1.

```
//operator overloading

#include<stdio.h>

struct Complex {
	int real,img;

	Complex() {
		this->real=0;
		this->img=0;
	}

	Complex(int real,int img) {
		this->real=real;
		this->img=img;
	}
	//Addition
	Complex operator+(Complex c) {
		printf("Addition of Complex Complex\n");
		Complex temp;
		temp.real=c.real+this->real;
		temp.img=c.img+this->img;
		return temp;
	}
	Complex operator+(int a) {
		printf("Addition of Complex int\n");
		Complex temp;
		temp.real=this->real+a;
		temp.img=this->img+a;
```

```
	return temp;
}

//Subtraction

Complex operator-(Complex c) {
	printf("Subtraction of Complex Complex\n");
	Complex temp;
	temp.real=c.real-this->real;
	temp.img=c.img-this->img;
	return temp;
}
Complex operator-(int a) {
	printf("Subtraction of Complex int\n");
	Complex temp;
	temp.real=this->real-a;
	temp.img=this->img-a;
	return temp;
}
//unary operator
void operator++() {
	printf("Post increment\n");
	Complex c;
	c.real=this->real++;
	c.img=this->img++;

}
void operator++(int a) {
	printf("Pre increment\n");
	Complex c;
	c.real=++this->real;
```

```
        c.img=++this->img;

}
Complex operator*(int a) {
        printf("multiplication operator\n");
        Complex c;
        c.real=a*this->real;
        c.img=a*this->img;
        return c;
}

//display
void displayAdd() {
        printf("%d+%di\n\n",this->real,this->img);
}

void displaySub() {
        printf("%d-(%d)i\n\n",this->real,this->img);
}
void display() {
        printf("Real=%d, Img=%d\n",this->real,this->img);
}

//logical
Complex operator&&(Complex c) {
        Complex temp;
        temp.real=this->real&&c.real;
        temp.img=this->img&&c.img;
        return temp;
}
```

```
Complex operator||(Complex c) {
	Complex temp;
	temp.real=this->real||c.real;
	temp.img=this->img||c.img;
	return temp;
}
Complex operator&&(int a) {
	Complex temp;
	temp.real=this->real&&a;
	temp.img=this->img&&a;
	return temp;
}
Complex operator||(int a) {
	Complex temp;
	temp.real=this->real||a;
	temp.img=this->img||a;
	return temp;
}

//relational
Complex operator>(Complex c) {
	Complex temp;
	if(this->real>c.real)
		temp.real=this->real;
	else
		temp.real=c.real;
	if(this->img>c.img)
		temp.img=this->img;
	else
		temp.img=c.img;
```

```
	return temp;
}
Complex operator<(Complex c) {
	Complex temp;
	if(this->real<c.real)
		temp.real=this->real;
	else
		temp.real=c.real;
	if(this->img<c.img)
		temp.img=this->img;
	else
		temp.img=c.img;

	return temp;
}
Complex operator<=(Complex c) {
	Complex temp;
	if(this->real<=c.real)
		temp.real=this->real;
	else
		temp.real=c.real;
	if(this->img<=c.img)
		temp.img=this->img;
	else
		temp.img=c.img;

	return temp;
}
Complex operator>=(Complex c) {
	Complex temp;
	if(this->real>=c.real)
```

```
                temp.real=this->real;
        else
                temp.real=c.real;
        if(this->img>=c.img)
                temp.img=this->img;
        else
                temp.img=c.img;

        return temp;
    }
    Complex operator!=(Complex c) {
        Complex temp;
        if(this->real!=c.real)
                temp.real=1;
        else
                temp.real=0;
        if(this->img!=c.img)
                temp.img=1;
        else
                temp.img=0;

        return temp;
    }

};
Complex operator+(int a,Complex c) {
    printf("Addition of Int Complex\n");
    Complex temp;
    temp.real=a+c.real;
    temp.img=a+c.img;
    return temp;
```

```
}
Complex operator-(int a,Complex c) {
	printf("Subtraction of Int Complex\n");
	Complex temp;
	temp.real=a-c.real;
