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

{

    int real;

    int img;

    public:

        friend void operator>>(istream &input, complex &c)

        {

            cout<<"enter the real value :  ";

            input>>c.real;

            cout<<"enter the imaginaroy value :  ";

            input>>c.img;

        }

        friend void operator<<(ostream &output, complex &c)

        {

            output<<c.real<<"  +  "<<c.img<<" i "<<endl;

        }

};

int main()

{

    complex c1,c2;

    cout<<"\n enter the complex number  "<<endl;

    cin>>c1;

    cout<<"\n enter the complex number : "<<endl;

    cin>>c2;

    cout<<"-------------------"<<endl;

    cout<<"the entered complex numbers are : "<<endl;

    cout<<c1;

    cout<<endl;

    cout<<c2;

    return 0;

}

output;

"c:\Users\tusha\Documents\coadind\assignemet\_27\assignment\_28\" ; if ($?) { g++ complex\_num.cpp -o complex\_num } ; if ($?) { .\complex\_num }

enter the complex number

enter the real value : 1

enter the imaginaroy value : 5

enter the complex number :

enter the real value : 5

enter the imaginaroy value : 4

-------------------

the entered complex numbers are :

1 + 5 i

5 + 4 i

-------------------

PS C:\Users\tusha\Documents\coadind\assignemet\_27\assignment\_28>

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;

    int img;

    public:

        complex()

        {

        }

        complex(int real, int img)

        {

            this->real= real;

            this->img=img;

        }

        friend ostream & operator<<( ostream &output , complex &c)

        {

            output<<"Real  :  "<<c.real<<"  Img  :  "<<c.img<<endl;

        }

};

int main()

{

    complex c1(1,2), c2(4,2);

    cout<<c1<<c2;

    return 0;

}

Output:

cd "c:\Users\tusha\Documents\coadind\assigment\_28.cpp\" ; if ($?) { g++ setDATA.cpp -o setDATA } ; if ($?) { .\setDATA }

Real : 1 Img : 2

Real : 4 Img : 2

PS C:\Users\tusha\Documents\coadind\assigment\_28.cpp>

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:

        Array()

        {

        }

        Array(int n, int index)

        {

            if(index>=size)

            {

                cout<<"array index is out of bound exception "<<endl;

                exit(0);

            }

            a[index]=n;

        }

        void display(int index)

        {

            if(index>=size)

            {

                cout<<"array index is out of bound exception "<<endl;

            }

            else

                cout<<"value at "<<index<<" is "<<a[index]<<endl;

        }

        int operator [](int index)

        {

            if(index>=size)

            {

                cout<<"array index is out of bound exception "<<endl;

                exit(0);

            }

            return a[index];

        }

};

int main()

{

    Array b(10,99);

    b.display(99);

    b.display(150);

    return 0;

}

Output:

"c:\Users\tusha\Documents\coadind\assigment\_28.cpp\" ; if ($?) { g++ upper\_bound.cpp -o upper\_bound } ; if ($?) { .\upper\_bound }

value at 99 is 10

array index is out of bound exception

PS C:\Users\tusha\Documents\coadind\assigment\_28.cpp>

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 set(int x, int y)

        {

            real=x;

            img=y;

        }

        void display()

        {

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

        }

        void operator=(complex c)

        {

            real=c.real;

            img=c.img;

        }

};

int main()

{

    complex c1, c2;

    c1.set(2,5);

    c2.set(7,2);

    cout<<"complex number before assigning : "<<endl;

    c1.display();

    c2.display();

    cout<<"complex number after assigning : "<<endl;

    c2=c1;

    c1.display();

    c2.display();

    return 0;

}

Output:

; if ($?) { g++ assignment\_operator.cpp -o assignment\_operator } ; if ($?) { .\assignment\_operator }

complex number before assigning :

2 + 5 i

7 + 2 i

complex number after assigning :

2 + 5 i

2 + 5 i

PS C:\Users\tusha\Documents\coadind\assigment\_28.cpp>

#include<iostream>

using namespace std;

class complex

{

    private:

        int real;

        int img;

    public:

        void set(int x, int y)

        {

            real=x;

            img=y;

        }

        void display()

        {

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

        }

        complex operator=(complex c) // thid will help to continuously a assognement operator (c3=c2=c1)

        {

            real=c.real;

            img=c.img;

            return c;

        }

};

int main()

{

    complex c1, c2,c3;

    c1.set(2,5);

    c2.set(7,2);

    cout<<"complex number before assigning : "<<endl;

    c1.display();

    c2.display();

    cout<<"complex number after assigning : "<<endl;

    c3=c2=c1;

    c1.display();

    c2.display();

    c3.display();

    return 0;

}

Ouput+

cd "c:\Users\tusha\Documents\coadind\assigment\_28.cpp\" ; if ($?) { g++ assignment\_operator.cpp -o assignment\_operator } ; if ($?) { .\assignment\_operator }

complex number before assigning :

