**Day\_7\_Assignments**

**1.**

#include <iostream>

#include <string>

using namespace std;

class User {

private:

string username;

string profilePic;

public:

User(const string& user, const string& picture) :username(user), profilePic(picture) {}

string getUsername() const {

return username;

}

string getProfilePic() const {

return profilePic;

}

};

int main() {

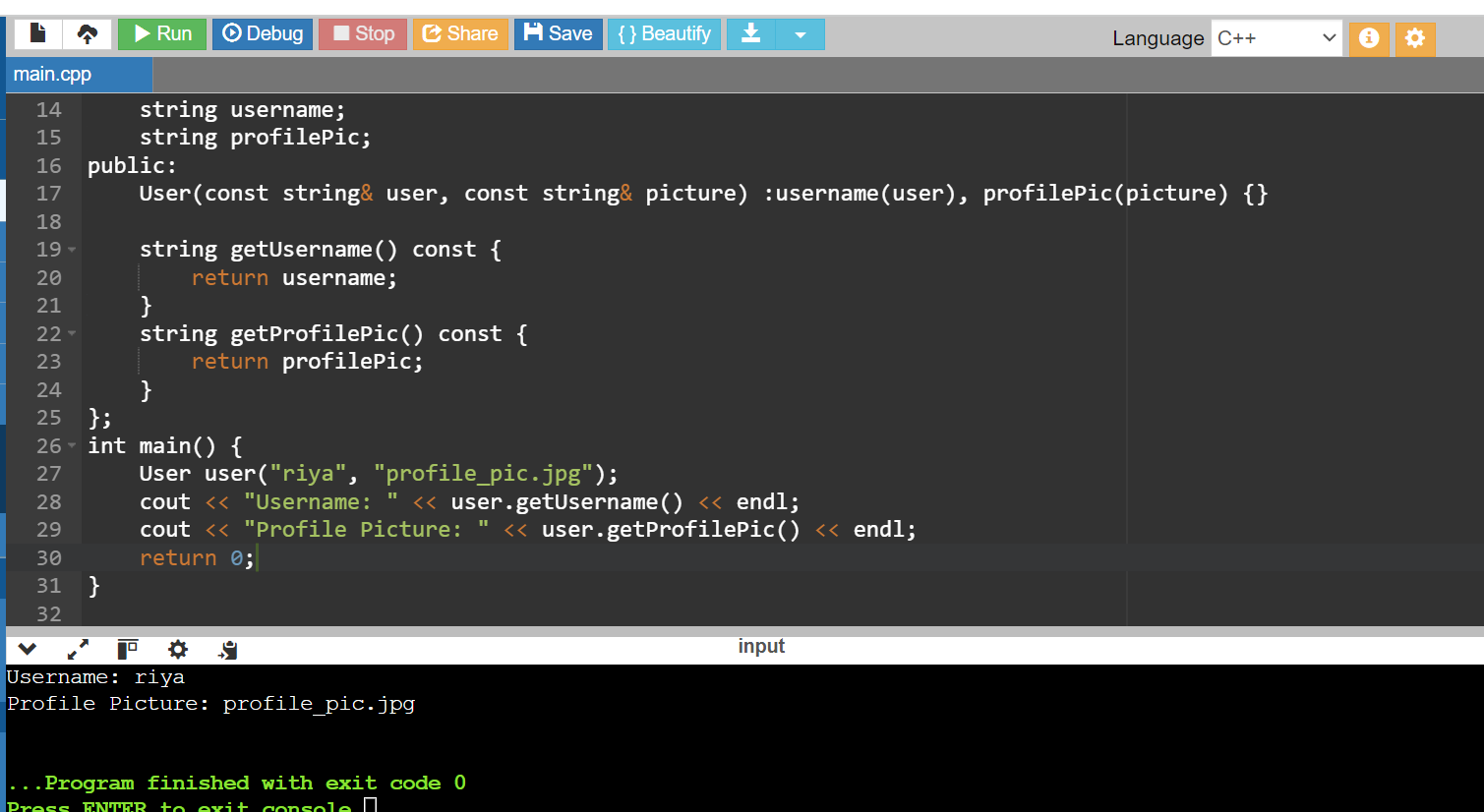
User user("riya", "profile\_pic.jpg");

cout << "Username: " << user.getUsername() << endl;

cout << "Profile Picture: " << user.getProfilePic() << endl;

return 0;

}



**2.**

#include <iostream>

#include <string>

using namespace std;

class User {

private:

string username;

string profilePicture;

public:

User(const string& user, const string& picture) : username(user), profilePicture(picture) {}

string getUsername() const {

return username;

}

string getProfilePicture() const {

return profilePicture;

}

};

class Post : public User {

private:

string postContent;

string timestamp;

public:

Post(const string& user, const string& picture, const string& content, const string& time)

: User(user, picture), postContent(content), timestamp(time) {}

string getPostInfo() const {

return "Username: " + getUsername() + "\nProfile Picture: " + getProfilePicture() +

"\n Post Content: " + postContent + "\nTimestamp: " + timestamp;

}

};

