ASSIGNMENT

1. WAP to swap two numbers using template function.

#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 swapping: x = " << x << ", y = " << y << std::endl;

swap(x, y);

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

double a = 3.14, b = 6.28;

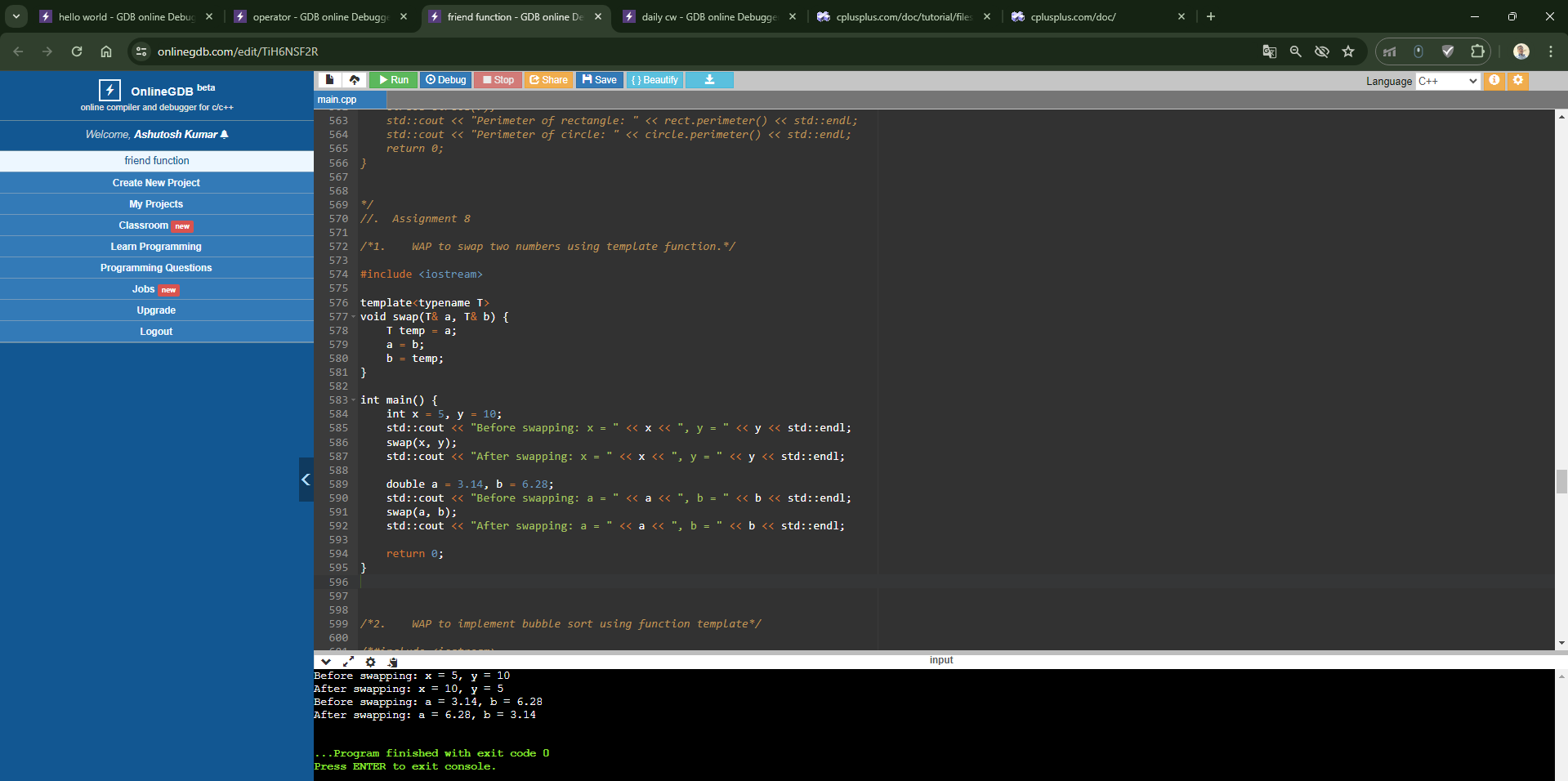
std::cout << "Before swapping: a = " << a << ", b = " << b << std::endl;

swap(a, b);

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

return 0;

