Q1. Define a class Complex with appropriate instance variables and member functions.
Overload following operators
a.<< insertion operator
b.>> extraction operator

```
#include <iostream>
using namespace std;

class Complex {

	private:
		int real, imag;

	public:

		friend istream &operator >>(istream &in, Complex &c1);
		friend ostream &operator <<(ostream &out, Complex &c1);

		void disp() {
			cout << real;
			cout << imag;
		}
};

istream &operator >>(istream &in, Complex &c1) {
	in >> c1.real;
	in >> c1.imag;

	return in;
}

ostream &operator<<(ostream &out, Complex &c1) {
	out << c1.real << " + " << c1.imag << "i";

	return out;
}


int main() {

	Complex c;

	cout << "Enter Complex Number : ";
	cin >> c;
	cout << c;

}
```

Q2. Define a class Complex with appropriate instance variables and member functions.
One of the functions should be setData() to set the properties of the object.
Make sure the names of formal arguments are the same as names of instance variables.

```
#include <iostream>
using namespace std;

class Complex {
	private :
		int real, imag;

	public:
		void setData (int real, int imag);
		void display(void);
};

void Complex::setData(int real, int imag) {

	this->real = real;
	this->imag = imag;

}

void Complex::display(void) {

	cout << real << " + " << imag << "i";
}

int main() {

	Complex c;

	c.setData(10, 7);
	c.display();

}
```

Q3. Overload subscript operator [] that will be useful when we want to check for an index out of bound.

```
#include <iostream>
using namespace std;

class Array {

        private:
                int *arr;
                int size;

        public:
                Array() {

                        cout << "Enter Size of array : ";
                        cin >> size;

                        arr = (int *)malloc(sizeof(int) * size);
                }

                int &operator[](int index) {

                        if (index >= size) {

                                cout << "Array out of bound" << endl;
                                exit(0);

                        }       else
                                return arr[index];
                }

};

int main() {

        Array z;

        z[0] = 58;
        z[1] = 10;
        z[2] = 20;

        cout << z[0] << z[1] << z[2];

}
```

Q4. Create a student class and overload new and delete operators as a member function of the class.

```
#include <iostream>
using namespace std;

class Student {

        private:
                int *p;

        public:

                void *operator new (size_t size) {

                        void *p = ::operator new (size);

                        return p;
                }

                void operator delete (void *p) {
                        free(p);
                }

                void disp(void) {
                        cout << "Operator overload successful";
                }
};



int main() {

        Student *p;

        p = (Student *)new Student();

        p->disp();

        delete (p);
}
```

Q5. Create a student class and overload new and delete operators outside the class.

```
#include <iostream>
using namespace std;

class Student {

        private:

                int a = 10;

        public:
                Student() {
                        cout << "constructor called" << endl;
                }

                void disp(void) {

                        cout << "New and Delete operator overloded";
                }

};

void *operator new (size_t size) {

        cout << "inside new" << endl;

        void *p = malloc (size);

        return p;
}

void operator delete (void *p) {

        free(p);
}

int main() {

        Student *p;

        p = new Student();

        p->disp();

        delete p;

}
```

Q6. Create a complex class and overload assignment operator for that class.

```
#include <iostream>
#include <string.h>

using namespace std;

class Complex {

	private:
		int real, imag;

	public:

		Complex() {
		}

		Complex(int real, int imag) {

			this->real = real;
			this->imag = imag;
		}

		Complex operator =(const Complex &x) {

			real = x.real;
			imag = x.imag;

			return x;
		}

		void disp() {
			cout << "real : " << real << endl <<  "imag : " << imag;
		}
};

int main() {

	Complex c1(100, 6);

	Complex c2;

	c2 = c1 ;

	c2.disp();



}
```

Q7. Create an Integer class and overload logical not operator for that class.

```
#include <iostream>
using namespace std;

class Integer {

	private:
		int x;

	public:

		Integer(int a): x(a) {
		}

		int operator !() {

			return -x;
		}

};

int main() {

	Integer i = -5;

	cout << !i;
}
```

Q8. Create a Coordinate class for 3 variables x,y and z and overload comma operator such that when you write c3 = (c1 , c2) then c2 is assigned to c3.
Where c1,c2,and c3 are objects of 3D coordinate class.

```
#include <iostream>
using namespace std;

class Coordinate {

	private:
		int x, y, z;

	public:

		Coordinate() {
		}

		Coordinate(int x, int y, int z) {

			this->x = x;
			this->y = y;
			this->z = z;
		}

		Coordinate operator, (Coordinate tmp) {

			x = tmp.x;
			y = tmp.y;
			z = tmp.z;

			return tmp;
		}

		void disp() {
			cout << "x : " << x << endl << "y : " << y << endl << "z : " << z;
		}
};

int main() {

	Coordinate c1(10, 20, 30);
	Coordinate c2(40, 50, 60);
	Coordinate c3;

	c3 = (c1, c2);

	c3.disp();
}
```

Q9.Create an Integer class that contains int x as an instance variable and overload casting int() operator that will type cast your Integer class object to int data type.

```
#include <iostream>
using namespace std;

class Integer {

	private:
		int x = 10;

	public:

		operator int() {

			return x;

		}
};

int main() {

	Integer i;

	int a = (int)i;

	cout << a;
}
```

Q10.Create a Distance class having 2 instance variable feet and inches. Also create default constructor and parameterized constructor takes 2 variables . Now overload () function call operator that takes 3 arguments a , b and c and set feet = a + c + 5 and inches = a+b + 15.

```
#include <iostream>
using namespace std;

class Distance {

	private:
		int feet, inches;


	public:
		Distance() {
		}

		Distance(int f, int i) {
			feet = f;
			inches = i;
		}

		void operator ()(int a, int b, int c ) {

			feet = a + c + 5;
			inches = a + b + 15;
		}
		void disp() {
			cout << "feet : " << feet << endl << "inches : " << inches;
		}
};

int main() {

	Distance d;
	d(10, 20, 30);
	d.disp();
}
```

Q11.Create a class Marks that have one member variable marks and one member function that will print marks. We know that we can access member functions using (.) dot operator. Now you need to overload (->) arrow operator to access that function.

```
#include <iostream>
using namespace std;

class Marks {

        private:
                int marks = 500 ;

        public:

                void disp() {

                        cout << marks;

                }

                Marks *operator ->() {

                        return this;
                }
};

int main() {

        Marks m;

        m->disp();

}
```