

#include <iostream>

using namespace std;

bool is\_leap\_year(int year) {

if (year % 400 == 0)

return true;

if (year % 100 == 0)

return false;

if (year % 4 == 0)

return true;

return false;

}

int main() {

int year;

cout << "Enter a year: ";

cin >> year;

if (is\_leap\_year(year)) {

cout << year << " is a leap year." << endl;

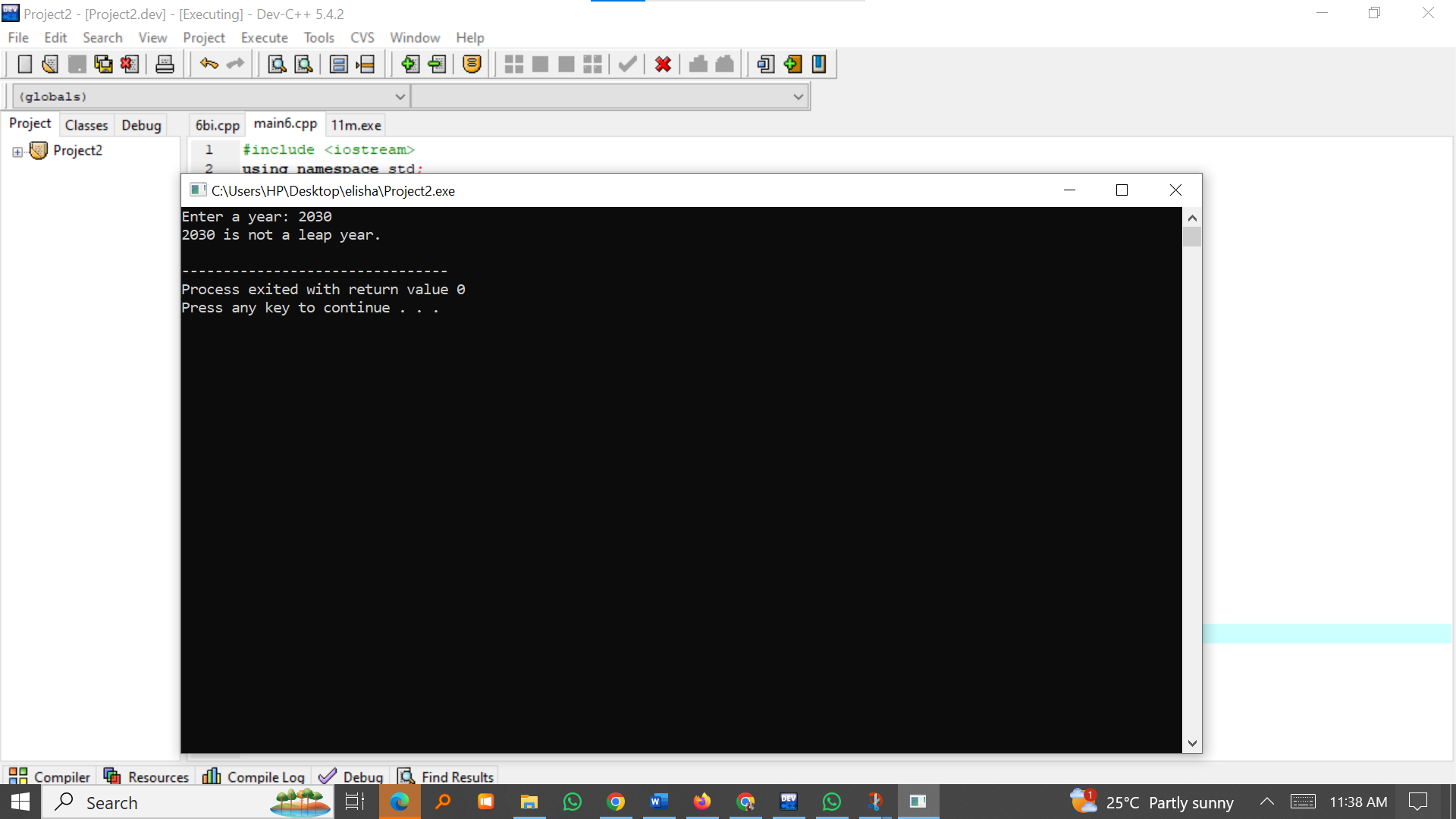
} else {

cout << year << " is not a leap year." << endl;

