1]Relational Operator == Overloading:

```
#include<iostream>
class Demo
{
    public:
        int a,b,c;

    Demo()
    {



    }

    Demo(int a,int b,int c)
    {
        this->a=a;
        this->b=b;
        this->c=c;
    }

    int operator==(const Demo& ref)
    {
        if(this->a==ref.a && this->b==ref.b && this->c==ref.c)
        {
            return 1;
        }
        else
        {
            return 0;
        }
    }
};
```

```
int main()
{
    Demo d1(10,20,30);
    //Demo d1(10,20,56);
    Demo d2(10,20,30);

    if(d1==d2)
    {
        std::cout<<"d1 and d2 are Equal."<<std::endl;
    }
    else
    {
        std::cout<<"d1 and d2 are not equal."<<std::endl;
    }
}
```

2]Relational Operator < Overloading:

```
#include<iostream>
class Demo
{
    public:
        int a,b,c;

    Demo()
    {

    }

    Demo(int a,int b,int c)
    {
        this->a=a;
        this->b=b;
        this->c=c;
```

```
    }

    int operator<(const Demo& ref)
    {
        if(this->a < ref.a && this->b < ref.b && this->c < ref.c)
        {
            return 1;
        }
        else
        {
            return 0;
        }
    }
};


int main()
{
    Demo d1(1,2,3);
    Demo d2(4,5,6);

    Demo d3(1,2,7);
    Demo d4(3,4,5);

    std::cout<<"(d1<d2)? = "<<(d1<d2)<<std::endl;
    std::cout<<"(d3<d4)? = "<<(d3<d4);
}
```

3]Relational Operator > Overloading:

```
#include<iostream>
class Demo
{
    public:
        int a,b,c;
    Demo()
    {

    }

    Demo(int a,int b,int c)
    {
        this->a=a;
        this->b=b;
        this->c=c;
    }

    int operator>(const Demo& ref)
    {
        if(this->a > ref.a && this->b > ref.b && this->c > ref.c)
        {
            return 1;
        }
        else
        {
            return 0;
        }
    }
};


int main()
```

```
{
    Demo d1(1,2,3);
    Demo d2(4,5,6);

    Demo d3(1,2,7);
    Demo d4(3,4,5);

    std::cout<<"(d2>d1)? = "<<(d2>d1)<<std::endl;
    std::cout<<"(d2>d1)? = "<<(d4>d3);
}
```