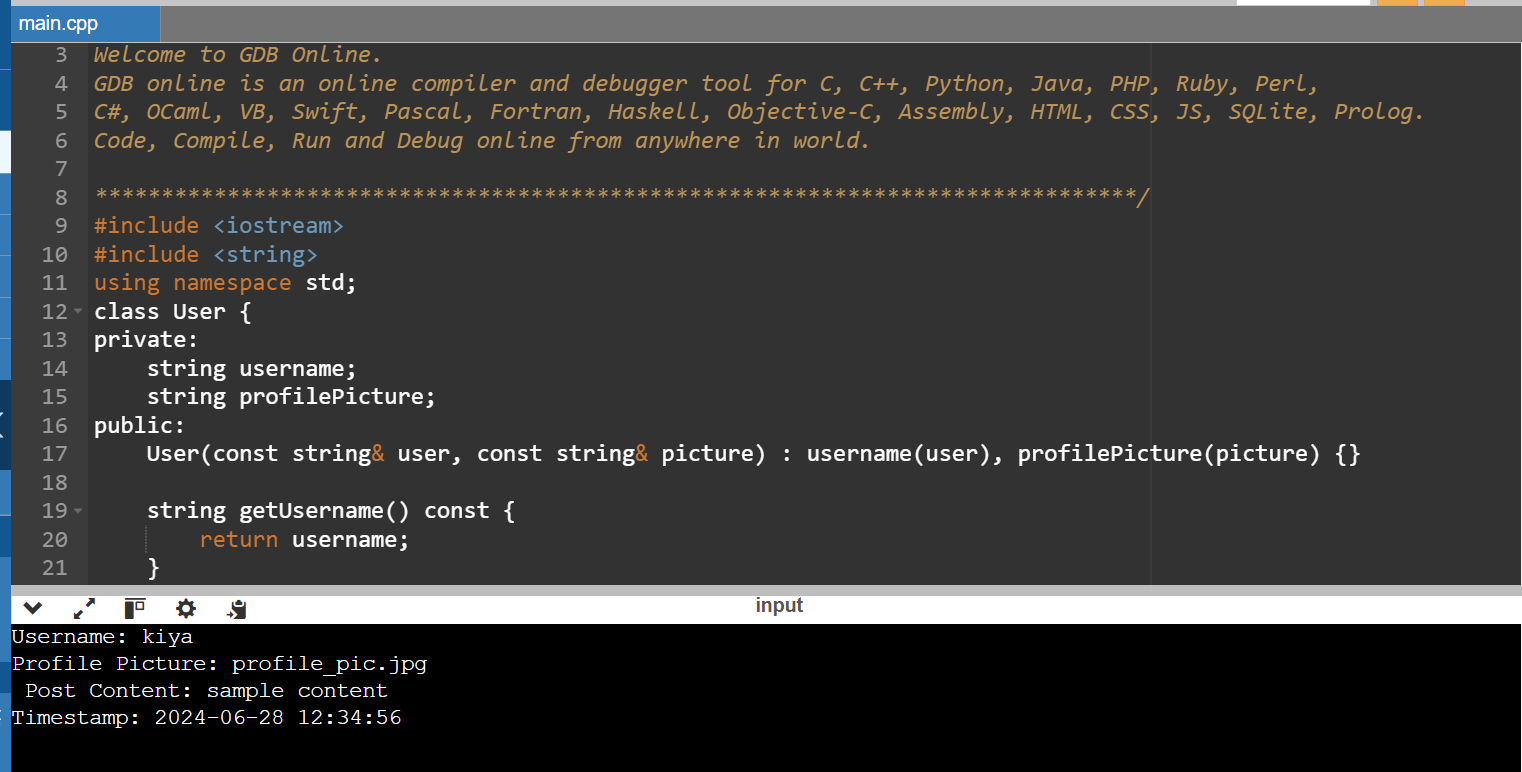
int main() {

Post post("kiya", "profile\_pic.jpg", "sample content", "2024-06-28 12:34:56");

cout << post.getPostInfo() << endl;

return 0;

}



**3.**

#include<iostream>

#include <string>

using namespace std;

class User {

public:

string name;

User(const string& name) : name(name) {}

};

void basicInteract(const User& user1, const User& user2) {

cout << user1.name << " interacts with " << user2.name << "." <<endl;

}

int main() {

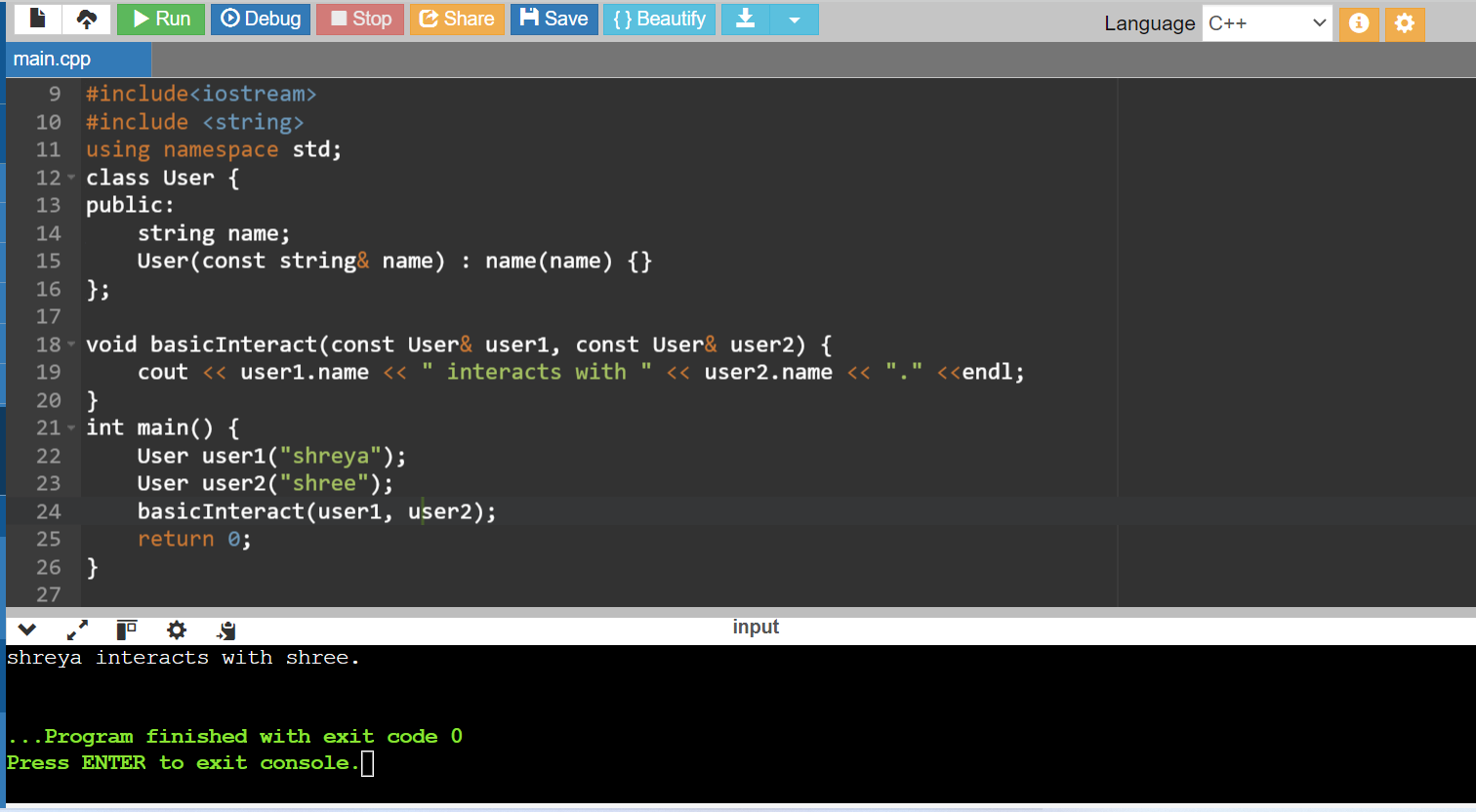
User user1("shreya");

User user2("shree");

basicInteract(user1, user2);

return 0;

}



**4.**

#include <iostream>

#include <string>

class User {

public:

std::string name;

