**Assignment 13**

1. Write a C++ program to implement Bubble Sort using templates in C++

**Code:**

#include <iostream>

using namespace std;

template <class T> void bubbleSort(T a[], int n)

{

    for (int i = 0; i < n - 1; i++)

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

            if (a[j] < a[j - 1])

                swap(a[j], a[j - 1]);

}

int main()

{

    int a[5] = { 100, 500, 300, 400, 200 };

    int n = sizeof(a) / sizeof(a[0]);

    bubbleSort<int>(a, n);

    cout << " Sorted array : ";

    for (int i = 0; i < n; i++)

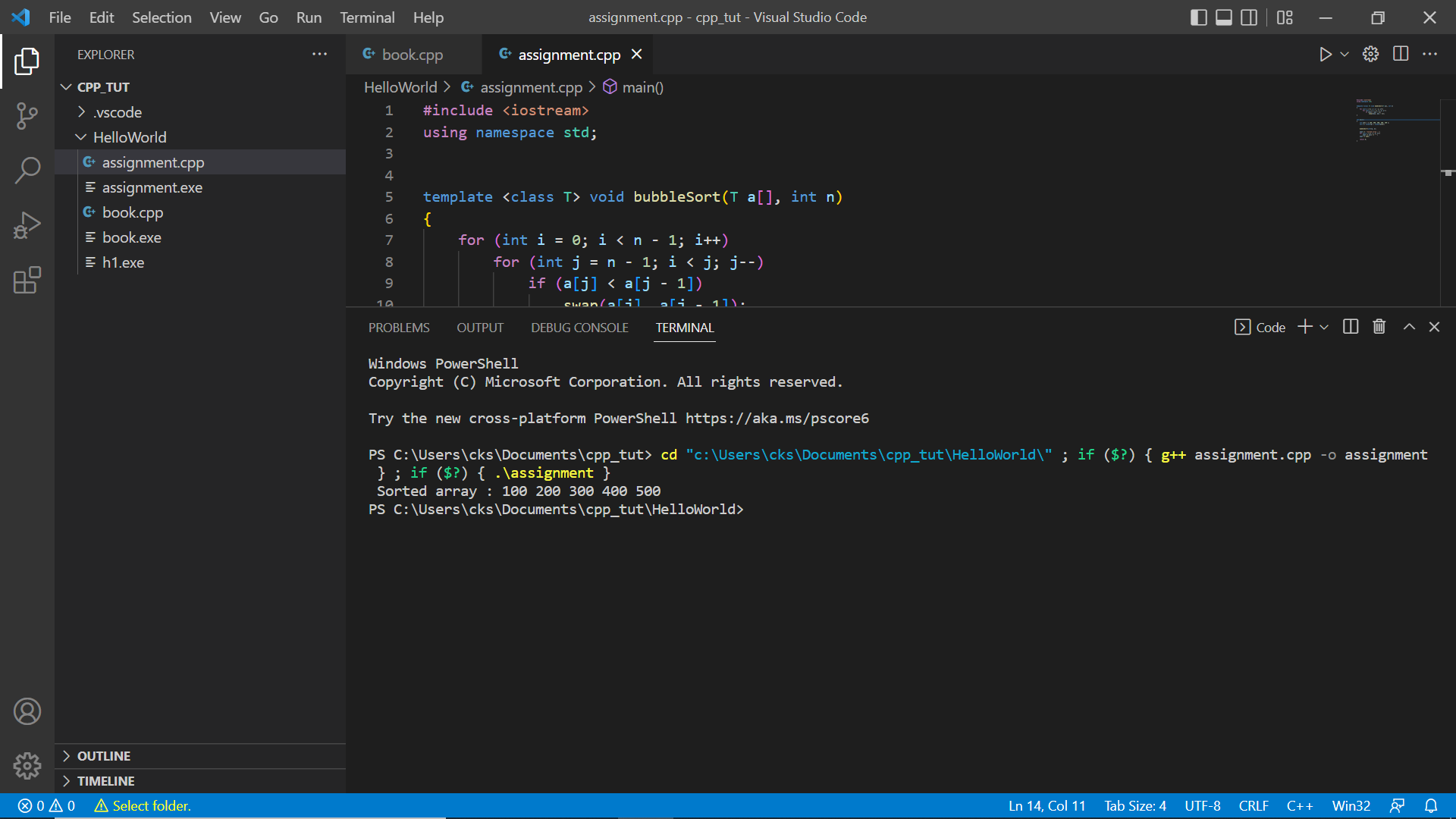
        cout << a[i] << " ";

    cout << endl;

    return 0;

}

**Output:**



**2 .** Write C++ Program to find Sum of Array using function template.

Code:

#include <iostream>

using namespace std;

template<class T> T sum(T a[], int length)

{

        T ret = a[0];

        for (int i = 1; i < length; i ++)

                ret += a[i];

        return ret;

}

int main(void) {

        int int\_data[5];

        float float\_data[5];

        int i = 0;

        cout << "\nInput 5 integers :: \n" << endl;

        for (; i < 5; i++)

                cin >> int\_data[i];

        cout << "\nSum of above is :: " << sum(int\_data, 5) << endl;

        cout << "\nInput 5 floats :: \n" << endl;

        for (i = 0; i < 5; i ++)

                cin >> float\_data[i];