2 + 5 i

7 + 2 i

complex number after assigning :

2 + 5 i

2 + 5 i

2 + 5 i

PS C:\Users\tusha\Documents\coadind\assigment\_28.cpp>

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

#include<iostream>

using namespace std;

class integer

{

    int i;

    public:

        integer(){

        }

        integer(int i): i(i)

        {

        }

        void display()

        {

            cout<<i;

        }

       int operator!()

       {

        return  -i;// we can do anything like return 0 or ++i or i/n this the .. //dvantage of not operator

       }

};

int main()

{

    integer x=10;

    cout<<!x;

    return 0;

}

cd "c:\Users\tusha\Documents\coadind\assigment\_28.cpp\" ; if ($?) { g++ not\_operator.cpp -o not\_operator } ; if ($?) { .\not\_operator }

-10

PS C:\Users\tusha\Documents\coadind\assigment\_28.cpp>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 coordinator

{

    int x,y,z;

    public:

        coordinator()

        {

        }

        coordinator(int a, int b, int c)

        {

            x=a;

            y=b;

            z=c;

        }

        void display()

        {

            cout<<"x= "<<x<<", y = "<<y<<", z = "<<z<<endl;

        }

        coordinator operator,(coordinator c)

        {

            coordinator temp;

            temp.x=c.x;

            temp.y=c.y;

            temp.z=c.z;

            return temp;

        }

};

int main()

{

    coordinator c1(1,5,4);

    coordinator c2(5,6,8);

    coordinator c3;

    cout<<"Before using the comma operator : "<<endl;

    c1.display();

    c2.display();

    cout<<"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_"<<endl;

    cout<<"after using comma operator c3=(c1,c2)"<<endl;

    c3=(c1,c2);

    c3.display();

    return 0;

}

Output:

gment\_28.cpp\" ; if ($?) { g++ coma\_operaotr.cpp -o coma\_operaotr } ; if ($?) { .\coma\_operaotr }

Before using the comma operator :

x= 1, y = 5, z = 4

x= 5, y = 6, z = 8

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

after using comma operator c3=(c1,c2)

x= 5, y = 6, z = 8

PS C:\Users\tusha\Documents\coadind\assigment\_28.cpp>

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 mark;

    public:

        void set\_mark(int m)

        {

            mark=m;

        }

        void display()

        {

            cout<<mark<<endl;

        }

        marks\* operator->()

        {

            return this;

        }

};

int main()

{

    marks m;

    m.set\_mark(50);

    m.display();

    m->display();

    return 0;

}

"c:\Users\tusha\Documents\coadind\assigment\_28.cpp\" ; if ($?) { g++ arrowopertor.cpp -o arrowopertor } ; if ($?) { .\arrowopertor }

50

50

PS C:\Users\tusha\Documents\coadind\assigment\_28.cpp>

#include<iostream>

using namespace std;

class shape

{

    float area;

    public:

        void set\_area(float a)

        {

            area=a;

        }

        float display()

        {

            cout<<"area  :  "<<area<<endl;

        }

};

class rectangle

{

    shape s;

    int l,b;

    public:

        rectangle()

        {

            l=0;

            b=0;

        }

        rectangle(int x, int y)

        {

            l=x;

            b=y;

        }

        void find\_area()

        {

            s.set\_area(l\*b);

        }

        shape \* operator->()

        {

            return &s;

        }

};

class circle

{

    shape s;

    int r;

    public:

        circle()

        {

            r=0;

        }

        circle(int x)

        {

            r=x;

        }

        void find\_area()

        {

            s.set\_area(3.1415 \* r \*r);

        }

        shape \* operator->()

        {

            return &s;

        }

};

class triangle

{

    shape s;

    int b,h;

    public:

        triangle()

        {

            h=0;

            b=0;

        }

        triangle(int x, int y)

        {

            b=x;

            h=y;

        }

        void find\_area()

        {

            s.set\_area(0.5\*b\*h);

        }

        shape \* operator->()

        {

            return &s;

        }

};

int main()

{

    circle c(2.5);

    c.find\_area();

    cout<<"circle ";

    c->display();

    cout<<"-------------------------"<<endl;

    rectangle r(5,4);

    r.find\_area();

    cout<<"rectangle ";

    r->display();

    cout<<"--------------------------"<<endl;

    triangle t(3,4);

    t.find\_area();

    cout<<"triangle ";

    t->display();

    cout<<"---------------------------"<<endl;

    return 0;

}

PS C:\Users\tusha\Documents\coadind\assigment\_28.cpp> cd "c:\Users\tusha\Documents\coadind\assigment\_28.cpp\" ; if ($?) { g++ arrow\_operator.cpp -o arrow\_operator } ; if ($?) { .\arrow\_operator }

circle area : 12.566

-------------------------

rectangle area : 20

--------------------------

triangle area : 6

---------------------------

PS C:\Users\tusha\Documents\coadind\assigment\_28.cpp>