Assignment - 27 A Job Ready Bootcamp in C++, DSA and IOT MySirG

Operator overloading and friend function

1. Define a class Complex with appropriate instance variables and member functions.

Define following operators in the class:

a. +

b. -

c. \*

d. ==

#include <iostream>

using namespace std;

class Complex

{

private:

int real, img;

public:

void setData(int x, int y)

{

real = x;

img = y;

}

void showData()

{

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

}

Complex operator+(Complex C) // Part - A

{

Complex temp;

temp.real = real + C.real;

temp.img = img + C.img;

return temp;

}

Complex operator-(Complex C) // Part - B

{

Complex temp;

temp.real = real - C.real;

temp.img = img - C.img;

return temp;

}

Complex operator\*(Complex C) // Part - 3

{

Complex temp;

temp.real = real \* C.real;

temp.img = img \* C.img;

return temp;

}

bool operator==(Complex C) // Part - 4

{

if (real == C.real && img == C.img)

return true;

return false;

}

};

int main()

{

Complex c1, c2, c3, c4, c5, c6;

c1.setData(5, 4);

c2.setData(5, 4);

c1.showData();

c2.showData();

c3 = c1 + c2; // c1.operator+(c2);

c3.showData();

c4 = c1 - c2; // c1.operator-(c2);

c4.showData();

c5 = c1 \* c2; // c1.operator\*(c2);

c5.showData();

bool result = c1 == c2; // c1.operator==(c2);

result ? cout << "Equal" : cout << "Not Equal ";

return 0;

}

2. Write a C++ program to overload unary operators that is increment and decrement.

#include <iostream>

using namespace std;

class Complex

{

private:

int real, img;

public:

void setData(int x, int y)

{

real = x;

img = y;

}

void showData()

{

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

}

Complex operator++() // Pre Increment

{

Complex temp;

temp.real = ++real;

temp.img = ++img;

return temp;

}

Complex operator++(int) // Post Increment, Just Pass Int for Deference ++(pre) and ++(post)

{

Complex temp;

temp.real = real++;

temp.img = img++;

return temp;

}

};

int main()

{

Complex c1, c2, c3, c4;

c1.setData(5, 4);

c2.setData(6, 7);

c1.showData();

c2.showData();

c4 = c2++; // Unary operator // c2.operator++(int);

c3 = ++c2; // Unary operator // c2.operator++();

c3.showData();

c4.showData();

return 0;

}

3. Write a C++ program to add two complex numbers using operator overloaded by a

friend function.

#include <iostream>

using namespace std;

class Complex

{

private:

int real, img;

public:

void setData(int x, int y)

{

real = x;

img = y;

}

void showData()

{

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

}

friend Complex operator+(Complex, Complex);

};

Complex operator+(Complex x, Complex y)

{

Complex temp;

temp.real = x.real + y.real;

temp.img = x.img + y.img;

return temp;

}

int main()

{

Complex c1, c2, c3;

c1.setData(5, 4);

c2.setData(6, 7);

c1.showData();

c2.showData();

c3 = c1 + c2; // operator+(c1,c2);

c3.showData();

return 0;

}

4. Create a class Time which contains:

- Hours

- Minutes

- Seconds

Write a C++ program using operator overloading for the following:

1. = = : To check whether two Times are the same or not.

2. >> : To accept the time.

3. << : To display the time.

Output –

#include <iostream>

using namespace std;

class Time

{

private:

int hours, minutes, second;

public:

friend void operator>>(istream &cin, Time &t);

friend void operator<<(ostream &cout, Time t);

friend bool operator==(Time t1, Time t2);

};

bool operator==(Time t1, Time t2)

{

if (t1.hours == t2.hours && t1.minutes == t2.minutes && t1.second == t2.second)

return true;

return false;

}

void operator<<(ostream &cout, Time t)

{

cout << "Hours: " << t.hours << endl;

cout << "Minutes: " << t.minutes << endl;

cout << "Second: " << t.second << endl;