}



1. WAP to implement bubble sort using function template

#include <iostream>

template<typename T>

void bubbleSort(T arr[], int n) {

for (int i = 0; i < n - 1; ++i) {

for (int j = 0; j < n - i - 1; ++j) {

if (arr[j] > arr[j + 1]) {

T temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

}

}

}

}

int main() {

int arr[] = {64, 34, 25, 12, 22, 11, 90};

int n = sizeof(arr) / sizeof(arr[0]);

bubbleSort(arr, n);

std::cout << "Sorted array: ";

for (int i = 0; i < n; ++i) {

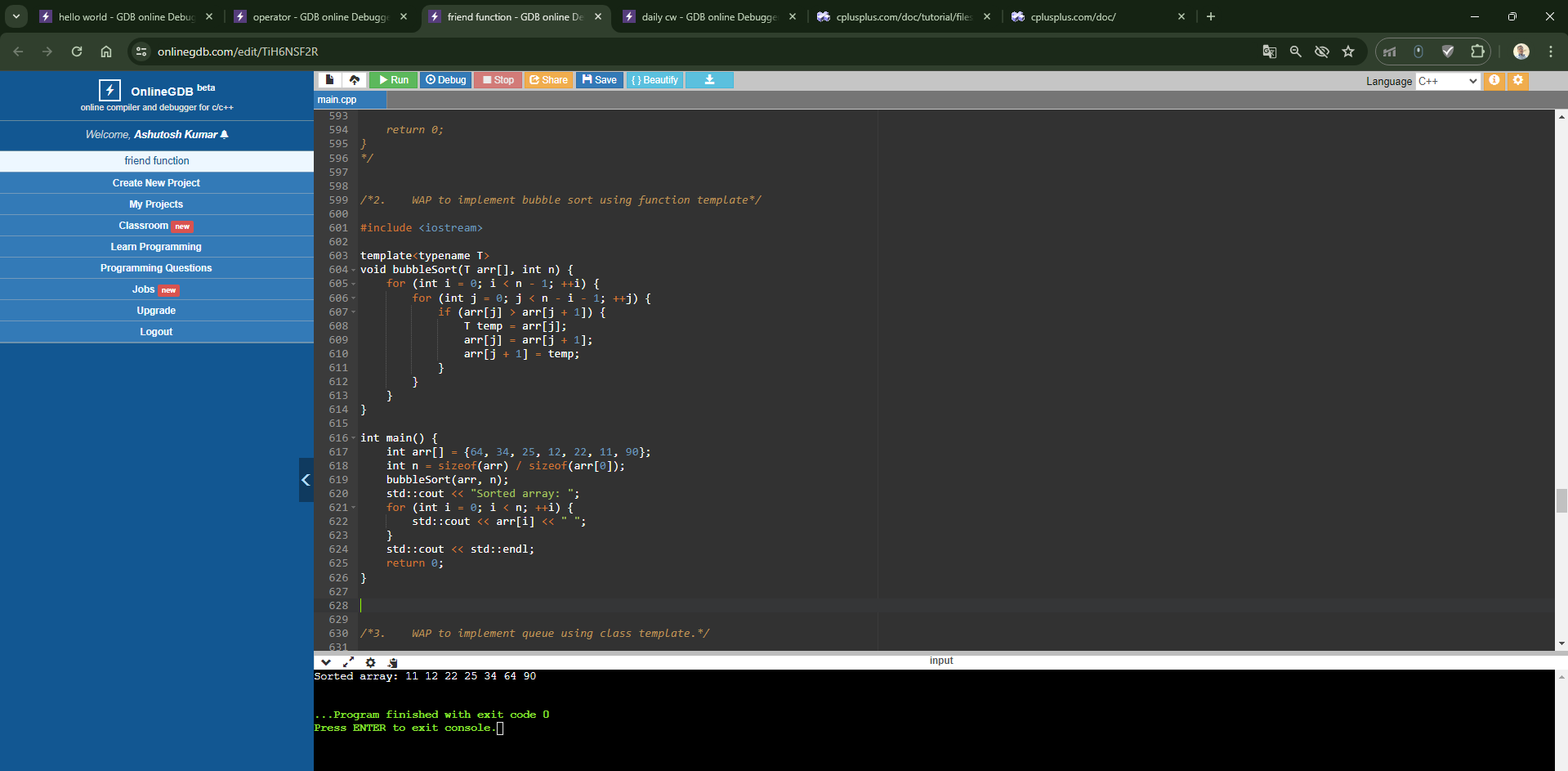
std::cout << arr[i] << " ";

}

std::cout << std::endl;

return 0;

}



1. WAP to implement queue using class template.

#include <iostream>

#include <queue>

template<typename T>

class Queue {

private:

std::queue<T> q;

public:

void enqueue(T val) {

q.push(val);

}

void dequeue() {

if (!q.empty()) {

q.pop();

}

}

T front() {

return q.front();

}

bool empty() {

return q.empty();

}

};

int main() {

Queue<int> q;

q.enqueue(1);

q.enqueue(2);

q.enqueue(3);

