张涵之 191220154

概念题

1. 构造函数：初始化对象存储空间中的数据成员。在创建对象时，使用对象前被调用。
析构函数：完成对象被删除前的清理工作。在对象消亡时，系统收回内存空间前被调用。

2. 成员对象初始化和消亡处理的次序由它们在类定义中的描述次序决定。
初始化时先执行成员对象类的构造函数，再执行本对象类的构造函数。多个成员对象按它们在本对象类中的说明次序进行。先调用本身类的构造函数，进入函数体之前，先调用成员对象类的构造函数，再执行本身类构造函数的函数体。

消亡处理时先执行本身类的析构函数，再执行成员对象类的析构函数。多个成员对象按它们在本对象类中的说明次序的逆序进行。先调用本身类的析构函数，本身类析构函数的函数体执行完之后，再调用成员对象类的析构函数。

3. 拷贝构造函数：参数类型为本类引用的构造函数。
创建一个对象时用另一个同类型的对象对其初始化，会调用对象类中的拷贝构造函数。

编程题

1. 定义一个固定高度箱体类 BoxWithFixedHeight：

```
class BoxWithFixedHeight {
    const int height = 10;
    const int area = 200;
    int length;
    int width;
public:
    BoxWithFixedHeight();
    void change_length(int l);
    int volumn();
};

BoxWithFixedHeight::BoxWithFixedHeight() {
    length = 5;
    width = 5;
}

void BoxWithFixedHeight::change_length(int l) {
    length = l;
    width = 10 - l;
}
```

```
int BoxWithFixedHeight::volumn() {
    return length * width * height;
}
```

2. 定义一个存储整型变量的矩阵类 Matrix:

```
class Matrix {
    int rows;
    int colums;
    int* m;
public:
    Matrix(int row, int col);
    ~Matrix();
    void input(int* a, int length);
    void print();
    void transpose();
};

Matrix::Matrix(int row, int col) {
    rows = row;
    colums = col;
    m = new int[row * col];
}

Matrix::~Matrix() {
    delete[]m;
}

void Matrix::input(int* a, int length) {
    int i;
    if (rows * colums > length) {
        for (i = 0; i < length; i++)
            m[i] = a[i];
        for (i = length; i < rows* colums; i++)
            m[i] = 0;
    }
    else {
        for (i = 0; i < rows * colums; i++)
            m[i] = a[i];
    }
}

void Matrix::print() {
    int i, j;
```

```
    for (i = 0; i < rows; i++) {
        cout << '[';
        for (j = 0; j < colums - 1; j++)
            cout << m[i * colums + j] << ", ";
        cout << m[(i + 1) * colums - 1] << ']' << endl;
    }
}

void Matrix::transpose() {
    int i, j, t, count = 0;
    int* temp = new int[rows * colums];
    for (j = 0; j < colums; j++) {
        for (i = 0; i < rows; i++) {
            temp[count] = m[i * colums + j];
            count++;
        }
    }
    t = rows;
    rows = colums;
    colums = t;
    for (i = 0; i < rows * colums; i++)
        m[i] = temp[i];
    delete[]temp;
}

int main()
{
    int r, l;
    cin >> r >> l;
    Matrix m(r, l);
    int* a, length;
    cin >> length;
    a = new int[length];
    for (int i = 0; i < length; i++)
        cin >> a[i];
    m.input(a, length);
    m.print();
    m.transpose();
    m.print();
    return 0;
}
```