}

void operator>>(istream &cin, Time &t)

{

cout << "Enter Hours: ";

cin >> t.hours;

cout << "Enter Minutes: ";

cin >> t.minutes;

cout << "Enter Second: ";

cin >> t.second;

}

int main()

{

Time t1, t2, t3;

cout << "Enter First Time: \n";

cin >> t1; // operator>>(cin,t1);

cout << "First Time: \n";

cout << t1; // operator << (cout, t1);

cout << "Enter Second Time: \n";

cin >> t2;

cout << t2;

bool result = (t1 == t2);

result ? cout << "Time are same" : cout << "Time are not same";

return 0;

}

5. Consider following class Numbers

class Numbers

{

int x,y,z;

public:

// methods

};

Overload the operator unary minus (-) to negate the numbers.

#include <iostream>

using namespace std;

class Number

{

private:

int x, y, z;

public:

void setData(int a, int b, int c)

{

x = a;

y = b;

z = c;

}

void showData()

{

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

}

Number operator-()

{

Number temp;

temp.x = -x;

temp.y = -y;

temp.z = -z;

return temp;

}

};

int main()

{

Number n1, n2, n3;

n1.setData(5, 4, 2);

n2 = -n1; // n1.operator-();

n1.showData();

n2.showData();

return 0;

}

6. Create a class CString to represent a string.

a) Overload the + operator to concatenate two strings.

b) == to compare 2 strings.

#include <iostream>

#include <string.h>

using namespace std;

class CString

{

private:

char str[100];

public:

void setString(char \*arr)

{

strcpy(str, arr);

}

void showString()

{

cout << str << endl;

}

CString operator+(CString s) // concatenate two string

{

CString temp;

strcpy(temp.str, str);

int length = strlen(temp.str);

strcpy(temp.str + length, s.str);

return temp;

}

bool operator==(CString s) // compare two string (==)

{

if (strcmp(str, s.str) == 0)

return 1;

return 0;

}

};

int main()

{

CString s1, s2, s3;

s1.setString("Welcome ");

s2.setString("MySirG");

s1.showString();

s2.showString();

s3 = s1 + s2; // s1.operator+(s2);

cout<<"Concatenate String: ";

s3.showString();

bool result = s1 == s2;

if (result)

cout << "String are equal";

else

cout << "String are not equal ";

return 0;

}

7. Define a C++ class fraction

class fraction

{

long numerator;

long denominator;

Public:

fraction (long n=0, long d=0);

}

Overload the following operators as member or friend:

a) Unary ++ (pre and post both)

b) Overload as friend functions: operators << and >>.

Output-

#include <iostream>

using namespace std;

class Fraction

{

private:

long numerator, denominator;

public:

void setFractionValue(long x = 0, long y = 0)

{

numerator = x;

denominator = y;

}

void showFractionValue()

{

cout << numerator << "/" << denominator << endl;

}

friend Fraction operator++(Fraction &);

friend Fraction operator++(Fraction &, int);

};

Fraction operator++(Fraction &f)

{

++f.numerator;

++f.denominator;

return f;

}

Fraction operator++(Fraction &f, int)

{

Fraction temp;

temp.numerator = f.numerator;

temp.denominator = f.denominator;

f.numerator++;

f.denominator++;

return temp;

}

int main()

{

Fraction f1, f2, f3, f4;

int x, y;

f1.setFractionValue();

f2.setFractionValue();

f1.showFractionValue();

f2.showFractionValue();

cout << endl;

cout << "Enter First Fraction Value: \n";

cout << "Numerator: ";

cin >> x;

cout << "Denomirator: ";

cin >> y;

f1.setFractionValue(x, y);

cout << endl;

cout << "f1++: ";

f1++;

f1.showFractionValue();

cout << "++f1: ";

++f1;

f1.showFractionValue();

cout << endl;

cout << "Enter Second Fraction Value: \n";

cout << "Numerator: ";

cin >> x;

cout << "Denomirator: ";

cin >> y;

f1.setFractionValue(x, y);

cout << endl;

cout << "f2 = ++f1";

f2 = ++f1;

cout << "f1 : ";

f1.showFractionValue();

cout << "f2 : ";

f2.showFractionValue();

cout << "f2 = f1++";

f2 = f1++;

cout << "f1 : ";

f1.showFractionValue();

cout << "f2 : ";

f2.showFractionValue();

cout << endl;

return 0;

}

8. Consider a class Matrix

Class Matrix

{

int a[3][3];

Public:

//methods;

};

Overload the - (Unary) should negate the numbers stored in the object.