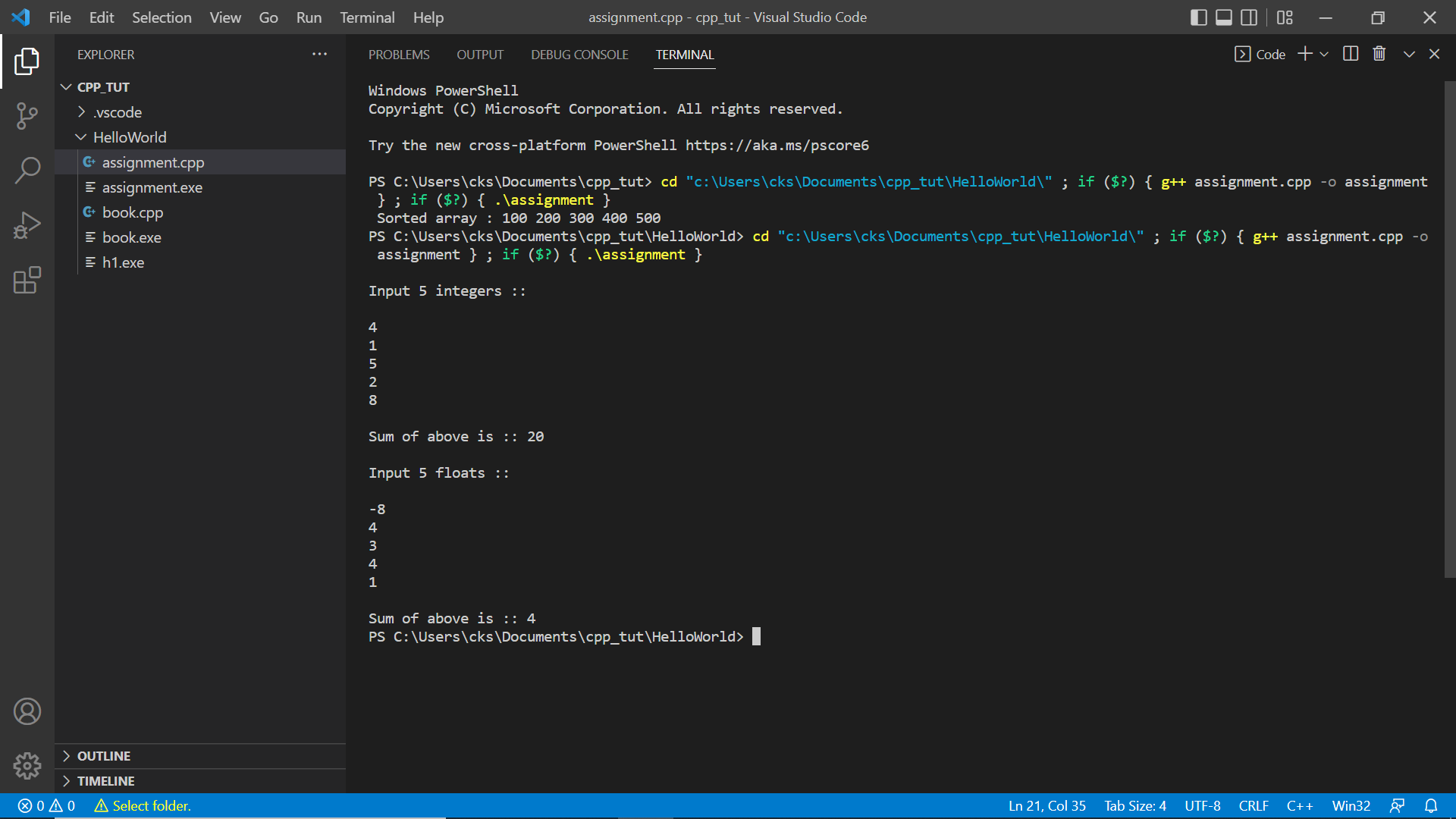
        cout << "\nSum of above is :: " << sum(float\_data, 5) << endl;

        cin.get();

        return 0;

}

**Output:**



1. Write C++ Program to Swap two input data using function template.

Code:

#include <iostream>

using namespace std;

template <class T>

int swap\_numbers(T& x, T& y)

{

    T t;

    t = x;

    x = y;

    y = t;

    return 0;

}

int main()

{

    int a, b;

    a = 7, b =11;