	temp.img=a-c.img;
	return temp;
}
Complex operator&&(int a,Complex c) {
	Complex temp;
	temp.real=c.real&&a;
	temp.img=c.img&&a;
	return temp;
}
Complex operator||(int a,Complex c) {
	Complex temp;
	temp.real=a||c.real;
	temp.img=a||c.img;
	return temp;
}

int main() {
	Complex c1(10,20),c2(30,40);
//Addition

	Complex c3=c1+c2;
	c3.displayAdd();

	Complex c4=10+c3;
	c4.displayAdd();
```

```
        Complex c5=c4+10;
        c5.displayAdd();

//Subtraction

        Complex c6=c5-10;
        c6.displaySub();

        Complex c7=10-c6;
        c7.displaySub();

        Complex c8=c6-c7;
        c8.displaySub();

// unary operator overloading
        c8++;
        c8.displayAdd();

        ++c8;
        c8.displayAdd();

//  multiplication operato
        Complex c9=c8*10;
        c9.display();

//  logical

        //logical and
        Complex c10=c1&&c2;
```

```
        c10.display();

        int a=0;
        Complex c15=a&&c1;
        c15.display();

        Complex c16=c1&&a;
        c16.display();

        //logical or
        Complex c11=c1||c2;
        c11.display();

        Complex o1=a||c2;
        o1.display();

        Complex o2=c1||a;
        o2.display();

//      relational

        Complex c12=c1>c2;
        c12.display();

        Complex i1=c1>=c2;
        i1.display();

        Complex i2=c1<c2;
        i2.display();

        Complex i3=c1<=c2;
```

```
	i3.display();

	Complex c14=c1!=c2;
	c14.display();

	return 0;
}
```

2.

```
//operator overloading

#include<stdio.h>

struct Distance {
	int inches,feet;

	Distance() {
		this->inches=0;
		this->feet=0;
	}
	Distance(int inches,int feet) {
		this->inches=inches;
		this->feet=feet;
	}
//  Addition
	Distance operator+(Distance c) {
		Distance temp;
		temp.inches=c.inches+this->inches;
		temp.feet=c.feet+this->feet;
		return temp;
	}
	Distance operator+(int a) {
```

```
        Distance temp;
        temp.inches=this->inches+a;
        temp.feet=this->feet+a;
        return temp;
}

//Subtraction

Distance operator-(Distance c) {
        Distance temp;
        temp.inches=c.inches-this->inches;
        temp.feet=c.feet-this->feet;
        return temp;
}
Distance operator-(int a) {
        Distance temp;
        temp.inches=this->inches-a;
        temp.feet=this->feet-a;
        return temp;
}

//unary operator
void operator++() {
        Distance c;
        c.inches=this->inches++;
        c.feet=this->feet++;

}
void operator++(int a) {
        Distance c;
        c.inches=++this->inches;
```

```
        c.feet=++this->feet;


}
Distance operator*(int a) {
        Distance c;
        c.inches=a*this->inches;
        c.feet=a*this->feet;
        return c;
}
//logical
Distance operator&&(Distance c) {
        Distance temp;
        temp.inches=this->inches&&c.inches;
        temp.feet=this->feet&&c.feet;
        return temp;
}


Distance operator||( Distance c) {
        Distance temp;
        temp.inches=this->inches||c.inches;
        temp.feet=this->feet||c.feet;
        return temp;
}
Distance operator&&(int a) {
        Distance temp;
        temp.inches=this->inches&&a;
        temp.feet=this->feet&&a;
        return temp;
}
Distance operator||(int a) {
        Distance temp;
```

```
        temp.inches=this->inches||a;
        temp.feet=this->feet||a;
        return temp;
}


//relational
Distance operator>( Distance c) {
        Distance temp;
        if(this->inches>c.inches)
                temp.inches=this->inches;
        else
                temp.inches=c.inches;
        if(this->feet>c.feet)
