國立臺北科技大學

進階 C 語言實務

Homework 1

老師：蔣政諺

班級：電機碩一

學號：111318133

姓名：魏千竣

日期：112/04/21

1. Program Description

1. 匯入所需的標頭檔和使用命名空間 std。

```
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <vector>

using namespace std;
```

2. 定義常數 MATRIX_SIZE，用於設定矩陣的大小（4x4）。

```
const int MATRIX_SIZE = 4;
```

3. 定義一系列函式，用於執行矩陣的各種操作：

- printMatrix：印出矩陣。依序印出每一行列的值。

```
//印出矩陣
void printMatrix(int* matrix[MATRIX_SIZE]) {
    for (int i = 0; i < MATRIX_SIZE; ++i) {
        for (int j = 0; j < MATRIX_SIZE; ++j) {
            cout << matrix[i][j] << "\t";
        }
        cout << endl;
    }
    cout << "------------------------------" << endl;
}
```

- copyMatrix：複製矩陣內容。為了不讓後續的操作影響到原始矩陣的值，所以每次需要操作到矩陣內容時，都先複製一次矩陣內容到新的變數內操作。

```
// 複製矩陣內容
void copyMatrix(int* src[MATRIX_SIZE], int* dest[MATRIX_SIZE]) {
    for (int i = 0; i < MATRIX_SIZE; ++i) {
        for (int j = 0; j < MATRIX_SIZE; ++j) {
            dest[i][j] = src[i][j];
        }
    }
}
```

- findMinMaxValues：查找矩陣中的最大值和最小值以及它們的索引。首先將矩陣的第一個元素 matrix[0][0] 設置為最小值 minVal 和最大值 maxVal 的初始值。接著初始化最小值 minIdx 和最大值 maxIdx 的 index 為 {0, 0}。接著藉由 for 迴圈跟 if 去逐行逐列判斷，並更新最小值最大值，及他們的 index。最後再印出最大值和最小值及其索引。

```
// 查找最大值和最小值
void findMinMaxValues(int* matrix[MATRIX_SIZE]) {
    int minVal = matrix[0][0];
    int maxVal = matrix[0][0];
    pair<int, int> minIdx = {0, 0};
    pair<int, int> maxIdx = {0, 0};

    // 查找最大值和最小值以及它們的索引
    for (int i = 0; i < MATRIX_SIZE; ++i) {
        for (int j = 0; j < MATRIX_SIZE; ++j) {
            if (matrix[i][j] < minVal) {
                minVal = matrix[i][j];
                minIdx = {i, j};
            }
            if (matrix[i][j] > maxVal) {
                maxVal = matrix[i][j];
                maxIdx = {i, j};
            }
        }
    }

    // 輸出最大值和最小值及其索引
    cout << "Minimum value: " << minVal << " at index (" << minIdx.first << ", " << minIdx.second << ")" << endl;
    cout << "Maximum value: " << maxVal << " at index (" << maxIdx.first << ", " << maxIdx.second << ")" << endl;
    cout << "------------------------------" << endl;
}
```

- transposeMatrix：轉置矩陣。使用雙 for 迴圈，並在迴圈中使用 swap 函數，將當前元素 matrix[i][j]與對應的轉置元素 matrix[j][i]互換。且為了避免重複交換元素，故使內層迴圈的起始值為 j = i + 1。

```
// 轉置矩陣
void transposeMatrix(int* matrix[MATRIX_SIZE]) {
    for (int i = 0; i < MATRIX_SIZE; ++i) {
        for (int j = i + 1; j < MATRIX_SIZE; ++j) {
            swap(matrix[i][j], matrix[j][i]);
        }
    }
}
```

