程式設計報告書

1072041 陳泳志

說明：

```
class dimension_exception : public exception {
public:
	const char* what() {
		return "dimension_exception!";
	}
};
class singular_exception : public exception {
public:
	const char* what() {
		return "singular_exception!";
	}
};
```

兩個繼承 exception 的類別，用來自定義例外

```
class Matrix {
public:
	Matrix* answer;
	float** matrix;
	int h, w;
	Matrix() {
		matrix = 0;
		answer = 0;
		h = w = 0;
	}//初始化
```

Matrix 類別，成員變數有：

answer：Matrix 指標，用來儲存反矩陣的資料

matrix：float 二維陣列，用來儲存輸入的矩陣資料

h,w：存這個陣列的高和寬

建構子會把這些東西都初始化

有參數的建構子看下一頁

```
Matrix(int h, int w) {
        if (h != w) throw dimension_exception();
        this->h = h;
        this->w = w;
        matrix = new float* [h];
        for (int i = 0; i < h; i++) {
            matrix[i] = new float[w];
        }//動態宣告
        answer = new Matrix();
        answer->h = h;
        answer->w = w * 2;
        answer->matrix = new float* [h];
        for (int i = 0; i < h; i++) {
            answer->matrix[i] = new float[w * 2];
        }//宣告儲存反矩陣的Matrix物件
    }
```

因為目標是算反矩陣，而反矩陣只有方陣才有，所以一開始就先比對高和寬，不一樣就丟出例外，如果一樣的話就存起來然後動態宣告。

這邊我的作法是把儲存答案的那個矩陣宣告兩倍寬，因為運算時旁邊會有一個單位矩陣，如果用兩個矩陣的話程式碼會很亂，乾脆直接放在一起比較好。

再來是運算反矩陣的函式，這邊所有運算都是在 answer 的矩陣做的，並沒有改到輸入的值。

```
Matrix& inverse() {
        int i = 0;
        int j = 0;
        while (i < h && j < w) {
            int maxi = i;
            for (int k = i + 1; k < h; k++) {
                if (abs(answer->matrix[k][j]) > abs(answer->matrix[maxi][j])) maxi
= k;
            }
```

從第 i 橫排開始往下找出第 j 直行的最大值。

```
                if (answer->matrix[maxi][j] != 0) {
                swap(answer->matrix[maxi], answer->matrix[i]);
```

找到有最大值的那個橫排後，如果這個值不是 0 就進行下一步，把最大值的那個橫排和第 i 橫排交換。

```
                float temp = answer->matrix[i][j];
                for (int k = 0; k < w * 2; k++) {
                    answer->matrix[i][k] /= temp;
                    check_zero(answer->matrix[i][k]);
                }
```

宣告一個變數儲存矩陣[i][j]的值，將第 i 橫排的所有值除以這個數，check_zero 函式用來處理浮點數運算後出現的誤差。

```
                for (int k = i + 1; k < h; k++) {
                    float temp = answer->matrix[k][j];
                    for (int l = j; l < w * 2; l++) {
                        answer->matrix[k][l] -= answer->matrix[i][l] * temp;
                        check_zero(answer->matrix[k][l]);
                    }
                }
```

上一步驟過後矩陣[i][j]的值會是 1，經過運算即可把其他排的第 j 行都變為 0，一樣 check_zero 用來處理誤差。

```
                i++;
            }
            j++;
        }
```

第 i 排處理完就換下一排，如果前面發現最大值是 0 的話會變成換下一直行，到這邊 while 結束，矩陣會變成一個列階梯形矩陣。

```
        for (int k = h - 1; k > 0; k--) {
            if (answer->matrix[k][k] == 0) throw singular_exception();
            for (int i = 0; i < k; i++) {
                float temp = answer->matrix[i][k];
                for (int j = 0; j < w * 2; j++) {
                    answer->matrix[i][j] -= answer->matrix[k][j] * temp;
                    check_zero(answer->matrix[i][j]);
                }
            }
        }
    return *answer;
}
```

經過之前的運算後，矩陣[k][k]的位置，也就是從右下到左上這條對角線，裡面的值都應該要是 1，如果是 0 的話代表沒有反矩陣，直接丟出例外。

如果確定有反矩陣，就用代入消去法求解，運算完即可得反矩陣，把值傳出結束。

```
    void swap(float*& m1, float*& m2) {
        float* temp = m1;
        m1 = m2;
        m2 = temp;
    }
```

用來交換橫排的函式

```
    void check_zero(float& f) {
        if (f<1e-5 && f>-(1e-5)) f = 0;
    }
```

用來處理浮點數運算後的誤差的函式

```
    friend istream& operator>>(istream& is, Matrix& m) {
        for (int i = 0; i < m.h; i++) {
            for (int j = 0; j < m.w; j++) {
                is >> m.matrix[i][j];
            }
        }
        for (int i = 0; i < m.h; i++) {
            for (int j = 0; j < m.w; j++) {
                m.answer->matrix[i][j] = m.matrix[i][j];
            }
            for (int j = m.w; j < m.w * 2; j++) {
                m.answer->matrix[i][j] = (j - i == m.h ? 1.0f : 0.0f);
            }
        }
        return is;
    }
```

重載>>運算子，可以直接 cin>>Matrix 物件，同時也會將輸入的值複製到 answer 的矩陣內，這個矩陣是兩倍寬，左半邊存輸入的，右半邊存單位矩陣。

```
    friend ostream& operator<<(ostream& os, Matrix& m) {
        for (int i = 0; i < m.h; i++) {
            for (int j = 0; j < m.w; j++) {
                os << setw(7) << setprecision(3) << m.matrix[i][j] << " ";
            }
            os << endl;
        }
        return os;
    }
```

重載<<運算子，可以直接 cout<<Matrix 物件，順便排版。

```
    ~Matrix() {
        delete answer;

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

解構子釋放記憶體空間

```
int main() {
	int M, N;
	cin >> M >> N;
	try {
		Matrix m(M, N);
		cin >> m;
		cout << m << endl;
		cout << m.inverse() << endl;
		cout << m << endl;
	}
	catch (dimension_exception& e) {
		cout << e.what() << endl;
	}
	catch (singular_exception& e) {
		cout << e.what() << endl;
	}
	catch (bad_alloc& e) {
		cout << e.what() << endl;
	}
}
```

輸入 M、N，宣告 m，輸入矩陣，輸出矩陣，輸出反矩陣，輸出矩陣
會去 catch 三種例外：不是方陣、沒有反矩陣、記憶體不夠用