Assignment - 28 A Job Ready Bootcamp in C++, DSA and IOT MySirG

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:

Complex(int x, int y)

{

real = x;

img = y;

}

Complex() {}

friend ostream &operator<<(ostream &output, Complex c);

friend istream &operator>>(istream &input, Complex c);

};

ostream &operator<<(ostream &output, Complex c)

{

output << "Real = " << c.real << " Imag = " << c.img << endl;

return output;

}

istream &operator>>(istream &input, Complex c)

{

input >> c.real >> c.img;

return input;

}

int main()

{

Complex c1(2, 3), c2(4, 4);

cout << c1 << 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, img;

public:

Complex(int x, int y)

{

real = x;

img = y;

}

Complex() {}

void setData(int real, int img)

{

this->real = real;

this->img = img;

}

friend ostream &operator<<(ostream &output, Complex c);

friend istream &operator>>(istream &input, Complex c);

};

ostream &operator<<(ostream &output, Complex c)

{

output << "Real = " << c.real << " Imag = " << c.img << endl;

return output;

}

istream &operator>>(istream &input, Complex c)

{

input >> c.real >> c.img;

return input;

}

int main()

{

Complex c1, c2;

c1.setData(3, 4);

c2.setData(5, 3);

cout << c1 << c2;

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[100];

const int size = 100;

public:

void setValue(int value, int index)

{

if (index >= size)

{

cout << "Array Index is Out of Bound Exception..!" << endl;

exit(0);

}

arr[index] = value;

}

void showValue(int index)

{

if (index >= size)

{

cout << "Array Index is out of bound exception...!" << endl;

exit(0);

}

cout << arr[index] << endl;

}

int operator[](int index)

{

if (index >= size)

{

cout << "Array index is out of bound exception...!" << endl;

exit(0);

}

return arr[index];

}

};

int main()

{

Array b;

b.setValue(35, 55);

b.setValue(343, 3);

// b.setValue(3,343);

b.showValue(55);

b.showValue(3);

// b.showValue(343);

cout << b[55] << endl;

return 0;

}

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

of the class.

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

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() {}

Complex(int x, int y)

{

real = x;

img = y;

}

void display() { cout << "Real = " << real << " Imag = " << img << endl; }

Complex operator=(Complex c)

{

// if we crete dma memory then first free this memory

// but first check both the object are not equal

// third point is c1=c2=c3

// here we don't need to check this all condition because we don't cretae dma memory

this->real = c.real;

this->img = c.img;

return \*this;

}

};

int main()

{

Complex c1(5, 4), c2(5, 3);

c1 = c2;

c1.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 i;

public:

Integer(int i) : i(i) {}

int operator!()

{

return -i; // return !i;

}

};

int main()

{

Integer x(5); // x = 5(PC constructor called )

cout << !x << endl;

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, y, z;

public:

Coordinate() {}

Coordinate(int x, int y, int z)

{

this->x = x;

this->y = y;

this->z = z;

}

void display() { cout << "x = " << x << " y = " << y << " z = " << z << endl; }

Coordinate operator,(Coordinate &c)

{

Coordinate temp;

temp.x = c.x;

temp.y = c.y;

temp.z = c.z;

return temp;

}

};

int main()

{

Coordinate c1(4, 3, 5), c2(3, 5, 3), c3, c4(5, 3, 3);

c3 = (c1, c2);

c3.display();

c3 = (c1, c2, c4);

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 Coordinate

{

private:

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 << "x = " << x << " y = " << y << " z = " << z << endl; }

Coordinate operator,(Coordinate &c)

{

Coordinate temp;

temp.x = c.x;

temp.y = c.y;

temp.z = c.z;

return temp;

}

};

int main()

{

Coordinate c1(4, 3, 5), c2(3, 5, 3), c3, c4(5, 3, 3);

c3 = (c1, c2);

c3.display();

c3 = (c1, c2, c4);

c3.display();

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, 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 display() { cout << "Feet = " << feet << " Inches = " << inches << endl; }

};

int main()

{

Distance d1(4,3), d2, d3;

d1(2,3,4);

d1.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() {}

Marks(int x) { marks = x; }

void printMarks() { cout << "Marks: " << marks << endl; }

Marks \*operator->()

{

return this;

}

};

int main()

{

Marks m1(55), m2;

m1->printMarks();

return 0;

}