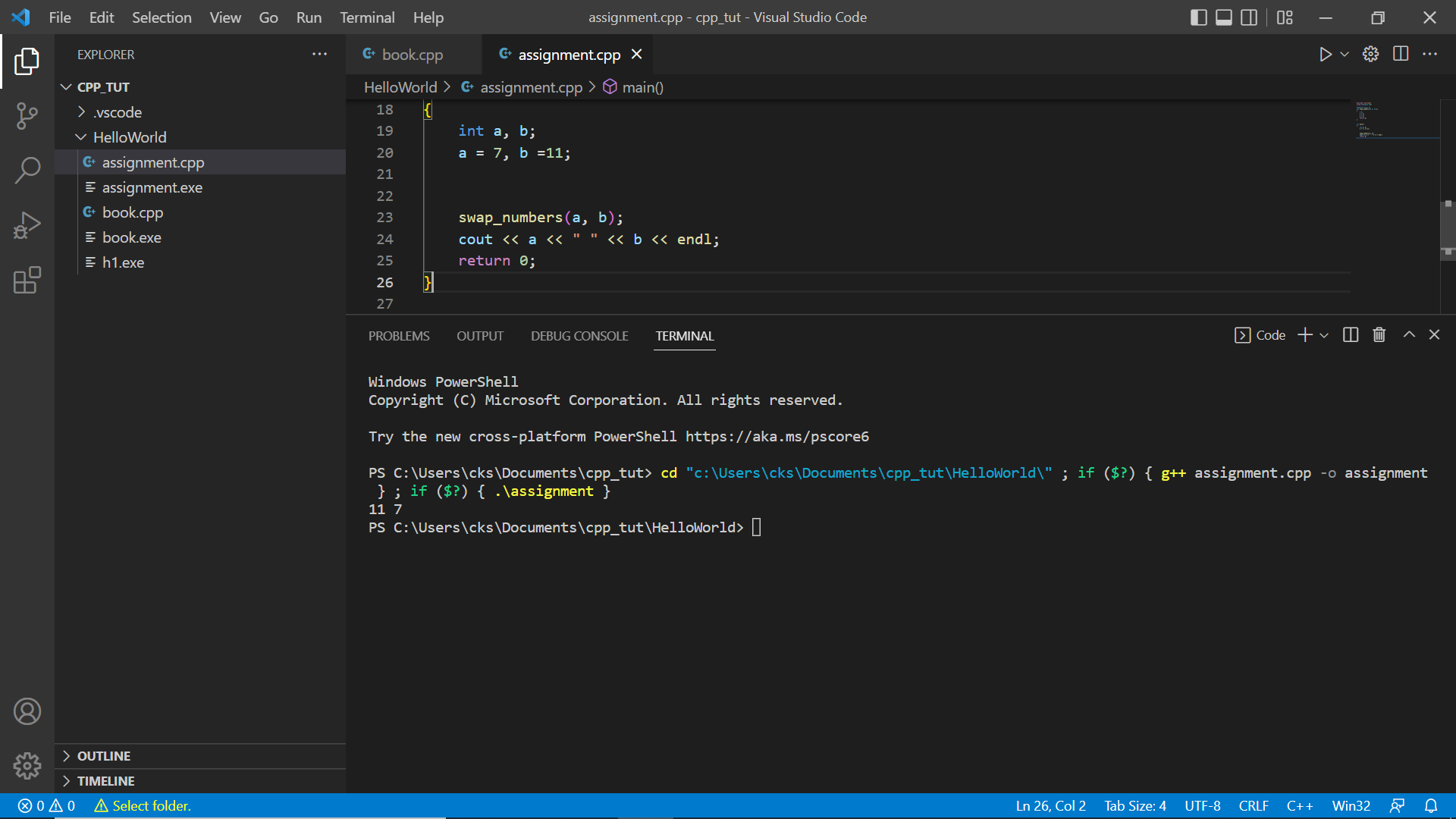
    swap\_numbers(a, b);

    cout << a << " " << b << endl;

    return 0;

}

**Output:**



#include <iostream>

using namespace std;

template <class T>

void swap\_(T\* x, T\* y)

{

    T temp = \*x;

    \*x = \*y;

    \*y = temp;

}

template <class T, int size>

void bubble\_sort(T arr[])

{

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

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

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

                swap\_(&arr[j], &arr[j + 1]);

            }

        }

    }

}

template <class T, int size>

void printArray(T arr[])

{

    int i;

    for (i = 0; i < size - 1; i++) {

        cout << arr[i] << ", ";

    }

    cout << arr[size - 1] << endl;

}

int main()

{

    float arr[] = { 3.2,4.3,4.5,6.2,7.2,1.1};

    const int size\_arr = sizeof(arr) / sizeof(arr[0]);

    bubble\_sort<float, size\_arr>(arr);

    cout << "Sorted Array is: ";

    printArray<float, size\_arr>(arr);

    return 0;

}

4. Write C++ Program to demonstrate an Example of Non-type parameters for templates.

Code:

#include <iostream>

using namespace std;

template <class T>

void swap\_(T\* x, T\* y)

{

    T temp = \*x;

    \*x = \*y;

    \*y = temp;

}

template <class T, int size>

void bubble\_sort(T arr[])

{

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

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

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

                swap\_(&arr[j], &arr[j + 1]);

            }

        }

    }

}

template <class T, int size>

void printArray(T arr[])

{

    int i;

    for (i = 0; i < size - 1; i++) {

        cout << arr[i] << ", ";

    }

    cout << arr[size - 1] << endl;

}

int main()

{

    float arr[] = { 3.2,4.3,4.5,6.2,7.2,1.1};

    const int size\_arr = sizeof(arr) / sizeof(arr[0]);

    bubble\_sort<float, size\_arr>(arr);