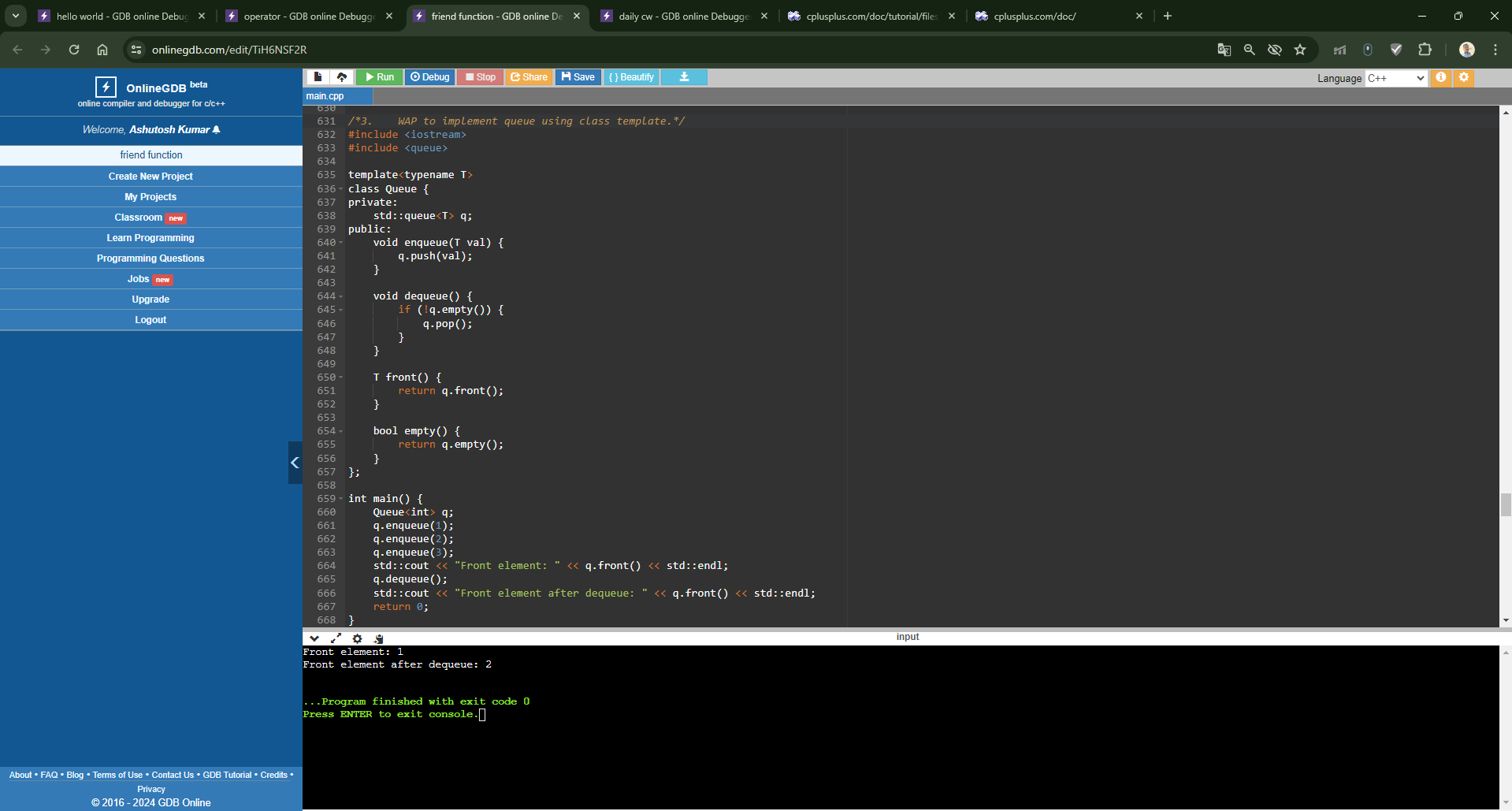
std::cout << "Front element: " << q.front() << std::endl;

q.dequeue();

std::cout << "Front element after dequeue: " << q.front() << std::endl;

return 0;

}



### WAP Simple calculator using Class template

#include <iostream>

template<typename T>

class Calculator {

public:

T add(T a, T b) {

return a + b;

}

T subtract(T a, T b) {

return a - b;

}

T multiply(T a, T b) {

return a \* b;

}

T divide(T a, T b) {

if (b == 0) {

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

}

return a / b;

}

};

int main() {

Calculator<int> calc;

std::cout << "Addition: " << calc.add(10, 5) << std::endl;

std::cout << "Subtraction: " << calc.subtract(10, 5) << std::endl;

std::cout << "Multiplication: " << calc.multiply(10, 5) << std::endl;

try {

std::cout << "Division: " << calc.divide(10, 5) << std::endl;

std::cout << "Division: " << calc.divide(10, 0) << std::endl; // Throws exception

