2.4组合类

组合类（理论的自拓展）

- 组合类（composition class）是指类中的数据成员是另一个类的对象，即一个类内嵌其它类的对象作为自己的成员。
- 利用组合类可以在已有类的基础上定义一个复杂的类，即在已有抽象的基础上实现更复杂的抽象。

例子：组合类

```
class CEye   { /* Some Code */};
class CNose  { /* Some Code */};
class CEar   { /* Some Code */};
class CMouth { /* Some Code */};


class CHead {
private:
    CEye m_lEye, m_rEye;
    CNose m_nose;
    CEar m_ear;
    CMouth m_mouth;
public:
    void look();   // this->m_lEye.look();
                   // this->m_rEye.look();
    void smell();  // this->m_nose.smell();
    void listen(); // this->m_ear.listen();
    void eat();    // this->m_mouth.eat();
};
```

组合类的构造与析构

- 构造顺序：
  - 由内而外：先构造内嵌类，再构造组合类
  - 内嵌类构造顺序：按内嵌对象在组合**类中定义的顺序**（与组合类构造函数的初始化列表无关），调用内嵌对象的构造函数。
- 构造方法：

```
类名::类名(对象形参，本类成员形参):对象1(参数)对象2(参数)
{文本类成员初始化}
```

  - 若未定义构造函数，系统**自动**为组合类添加。
  - 若内嵌对象未出现在组合类构造函数的初始化列表中，系统自动调用内嵌对象的默认构造函数。

- 析构函数的调用顺序构造函数相反。

组合类的拷贝构造函数

- 若组合类未定义拷贝构造函数编译系统会自动为组合类添加一个默认的拷贝构造函数。
- 默认的拷贝构造函数的功能是：自动调用内嵌对象的拷贝构造函数，对各个内嵌对象成员进行初始化。
- 假定组合C类包含A类对象成员a，组合类C拷贝构造函数声明的形式如下：

```
C::C(C&c):a(c.a)
{...}
```

- 例如：

```
Line::Line(Line&line):start(line.start),end(line.end)
{...}
```

例子：组合类的构造

```
#include <bits/stdc++.h>
using namespace std;
// CPoint.hpp
//  #include <iomanip>
class CPoint
{
private:
    double x, y;

public:
    CPoint(double x = 0, double y = 0);
    CPoint(const CPoint &point);
    ~CPoint();

    friend ostream &operator<<(ostream &os, const CPoint &point)
    {
        os << "(" << point.x << "," << point.y << ")";
        return os;
    }
};

// CLine.hpp
// #include <iomanip>
class CLine
{
private:
    CPoint start, end;

public:
    CLine(const CPoint& start,const CPoint& end);
    CLine(const CLine& line);
    ~CLine();

    friend ostream &operator<<(ostream &os, const CLine &line)
    {
        os << line.start << "-->" << line.end;
```

```
        return os;
    }
};

// #include "CPoint.hpp"
CPoint::CPoint(double x, double y)
{
    this->x = x;
    this->y = y;
    cout << *this << ", constructer..." << endl;
}

CPoint::CPoint(const CPoint &point)
{
    this->x = point.x;
    this->y = point.y;
    cout << *this << ", copy constructer..." << endl;
}

CPoint::~CPoint()
{
    cout << *this << ", deconstructer..." << endl;
}

CLine::CLine(const CPoint &start, const CPoint &end) : start(start), end(end)
{
    cout << *this << ",Line constructer..." << endl;
}

CLine::CLine(const CLine &line) : start(line.start), end(line.end)
{
    cout << *this << ",Line copy constructer..." << endl;
}

CLine::~CLine()
{
    cout << *this << ",Line deconstructer..." << endl;
}

int main()
{
    CPoint p1, p2(2, 2);
    CLine line1(p1, p2), line2 = line1;

    cout << "p1: " << p1 << endl;
    cout << "p2: " << p2 << endl;
    cout << "line1: " << line1 << endl;
    cout << "line2: " << line2 << endl;
    return 0;
}
```