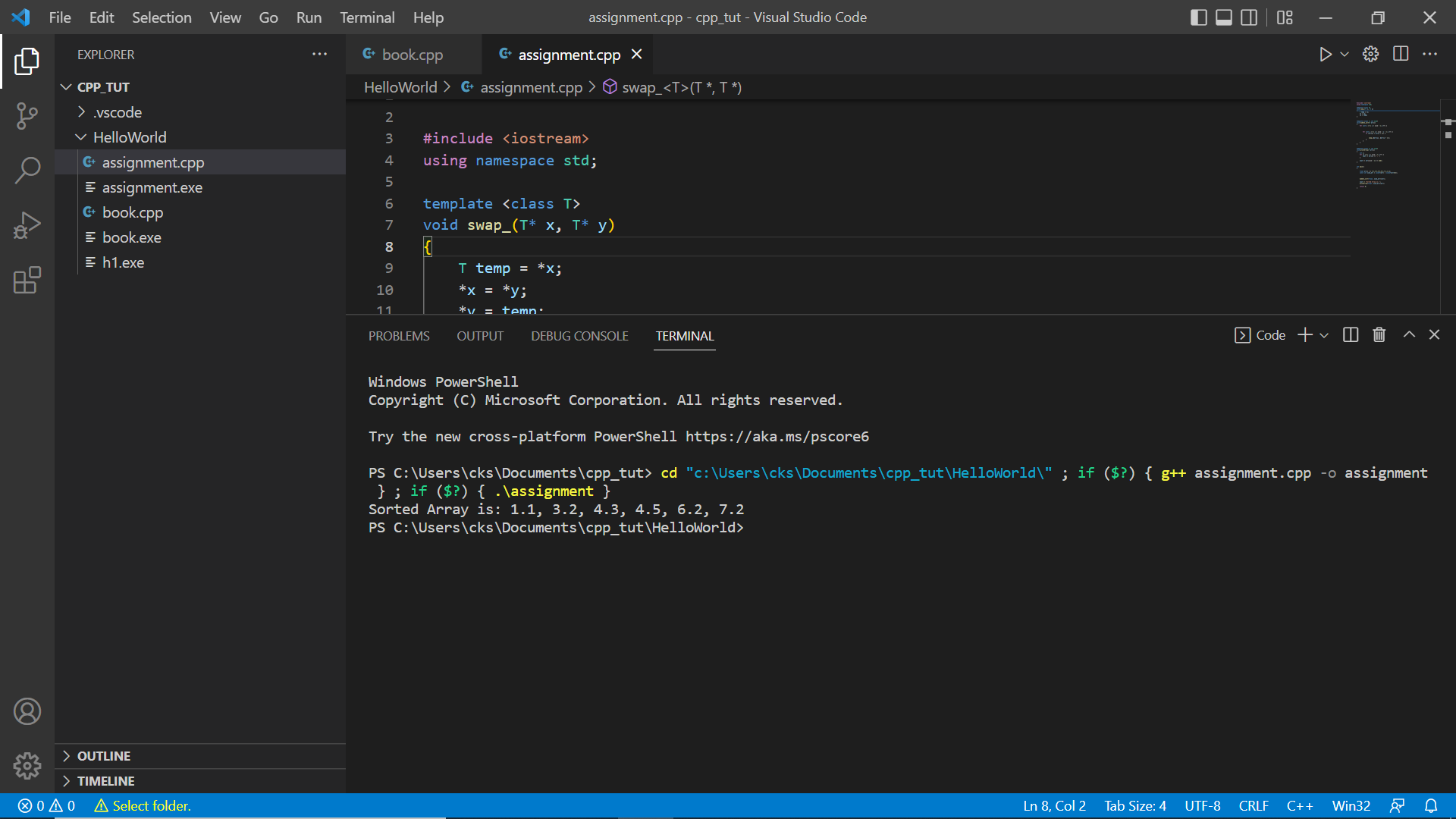
    cout << "Sorted Array is: ";

    printArray<float, size\_arr>(arr);

    return 0;

}

**Output:**



1. Write C++ Program to show:
2. Non-Templated class derived from Templated base class.

Code:

#include <iostream>

#include <math.h>

#include <string>

using namespace std;

double pi = 3.14;

enum eColor { none = 0, red, white, blue, yellow, green, black };

class Color {

public:

Color(eColor color);

void setColor(eColor color);

eColor getColor() { return mColor; };

std::string getStrColor();

protected:

eColor mColor;

};

Color::Color(eColor \_color) { mColor = \_color; }

void Color::setColor(eColor \_color) { mColor = \_color; }

std::string Color::getStrColor() {

switch (mColor) {

case red:

return "red";

case white:

return "white";

case blue:

return "blue";

case yellow:

return "yellow";

case green:

return "green";

case black:

return "black";

case none:

default:

return "none";

}

}

template <typename T> class Circle : public Color {

public:

Circle(T centerX, T centerY, T radius, eColor color);

Circle(T centerX, T centerY, T radius);

Circle(T radius);

Circle();

T area();

T circumference();

T getX();

T getY();

T getRadius();

protected:

T x;

T y;

T radius;

};

template <typename T>

Circle<T>::Circle(T \_x, T \_y, T \_radius, eColor \_color) : Color(\_color) {

x = \_x;

y = \_y;

radius = \_radius;

}

template <typename T> Circle<T>::Circle(T \_x, T \_y, T \_radius) : Color(none) {

x = \_x;

y = \_y;

radius = \_radius;

}

template <typename T> Circle<T>::Circle(T \_radius) : Color(none) {

x = static\_cast<T>(0);

y = static\_cast<T>(0);

radius = \_radius;

}

template <typename T> Circle<T>::Circle() : Color(none) {

x = static\_cast<T>(0);

y = static\_cast<T>(0);

radius = static\_cast<T>(1);

}

template <typename T> T Circle<T>::area() { return M\_PI \* radius \* radius; }

template <typename T> T Circle<T>::circumference() {

return static\_cast<T>(2) \* M\_PI \* radius;

}

class Sphere : public Circle<float> {

public:

Sphere(float centerZ, float centerX, float centerY, float radius,

eColor color);

Sphere(float radius);

Sphere();

float surfaceArea();

float volume();

float getZ();

private:

float z;

};

Sphere::Sphere(float \_x, float \_y, float \_z, float \_radius, eColor \_color)

: Circle<float>::Circle(\_x, \_y, \_radius, \_color) {

this->z = \_z;

}

Sphere::Sphere(float \_radius) : Circle::Circle(\_radius) {

this->x = static\_cast<float>(0);

this->y = static\_cast<float>(0);

this->z = static\_cast<float>(0);

this->radius = \_radius;

}

Sphere::Sphere() {

this->x = static\_cast<float>(0);

this->y = static\_cast<float>(0);

this->z = static\_cast<float>(0);

this->radius = static\_cast<float>(1);

}

float Sphere::surfaceArea() {

return static\_cast<float>(4) \* M\_PI \* this->radius \* this->radius;