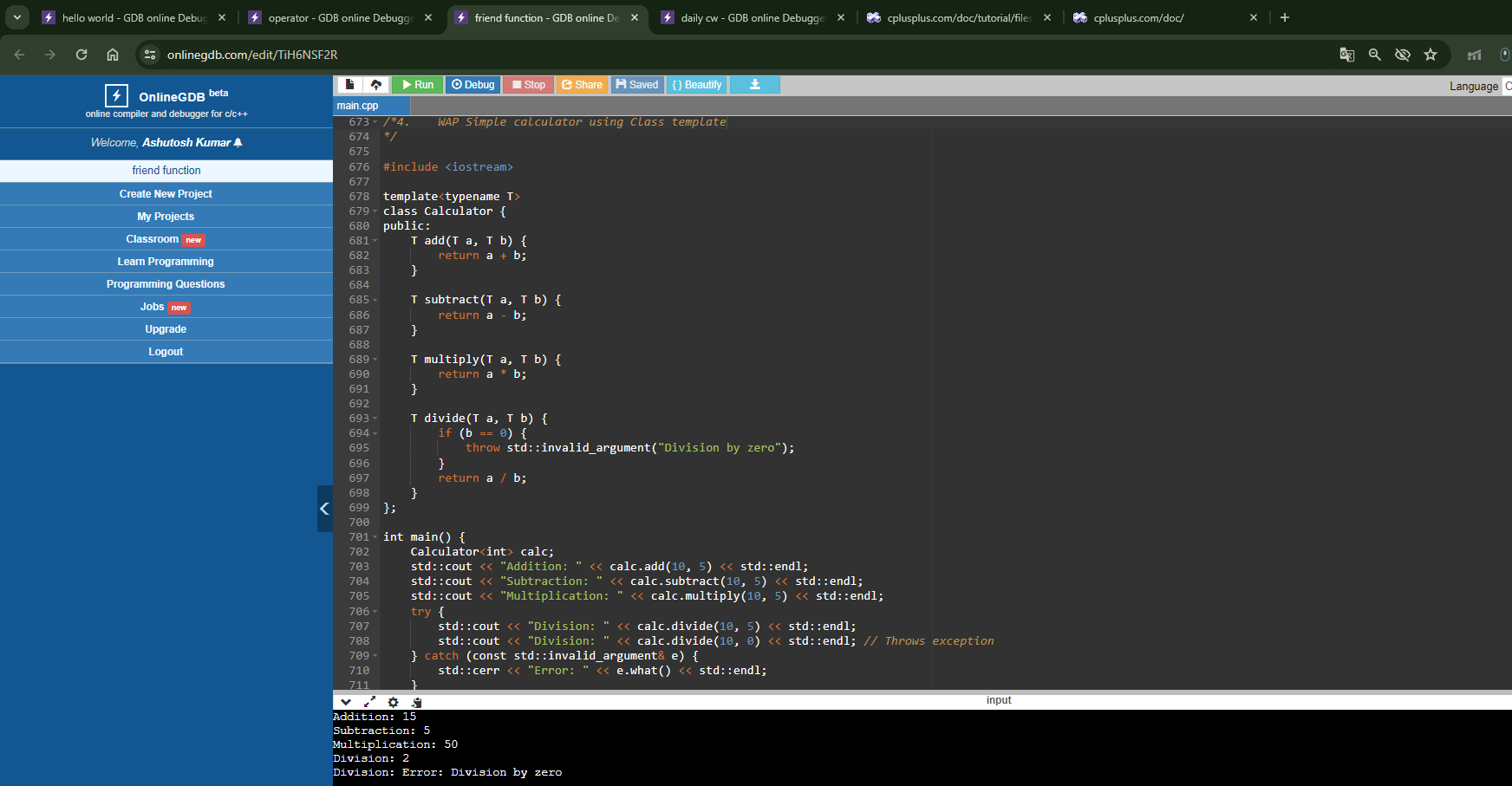
} catch (const std::invalid\_argument& e) {

std::cerr << "Error: " << e.what() << std::endl;

}

return 0;

}



1. WAP Exception Handling for Divide by zero condition for the division of two numbers.

#include <iostream>

#include <stdexcept>

int main() {

try {

int a = 10, b = 0;

if (b == 0) {

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

}

std::cout << "Result: " << a / b << std::endl;

