Operator Overloading, friend operator and this pointers

1. 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, img;

public:
    friend istream &operator>>(istream &in, Complex &C);
    friend ostream &operator<<(ostream &out, Complex &SC);
    void setData(int a, int b)
    {
        real = a;
        img = b;
    }
    void showData()
    {
        if (img >= 0)
            cout << real << " + " << img << "i" << endl;
        else
            cout << real << " - " << -img << "i" << endl;
    }
};
istream &operator>>(istream &in, Complex &C)
{
    cout << "Enter real part : ";
    in >> C.real;
    cout << "Enter img part : ";
    in >> C.img;
    return in;
}
ostream &operator<<(ostream &out, Complex &C)
{
    if (C.img >= 0)
        out << C.real << " + " << C.img << "i" << endl;
    else
        out << C.real << " - " << -C.img << "i" << endl;

    return out;
}

int main()
{
    Complex c1;
    cin >> c1;
    cout << "Complex number is : ";
    cout << c1;
    return 0;
}
===============================================================================
Output:
Enter real part : 8
Enter img part : 9
Complex number is : 8 + 9i
```

2. 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, img;
    public:
        void setData(int real, int img)
        {
            this->real = real;
            this->img = img;
        }
        void showData()
        {
            if (img >= 0)
                cout << real << " + " << img << "i" << endl;
            else
                cout << real << " - " << -img << "i" << endl;
        }
};

int main()
{
    Complex c;
    c.setData(-10, 22);
    cout << "Complex number is : ";
    c.showData();
    return 0;
}
=============================================================================
Output:
Complex number is : -10 + 22i
```

3. 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 a[100];
    const int size = 100;

public:
    void setArray(int n, int index)
    {
        a[index] = n;
    }
    void display(int index)
    {
        if (index > size)
        {
            cout << "Array index out of bound" << endl;
            exit(0);
        }
        cout << a[index] << endl;
    }
    int operator[](int index)
    {
        if (index > size)
        {
            cout << "Array index out of bound" << endl;
            exit(0);
        }
        return a[index];
    }
};
int main()
{
    Array obj1;
    obj1.setArray(20, 8);
    obj1.display(8);
    cout << obj1[120];

    return 0;
}
=============================================================================
Output:
20
Array index out of bound
```

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

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

class Student
{
private:
    char *a;
    int size;

public:
    Student()
    {
        a = new char;
        size = 1;
    }
    Student(char *str)
    {
        size = strlen(str);
        a = new char[size + 1];
        strcpy(a, str);
    }
    Student(const Student &s)
    {
        size = strlen(s.a);
        a = new char[size + 1];
        strcpy(a, s.a);
    }
    Student &operator=(const Student &s)
    {
        if (a == s.a)
        {
            return *this;
        }
        delete a;
        size = strlen(s.a);
        a = new char[size + 1];
        strcpy(a, s.a);
        return *this;
    }
    void display()
    {
        cout << a << endl;
    }
    void change(char *str)
    {
        delete a;
        size = strlen(str);
        a = new char[size + 1];
        strcpy(a, str);
    }
};

int main()
{
    Student s1("Akhtar");
    Student s2 = s1;
    Student s3;
    Student s4("Quraishi");
    Student s5;
    s3 = s1;
    s4 = s1;
    s1 = s1;
    s5 = s4 = s1;
    s1.change("Shekh");
    s1.display();
    s2.display();
    s3.display();
    s4.display();
    s5.display();
    return 0;
}
==============================================================================
Output:
Shekh
Akhtar
Akhtar
Akhtar
Akhtar
```

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

```
#include <iostream>
#include <cstdlib>
using namespace std;
// Overloading Global new operator
void *operator new(size_t sz)
{
    void *m = malloc(sz);
    cout << "User Defined :: Operator new" << std::endl;

    return m;
}
// Overloading Global delete operator
void operator delete(void *m)
{
    cout << "User Defined :: Operator delete" << std::endl;
    free(m);
}
// Overloading Global new[] operator
void *operator new[](size_t sz)
{
    cout << "User Defined :: Operator new []" << std::endl;
    void *m = malloc(sz);
    return m;
}
// Overloading Global delete[] operator
```

```
void operator delete[](void *m)
{
    cout << "User Defined :: Operator delete[]" << std::endl;
    free(m);
}
class Dummy
{
public:
    Dummy()
    {
        std::cout << "Dummy :: Constructor" << std::endl;
    }
    ~Dummy()
    {
        std::cout << "Dummy :: Destructor" << std::endl;
    }
};
int main()
{
    int *ptr = new int;
    delete ptr;
    Dummy *dummyPtr = new Dummy;
    delete dummyPtr;
    int *ptrArr = new int[5];
    delete[] ptrArr;
    return 0;
}
=============================================================================
Output:
User Defined :: Operator new
User Defined :: Operator delete
User Defined :: Operator new
Dummy :: Constructor
Dummy :: Destructor
User Defined :: Operator delete
User Defined :: Operator new []
User Defined :: Operator delete[]
```