}

float Sphere::volume() {

float three = 3;

float four = 4;

return four \* M\_PI \* this->radius \* this->radius \* this->radius / three;

}

int main(int argc, char \*argv[]) {

Sphere sphereA(0.0, 0.0, 0.0, 10.0, blue);

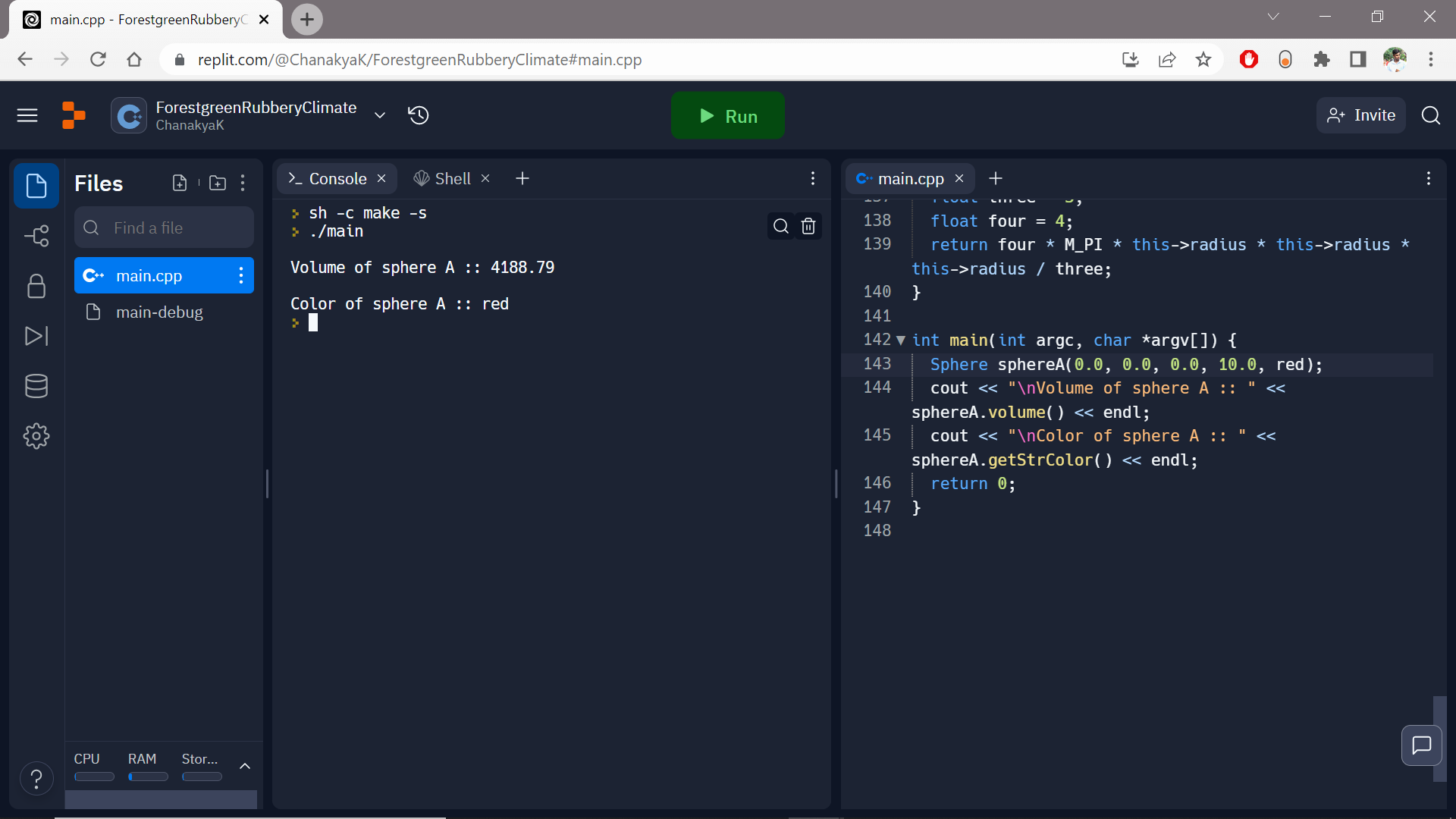
cout << "\nVolume of sphere A :: " << sphereA.volume() << endl;

cout << "\nColor of sphere A :: " << sphereA.getStrColor() << endl;

return 0;

}

Output:



1. Object Creation of derived class

Code:

#include <iostream>

using namespace std;

class Shape {

public:

void setWidth(int w) {

width = w;

}

void setHeight(int h) {

height = h;

}

protected:

int width;

int height;

};

class Rectangle: public Shape {

public:

int getArea() {

return (width \* height);

}

};

int main(void) {

Rectangle Rect;

Rect.setWidth(5);

