (int i = 0; i < m; i++)
        {
                for (int j = 0; j < n; j++)
                        cout << laby[i][j] << " ";
                cout << endl;
        }
}

void
main ()
{
        const int m = 9, n = 9;
        int j;
        MatrMN laby = {
                {1, 0, 1, 1, 1, 1, 1, 1, 1},
                {1, 0, 1, 1, 1, 1, 1, 1, 1},
                {1, 0, 0, 0, 0, 0, 1, 1, 1},
                {1, 0, 1, 0, 1, 1, 1, 1, 1},
                {1, 0, 1, 0, 0, 0, 1, 1, 1},
                {1, 1, 1, 0, 1, 0, 0, 0, 0},
                {1, 1, 1, 0, 1, 0, 1, 1, 1},
                {1, 1, 1, 0, 0, 0, 1, 1, 1},
                {1, 1, 1, 1, 1, 1, 1, 1, 1}
        };


        lecture (laby, m, n);
        cout << "Longueur du trajet en suivant le mur de gauche: " <<
                gauche (laby, m,
                        n) << ", longueur en suivant le mur de droite: " <<
                droite (laby, m, n) << endl;
        cout << "Le chemin le plus court est celui parcouru en suivant le mur de ";
        if (meilleur (laby, m, n))
                cout << "droite" << endl;
        else
                cout << "gauche" << endl;
}
```