User(const std::string& name) : name(name) {}

};

class Post {

public:

std::string content;

Post(const std::string& content) : content(content) {}

};

void likePost(User& user, Post& post) {

std::cout << user.name << " liked the post: \"" << post.content << "\"." << std::endl;

}

void followUser(User& follower, User& followed) {

std::cout << follower.name << " started following " << followed.name << "." << std::endl;

}

int main() {

User user1("Alice");

User user2("Bob");

Post post("This is a sample post.");

likePost(user1, post);

followUser(user1, user2);

return 0;

}



**5.**

#include<iostream>

using namespace std;

class MyClass{

private:

static int counter;

int count;

public:

MyClass(){

count ++;

counter++;

}

static int getCounter(){

return counter;

}

int getCount(){

return count;

}

};

int MyClass::counter =0;

int main(){

MyClass obj1;

MyClass obj2;

MyClass obj3;

cout<<"Number of objects created ="<<MyClass::getCounter()<<endl;

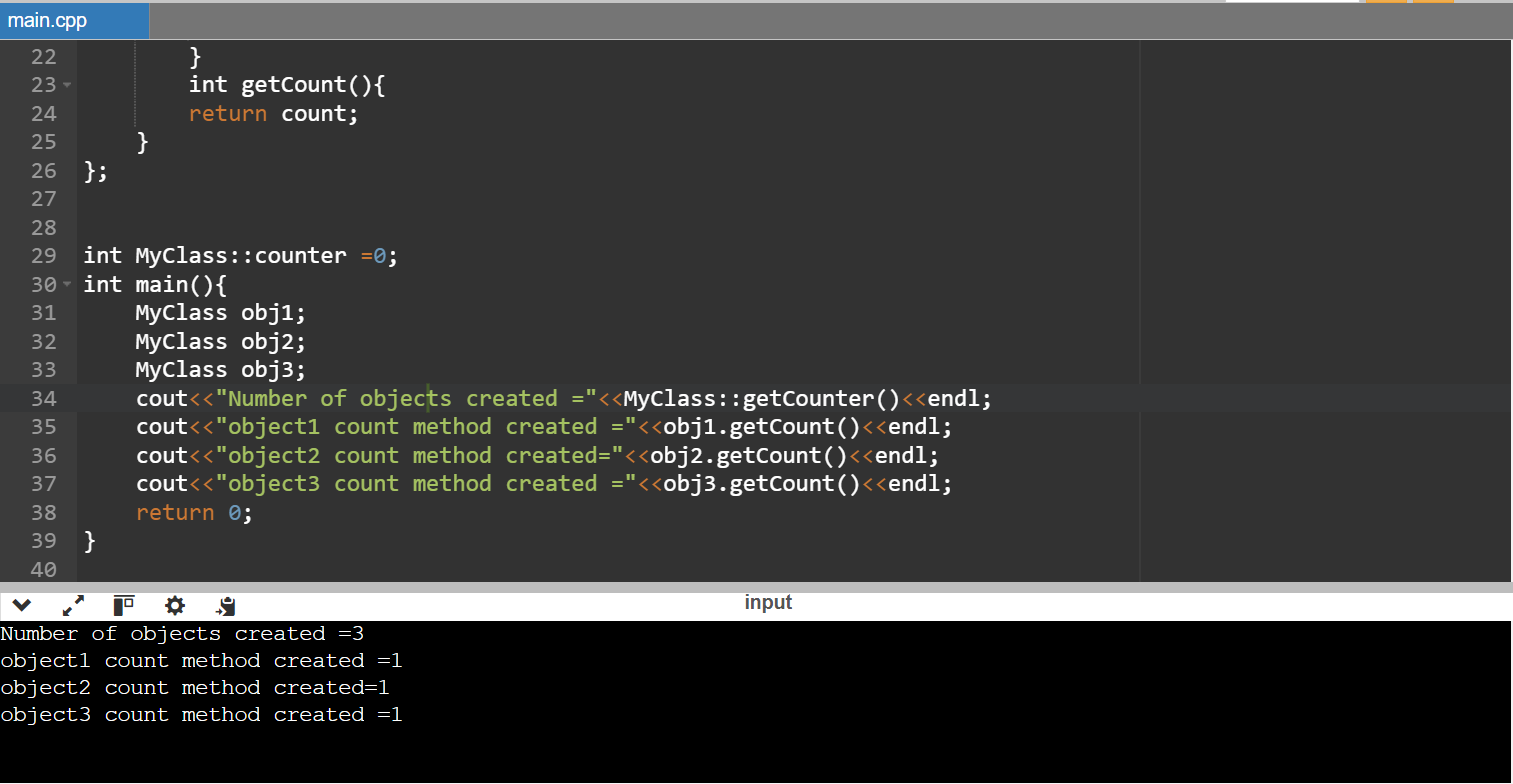
cout<<"object1 count method created ="<<obj1.getCount()<<endl;

cout<<"object2 count method created="<<obj2.getCount()<<endl;

cout<<"object3 count method created ="<<obj3.getCount()<<endl;

return 0;

}



**6.**

#include <iostream>

using namespace std;

class DistanceConverter {

public:

static double convertMilesToKm(double miles) {

return miles \* 1.60934;

}

static double convertKmToMiles(double kilometers) {

return kilometers / 1.60934;

}

};

int main() {

double distance;

char unit;

cout << "Enter distance: ";

cin >> distance;

cout << "Enter unit m for miles, k for kilometers: ";

cin >> unit;

if (unit == 'm' || unit == 'M') {

double km = DistanceConverter::convertMilesToKm(distance);

cout << distance << " miles is " << km << " kilometers." << endl;

} else if (unit == 'k' || unit == 'K') {

double miles = DistanceConverter::convertKmToMiles(distance);

cout << distance << " kilometers is " << miles << " miles." << endl;