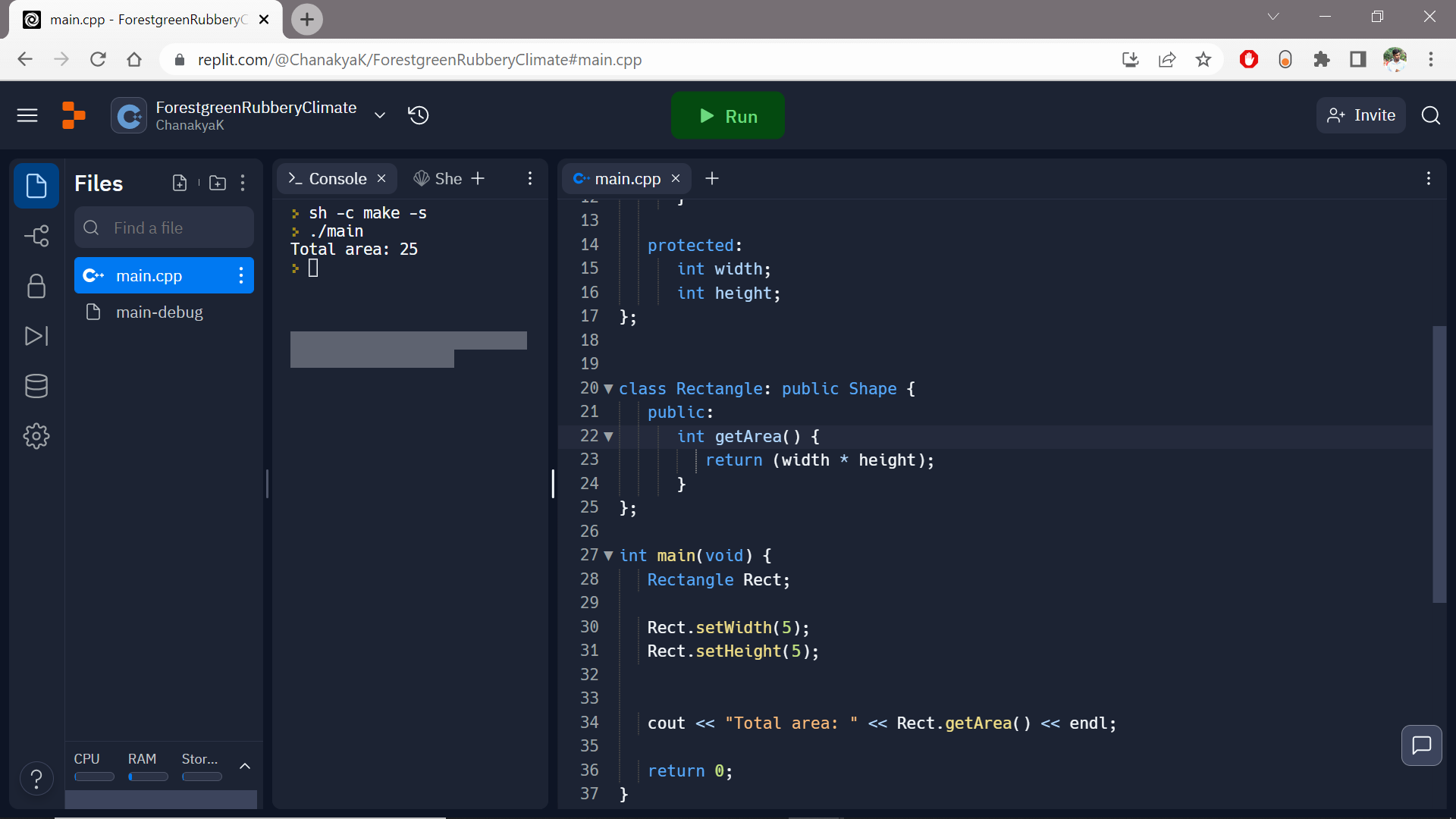
Rect.setHeight(5);

cout << "Total area: " << Rect.getArea() << endl;

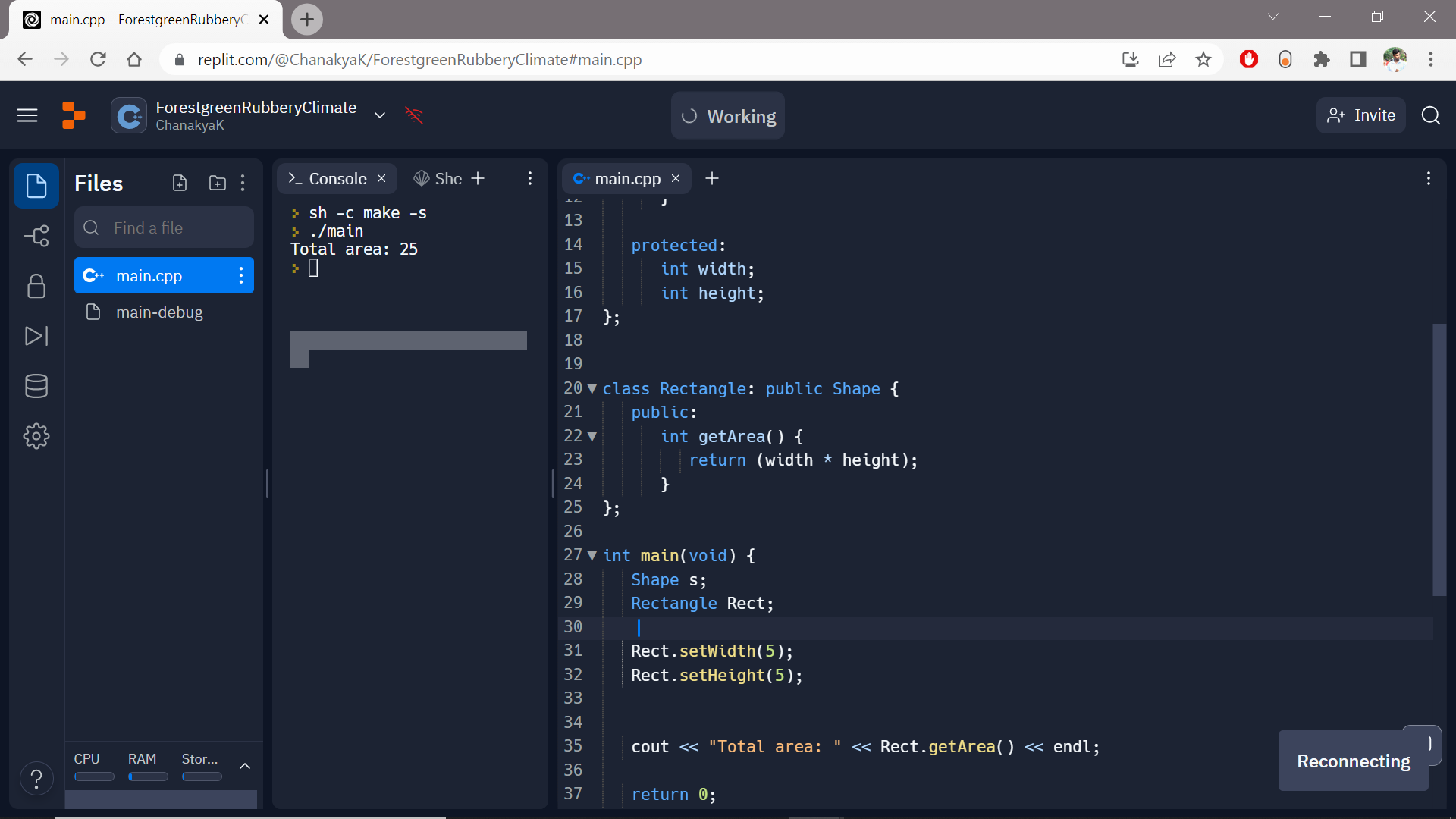
return 0;

