**Assignment-28:**

**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

sol-

// 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{

    int real,img;

    public:

        friend ostream & operator<<(ostream &o, Complex c){

            if(c.img>0){

                cout<<"C.N - "<<c.real<<"+"<<c.img<<"i"<<endl;

            }else{

                cout<<"C.N - "<<c.real<<c.img<<"i"<<endl;

            }

            return o;

        }

        friend istream& operator>>(istream& i, Complex &x){

            cout<<"Enter Real Part: ";

            cin>>x.real;

            cout<<"Enter Imaginary Part: ";

            cin>>x.img;

            return i;

        }

};

int main(){

    Complex c1;

    cin>>c1;

    cout<<c1;

    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.

Sol-

// 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{

    int real, img;

    public:

    void setData(int real, int img){

        this->real = real;

        this->img = img;

    }

    void printData(){

        if(img>0){

            cout<<real<<" + "<<img<<"i"<<endl;

        }

        else cout<<real<<" "<<img<<"i"<<endl;

    }

};

int main(){

    Complex c1,c2;

    c1.setData(5,3);

    c1.printData();

    c2.setData(8,-1);

    c2.printData();

    return 0;

}

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

Sol-

// 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{

    int a[100];

    const int size = 100;

    public:

        void setData(int n, int index){

            if(index>=size){

                cout<<"Array Out of Bound Exception"<<endl;

                exit(0);

            }

            a[index] = n;

        }

        void display(int index){

            cout<<a[index]<<endl;

        }

        int operator[](int index){

            if(index>=size){

                cout<<"Array Out of Bound Exception"<<endl;

                exit(0);

            }

            return a[index];

        }

};

int main(){

    Array arr;

    arr.setData(5,0);

    arr.display(0);

    arr[100];

    return 0;

}

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

Sol-

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

#include<iostream>

using namespace std;

class Student{

    string name;

    int RollNo,age;

    public:

        Student(){}

        Student(string s,int r, int a){

            name = s;

            RollNo = r;

            age = a;

        }

        void display(){

            cout<<"Name: "<<name<<endl<<"Roll No: "<<RollNo<<endl<<"Age: "<<age<<endl;

        }

        void \* operator new(size\_t size){

            void \*p = malloc(size);

            cout<<"Overloaded new operator with size: "<<size<<endl;

            return p;

        }

        void operator delete(void \*p){

            free(p);

        }

};

int main(){

    Student s("Om Pant", 521, 15);

    s.display();

    Student \*s1 = new Student("Akash",152,16);

    s1->display();

    delete s1;

    return 0;

}

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

Sol-

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

#include<iostream>

using namespace std;

class Student{

    string name;

    int RollNo,age;

    public:

        Student(){}

        Student(string s,int r, int a){

            name = s;

            RollNo = r;

            age = a;

        }

        void display(){

            cout<<"Name: "<<name<<endl<<"Roll No: "<<RollNo<<endl<<"Age: "<<age<<endl;

        }

};

void \* operator new(size\_t size){

    void \*p = malloc(size);

    cout<<"Overloaded new operator with size: "<<size<<endl;

    return p;

}

void operator delete(void \*p){

    free(p);

}

int main(){

    Student s("Om Pant", 521, 15);

    s.display();

    Student \*s1 = new Student("Akash",152,16);

    int \*p = new int;

    s1->display();

    delete s1;

    return 0;

}

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

Sol-

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

#include<iostream>

using namespace std;

class Complex{

    int real, img;

    public:

        void setData(int x, int y){

            real = x;

            img = y;

        }

        void printData(){

            if(img>0){

                cout<<real<<" + "<<img<<"i"<<endl;

            }

            else cout<<real<<" "<<img<<"i"<<endl;

        }

        Complex& operator=(Complex &x){

            if(this == &x){

                return x;

            }

            real = x.real;

            img  = x.img;

            return x;

        }

};

int main(){

    Complex c1,c2, c3;

    c1.setData(5,3);

    c1.printData();

    c3 = c2 = c1;

    c2.printData();

    c3.printData();

    return 0;

}

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

Sol-

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

#include<iostream>

using namespace std;

class Integer{

    int x;

    public:

    void setVal(int a){

        x = a;

    }

    void getVal(){

        cout<<x<<endl;

    }

    void operator!(){

        x \*= -1;

    }

};

int main(){

    Integer a,b;

    a.setVal(5);

    b.setVal(-10);

    cout<<"Values Are: "<<endl;

    a.getVal();

    b.getVal();

    !a;

    !b;

    cout<<"After negation values Are: "<<endl;

    a.getVal();

    b.getVal();

    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.

Sol-

// 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{

    int x,y,z;

    public:

        Coordinate(){}

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

            this->x = x;

            this->y = y;

            this->z = z;

        }

        void display(){

            cout<<"Coordinates: "<<x<<"x "<<y<<"y "<<z<<"z "<<endl;

        }

        Coordinate operator,(Coordinate c){

            return c;

        }

};

int main(){

    Coordinate c1 = Coordinate(1,2,3);

    Coordinate c2 = Coordinate(4,5,6);

    Coordinate c3;

    c1.display();

    c2.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.

Sol-

// 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

{

    int x;

    public:

        Integer(int i){

            this->x = i;

        }

        operator int(){

            return x;

        }

};

int main(){

    Integer a = 5;

    int b = a;

    cout<<b;

    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.

Sol-

// 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{

    int feet,inches;

    public:

        Distance(){}

        Distance(int ft, int inch){

            feet = ft;

            inches = inch;

        }

        void display(){

            cout<<"Distance: "<<feet<<" Feet "<<inches<<" Inches "<<endl;

        }

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

            feet = a + c +5;

            inches = a+b+15;

        }

};

int main(){

    Distance d1 = Distance(5,3);

    Distance d2;

    d1.display();

    d2(5,2,3);

    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.

Sol-

// 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{

    int marks;

    public:

        Marks(int x){

            marks = x;

        }

        void printMarks(){

            cout<<marks<<endl;

        }

        Marks \* operator->(){

            return this;

        }

};

int main(){

    Marks m1 = 5,m2 = 6;

    m1->printMarks();

    m2->printMarks();

    return 0;

}