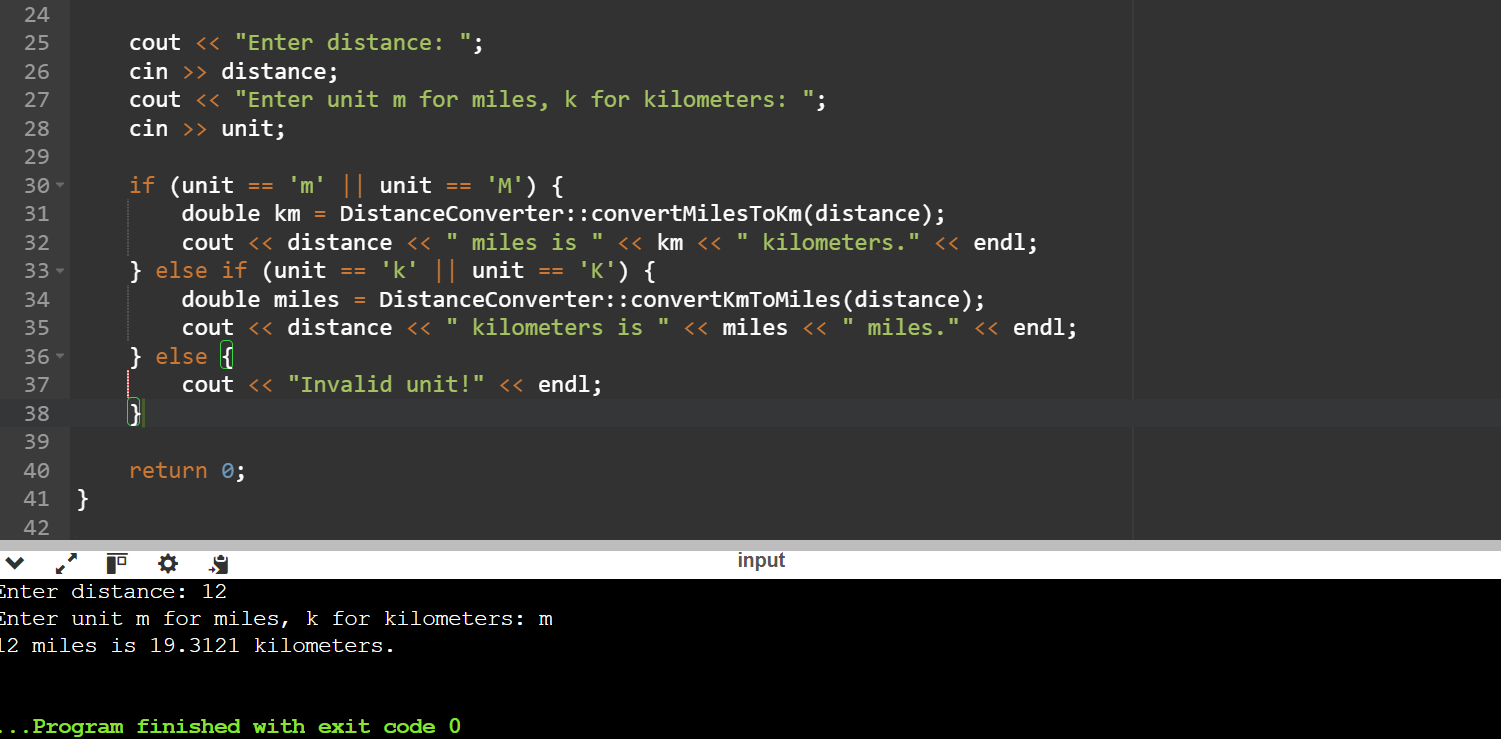
} else {

cout << "Invalid unit!" << endl;

}

return 0;

}



**7**.

#include <iostream>

using namespace std;

class MathUtil {

public:

static int add(int a, int b) {

return a + b;

}

static int subtract(int a, int b) {

return a - b;

}

static int multiply(int a, int b) {

return a \* b;

}

static double divide(int a, int b) {

if (b == 0) {

throw invalid\_argument("Error: Division by zero");

}

return static\_cast<double>(a) / b;

}

};

int main() {

int num1,num2;

char operation;

cout << "Enter first number: ";

cin >> num1;

cout << "Enter second number: ";

cin >> num2;

cout << "Enter operation (+, -, \*, /): ";

cin >> operation;

try {

switch (operation) {

case '+':

cout << "Result: "<<MathUtil::add(num1, num2)<<endl;

break;

case '-':

cout << "Result: "<<MathUtil::subtract(num1,num2)<<endl;

break;

case '\*':

cout << "Result: "<<MathUtil::multiply(num1,num2)<<endl;

break;

case '/':

cout << "Result: "<<MathUtil::divide(num1,num2)<<endl;

break;

default:

cout << "Invalid operation!" << endl;

break;

}

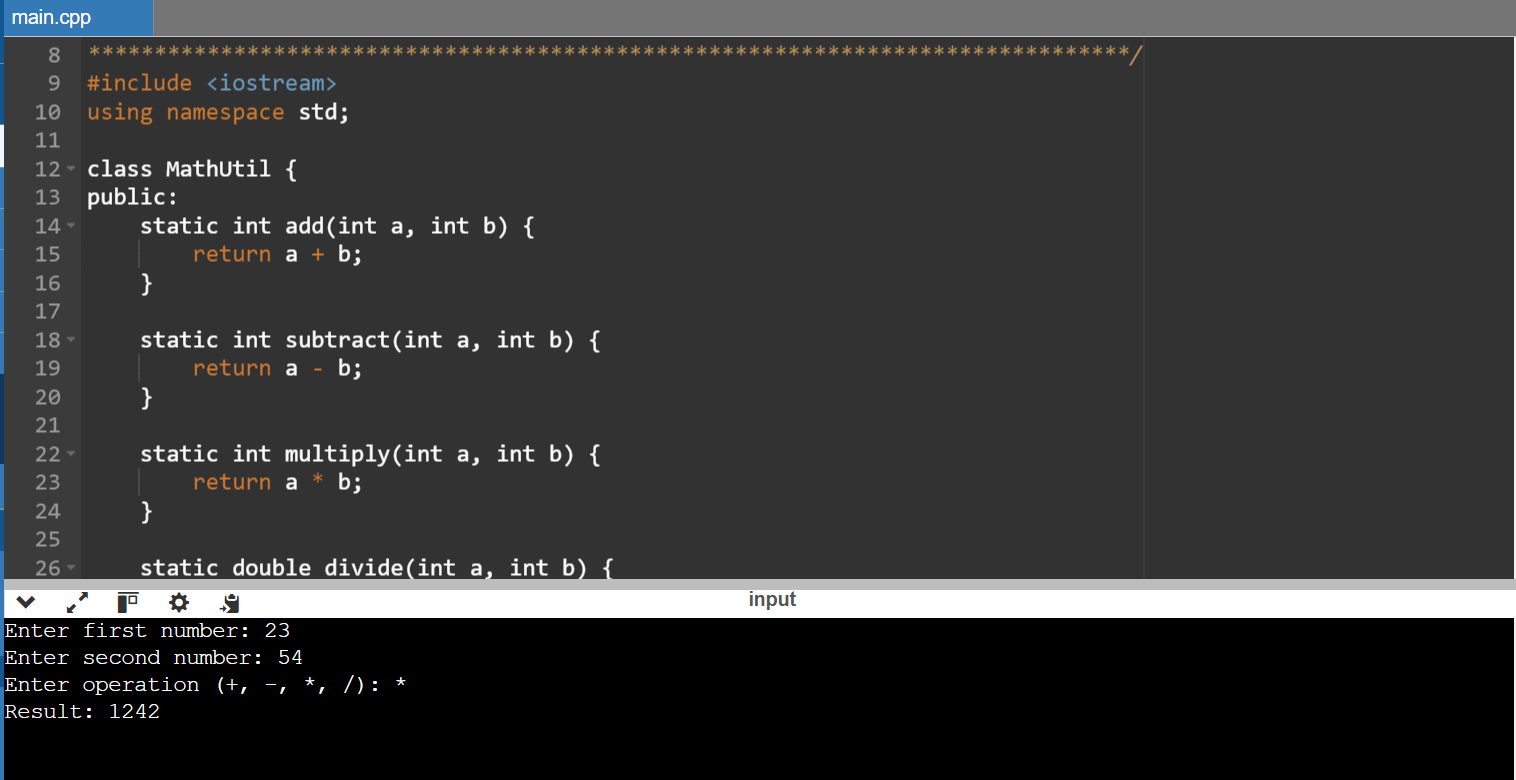
} catch (const invalid\_argument& e) {

cout << e.what() << endl;

}

return 0;

}



**8.**

#include <iostream>

using namespace std;

class CurrencyConverter {

public:

static double exchangeRate;

static double convertToEur(double amount) {

return amount \* exchangeRate;

}

static double convertFromEur(double amount) {

return amount / exchangeRate;

}

};

double CurrencyConverter::exchangeRate = 0.85;

int main() {

double amount;

char direction;

cout << "Enter amount: ";

cin >> amount;

cout << "Convert to (E)UR or from (E)UR to USD? (enter 'T' for to EUR, 'F' for from EUR): ";

cin >> direction;

if (direction == 'T' || direction == 't') {

double result = CurrencyConverter::convertToEur(amount);

cout << amount <<" USD is "<< result <<" EUR" << endl;

} else if (direction == 'F' || direction == 'f') {

double result = CurrencyConverter::convertFromEur(amount);

cout << amount << " EUR is " << result << " USD" << endl;

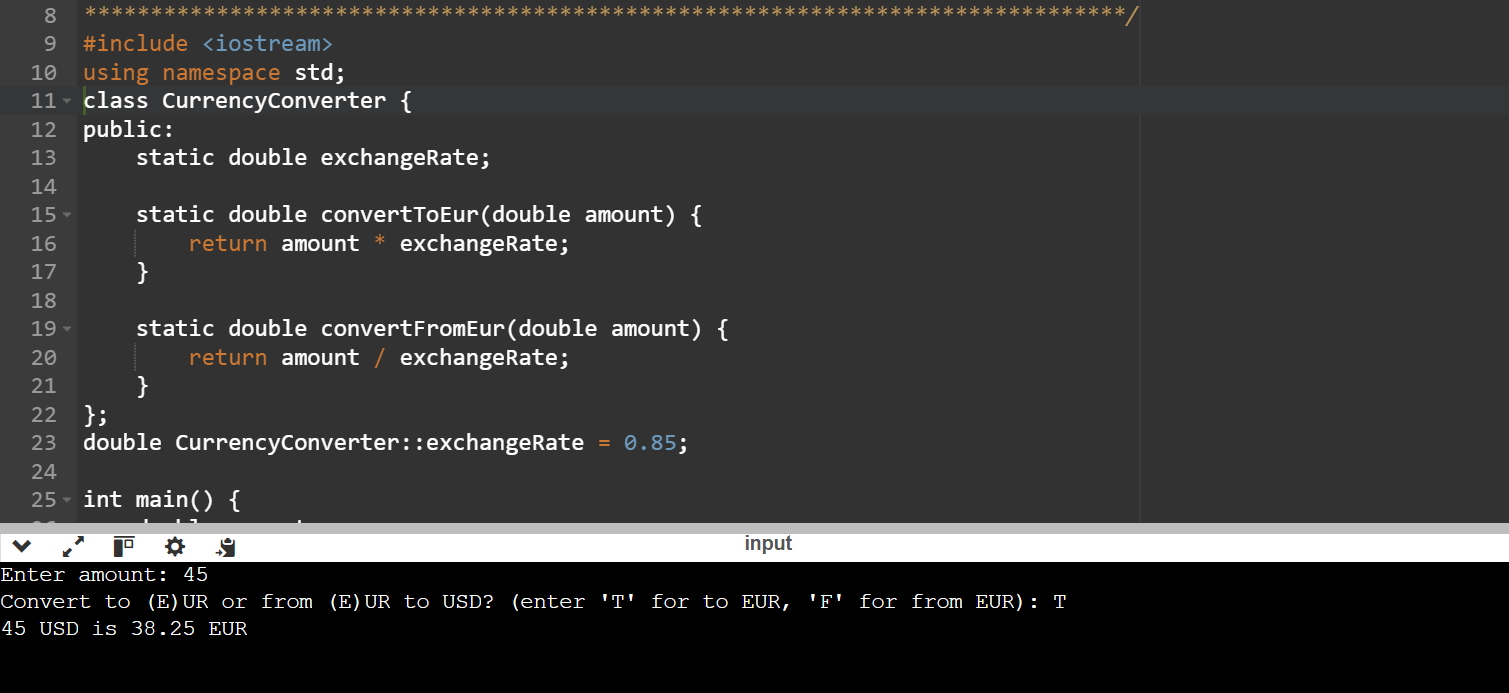
} else {

cout << "Invalid conversion direction!" << endl;

}

return 0;

}



**9.**

#include<iostream>

using namespace std;

template<class T>T add(T &a,T &b)

{

T result = a+b;

return result;

}

int main()

{

int i=2;

int j=3;

float m=2.3;

float n=1.2;