}

Output:



1. Object Creation of Base class



1. Initialize values of private member variables of both classes.

Code:

#include <iostream>

using namespace std;

class MyClass{

private:

static int st\_var;

public:

MyClass(){

st\_var++;

}

static int getStaticVar() {

return st\_var;

}

};

int MyClass::st\_var = 0;

int main() {

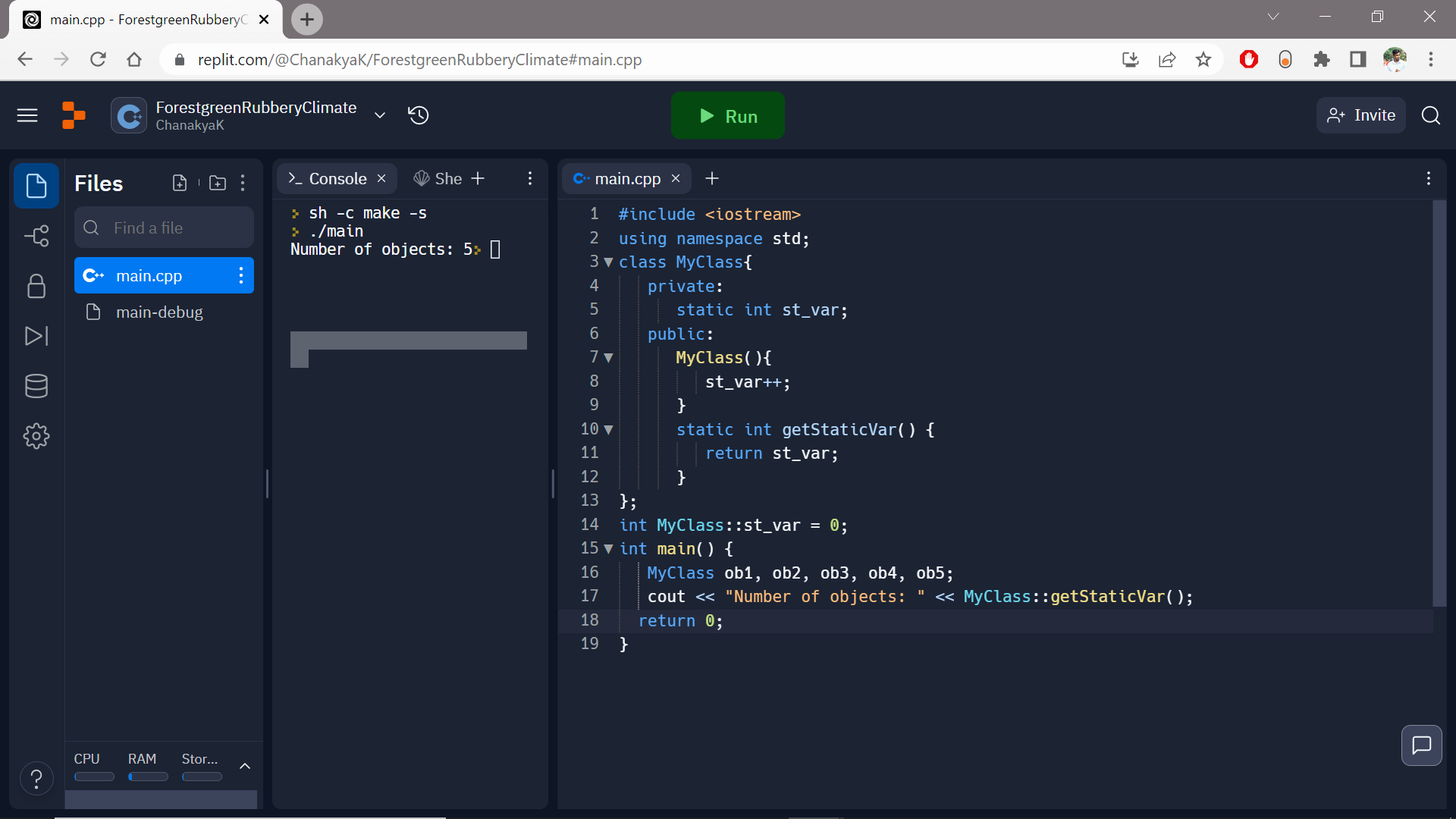
MyClass ob1, ob2, ob3, ob4, ob5;

cout << "Number of objects: " << MyClass::getStaticVar();

return 0;

}

Output:



1. How many parameters are legal for non-type template?

A. 1

B. 2

C. 3

D. 4

Ans. D. 4 because The following are legal for non-type template parameters: integral or enumeration type, Pointer to object or pointer to function, Reference to object or reference to function, Pointer to member.

