一、 设计实现-圆类(Class MyCircle)

设计一个二维平面上的圆类(Class MyCircle)，其中包含圆的属性：圆心的坐标(coordinate)，圆的半径(radius)；以及计算：1）圆周长；2）圆面积；3）两个圆的空间关系(是否相离、相切、相交和包含)；4）两个圆的相交面积的成员函数。注意：还需要设计一个有参数的构造函数和一个拷贝构造函数。编程要求如下：

a) 要求采用接口(MyCircle.h)和实现(MyCircle.cpp)分离的方案，定义并实现上面的 Class Circle。

b) 写一个主程序(main.cpp)来测试和验证其正确性。

```
// 参考代码
// 头文件 MyCircle.h
#ifndef MY_CIRCLE_HEADER_
#define MY_CIRCLE_HEADER_
#include<iostream>
#include<cmath>
using namespace std;

// 定义两个圆的相离、相切(外切)、相交(非包含)、包含 4 种空间关系
enum SpatialRelations{Separation, Tangency, Intersection, Inclusion};
const double PI = acos(-1.0);
const double AZERO = 1e-6;
struct MyPoint{
	double x;
	double y;
};

class MyCircle{
	MyPoint cc;   // 圆心
	double radius;   //   半径
	double cal_Distance(const MyCircle& c) const; // 计算距离
 public:
	MyCircle(double x, double y, double r);
	MyCircle(MyPoint p, double r);
	MyCircle(const MyCircle& c);     //   拷贝构造函数
	double cal_Circumference() const;   // 计算圆周
	double cal_Area() const;     // 计算面积
	// 计算空间关系
	SpatialRelations cal_Spatial_Relation(const MyCircle& c) const;
```

```
    double cal_Intersection_Area(const MyCircle& c) const; //  计算相交面积
    friend ostream& operator<<(ostream& out,MyCircle&);
};
#endif

//  实现源文件 MyCircle.cpp
#include "Mycircle.h"
double MyCircle::cal_Distance(const MyCircle& c) const {
    return sqrt((cc.x-c.cc.x)*(cc.x-c.cc.x) +
                (cc.y-c.cc.y)*(cc.y-c.cc.y));
}

MyCircle::MyCircle(double x, double y, double r){
    cc.x = x; cc.y = y; radius = r;
}

MyCircle::MyCircle(MyPoint p, double r):cc(p),radius(r){
}

MyCircle::MyCircle(const MyCircle& c){
    cc.x = c.cc.x; cc.y = c.cc.y; radius = c.radius;
}

double MyCircle::cal_Circumference() const {
    return 2 * PI * radius;
}

double MyCircle::cal_Area() const{
    return PI * radius * radius;
}

SpatialRelations MyCircle::cal_Spatial_Relation(const MyCircle& c) const{
    double d = this->cal_Distance(c);
    if (d - radius - c.radius > AZERO)
        return Separation;
    if (fabs(d - radius - c.radius) <= AZERO)
        return Tangency;

    if (fabs(radius - c.radius) - d > AZERO ||
            fabs(fabs(radius - c.radius) - d)<= AZERO)
        return Inclusion;
    return Intersection;
}
```

```
double MyCircle::cal_Intersection_Area(const MyCircle& c) const{
	switch (cal_Spatial_Relation(c))
	{
	case Separation:
	case Tangency:	return 0;
	case Inclusion:
		if (radius - c.radius > AZERO)
			return c.cal_Area();
		else
			return cal_Area();
	case Intersection:
		double d = this->cal_Distance(c);
		double ang1=acos((radius*radius+d*d-c.radius*c.radius)
					/(2*radius*d));
		double ang2=acos((c.radius*c.radius+d*d-radius*radius)
					/(2*c.radius*d));
		return ang1*radius*radius + ang2*c.radius*c.radius
				- radius*d*sin(ang1);
	}
}

ostream& operator<<(ostream& out,MyCircle& c){
	out<<"Center of Circle("<<c.cc.x<<", "<<c.cc.y<<")"
	   << ", Radius(" << c.radius << ")";
	return out;
}

// 测试源程序 main.cpp
#include "MyCircle.h"
#include <string>
int main()
{
	string srStr[] = {"Separation", "Tangency", "Intersection", "Inclusion"};

	MyCircle c1(0,0,3);
	cout << "Circle c1 : " << c1 << endl;
	cout << "Area of c1 : " << c1.cal_Area() << endl;
	cout << "Circumference of c1 : " << c1.cal_Circumference()
		 << endl << endl;

	MyPoint p = {6,0};
	MyCircle c2(p,3);
	cout << "Circle c2 : " << c2 << endl;
	cout << "Spatial Relation of c2 and c1 : "
```

```
        << srStr[c2.cal_Spatial_Relation(c1)] << endl;
    cout << "Intersection Area of c2 and c1 : "
        << c2.cal_Intersection_Area(c1) << endl << endl;

    MyCircle c3(c1);
    cout << "Circle c3 : " << c3 << endl;
    cout << "Spatial Relation of c3 and c1 : "
        << srStr[c3.cal_Spatial_Relation(c1)] << endl;
    cout << "Intersection Area of c3 and c1 : "
        << c3.cal_Intersection_Area(c1) << endl<< endl;

    MyCircle c4(5,7,6);
    cout << "Circle c4 : " << c4 << endl;
    cout << "Area of c4 : " << c4.cal_Area() << endl;
    cout << "Circumference of c4 : " << c4.cal_Circumference() << endl;

    cout << "Spatial Relation of c4 and c1 : "
        << srStr[c4.cal_Spatial_Relation(c1)] << endl;
    cout << "Intersection Area of c4 and c1 : "
        << c4.cal_Intersection_Area(c1) << endl << endl;

    MyCircle c5(1, 0, 2);
    cout << "Circle c5 : " << c5 << endl;
    cout << "Area of c5 : " << c5.cal_Area() << endl;
    cout << "Circumference of c5 : " << c5.cal_Circumference() << endl;
    cout << "Spatial Relation of c5 and c1 : "
        << srStr[c5.cal_Spatial_Relation(c1)] << endl;
    cout << "Intersection Area of c5 and c1 : "
        << c5.cal_Intersection_Area(c1) << endl<< endl;

    return 0;
}
```

二、 设计实现时间类(Class MyTime)

在第六次作业的基础上，定义一个时间类 Class MyTime，通过运算符重载实现时间的比较（关系运算）、时间增加/减少若干秒（+=和-=）、时间增加/减少 1 秒（++和--）以及输出时间对象的值（时—分—秒）。编程要求：

a) 采用接口(MyTime.h)和实现(MyTime.cpp)分离的方案，定义并实现上面的 Class MyTime。

b) 写一个主程序(main.cpp)来测试和验证其正确性。

```
// 参考代码
// 头文件 MyTime.h
#ifndef MY_TIME_HEADER_
#define MY_TIME_HEADER_
#include<iostream>
using namespace std;

class MyTime{
	friend int operator-(const MyTime &t1, const MyTime &t2);
	friend ostream &operator<<(ostream &os, const MyTime &t);
	friend bool operator>(const MyTime &t1, const MyTime &t2);
	friend bool operator>=(const MyTime &t1, const MyTime &t2);
	friend bool operator==(const MyTime &t1, const MyTime &t2);
	friend bool operator!=(const MyTime &t1, const MyTime &t2);
	friend bool operator<(const MyTime &t1, const MyTime &t2);
	friend bool operator<=(const MyTime &t1, const MyTime &t2);

	int second;

public:
	MyTime (int tt=0, int mm = 0, int ss = 0) ;
	MyTime &operator++() ;
	MyTime operator++(int x);

	MyTime &operator--() ;
	MyTime operator--(int x) ;

	MyTime &operator+=(const MyTime &other) ;
	MyTime &operator-=(const MyTime &other);

};
#endif

// 实现源文件 MyTime.cpp
#include "MyTime.h"
ostream &operator<<(ostream &os, const MyTime &t)
{
	int tt, mm, ss;
	tt = t.second/3600;
	mm = t.second%3600/60;
	ss = t.second%60;
```

```
    os << tt << '-' << mm<<'-' << ss;
    return os;
}

    int operator-(const MyTime &t1, const MyTime &t2)
    {return t1.second-t2.second;}

    bool operator>(const MyTime &t1, const MyTime &t2)
    {return t1.second > t2.second;}

    bool operator>=(const MyTime &t1, const MyTime &t2)
    {return t1.second >= t2.second;}

    bool operator==(const MyTime &t1, const MyTime &t2)
    {return t1.second == t2.second;}

    bool operator!=(const MyTime &t1, const MyTime &t2)
    {return t1.second != t2.second;}

    bool operator<(const MyTime &t1, const MyTime &t2)
    {return t1.second < t2.second;}

    bool operator<=(const MyTime &t1, const MyTime &t2)
    {return t1.second <= t2.second;}

MyTime::MyTime (int tt, int mm, int ss)
    {second = ss + mm * 60 + tt*3600;}
    MyTime& MyTime::operator++() {++second; return *this;}
    MyTime MyTime::operator++(int x) {
        MyTime tmp = *this;
        ++second;
        return tmp;
    }

    MyTime& MyTime::operator--() {--second; return *this;}
    MyTime MyTime::operator--(int x) {
        MyTime tmp = * this; -- second;
        return tmp;
    }

MyTime& MyTime::operator+=(const MyTime &other)
    {second += other.second; return *this;}
MyTime& MyTime::operator-=(const MyTime &other)
    {second -= other.second; return *this;}
```

```
// 测试源程序 main.cpp
#include "MyTime.h"

int main()
{
	MyTime MyTime1(16,28,46);
	MyTime MyTime2(18,20,34);
	cout<<"We have MyTime1&MyTime2: " <<endl;
	cout<<MyTime1<<endl<<MyTime2<<endl;
	bool flag=MyTime1>MyTime2;
	cout<<"MyTime1>MyTime2? "<<flag<<endl;
	flag=MyTime1<MyTime2;
	cout<<"MyTime1<MyTime2? "<<flag<<endl;
	flag=(MyTime1==MyTime2);
	cout<<"MyTime1==MyTime2? "<<flag<<endl;
	flag=MyTime1!=MyTime2;
	cout<<"MyTime1!=MyTime2? "<<flag<<endl;
	cout<<"MyTime2-MyTime1="<<MyTime2-MyTime1<<"s"<<endl;
	cout<<MyTime1<<"+1s="<<++MyTime1<<endl;
	cout<<MyTime2<<"-1s="<<--MyTime2<<endl;

	return 0;
}
```

三、设计实现三维向量类（Class MyVector3D）

设计和实现一个三维向量类（Class MyVector3D），要求如下：

a) 计算向量的模长；

b) 计算两个向量的夹角、内积和外积；

c) 重载加减乘除运算符，实现向量与数值的加减乘除运算；

d) 重载加减运算符，实现向量之间的加减运算；

e) 实现拷贝构造函数，重载赋值运算符。

f) 重载插入运算符<<和提取运算符>>。

参考代码

```
// 头文件(MyVector3D.h)
#ifndef MY_VECTOR3D_HEADER_
#define MY_VECTOR3D_HEADER_
#include<iostream>
using namespace std;
extern const double uZero;

class MyVector3D
{
    double x, y, z;
  public:
    MyVector3D():x(0), y(0), z(0){}
    MyVector3D(double x1, double y1, double z1):x(x1), y(y1), z(z1){}
    MyVector3D(const MyVector3D &v);
    void operator=(const MyVector3D &v);
    MyVector3D operator+(const MyVector3D &v);
    MyVector3D operator-(const MyVector3D &v);
    MyVector3D operator/(const MyVector3D &v);
    MyVector3D operator*(const MyVector3D &v);
    MyVector3D operator+(double f);
    MyVector3D operator-(double f);
    MyVector3D operator/(double f);
    MyVector3D operator*(double f);

    friend ostream& operator<<(ostream& out,MyVector3D &);
    friend istream& operator>>(istream& in,MyVector3D &);

    double cal_inner(const MyVector3D& V) const;
    double cal_angle(const MyVector3D& V) const;
    double cal_outer(const MyVector3D& V);
    double dot(const MyVector3D &v) const;
    double length() const;
    void normalize();
    MyVector3D crossProduct(const MyVector3D &v);
    void printVec3();
};

#endif

//实现源文件（MyVector3D.cpp)
#include "MyVector3D.h"
#include<cmath>
```

```
const double uZero = 1e-6;

//Copy 构造函数，必须为常量引用参数，否则编译不通过
MyVector3D::MyVector3D(const MyVector3D &v):x(v.x), y(v.y), z(v.z)
{
}

ostream& operator << (ostream& out,MyVector3D& v) //定义重载运算符“<<”
{
	out<<"("<<v.x<<","<<v.y<<","<<v.z<<")";
	return out;
}
istream& operator >> (istream& in,MyVector3D& v)   //定义重载运算符“>>”
{
	double x,y,z;
	in>>x>>y>>z;
	MyVector3D tmp(x,y,z);
	v=tmp;
	return in;
}
void MyVector3D::operator=(const MyVector3D &v)
{
	x = v.x;	y = v.y;	z = v.z;
}

MyVector3D MyVector3D::operator+(const MyVector3D &v)
{
	return MyVector3D(x+v.x, y+v.y, z+v.z);
}

MyVector3D MyVector3D::operator-(const MyVector3D &v)
{
	return MyVector3D(x-v.x, y-v.y, z-v.z);
}

MyVector3D MyVector3D::operator/(const MyVector3D &v)
{
	if (fabsf(v.x) <= uZero || fabsf(v.y) <= uZero || fabsf(v.z) <= uZero)
	{
		std::cerr<<"Over flow!\n";
		return *this;
	}
	return MyVector3D(x/v.x, y/v.y, z/v.z);
}
```

```
MyVector3D MyVector3D::operator*(const MyVector3D &v)
{
	return MyVector3D(x*v.x, y*v.y, z*v.z);
}

MyVector3D MyVector3D::operator+(double f)
{
	return MyVector3D(x+f, y+f, z+f);
}

MyVector3D MyVector3D::operator-(double f)
{
	return MyVector3D(x-f, y-f, z-f);
}

MyVector3D MyVector3D::operator/(double f)
{
	if (fabsf(f) < uZero)
	{
		std::cerr<<"Over flow!\n";
		return *this;
	}
	return MyVector3D(x/f, y/f, z/f);
}

MyVector3D MyVector3D::operator*(double f)
{
	return MyVector3D(x*f, y*f, z*f);
}

double MyVector3D::dot(const MyVector3D &v) const
{
	return x*v.x + y*v.y + z*v.z;
}

double MyVector3D::length() const
{
	return sqrtf(dot(*this));
}

void MyVector3D::normalize()
{
	double len = length();
```

```
    if (len < uZero) len = 1;
    len = 1/len;

    x *= len;       y *= len;       z *= len;
}

/*
Cross Product 叉乘公式
aXb = | i,    j,    k    |
      | a.x a.y a.z|
      | b.x b.y b.z| = (a.x*b.z -a.z*b.y)i + (a.z*b.x - a.x*b.z)j
                                + (a.x+b.y - a.y*b.x)k
*/
MyVector3D MyVector3D::crossProduct(const MyVector3D &v)
{
    return MyVector3D(y * v.z - z * v.y,
                                z * v.x - x * v.z,
                                x * v.y - y * v.x);
}

void MyVector3D::printVec3()
{
    std::cout<<"("<<x<<", "<<y<<", "<<z<<")"<<std::endl;
}

double MyVector3D::cal_inner(const MyVector3D& V) const
{
    return this->x*V.x + this->y*V.y + this->z*V.z;
}

double MyVector3D::cal_angle(const MyVector3D& V)const
{
    double cos = this->cal_inner(V) / (this->length()* V.length());
    return acosf(cos);
}

//const 对象不可以调用非 const 成员函数，因此函数中调用的成员函数
//应设置为 const
double MyVector3D::cal_outer( const MyVector3D& V)
{
    double p= this->length() * V.length() * sinf(this->cal_angle(V));
    return p;
}
```

```
// 测试主程序（main.cpp)
#include"MyVector3D.h"
using namespace std;

int main()
{
	MyVector3D v31;
	MyVector3D v32(2.0f,3.0f,4.0f);
	MyVector3D v33(v32 - 1.0f);

	cout<<"We have original MyVector3Ds:" <<endl ;
	v31.printVec3();
	v32.printVec3();
	v33.printVec3();

	cout<<"Their length:" <<endl;
	cout<<v31.length()<<endl;
	cout<<v32.length()<<endl;
	cout<<v33.length()<<endl;

	cout<<"v32 &v33 innerProduct:"<<endl;
	cout<<v32.cal_inner(v33)<<endl;
	cout<<"v32 &v33 outterProduct:"<<endl;
	cout<<v32.cal_outer(v33)<<endl;
	cout<<"v32 &v33 angle:"<<endl;
	cout<<v32.cal_angle(v33)<<endl;
	cout<<"v32 crossproduct v33 is:";
	MyVector3D v3233 = v32.crossProduct(v33);
	v3233.printVec3();
	cout<<"v32+v33=";
	MyVector3D vand=v32+v33;
	vand.printVec3();
	cout<<"v32-v33=";
	MyVector3D vminus=v32-v33;
	vminus.printVec3();
	cout<<"v32*v33=";
	MyVector3D vmulti=v32*v33;
	vmulti.printVec3();
	cout<<"v32/v33=";
	MyVector3D vdiv=v32/v33;
	vdiv.printVec3();

	cout<<"Now we normalize them:" <<endl;
	v31.normalize();
```

```
    v32.normalize();
    v33.normalize();
    v3233.normalize();
    v31.printVec3();
    v32.printVec3();
    v33.printVec3();
    v3233.printVec3();

    MyVector3D v;//重载输入输出
    cout<<"Now input new vector3D:";
    cin>>v;
    cout<<v;

    return 0;
}
```