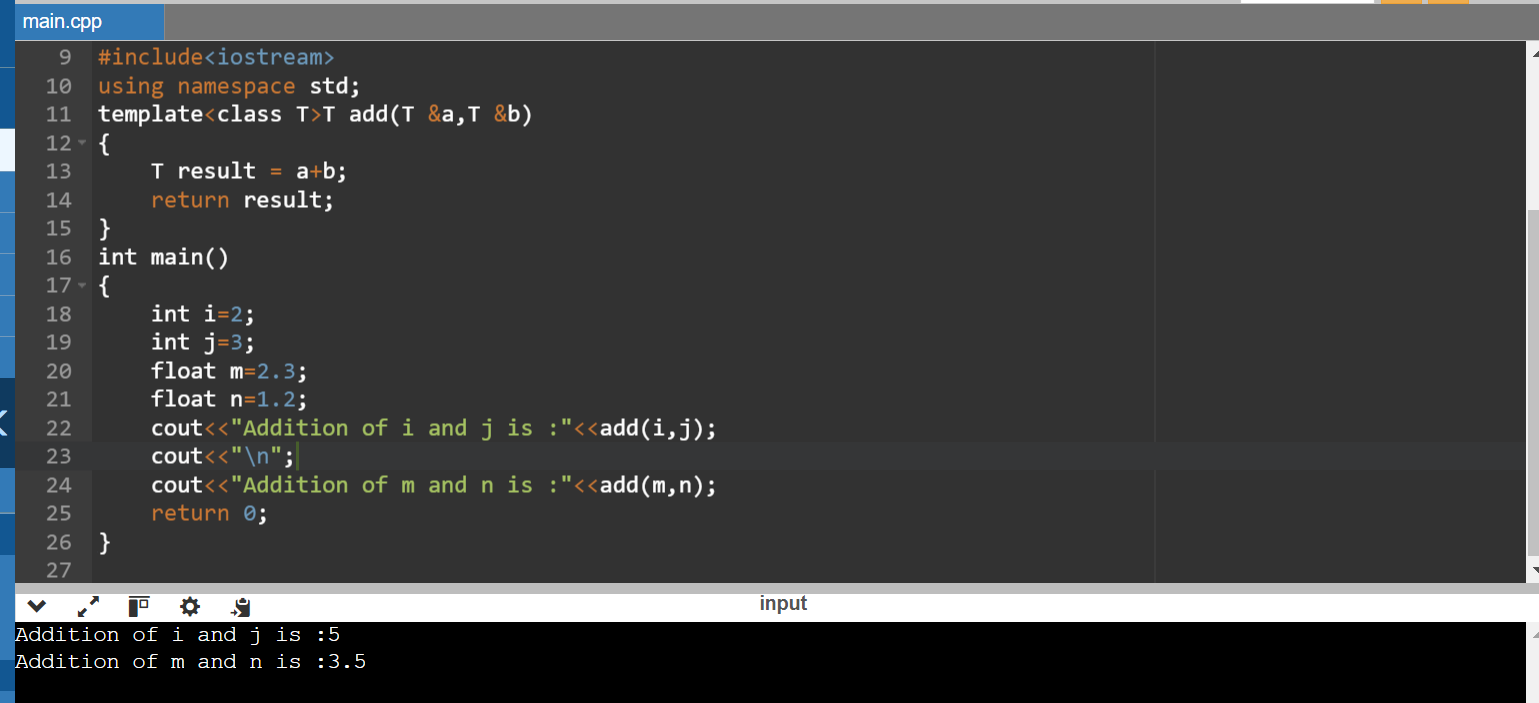
cout<<"Addition of i and j is :"<<add(i,j);

cout<<"\n";

cout<<"Addition of m and n is :"<<add(m,n);

return 0;

}



**10.**

#include<iostream>

using namespace std;

template<class X,class Y>void fun(X a,Y b)

{

cout<<"value of a is "<<a<<endl;

cout<<"Value of b is "<<b<<endl;

}

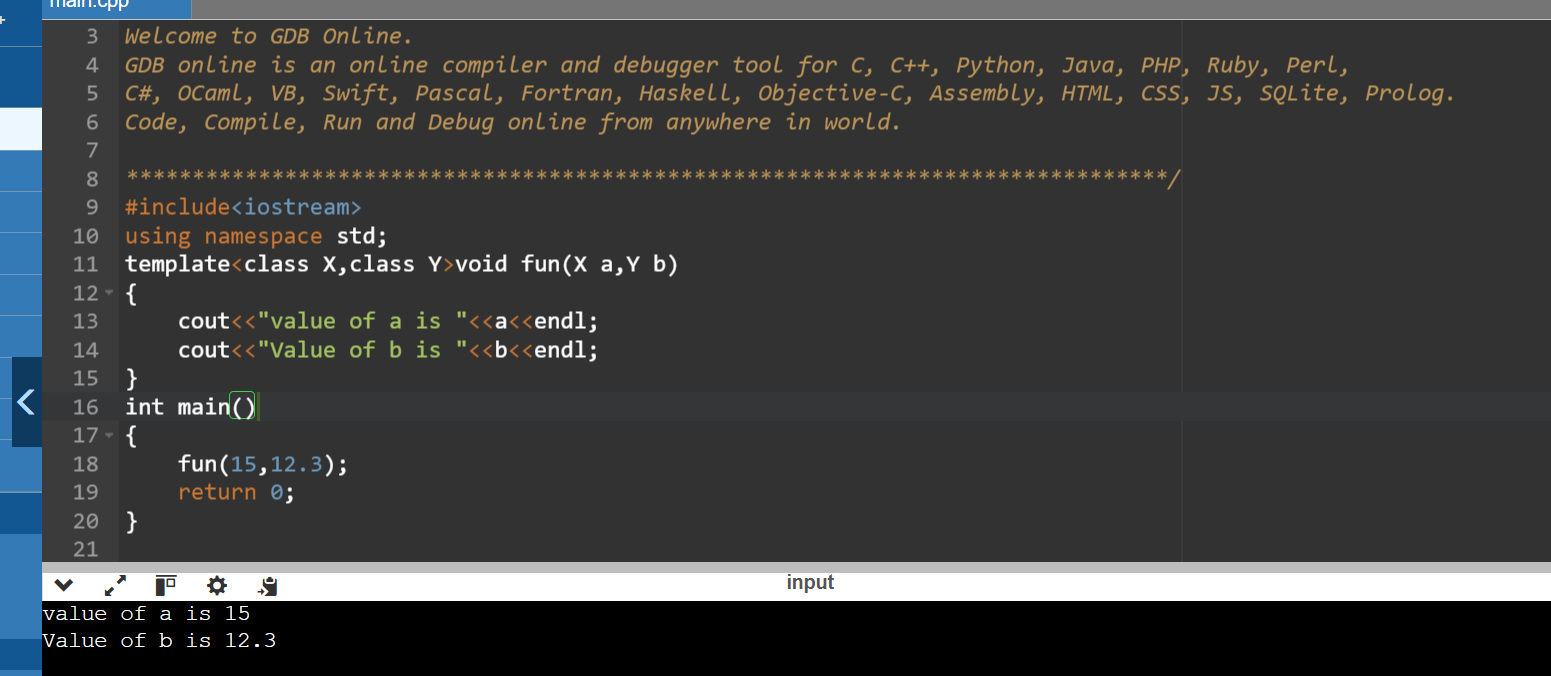
int main()

{

fun(15,12.3);

return 0;

}



**11.**

#include<iostream>

using namespace std;

template<class X>void fun(X a)

{

cout<<"value of a:"<<a<<endl;