}

return 0;

}

****

**B**

#include <iostream>

using namespace std;

int main() {

double sum = 0;

for (int i = 1, j = 3; i <= 95 && j <= 97; i += 2, j += 2) {

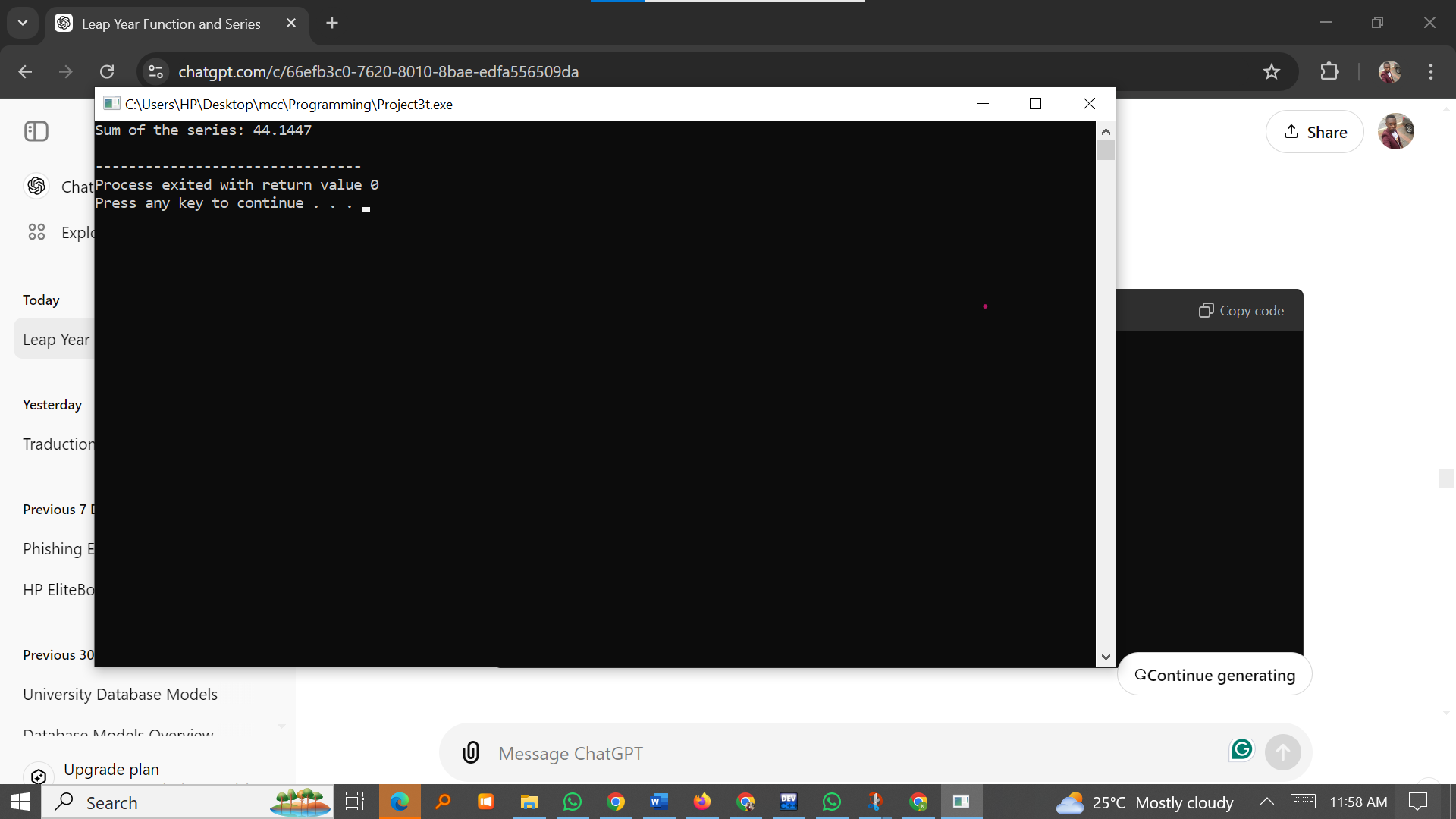
sum += (double)i / j;

}

cout << "Sum of the series: " << sum << endl;

return 0;

}



**Number Three**

**A(i)**

Arrays in C++ are a collection of elements, all of the same data type, stored in contiguous memory locations.

