Assignment – 28

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;
        int img;
    public:
        Complex()
        {
            real = 0;
            img = 0;
        }
        friend istream & operator>>(istream &in,Complex &c);
        friend ostream & operator<<(ostream &out,const Complex &c);
};
istream & operator>>(istream &in,Complex &c)
{
    cout<<"Enter real part : ";
    in>>c.real;
    cout<<"Enter imaginary part : ";
    in>>c.img;
    return in;
}
ostream & operator<<(ostream &out,const Complex &c)
{
    out<<"Entered Value is :"<<endl;
    if(c.img>=0)
        out<<c.real<<" + "<<c.img<<"i"<<endl;
    else
        out<<c.real<<" - "<<c.img*(-1)<<"i"<<endl;
    return out;
}


int main()
{
    Complex c1,c2;
    cin>>c1;
    cout<<c1;
    cout<<endl;
    cin>>c2;
    cout<<c2;
    return 0;
}
```

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;
        int img;
    public:
        void setData(int real,int img)
        {
            this->real = real;
            this->img = img;
        }
        void display()
        {
            if(img>=0)
                cout<<real<<" + "<<img<<"i"<<endl;
            else
                cout<<real<<" - "<<img*(-1)<<"i"<<endl;
        }
};

int main()
{
    Complex c1,c2;
    c1.setData(2,8);
    c2.setData(3,-9);
    c1.display();
    c2.display();
    return 0;
}
```

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* arr;
        int size;
    public:
        Array(int* p=NULL,int s=0)
        {
            size = s;
            arr =NULL;
            if(s!=0)
            {
                arr = new int[s];
                for(int i=0;i<s;i++)
```

```
                {
                    arr[i] = p[i];
                }
            }
        }
        int & operator[](int s)       // operator[] overloading
        {
            if(s>=size)
            {
                cout<<"Array index out of bound "<<endl;
                exit (0);
            }
            return arr[s];
        }
        void print()
        {
            for(int i=0;i<size;i++)
            {
                cout<<arr[i]<<" ";
            }
            cout<<endl;
        }
};

int main()
{
    int a[] = {1,2,3,4,5,6};
    Array b(a,6);
    b[3] = 11;
    b.print();
    b[7]=1;
    return 0;
}
```

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:
        string name;
        int age;
    public:
        Student()
        {

        }
        Student(string name,int age)
        {
            cout<<"Constuctor called"<<endl;
            this->name = name;
            this->age = age;
```

```
        }
        void display()
        {
            cout<<"Name : "<<name<<endl;
            cout<<"Age : "<<age<<endl;
        }
        void * operator new(size_t size)
        {
            cout<<"new operator called "<<endl;
            void *p = malloc(size);
            return p;
        }
        void operator delete(void *p)
        {
            cout<<"delete operator called "<<endl;
            free(p);
        }
};

int main()
{
    Student *p = new Student("Upesh",24);
    p->display();
    delete p;
}
```

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

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

class Student
{
    private:
        string name;
        int age;
    public:
        Student()
        {

        }
        Student(string name,int age)
        {
            cout<<"Constuctor called"<<endl;
            this->name = name;
            this->age = age;
        }
        void display()
        {
            cout<<"Name : "<<name<<endl;
            cout<<"Age : "<<age<<endl;
        }
        friend void * operator new(size_t size);
        friend void operator delete(void *p);
};
```

```
void * operator new(size_t size)
{
    cout<<"new operator called "<<endl;
    void *p = malloc(size);
    return p;
}
void operator delete(void *p)
{
    cout<<"delete operator called "<<endl;
    free(p);
}

int main()
{
    Student *p = new Student("Upesh",24);
    p->display();
    delete p;
    return 0;
}
```

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

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

class Complex
{
    private:
        int real;
        int img;
    public:
        void setData(int real,int img)
        {
            this->real = real;
            this->img = img;
        }
        void display()
        {
            if(img>=0)
                cout<<real<<" + "<<img<<"i"<<endl;
            else
                cout<<real<<" - "<<img*(-1)<<"i"<<endl;
        }
        void operator=(Complex c) // = operator overloading
        {
            real = c.real;
            img = c.img;
        }
};

int main()
{
    Complex c1,c2;
    c1.setData(2,8);
    c2.setData(3,-9);
    c1.display();
    c2.display();
```

```
    c2=c1;
    c2.display();
    return 0;
}
```

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

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

class Integer
{
    private:
        int x;
    public:
        void setData(int a)
        {
            x = a;
        }
        void display()
        {
            cout<<"x = "<<x<<endl;
        }
        void operator!()
        {
            cout<<"! operator overloading called "<<endl;
            x = !x;
        }
};
int main()
{
    Integer i;
    i.setData(120);
    i.display();
    !i;
    i.display();
    return 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;
        int y;
        int z;
    public:
        Coordinate(int a=0,int b=0,int c=0)
        {
            x=a;
            y=b;
```

```
            z=c;
        }
        void display()
        {
            cout<<"Values of coordinate are : ";
            cout<<"x="<<x<<" y="<<y<<" z="<<z<<endl<<endl;
        }
        Coordinate operator,(Coordinate c)
        {
            return c;
        }
};

int main()
{
    Coordinate c1(5,2,3);
    Coordinate c2(7,9,5);
    Coordinate c3;
    c1.display();
    c2.display();
    c3.display();
    c3=(c1,c2);
    c3.display();
    return 0;
}
```

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=5;
        }
        void display()
        {
            cout<<"x = "<<x<<endl;
        }
        operator int()
        {
            return x;
        }
};

int main()
{
    Integer i;
    int j=9;
    i.display();
```

```
    cout<<"j = "<<j<<endl;
    j = i;
    cout<<"j = "<<j<<endl;
    return 0;
}
```

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;
        int inches;
    public:
        Distance()
        {
            feet = 0;
            inches = 0;
        }
        Distance(int x,int y)
        {
            feet = x;
            inches = y;
        }
        void display()
        {
            cout<<"feet = "<<feet<<"  inches = "<<inches<<endl;
        }
        void operator() (int a,int b,int c)
        {
            cout<<"() overloading called "<<endl;
            feet = a+c+5;
            inches = a+b+15;
        }
};

int main()
{
    Distance d1(2,3),d2;
    d1.display();
    d2.display();
    d1(5,6,7);
    d1.display();
    d2(1,2,3);
    d2.display();
    return 0;
}
```

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(int x=0)
        {
            marks = x;
        }
        void print()
        {
            cout<<"Marks = "<<marks<<endl;
        }
        Marks *operator->()
        {
            cout<<"-> overloading called "<<endl;
            return this;
        }
};

int main()
{
    Marks m1(91),m2;
    m1.print();
    m1->print();
    m2.print();
    m2->print();
    return 0;
}
```