}

template<class X,class Y>void fun(X b,Y c)

{

cout<<"value of b:"<<b<<endl;

cout<<"value of c:"<<c<<endl;

}

int main()

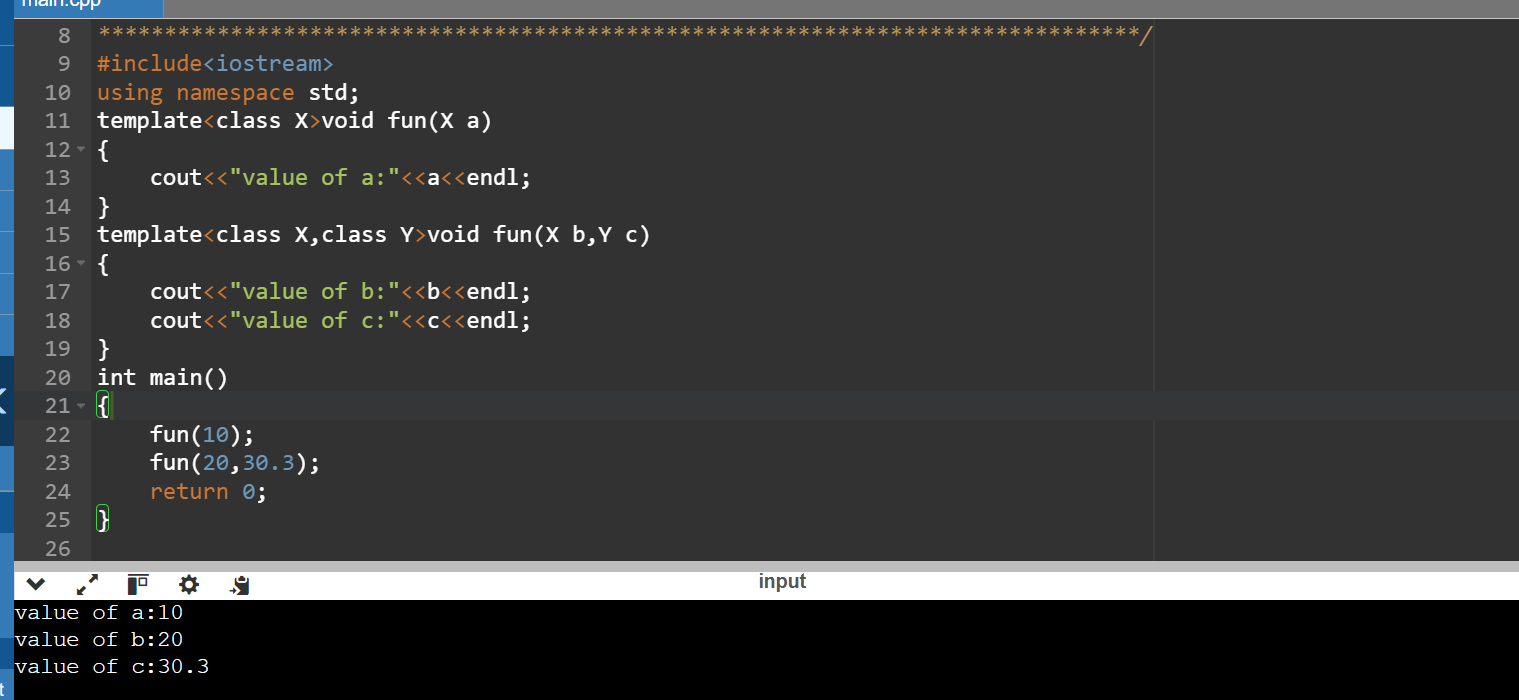
{

fun(10);

fun(20,30.3);

return 0;

}



**12.**

#include <iostream>

template <typename T>

bool compare(const T& a, const T& b) {

if (a > b) {

std::cout << "First argument is greater than the second argument." << std::endl;

return true;

} else if (a < b) {

std::cout << "First argument is less than the second argument." << std::endl;

return false;

} else {

std::cout << "Both arguments are equal." << std::endl;

return false;

}

}

class CustomType {

public:

int value;

CustomType(int v) : value(v) {}

// Overload the comparison operators

bool operator>(const CustomType& other) const {

return this->value > other.value;

}

bool operator<(const CustomType& other) const {

return this->value < other.value;

}

bool operator==(const CustomType& other) const {

return this->value == other.value;

}

};

int main() {

int a = 5, b = 10;

compare(a, b);

double x = 5.5, y = 2.2;

compare(x, y);

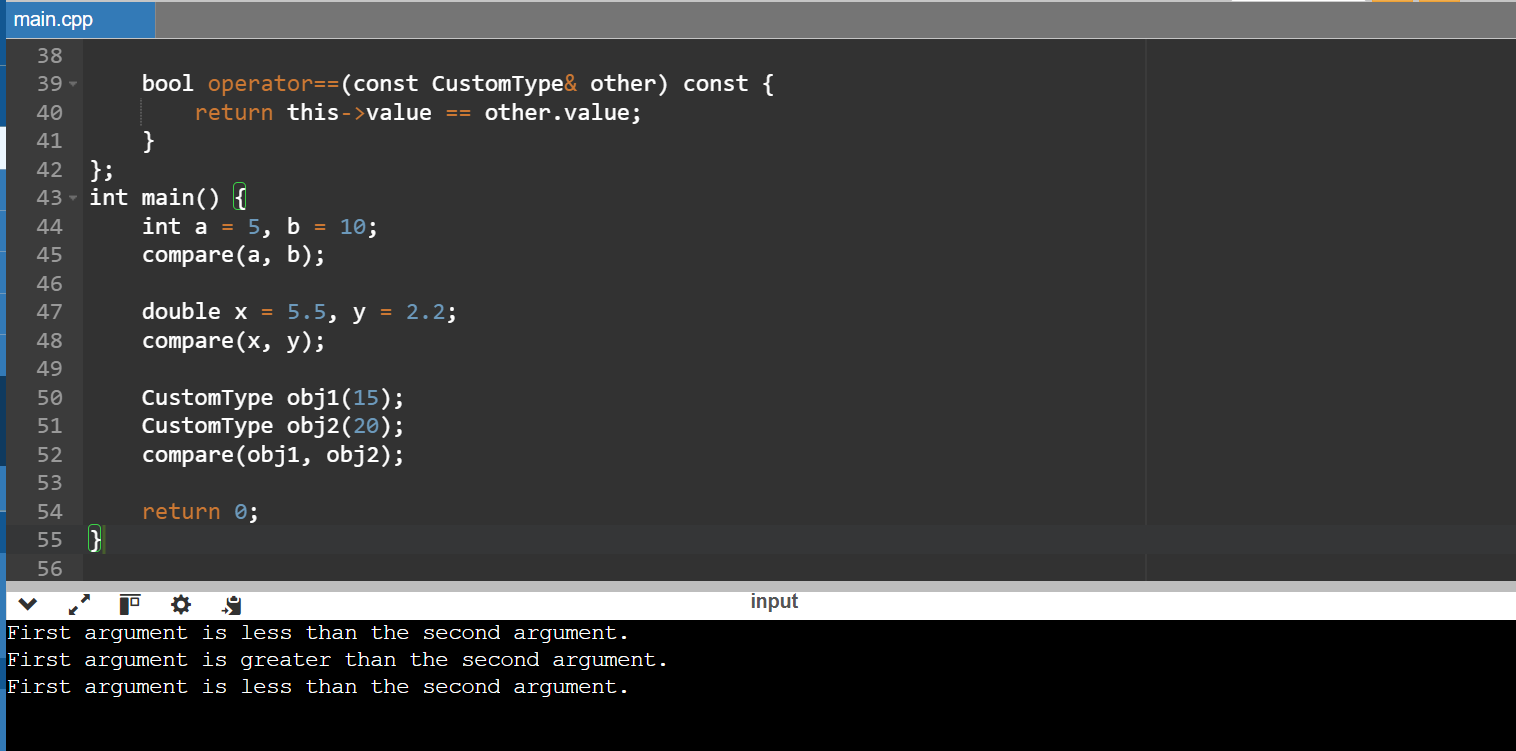
CustomType obj1(15);