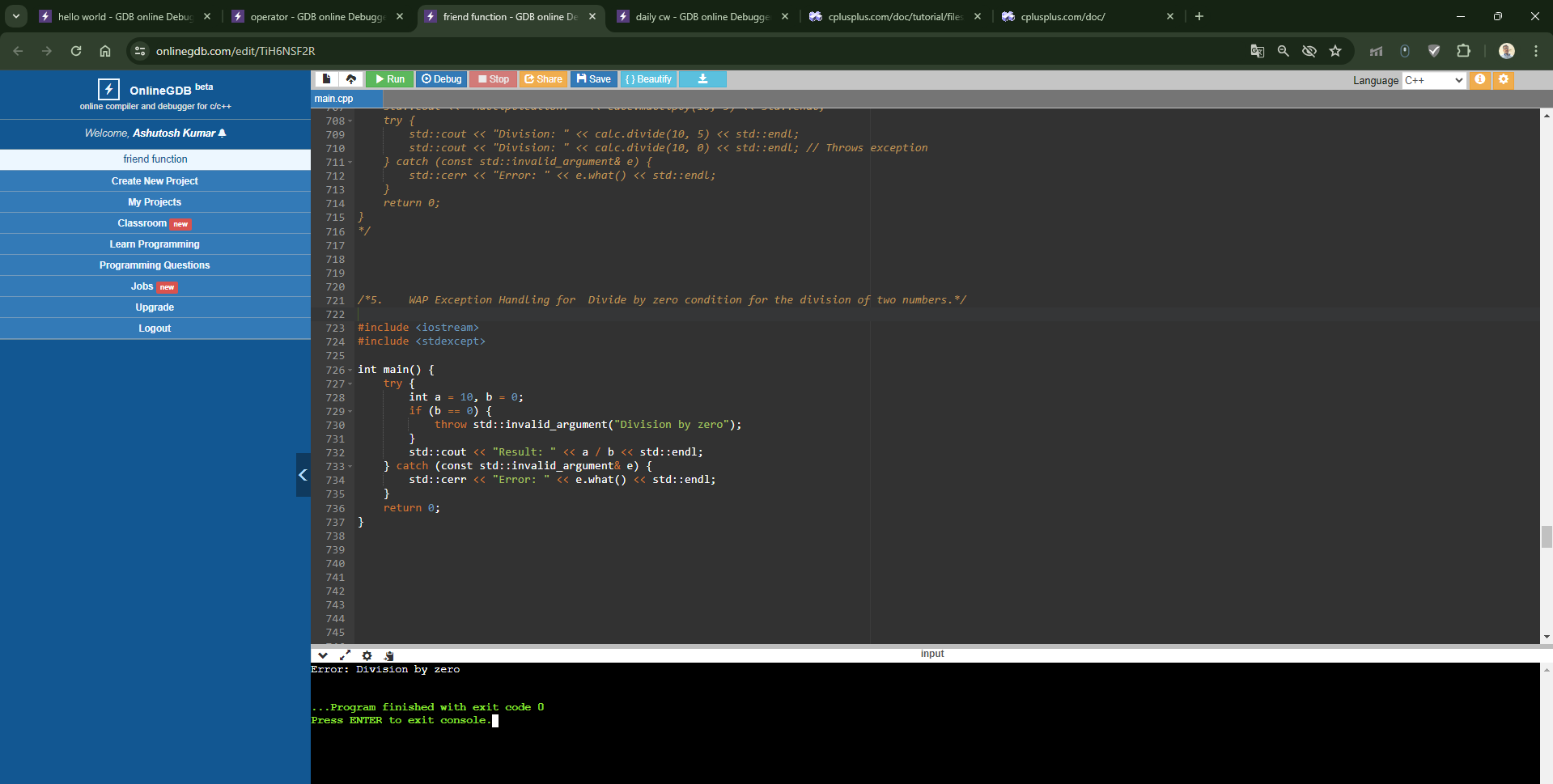
} catch (const std::invalid\_argument& e) {

std::cerr << "Error: " << e.what() << std::endl;

}

return 0;

}



1. WAP for multiple catch statements to handle various types of exceptions for 3 different values of a variable w.r.t int,char,double throws.

#include <iostream>

#include <stdexcept>

int main() {

try {

// Division by zero

int a = 10, b = 0;

std::cout << "Result: " << a / b << std::endl;

} catch (const std::invalid\_argument& e) {

std::cerr << "Error: " << e.what() << std::endl;

} catch (const std::out\_of\_range& e) {

std::cerr << "Error: " << e.what() << std::endl;

} catch (...) {

std::cerr << "Unknown error occurred" << std::endl;

}

try {

// String to int conversion

std::string str = "abc";

int x = std::stoi(str);

std::cout << "String to int: " << x << std::endl;

} catch (const std::invalid\_argument& e) {

std::cerr << "Error: " << e.what() << std::endl;

} catch (const std::out\_of\_range& e) {

std::cerr << "Error: " << e.what() << std::endl;

} catch (...) {

std::cerr << "Unknown error occurred" << std::endl;

}

try {

// Invalid conversion

char ch = 'A';

double d = static\_cast<double>(ch);

std::cout << "Char to double: " << d << std::endl;

} catch (const std::invalid\_argument& e) {

std::cerr << "Error: " << e.what() << std::endl;

} catch (const std::out\_of\_range& e) {

std::cerr << "Error: " << e.what() << std::endl;