1. Can we have overloading of the function templates?

A. Yes

B. No

C. May Be

D. Can't Say

Ans: A **template function can be overloaded either by a non-template function or using an ordinary function template.**

1. Write a program for Nested Exception Handling.

Code:

#include <iostream>

using namespace std;

int main()

{

int x = -1;

cout << "Before try \n";

try {

cout << "Inside try \n";

if (x < 0)

{

throw x;

cout << "After throw (Never executed) \n";

}

}

catch (int x ) {

cout << "Exception Caught \n";

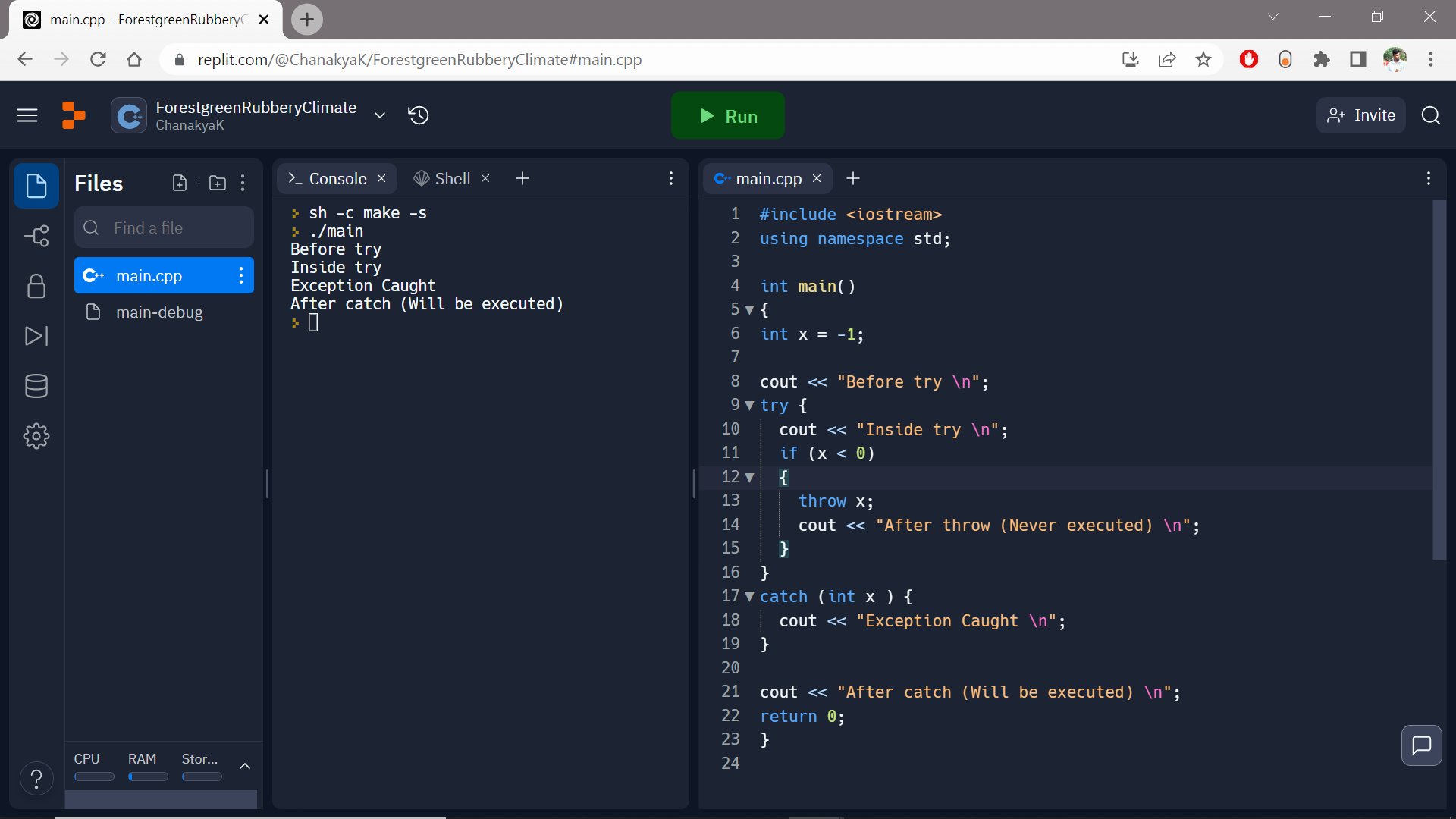
}

cout << "After catch (Will be executed) \n";

return 0;

}

Output:



1. Write a program for Rethrowing Exception Handling in Function.

Code: #include <iostream>

using namespace std;

void exceptionFunction() {

try {

throw 0;

} catch (int i) {

cout << "\nIn Function : Wrong Input :" << i;

throw;

}

}

int main() {

int var = 0;

cout << "Simple C++ Program for Rethrowing Exception Handling : In Function\n";

try {

exceptionFunction();

}

catch (int ex) {

cout << "\nIn Main : Wrong Input :" << ex;

}

return 0;

}

Output:

