ASSIGNMENT

1. Create a class FLOAT that contains one float data member .Overload all the four

arithmetic operators so that they operate on the objects of FLOAT.

#include <iostream>

using namespace std;

class FLOAT {

private:

float num;

public:

FLOAT(float n) : num(n) {}

// Overloading arithmetic operators

FLOAT operator+(const FLOAT& f) {

return FLOAT(num + f.num);

}

FLOAT operator-(const FLOAT& f) {

return FLOAT(num - f.num);

}

FLOAT operator\*(const FLOAT& f) {

return FLOAT(num \* f.num);

}

FLOAT operator/(const FLOAT& f) {

if (f.num != 0) {

return FLOAT(num / f.num);

} else {

cerr << "Error: Division by zero" << endl;

return FLOAT(0);

}

}

// Overloading insertion operator for easy printing

friend ostream& operator<<(ostream& out, const FLOAT& f) {

out << f.num;

return out;

}

};

int main() {

FLOAT a(5.5), b(2.5);

// Testing arithmetic operations

cout << "Addition: " << a + b << endl;

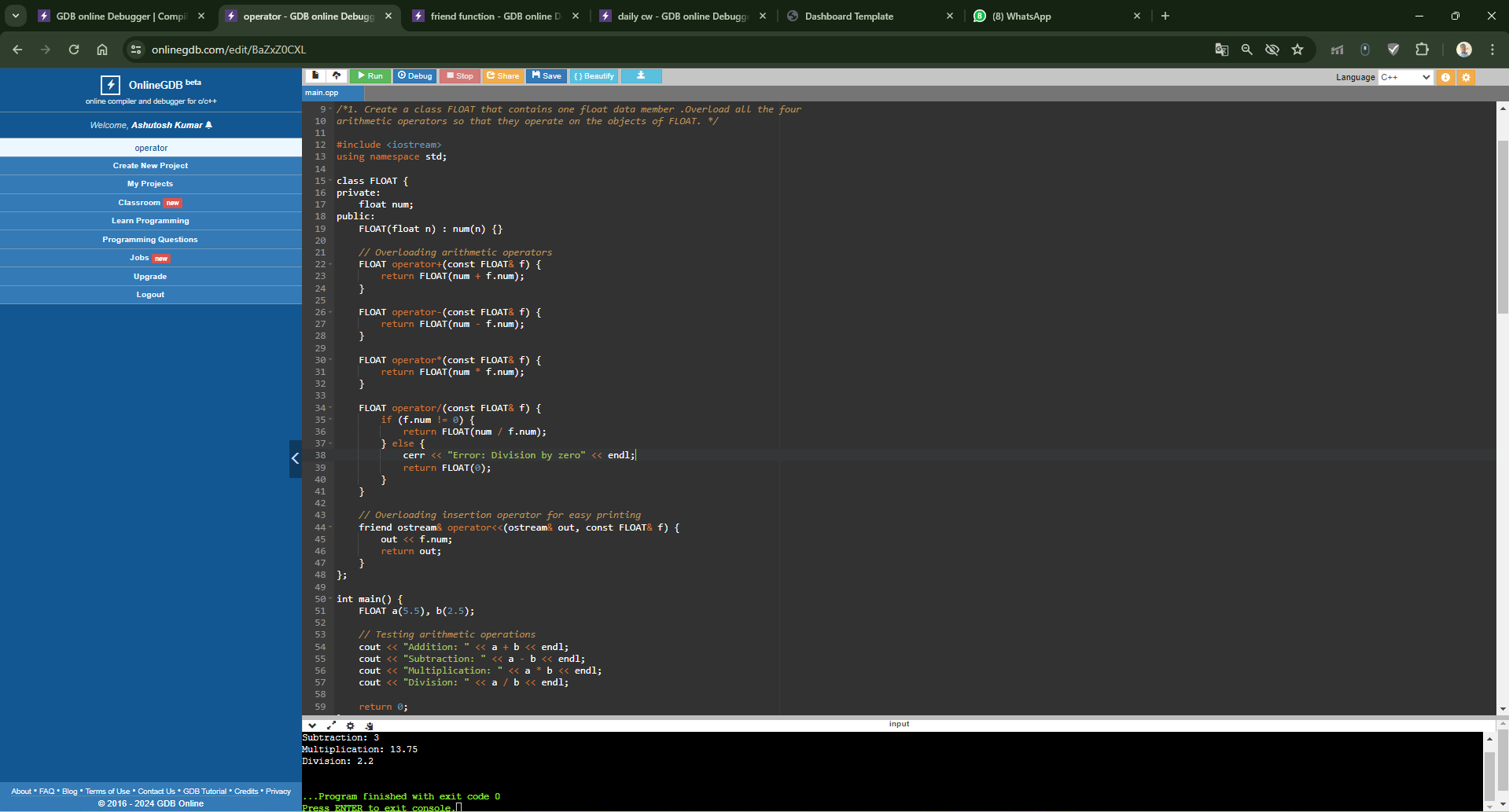
cout << "Subtraction: " << a - b << endl;

cout << "Multiplication: " << a \* b << endl;

cout << "Division: " << a / b << endl;

return 0;

}



2. Define a class string. Overlaod ==operator to compare 2 strings.

#include <iostream>

#include <cstring>

using namespace std;

class String {

private:

char\* str;

public:

String(const char\* s) {

str = new char[strlen(s) + 1];

strcpy(str, s);

}

// Overloading == operator to compare strings

bool operator==(const String& s) {

return strcmp(str, s.str) == 0;

}

};

int main() {

String s1("Hello"), s2("World"), s3("Hello");

// Testing string comparison

if (s1 == s2) {

cout << "Strings are equal" << endl;

} else {

cout << "Strings are not equal" << endl;

}

if (s1 == s3) {

cout << "Strings are equal" << endl;

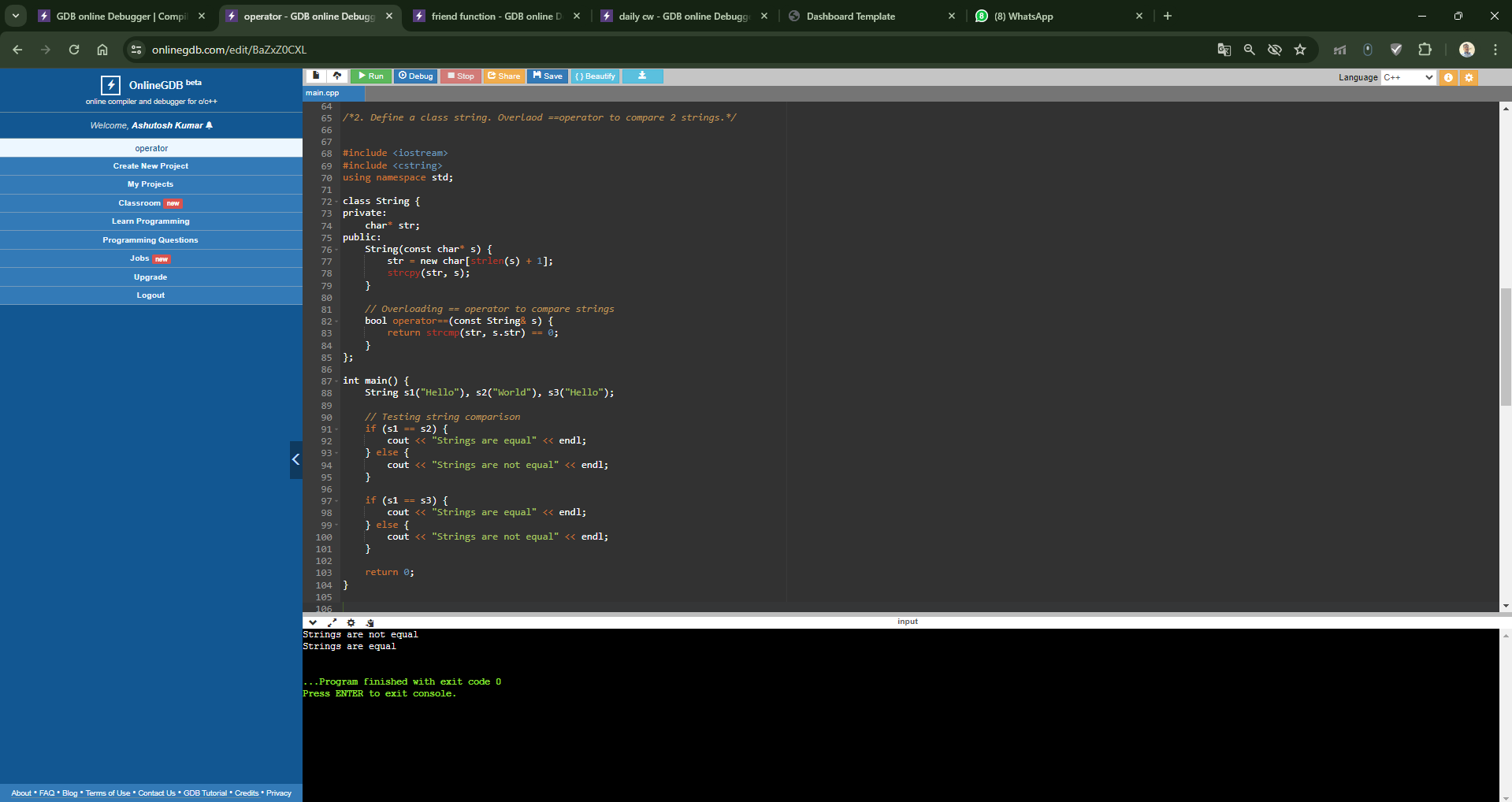
} else {

cout << "Strings are not equal" << endl;

}

return 0;

}



3. Create a Complex class that has real(int) and img(int) as member data, and has getData

and showData functions. Then also overload the following operators for Complex class. =,

==, +, ++, --,

#include <iostream>

using namespace std;

class Complex {

private:

int real;

int img;

public:

Complex(int r = 0, int i = 0) : real(r), img(i) {}

// Getter function to get data

void getData() {

cout << "Enter real and imaginary parts: ";

cin >> real >> img;

}

// Display function to show data

void showData() {

cout << "Complex number: " << real << " + " << img << "i" << endl;

}

// Overloading assignment operator

Complex& operator=(const Complex& c) {

real = c.real;

img = c.img;

return \*this;

}

// Overloading equality operator

bool operator==(const Complex& c) {

return (real == c.real) && (img == c.img);

}

// Overloading addition operator

Complex operator+(const Complex& c) {

Complex temp;

temp.real = real + c.real;

temp.img = img + c.img;

return temp;

}

// Overloading pre-increment operator

Complex operator++() {

++real;

++img;

return \*this;

}

// Overloading post-increment operator

Complex operator++(int) {

Complex temp = \*this;

++real;

++img;

return temp;

}

// Overloading pre-decrement operator

Complex operator--() {

--real;

--img;

return \*this;

}

// Overloading post-decrement operator

Complex operator--(int) {

Complex temp = \*this;

--real;

--img;

return temp;

}

};

int main() {

Complex c1(3, 4), c2(2, 6), c3;

// Testing overloaded operators

c3 = c1 + c2;

c3.showData();

++c3;

c3.showData();

c3++;

c3.showData();

--c3;

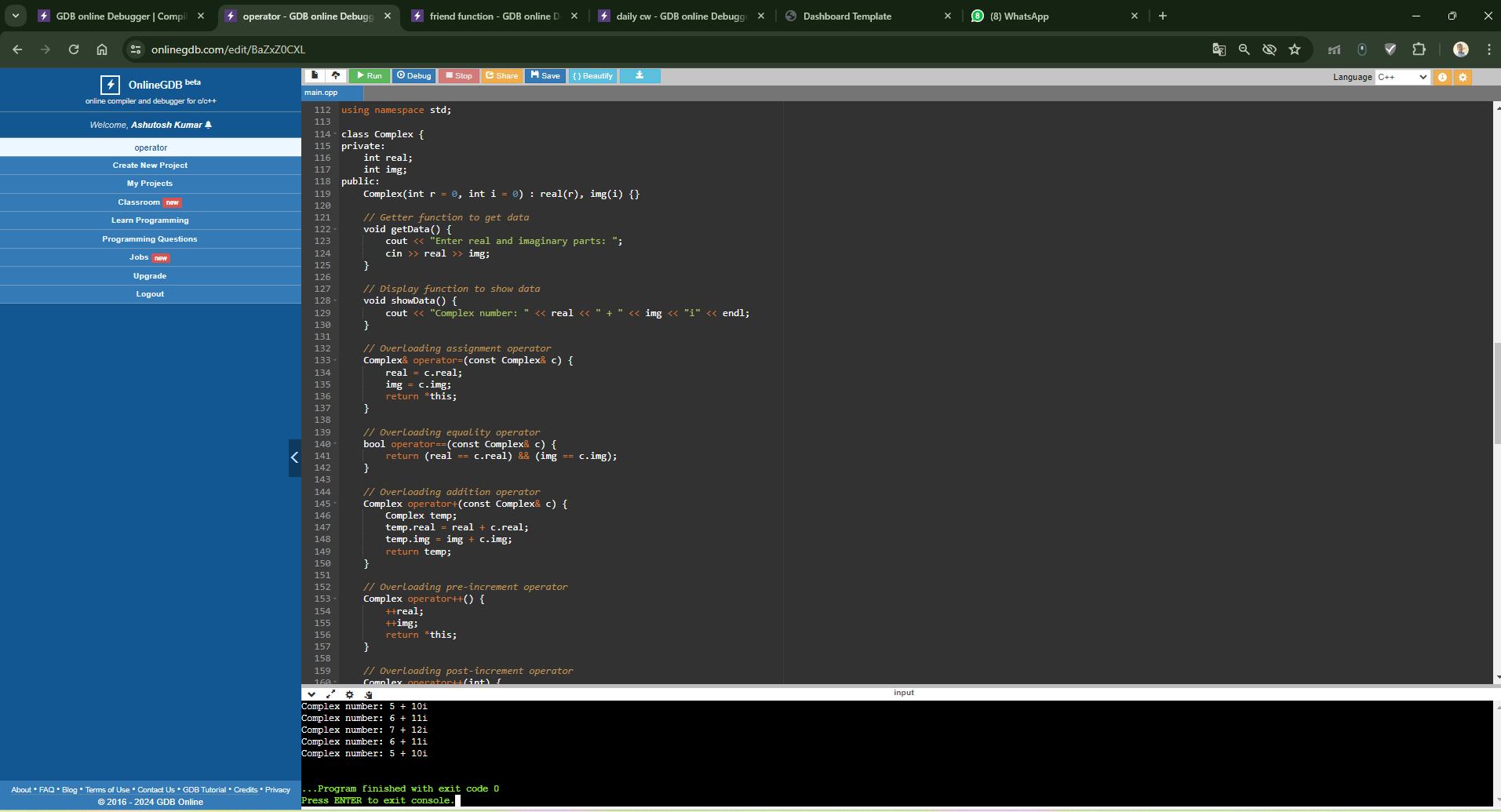
c3.showData();

c3--;

c3.showData();

return 0;

}



4. Write a C++ program to overload ‘!’ operator using friend function

#include <iostream>

using namespace std;

class Box {

private:

double width;

public:

Box(double w) : width(w) {}

friend void printWidth(Box box);

};

// Friend function to access private member

void printWidth(Box box) {

cout << "Width of box: " << box.width << endl;