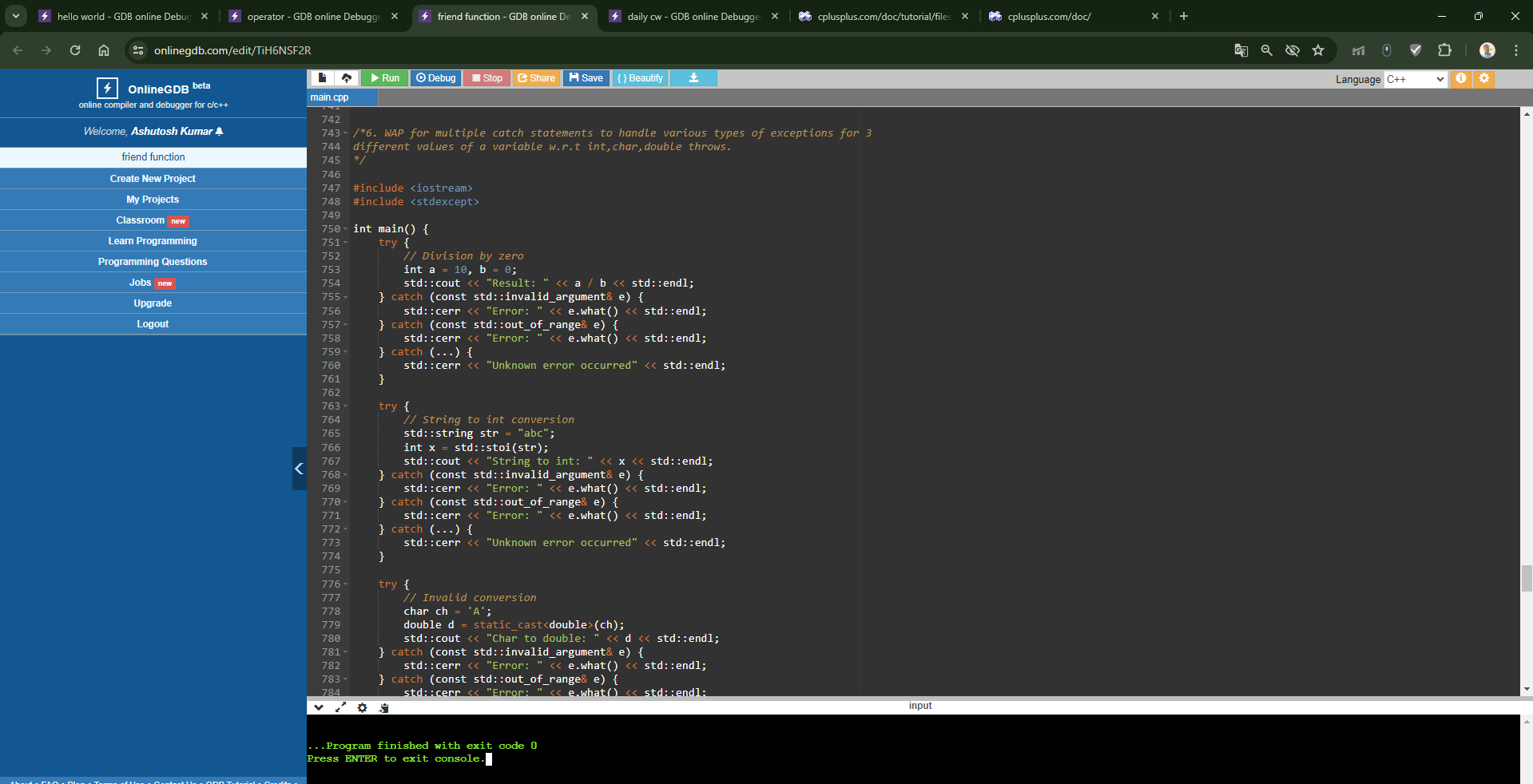
} catch (...) {

std::cerr << "Unknown error occurred" << std::endl;

}

return 0;

}



### Wap to read from a file and print on console.

#include <iostream>

#include <fstream>

#include <string>

int main() {

std::ifstream fin("input.txt");

if (fin.is\_open()) {

std::string line;

while (getline(fin, line)) {

std::cout << line << std::endl;

}

fin.close();

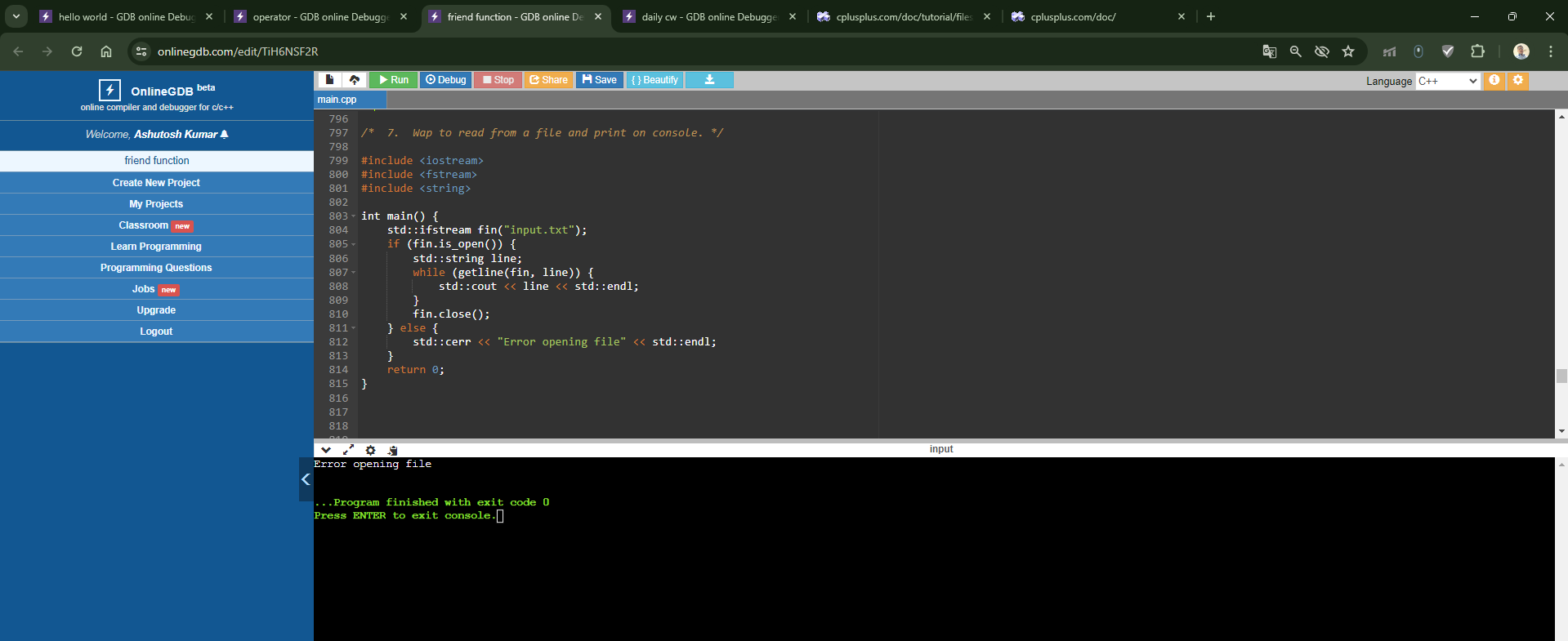
} else {

std::cerr << "Error opening file" << std::endl;

}

return 0;

}



### WAP to create a file and write data on it.

#include <iostream>

#include <fstream>

int main() {

std::ofstream fout("output.txt");

if (fout.is\_open()) {

fout << "Hello, world!" << std::endl;

fout << "This is a test file." << std::endl;

fout.close();

std::cout << "File created and data written successfully" << std::endl;

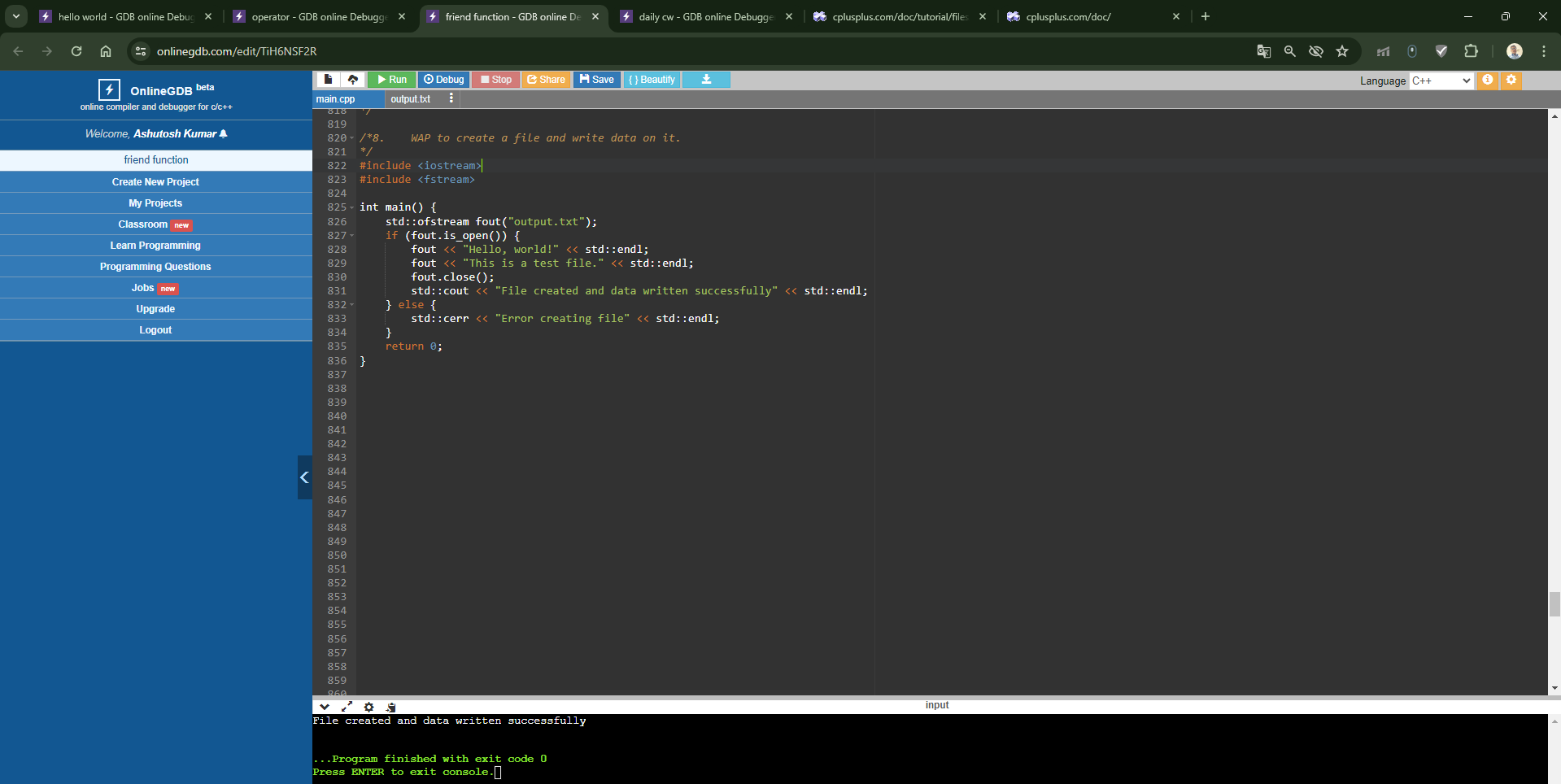
} else {

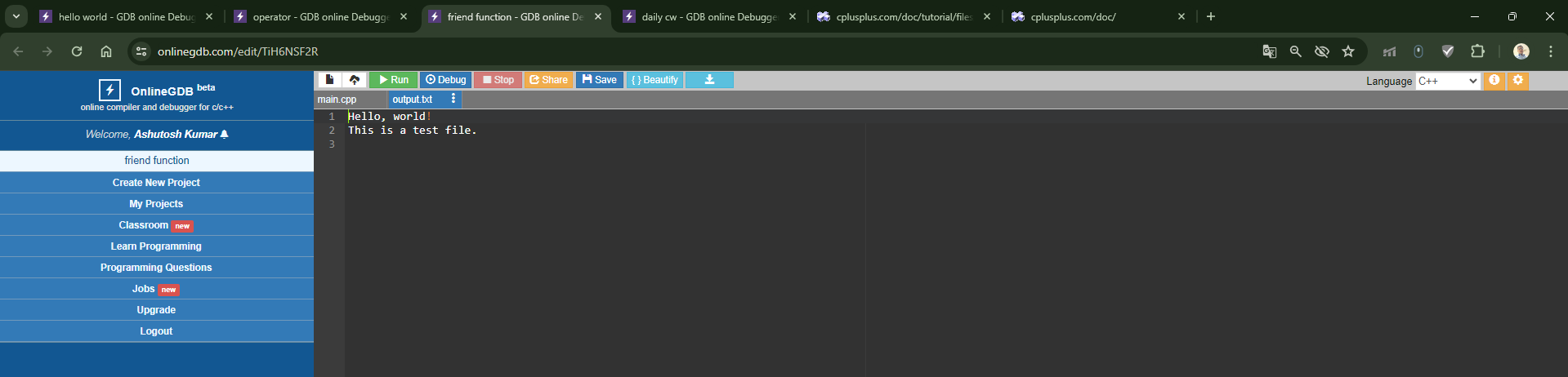
std::cerr << "Error creating file" << std::endl;

}

return 0;

}





### WAP to copy the data of one to other file;

#include <iostream>

#include <fstream>

int main() {

std::ifstream fin("input.txt");

std::ofstream fout("output.txt");

if (fin.is\_open() && fout.is\_open()) {

std::string line;

while (getline(fin, line)) {

fout << line << std::endl;

}

fin.close();

fout.close();

std::cout << "File copied successfully" << std::endl;

} else {

std::cerr << "Error opening files" << std::endl;

}

return 0;

}