Output –

#include <iostream>

using namespace std;

class Matrix

{

private:

int a[3][3];

public:

void setMatrix();

void showMatrix();

friend Matrix operator-(Matrix);

};

Matrix operator-(Matrix m)

{

for (int i = 0; i < 3; i++)

for (int j = 0; j < 3; j++)

m.a[i][j] = -m.a[i][j];

return m;

}

void Matrix::setMatrix()

{

cout << "Enter Matrix Element (3\*3) \n";

for (int i = 0; i < 3; i++)

for (int j = 0; j < 3; j++)

cin >> a[i][j];

}

void Matrix::showMatrix()

{

{

cout << "Matrix is: " << endl;

for (int i = 0; i < 3; i++)

{

for (int j = 0; j < 3; j++)

{

cout << a[i][j] << "\t";

}

cout << endl;

}

}

}

int main()

{

Matrix m1, m2;

m1.setMatrix();

m1.showMatrix();

m2 = -m1; //

m2.showMatrix();

return 0;

}

9. Consider the following class mystring

Class mystring

{

char str [100];

Public:

// methods

};

Overload operator “!” to reverse the case of each alphabet in the string

(Uppercase to Lowercase and vice versa).

#include <iostream>

using namespace std;

class Matrix

{

private:

int a[3][3];

public:

void setMatrix();

void showMatrix();

friend Matrix operator-(Matrix);

};

Matrix operator-(Matrix m)

{

for (int i = 0; i < 3; i++)

for (int j = 0; j < 3; j++)

m.a[i][j] = -m.a[i][j];

return m;

}

void Matrix::setMatrix()

{

cout << "Enter Matrix Element (3\*3) \n";

for (int i = 0; i < 3; i++)

for (int j = 0; j < 3; j++)

cin >> a[i][j];

}

void Matrix::showMatrix()

{

{

cout << "Matrix is: " << endl;

for (int i = 0; i < 3; i++)

{

for (int j = 0; j < 3; j++)

{

cout << a[i][j] << "\t";

}

cout << endl;

}

}

}

int main()

{

Matrix m1, m2;

m1.setMatrix();

m1.showMatrix();

m2 = -m1; //

m2.showMatrix();

return 0;

}

10.Class Matrix

{

int a[3][3];

Public:

//methods;

};

Let m1 and m2 are two matrices. Find out m3=m1+m2 (use operator

overloading).

Output –

#include <iostream>

using namespace std;

class Matrix

{

private:

int a[3][3];

public:

void inputMatrix()

{

cout << "Enter Matrix Element (3\*3): \n";

for (int i = 0; i < 3; i++)

for (int j = 0; j < 3; j++)

cin >> a[i][j];

}

void showMatrix()

{

cout << "Matrix is: \n";

for (int i = 0; i < 3; i++)

{

for (int j = 0; j < 3; j++)

{

cout << a[i][j] << " ";

}

cout << endl;

}

}

Matrix operator+(Matrix m)

{

Matrix temp;

for (int i = 0; i < 3; i++)

{

for (int j = 0; j < 3; j++)

{

temp.a[i][j] = a[i][j] + m.a[i][j];

}

}

return temp;

}

};

int main()

{

Matrix m1, m2, m3;

m1.inputMatrix();

m1.showMatrix();

m2.inputMatrix();

m2.showMatrix();

m3 = m1 + m2;

m3.showMatrix();

return 0;

}