How to Deploy:

To declare an array, specify the data type and size of the array. Example: int arr[5]; creates an array of 5 integers.

Arrays can be used to store multiple values in a single variable (e.g., marks of 5 students, names, etc.).

Accessing Elements: Use an index to access array elements (starting from 0). Example: arr[0] accesses the first element.

Arrays are useful when dealing with a fixed number of items and when you need to perform operations like sorting or searching.

Example**:** int arr[5] = {10, 20, 30, 40, 50}; // Array with 5 integers

for (int i = 0; i < 5; i++) {

cout << arr[i] << " "; // Access and print each element

}

**(ii)**

Multi-dimensional arrays are arrays of arrays, allowing for storage in tabular format (e.g., matrices).

How to Deploy:

A 2D array is declared as data\_type array\_name[rows][columns];. Example: int matrix[3][3];.

Use nested loops to traverse and manipulate multi-dimensional arrays.

Multi-dimensional arrays are useful for complex data structures like matrices, grids, or tables.

Accessing Elements: Use two indices for 2D arrays. Example: matrix[1][2] accesses the element in the second row and third column.

Example:

int matrix[2][3] = {{1, 2, 3}, {4, 5, 6}}; // 2x3 matrix

for (int i = 0; i < 2; i++) {

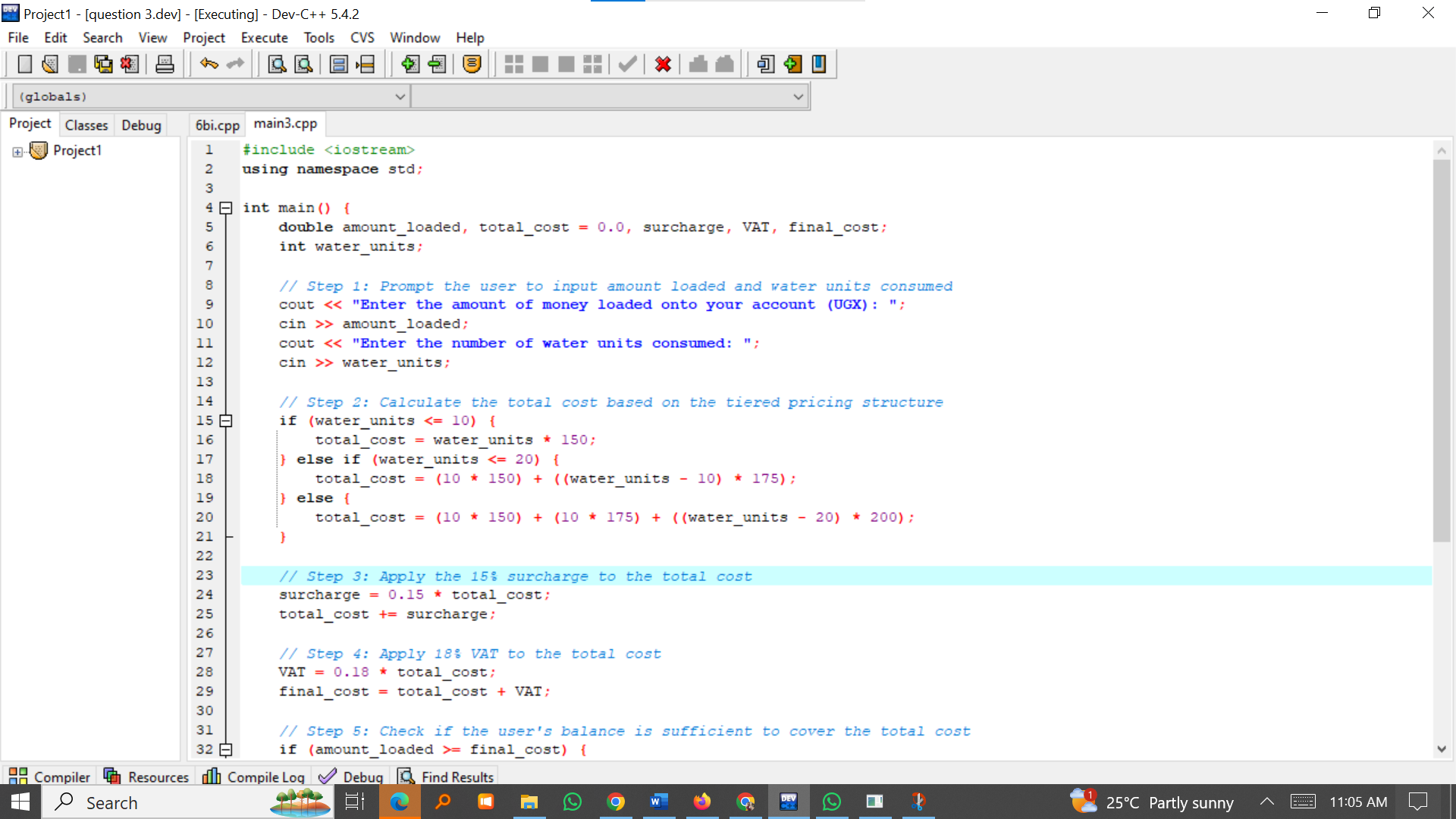
for (int j = 0; j < 3; j++) {

cout << matrix[i][j] << " "; // Access and print each element

}

}

B

 #include <iostream>