- rotateMatrix：將矩陣順時針旋轉 90 度。首先創建一個新的二維整數陣列 temp，大小與輸入矩陣相同，用來存儲旋轉後的矩陣。接著使用 for 迴圈，將當前元素 matrix[i][j]複製到旋轉 90 度後元素的新位置 temp[j][MATRIX_SIZE - 1 - i]。再來將 temp 中的元素值複製回輸入矩陣。最後，釋放 temp 佔用的內存。

```
// 順時針轉90度
void rotateMatrix(int* matrix[MATRIX_SIZE]) {
    int** temp = new int*[MATRIX_SIZE];
    for (int i = 0; i < MATRIX_SIZE; ++i) {
        temp[i] = new int[MATRIX_SIZE];
    }

    for (int i = 0; i < MATRIX_SIZE; ++i) {
        for (int j = 0; j < MATRIX_SIZE; ++j) {
            temp[j][MATRIX_SIZE - 1 - i] = matrix[i][j];
        }
    }

    for (int i = 0; i < MATRIX_SIZE; ++i) {
        for (int j = 0; j < MATRIX_SIZE; ++j) {
            matrix[i][j] = temp[i][j];
        }
    }

    for (int i = 0; i < MATRIX_SIZE; ++i) {
        delete[] temp[i];
    }
    delete[] temp;
}
```

- GetRow、GetColumn、GetDiagonal 和 GetInverseDiagonal：分別獲取矩陣的行、列、對角線和逆對角線。printRow、printCol、printDia 和 printInvD：分別印出行、列、對角線和逆對角線。它們的操作方式都是，首先創建一個整數向量，使用 for 迴圈將矩陣中需要的元素添加到向量中，最後返回向量。及引入向量，並用 for 迴圈將像量中的所有值印出來。

```cpp
// Get Row
vector<int> GetRow(int* matrix[MATRIX_SIZE], int rowIndex) {
    vector<int> row(MATRIX_SIZE);
    for (int j = 0; j < MATRIX_SIZE; ++j) {
        row[j] = matrix[rowIndex][j];
    }
    return row;
}

// 印出row
void printRow(const vector<int>& row) {
    for (int value : row) {
        cout << value << " ";
    }
    cout << endl;
    cout << "------------------------------" << endl;
}

// Get Column
vector<int> GetColumn(int* matrix[MATRIX_SIZE], int colIndex) {
    vector<int> column(MATRIX_SIZE);
    for (int j = 0; j < MATRIX_SIZE; ++j) {
        column[j] = matrix[j][colIndex];
    }
    return column;
}

// 印出Column
void printCol(const vector<int>& column) {
    for (int value : column) {
        cout << value << " ";
    }
    cout << endl;
    cout << "------------------------------" << endl;
}
```

```cpp
// Get Diagonal
vector<int> GetDiagonal(int* matrix[MATRIX_SIZE]) {
    vector<int> diagonal(MATRIX_SIZE);
    for (int i = 0; i < MATRIX_SIZE; ++i) {
        diagonal[i] = matrix[i][i];
    }
    return diagonal;
}

// 印出 Diagonal
void printDia(const vector<int>& diagonal) {
    for (int value : diagonal) {
        cout << value << " ";
    }
    cout << endl;
    cout << "------------------------------" << endl;
}

// Get Inverse Diagonal
vector<int> GetInverseDiagonal(int* matrix[MATRIX_SIZE]) {
    vector<int> inverse_diagonal(MATRIX_SIZE);
    for (int i = 0; i < MATRIX_SIZE; ++i) {
        inverse_diagonal[i] = matrix[i][MATRIX_SIZE-i-1];
    }
    return inverse_diagonal;
}

// 印出 Inverse Diagonal
void printInvD(const vector<int>& invdiagonal) {
    for (int value : invdiagonal) {
        cout << value << " ";
    }
    cout << endl;
    cout << "------------------------------" << endl;
}
```