}

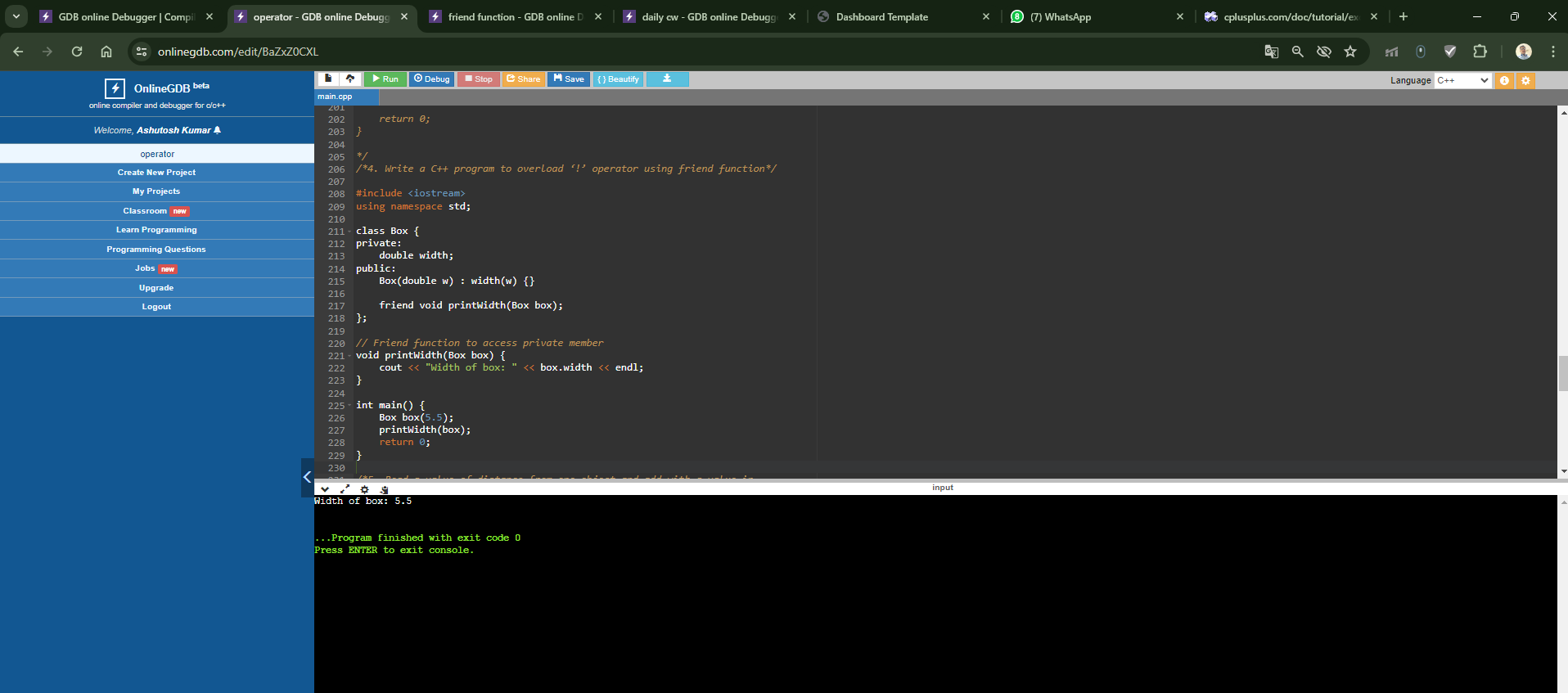
int main() {

Box box(5.5);

printWidth(box);

return 0;

}



5. Read a value of distance from one object and add with a value in

another object using friend function.

#include <iostream>

using namespace std;

class Distance {

private:

float distance;

public:

Distance(float d) : distance(d) {}

friend void addDistance(Distance& d1, Distance& d2);

};

// Friend function to add distances

void addDistance(Distance& d1, Distance& d2) {

float sum = d1.distance + d2.distance;

cout << "Sum of distances: " << sum << endl;

}

int main() {

Distance d1(10.5), d2(5.5);

addDistance(d1, d2);

return 0;

}