using namespace std;

int main() {

double amount\_loaded, total\_cost = 0.0, surcharge, VAT, final\_cost;

int water\_units;

// Step 1: Prompt the user to input amount loaded and water units consumed

cout << "Enter the amount of money loaded onto your account (UGX): ";

cin >> amount\_loaded;

cout << "Enter the number of water units consumed: ";

cin >> water\_units;

// Step 2: Calculate the total cost based on the tiered pricing structure

if (water\_units <= 10) {

total\_cost = water\_units \* 150;

} else if (water\_units <= 20) {

total\_cost = (10 \* 150) + ((water\_units - 10) \* 175);

} else {

total\_cost = (10 \* 150) + (10 \* 175) + ((water\_units - 20) \* 200);

}

// Step 3: Apply the 15% surcharge to the total cost

surcharge = 0.15 \* total\_cost;

total\_cost += surcharge;

// Step 4: Apply 18% VAT to the total cost

VAT = 0.18 \* total\_cost;

final\_cost = total\_cost + VAT;

// Step 5: Check if the user's balance is sufficient to cover the total cost

if (amount\_loaded >= final\_cost) {

// Transaction successful, calculate remaining balance

double remaining\_balance = amount\_loaded - final\_cost;

cout << "Transaction successful!" << endl;

cout << "Total cost (including surcharge and VAT): " << final\_cost << " UGX" << endl;

cout << "Remaining balance: " << remaining\_balance << " UGX" << endl;

} else {

// Insufficient balance

cout << "Error: Insufficient balance!" << endl;