- 使用公式分別計算 3x3 (determinant3x3)、4x4 (determinant)矩陣的行列式。

```
// 3*3determinant
int determinant3x3(int a, int b, int c, int d, int e, int f, int g, int h, int i) {
    return a * e * i + b * f * g + c * d * h - c * e * g - b * d * i - a * f * h;
}

// 4*4determinant
int determinant(int* matrix[MATRIX_SIZE]) {
    int result = 0; //初始化result
    for (int i = 0; i < MATRIX_SIZE; ++i) {
        int sign = (i % 2 == 0) ? 1 : -1; //使用變數 sign 記錄當前代數余子式的符號，當列的編號為偶數時，sign為1，否則為-1
        int sub_determinant = determinant3x3(
            matrix[1][(i + 1) % MATRIX_SIZE], matrix[1][(i + 2) % MATRIX_SIZE], matrix[1][(i + 3) % MATRIX_SIZE],
            matrix[2][(i + 1) % MATRIX_SIZE], matrix[2][(i + 2) % MATRIX_SIZE], matrix[2][(i + 3) % MATRIX_SIZE],
            matrix[3][(i + 1) % MATRIX_SIZE], matrix[3][(i + 2) % MATRIX_SIZE], matrix[3][(i + 3) % MATRIX_SIZE]
        );  //呼叫 determinant3x3 函數計算當前代數余子式的值，並儲存在變數 sub_determinant 中。
        result += sign * matrix[0][i] * sub_determinant; //用展開定理
    }
    return result;
}
```

- 使用公式分別計算矩陣的餘子式(getCofactor)和逆矩陣(inverseMatrix)，裡面有用 if 判斷是否有逆矩陣，如果不存在會印出"This matrix is singular, cannot find its inverse."。及印出逆矩陣(printInverseMatrix)。

```
// 計算cofactor matrix
void getCofactor(int* matrix[MATRIX_SIZE], int* temp[MATRIX_SIZE], int p, int q, int n) {
    int i = 0, j = 0;
    for (int row = 0; row < n; row++) {
        for (int col = 0; col < n; col++) {
            if (row != p && col != q) {
                temp[i][j++] = matrix[row][col];
                if (j == n - 1) {
                    j = 0;
                    i++;
                }
            }
        }
    }
}
```

```
// 計算逆矩陣
bool inverseMatrix(int* matrix[MATRIX_SIZE], double* inverse[MATRIX_SIZE]) {
    int det = determinant(matrix);
    if (det == 0) {
        cout << "This matrix is singular, cannot find its inverse." << endl;
        return false;
    }
    int** temp = new int*[MATRIX_SIZE];
    for (int i = 0; i < MATRIX_SIZE; ++i) {
        temp[i] = new int[MATRIX_SIZE];
    }

    for (int i = 0; i < MATRIX_SIZE; i++) {
        for (int j = 0; j < MATRIX_SIZE; j++) {
            getCofactor(matrix, temp, i, j, MATRIX_SIZE);
            int sign = ((i + j) % 2 == 0) ? 1 : -1;
            inverse[j][i] = (double)(sign * determinant(temp)) / (double)det;
        }
    }

    for (int i = 0; i < MATRIX_SIZE; ++i) {
        delete[] temp[i];
    }
    delete[] temp;

    return true;
}

// 印出逆矩陣
void printInverseMatrix(double* inverse[MATRIX_SIZE]) {
    for (int i = 0; i < MATRIX_SIZE; ++i) {
        for (int j = 0; j < MATRIX_SIZE; ++j) {
            cout << inverse[i][j] << " ";
        }
        cout << endl;
    }
    cout << "-----------------------------" << endl;
}
```

4.主程式:

- 動態分配二維陣列，用於儲存原始矩陣、轉置矩陣、旋轉矩陣和逆矩陣。

```
int main() {
    // 動態分配二維陣列
    int** matrix = new int*[MATRIX_SIZE];
    int** transpose = new int*[MATRIX_SIZE];
    int** rotate = new int*[MATRIX_SIZE];
    double** inverse = new double*[MATRIX_SIZE];
    for (int i = 0; i < MATRIX_SIZE; ++i) {
        matrix[i] = new int[MATRIX_SIZE];
        transpose[i] = new int[MATRIX_SIZE];
        rotate[i] = new int[MATRIX_SIZE];
        inverse[i] = new double[MATRIX_SIZE];
    }
```

- 用隨機數填充原始矩陣，並印出原始矩陣。

```
// 填充隨機數據
srand(time(0));
for (int i = 0; i < MATRIX_SIZE; ++i) {
    for (int j = 0; j < MATRIX_SIZE; ++j) {
        matrix[i][j] = rand() % 100;
    }
}

cout << "matrix:" << endl;
printMatrix(matrix);
```

- 尋找並印出原始矩陣中的最大值和最小值及其索引。

```
// 印出最大最小值及index
findMinMaxValues(matrix);
```

- 計算並印出轉置矩陣和旋轉矩陣。為了避免影響原始矩陣，故一開始都有先複製原始矩陣。

```
//印出transpose
copyMatrix(matrix, transpose); //使matrix的值先複製到transpose
transposeMatrix(transpose);
cout << "traspose:" << endl;
printMatrix(transpose);

//印出Rot90
copyMatrix(matrix, rotate); //使matrix的值先複製到rotate
rotateMatrix(rotate);
cout << "Rot90:" << endl;
printMatrix(rotate);
```

- 獲取並印出矩陣的行、列、對角線和逆對角線。

```
// 印出ROW
int rowIndex = 1; // 以第1行為例(從0開始計數)
vector<int> row = GetRow(matrix, rowIndex);
cout << "NO. " << rowIndex << " row:" << endl;
printRow(row);

// 印出COLUMN
int colIndex = 2; // 以第1列為例(從0開始計數)
vector<int> col = GetColumn(matrix, colIndex);
cout << "NO. " << colIndex << " column:" << endl;
printCol(col);

// 印出DIAGONAL
vector<int> dia = GetDiagonal(matrix);
cout << "diagonal:" << endl;
printDia(dia);

// 印出Inverse DIAGONAL
vector<int> invdia = GetInverseDiagonal(matrix);
cout << "inverse diagonal:" << endl;
printInvD(invdia);
```

- 計算並印出矩陣的行列式。

```
// 印出determinant
int det = determinant(matrix);
cout << "determinant: " << det << std::endl;
cout << "------------------------------" << endl;
```

- 計算並印出矩陣的逆矩陣（如果存在）。

```
// 印出inverse matrix
if (inverseMatrix(matrix, inverse)) {
    cout << "Inverse matrix:" << endl;
    printInverseMatrix(inverse);
}
```

- 釋放動態分配的記憶體。

```
    // 釋放記憶體
    for (int i = 0; i < MATRIX_SIZE; ++i) {
        delete[] matrix[i];
        delete[] transpose[i];
        delete[] rotate[i];
        delete[] inverse[i];
    }
    delete[] matrix;
    delete[] transpose;
    delete[] rotate;
    delete[] inverse;

    return 0;
}
```

2. Result Display

```
PS C:\Users\503503\Desktop\AdvC> ./hw1.exe
matrix:
55      35      11      68
32      9       70      36
80      70      90      46
29      26      79      51
------------------------------
Minimum value: 9 at index (1, 1)
Maximum value: 90 at index (2, 2)
------------------------------
traspose:
55      32      80      29
35      9       70      26
11      70      90      79
68      36      46      51
------------------------------
Rot90:
29      80      32      55
26      70      9       35
79      90      70      11
51      46      36      68
------------------------------
NO. 1 row:
32 9 70 36
------------------------------
NO. 2 column:
11 70 90 79
------------------------------
diagonal:
55 9 90 51
------------------------------
inverse diagonal:
68 70 70 29
------------------------------
determinant: -5693426
------------------------------
Inverse matrix:
-190.438 133.709 -0.442383 92.0204
203.716 -173.079 35.883 289.733
-63.2096 -310.603 179.229 103.21
-151.421 93.246 -374.822 -120.824
------------------------------
```