Assignment 5

一. 概念题

1.1 C++所提供的隐式赋值操作存在什么样的问题？如何解决这样的问题?

解： 如果某些对象指向了一个动态区域，提供的隐式赋值操作会让两个对象指向同一块内存区域，会产生与隐式拷贝构造函数类似的问题，还可能会导致内存泄露，因为被赋值的那个对象之前指向的动态内存区域找不到了；解决办法是自己定义赋值操作符重载函数。

1.2 请简述拷贝构造函数与赋值操作符"="重载函数的区别.

解：创建一个新对象时用另一个已存在的同类对象对其初始化，调用拷贝构造函数；对两个已存在的对象，用其中一个对象改变另一个对象的状态时，调用赋值操作符重载函数。

1.3 为什么会对操作符new和delete进行重载?

解：系统提供的new和delete操作所涉及的空间分配和释放是通过系统的堆区管理系统来进行的，效率常常不高，对操作符new和delete进行重载，使得程序能以自己的方式来实现动态对象空间的分配和释放功能。

1.4 C++是如何实现λ表达式的?

解：在C++中，λ表达式是通过函数对象来实现的。

二. 编程题

2.1 完成int型矩阵类Matrix的实现，要求补充 '?' 处内容并完成如下的接口.

```
class Matrix
{
    int **p_data; //表示矩阵数据
    int row, col; //表示矩阵的行数和列数
public:
    Matrix(int r, int c);                       //构造函数
    ~Matrix();                                  //析构函数
    int *&operator[](int i);                    //重载[], 对于Matrix对象m, 能够通过m[i][j]访问第i+1行、第j+1列元素
    Matrix &operator=(const Matrix &m);         //重载=, 实现矩阵整体赋值, 若行/列不等, 归还空间并重新分配
    bool operator==(const Matrix &m) const;     //重载==, 判断矩阵是否相等
    Matrix operator+(const Matrix &m) const;    //重载+, 完成矩阵加法, 可假设两矩阵满足加法条件(两矩阵行、列分别相等)
    Matrix operator*(const Matrix &m) const;    //重载*, 完成矩阵乘法, 可假设两矩阵满足乘法条件(this.col = m.row)
};
```

```
Matrix::Matrix(int r, int c)
{
    p_data = new int *[r];
    for (int i = 0; i < r; i++)
    {
        p_data[i] = new int[c];
    }
    row = r;
    col = c;
}

Matrix::~Matrix()
{
    for (int i = 0; i < row; i++)
    {
        delete[] p_data[i];
    }
    delete[] p_data;
}

int *&Matrix::operator[](int i)
{
    return p_data[i];
}

Matrix &Matrix::operator=(const Matrix &m)
{
    if (row == m.row && col == m.col)
    {
        for (int i = 0; i < row; i++)
        {
            for (int j = 0; j < col; j++)
            {
                p_data[i][j] = m.p_data[i][j];
            }
        }
        return *this;
    }
    for (int i = 0; i < row; i++)
    {
        delete[] p_data[i];
    }
    delete[] p_data;
    p_data = new int *[m.row];
    for (int i = 0; i < m.row; i++)
    {
        p_data[i] = new int[m.col];
    }
    row = m.row;
    col = m.col;
    for (int i = 0; i < row; i++)
    {
        for (int j = 0; j < col; j++)
        {
            p_data[i][j] = m.p_data[i][j];
        }
    }
    return *this;
```

```
}

bool Matrix::operator==(const Matrix &m) const
{
    if (row != m.row || col != m.col)
    {
        return false;
    }
    for (int i = 0; i < row; i++)
    {
        for (int j = 0; j < col; j++)
        {
            if (p_data[i][j] != m.p_data[i][j])
            {
                return false;
            }
        }
    }
    return true;
}

Matrix Matrix::operator+(const Matrix &m) const
{
    Matrix result(row, col);
    for (int i = 0; i < row; i++)
    {
        for (int j = 0; j < col; j++)
        {
            result[i][j] = p_data[i][j] + m.p_data[i][j];
        }
    }
    return result;
}

Matrix Matrix::operator*(const Matrix &m) const
{
    Matrix result(row, m.col);
    for (int i = 0; i < row; i++)
    {
        for (int j = 0; j < m.col; j++)
        {
            result[i][j] = 0;
            for (int k = 0; k < col; k++)
            {
                result[i][j] += (p_data[i][k] * m.p_data[k][j]);
            }
        }
    }
    return result;
}
```

2.2 设计一种能解决教材例6-10中把存储块归还到堆空间的方法. (提示：可以在每次申请存储块时多申请一个能存储一个指针的空间，用该指针把每个存储块链接起来.)

```
#include <cstring>
#include <cstdlib>

class A
{
    //......
  public:
    static void *operator new(size_t size);
    static void operator delete(void *p);
  private:
    A *next;
    static A *p_free;
    static int counter;
    static A *blocks;
};

A *A::p_free = NULL;
int A::counter = 0;
A *A::blocks = NULL;
const int NUM = 1;

void *A::operator new(size_t size)
{
    if (p_free == NULL)
    {
        p_free = (A *)malloc(size*(NUM+1));
        for (int i = 0; i < NUM-1; i++)
            p_free[i].next = &p_free[i+1];
        p_free[NUM-1].next = NULL;
        p_free[NUM].next = blocks;
        blocks = p_free;
    }
    A *p = p_free;
    p_free = p_free->next;
    memset(p,0,size);
    counter++;
    return p;
}

void A::operator delete(void *p)
{
    ((A *)p)->next = p_free;
    p_free = (A *)p;
    counter--;
    if (counter == 0) {
        while (blocks != NULL) {
            A *next_block = blocks[NUM].next;
            free(blocks);
            blocks = next_block;
        }
    }
}
```