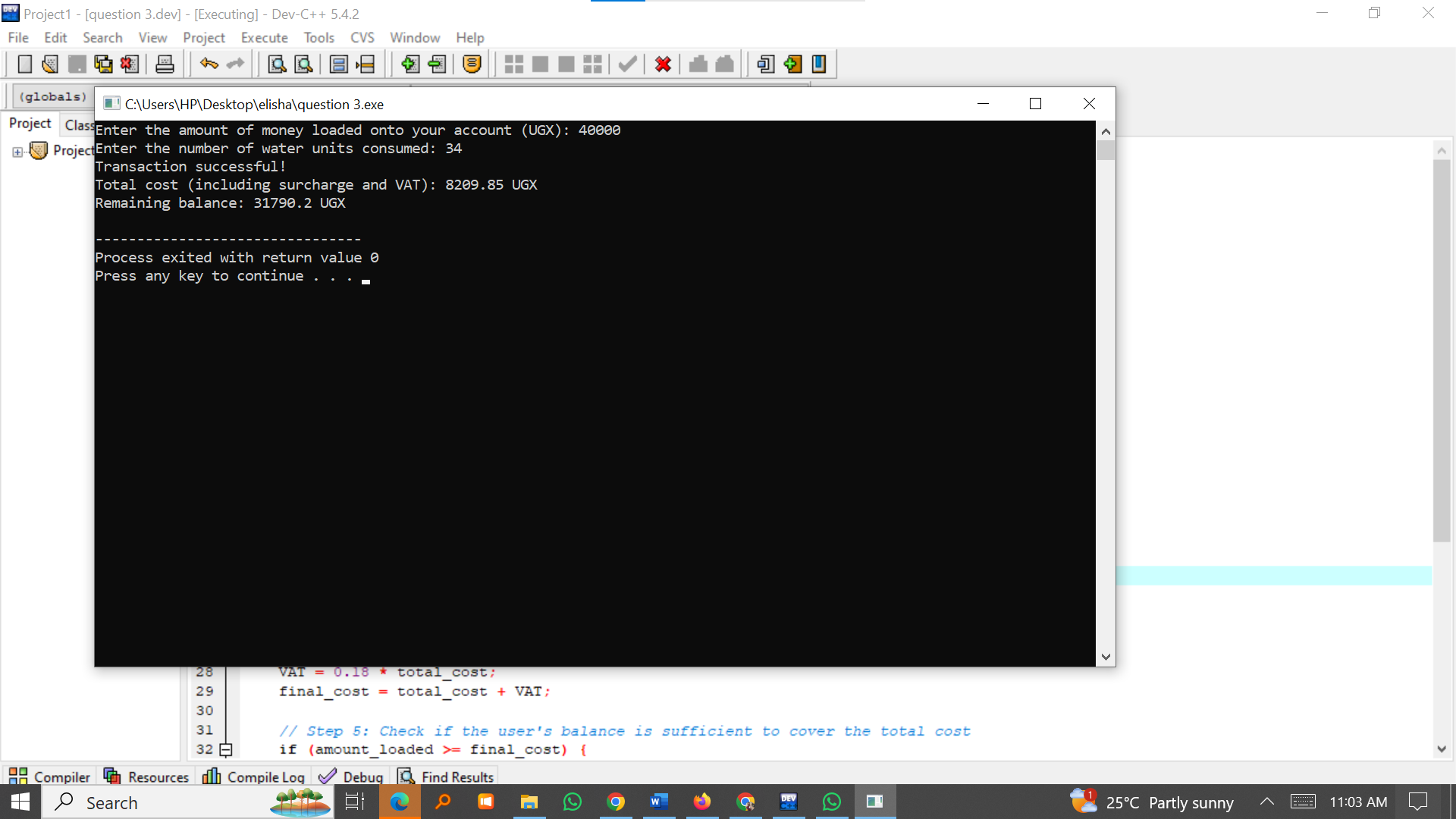
cout << "Your loaded amount is: " << amount\_loaded << " UGX" << endl;

cout << "You need: " << final\_cost << " UGX to cover the total cost." << endl;

}

return 0;