CustomType obj2(20);

compare(obj1, obj2);

return 0;

}



13.

#include <iostream>

template <typename T>

void swap(T& a, T& b) {

T temp = a;

a = b;

b = temp;

}

int main() {

int x = 5, y = 10;

std::cout << "Before swap: x = " << x << ", y = " << y << std::endl;

swap(x, y);

std::cout << "After swap: x = " << x << ", y = " << y << std::endl;

double a = 1.1, b = 2.2;

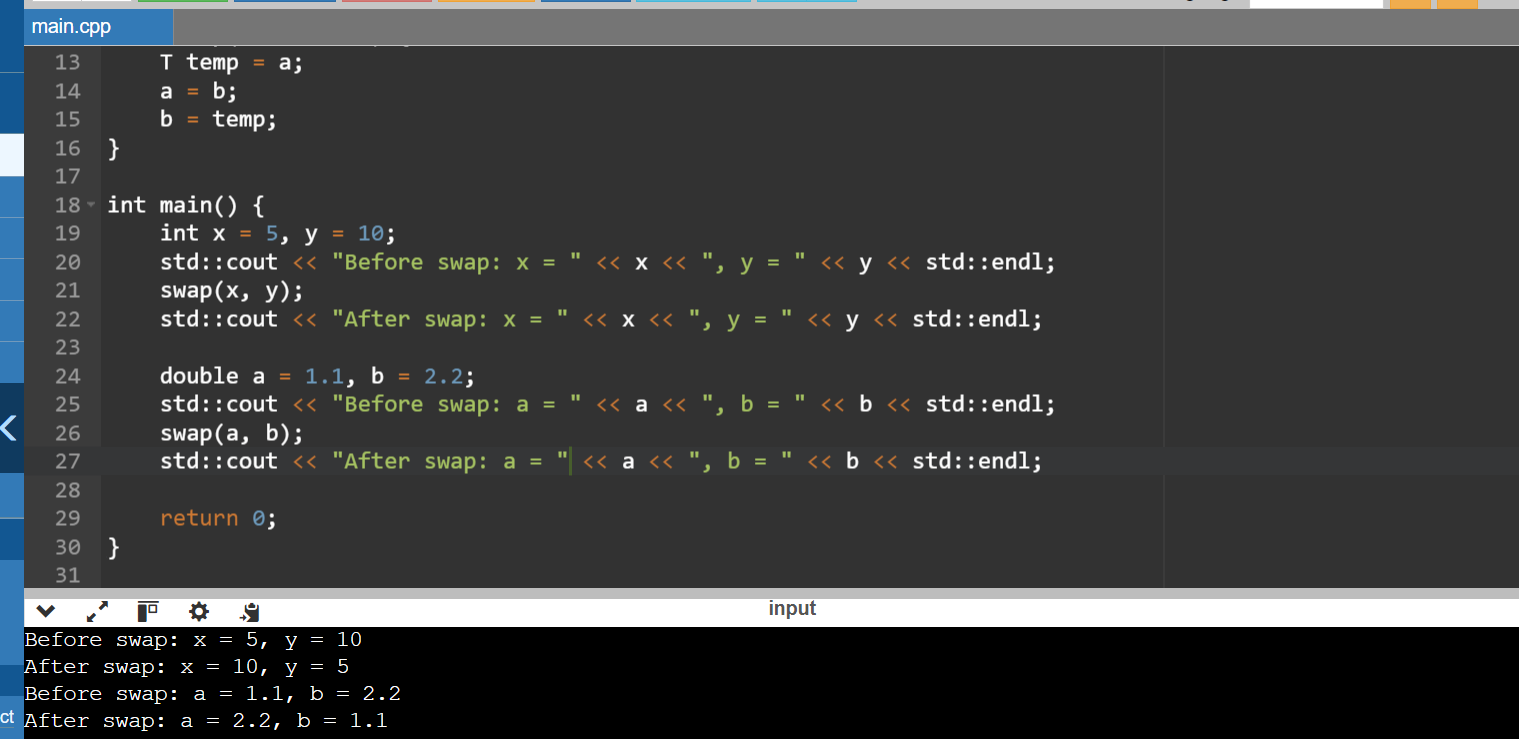
std::cout << "Before swap: a = " << a << ", b = " << b << std::endl;

swap(a, b);

std::cout << "After swap: a = " << a << ", b = " << b << std::endl;

return 0;

}



14.

#include <iostream>

template <typename T>

T findMin(T arr[], int size) {

if (size <= 0) throw std::invalid\_argument("Array size must be positive.");

T min = arr[0];

for (int i = 1; i < size; ++i) {

if (arr[i] < min) {

min = arr[i];

}

}

return min;

}

int main() {

int intArr[] = {5, 3, 8, 1, 2};

double doubleArr[] = {3.5, 2.1, 8.8, 1.1, 2.5};

int intSize = sizeof(intArr) / sizeof(intArr[0]);

double doubleSize = sizeof(doubleArr) / sizeof(doubleArr[0]);

std::cout << "Minimum int value: " << findMin(intArr, intSize) << std::endl;

std::cout << "Minimum double value: " << findMin(doubleArr, doubleSize) << std::endl;

return 0;

}