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

```
#include <iostream>
using namespace std;
class complex
{
    private:
        int real, img;
    public:
        complex(){ real = 0; img = 0; }
        void setData(int x, int y){ real = x; img = y; }
        void operator=(complex X){ real = X.real; img = X.img;}
        void showData()
        {
            if (img >= 0)
                cout << real << " + " << img << "i" << endl;
            else
                cout << real << " - " << -img << "i" << endl;
        }
};
int main()
{
    complex c1, c2;
    c1.setData(10, -15);
    cout << "First object " << endl;
    c1.showData();
    c2 = c1;
    cout << "Second object " << endl;
    c2.showData();
    return 0;
}
=============================================================================
Output:
First object
10 - 15i
Second object
10 - 15i
```

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

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

class Integer
{
    private:
        int a;
    public:
        int setData(int x){ a = x; return a; }
        void showData(){ cout << a; }
        int operator!()
        {
            cout << "! operator called" << endl;
            if(a == 0)
                return 1;
            else
                return 0;
        }
};

int main()
{
    Integer n;
    n.setData('j');
    int result = !n;
    cout << result;
    return 0;
}
=============================================================================
Output:
! operator called
0
```

8. 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()
    {
        x = 0;
        y = 0;
        z = 0;
    }
    Coordinate(int a, int b, int c)
    {
        x = a;
        y = b;
        z = c;
    }
    Coordinate operator,(Coordinate c)
    {
        cout << "Comma operator called" << endl;
        Coordinate temp;
        temp.x = c.x;
        temp.y = c.y;
        temp.z = c.z;
        return temp;
    }
    void display()
    {
        cout << "x   =    " << x << "      y     =    " << y << "     z    =    " <<
z << endl;
    }
};

int main()
{
    Coordinate c1(4, 5, 6), c2(7, 8, 9), c3;
    c3 = (c1, c2);
    c1.display();
    c2.display();
    c3.display();
    return 0;
}
==============================================================================
Output:
Comma operator called
x   =    4      y     =    5     z    =    6
x   =    7      y     =    8     z    =    9
x   =    7      y     =    8     z    =    9
```

9. 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;

public:
    Integer()
    {
        x = 0;
    }
    Integer(int a)
    {
        x = a;
    }
    operator int()
    {
        return x;
    }
    void display()
    {
        cout << "Value : " << x << endl;
    }
};

int main()
{
    int a = 5;
    Integer i = a;
    i.display();
    int b ;
    b = (int)i;
    cout << b << endl;
    return 0;
}
==============================================================================
Output:
Value : 5
5
```

10. 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()
        {
            feet = 0;
            inches = 0;
        }
        Distance(int x, int y)
        {
            feet = x;
            inches = y;
        }
        Distance operator()(int a, int b, int c)
        {
            Distance obj;
            obj.feet = a + c + 5;
            obj.inches = a + b +  15;
            return obj;
        }
        void diplay()
        {
            cout << "Feet = " << feet << " Inches = " << inches << endl;
        }
};

int main()
{
    Distance d1(10, 12);
    Distance d2;
    d1.diplay();
    d2 = d1(10, 20, 30);
    d1.diplay();
    d2.diplay();
    return 0;
}
==============================================================================
Output:
```

```
Feet = 10 Inches = 12
Feet = 10 Inches = 12
Feet = 45 Inches = 45
```

11. 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;

public:
    Marks()
    {
        marks = 0;
    }
    Marks(int x)
    {
        marks = x;
    }
    void display()
    {
        cout << "Marks   :    " << marks << endl;
    }
    Marks *operator->()
    {
        return this;
    }
};

int main()
{
    Marks m(54);
    m.display();
    m->display();
    return 0;
}
===========================================================================
Output:
Marks   :    54
Marks   :    54
```