                temp.feet=this->feet;
        else
                temp.feet=c.feet;

        return temp;
}
Distance operator<( Distance c) {
        Distance temp;
        if(this->inches<c.inches)
                temp.inches=this->inches;
        else
                temp.inches=c.inches;
        if(this->feet<c.feet)
                temp.feet=this->feet;
        else
                temp.feet=c.feet;
```

```
        return temp;
}
Distance operator<=( Distance c) {
        Distance temp;
        if(this->inches<=c.inches)
                temp.inches=this->inches;
        else
                temp.inches=c.inches;
        if(this->feet<=c.feet)
                temp.feet=this->feet;
        else
                temp.feet=c.feet;

        return temp;
}
Distance operator>=( Distance c) {
        Distance temp;
        if(this->inches>=c.inches)
                temp.inches=this->inches;
        else
                temp.inches=c.inches;
        if(this->feet>=c.feet)
                temp.feet=this->feet;
        else
                temp.feet=c.feet;

        return temp;
}
Distance operator!=( Distance c) {
        Distance temp;
        if(this->inches!=c.inches)
```

```
                temp.inches=1;
        else
                temp.inches=0;
        if(this->feet!=c.feet)
                temp.feet=1;
        else
                temp.feet=0;

        return temp;
    }

    //display
    void displayAdd() {
        printf("%d+%di\n\n",this->inches,this->feet);
    }

    void displaySub() {
        printf("%d-(%d)i\n\n",this->inches,this->feet);
    }
    void display() {
        printf("inches=%d, feet=%d\n",this->inches,this->feet);
    }


};
Distance operator+(int a,Distance c) {
    Distance temp;
    temp.inches=a+c.inches;
    temp.feet=a+c.feet;
    return temp;
}
```

```
Distance operator-(int a,Distance c) {
	Distance temp;
	temp.inches=a-c.inches;
	temp.feet=a-c.feet;
	return temp;
}
Distance operator&&(int a,Distance c) {
	Distance temp;
	temp.inches=a&&c.inches;
	temp.feet=a&&c.feet;
	return temp;
}

Distance operator||(int a, Distance c) {
	Distance temp;
	temp.inches=a||c.inches;
	temp.feet=a||c.feet;
	return temp;
}

int main() {
	Distance c1(20,20),c2(10,10);
	c1.display();
	c2.display();
//Addition

	Distance c3=c1+c2;
	c3.display();
	c3.displayAdd();

	Distance c4=10+c3;
```

```
        c4.display();
        c4.displayAdd();

        Distance c5=c4+10;
        c5.display();
        c5.displayAdd();


//Subtraction

        Distance c6=c5-10;
        c6.display();
        c6.displaySub();

        Distance c7=10-c6;
        c7.display();
        c7.displaySub();

        Distance c8=c6-c7;
        c8.display();
        c8.displaySub();


//unary operator overloading
        c8++;
        c8.display();

        ++c8;
        c8.display();

//multiplication operato
```

```
        Distance c9=c8*10;
        c9.display();

//logical
        Distance c10=c1&&c1;
        c10.display();

        int a=0;
        Distance c15=a&&c1;
        c15.display();

        Distance c16=c1&&a;
        c16.display();

        //logical or
        Distance c11=c1||c2;
        c11.display();

        Distance o1=a||c2;
        o1.display();

        Distance o2=c1||a;
        o2.display();
//Relational

        Distance c12=c1>c2;
        c12.display();

        Distance i1=c1>=c2;
        i1.display();
```

```
        Distance i2=c1<c2;
        i2.display();

        Distance i3=c1<=c2;
        i3.display();

        Distance c14=c1!=c2;
        c14.display();



        return 0;
}
```