Assignment 5

191220022 丁一凡

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?

当一个类有指针类型的数据成员时，隐式赋值操作会使得该类的两个对象的数据成员中的指针指向同一块内存区域，使得两个对象相互干扰、一块空间被归还两次以及被复制对象含有的指针原来指向的内存空间被丢掉导致内存泄露问题。

可以自己定义赋值操作符重载函数解决这一问题。

1.2 请简述拷贝构造函数与赋值操作符"="重载函数的区别

①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，则调用拷贝构造函数；

②对两个已存在的对象，用其中一个对象来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

原操作符new和delete将调用系统的堆内存管理来进行动态内存的分配与归还。对一些类而言，系统的堆内存管理的效率不高，因为系统的堆内存管理要考虑各种大小的堆内存的分配与归还。

特别地，系统的堆内存管理会面临“碎片”问题，即经过多次分配与归还后，在系统的堆内存中可能会出现很多不连续的很小的自由空间，对于一个新的分配空间请求，有时会面临这些自由空间中每一个的大小都不能满足要求，但它们的总和能够满足的问题。

这时可以重载new和delete来实现堆内存的管理

1.4 C++是如何实现λ表达式的

C++的λ表达式实际上通过创建一个函数对象来实现的，系统将隐式地定义一个类，并为该类重载函数调用操作符，然后隐式地使用这个类创建一个函数对象。

二、 编程题

2.1 完成int类型矩阵类Matrix的实现

```
class Matrix
{
```

```
    vector<vector<int>> p_data;
    int row, col;
public:
    Matrix(int r, int c)
    {
        row = r;
        col = c;
        for(int i = 0; i < row; i++)
        {
            vector<int>temp(col,0);
            p_data.push_back(temp);
        }
    }
    ~Matrix()
    {
        p_data.clear();
    }
    vector<int> &operator[] (int i)
    {
        return pdata[i];
    }
    Matrix &operator = (const Matrix& m)
    {
        if(&m == this)
            return *this;
        row = m.row, col = m.col;
        p_data.clear();
        for(int i = 0; i < row; i++)
        {
            vector<int>temp(col,0);
            p_data.push_back(temp);
        }
        for(int i = 0; i < row; i++)
        {
            for(int j = 0; j < col; j++)
            {
                p_data[i][j] = m.p_data[i][j];
            }
        }
    }
    bool 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 operator + (const Matrix& m)
    {
        Matrix temp(row,col);
        for(int i = 0; i < row; i++)
```

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

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

```
class A
{
    ......
public:
    ......
private:
    ......
    static A *p_uesd;
    A *next_uesd;
}

void  *A::operator new(size_t size)
{
    A *p;
    if(p_free == NULL)
    {
        p_free = (A *)malloc(size*NUM);
        for(p=p_free;p!=p_free+NUM-1;p++)
            p->next = p+1;
        p->next = NULL;
    }
    p = p_free;
    p->next_used = p_used;
    p_used = p;
    p_free = p_free->next;
    memset(p,0,size);
    return p;
}
```

```
void A::delete_all()
{
    for(A *p = p_used; p!=NULL;p++)
    {
        free(p);
    }
}
```