}

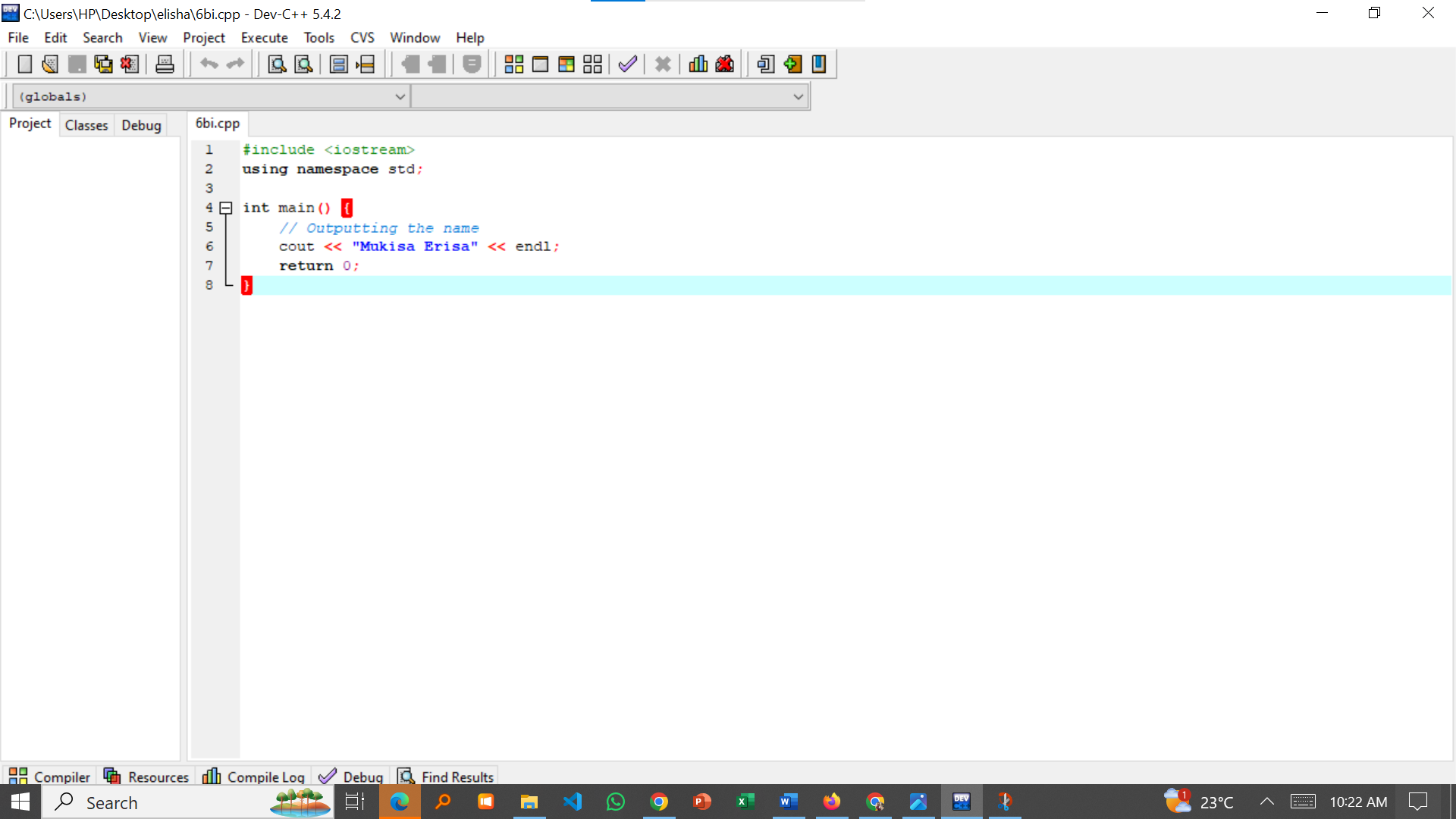
****

**5(a)**

**: Definitions in C++ Programming Language**

1. **Data type:**
   * A data type in C++ defines the type of data a variable can hold. Examples include:
     + int for integers,
     + float for floating-point numbers,
     + char for characters, and
     + bool for boolean values.
   * It specifies the range of values that the variable can store and the operations that can be performed on it.
2. **Variable:**
   * A variable in C++ is a named storage location that holds data that can be modified during program execution. It is declared with a specific data type (e.g., int x, float y).
   * Variables must be declared before they can be used.
3. **Algorithm:**
   * An algorithm is a step-by-step procedure or set of rules used to perform a specific task or solve a particular problem. In C++ programming, algorithms are often implemented through sequences of code instructions.
4. **Function:**
   * A function in C++ is a block of code that performs a specific task. Functions are reusable pieces of code that can be called by their name, passing data (arguments) to them, and can return a value. They allow modularity and code reusability.
   * Example: int add(int a, int b) { return a + b; } is a function that adds two numbers.

**5b(i)**

****

#include <iostream>

using namespace std;

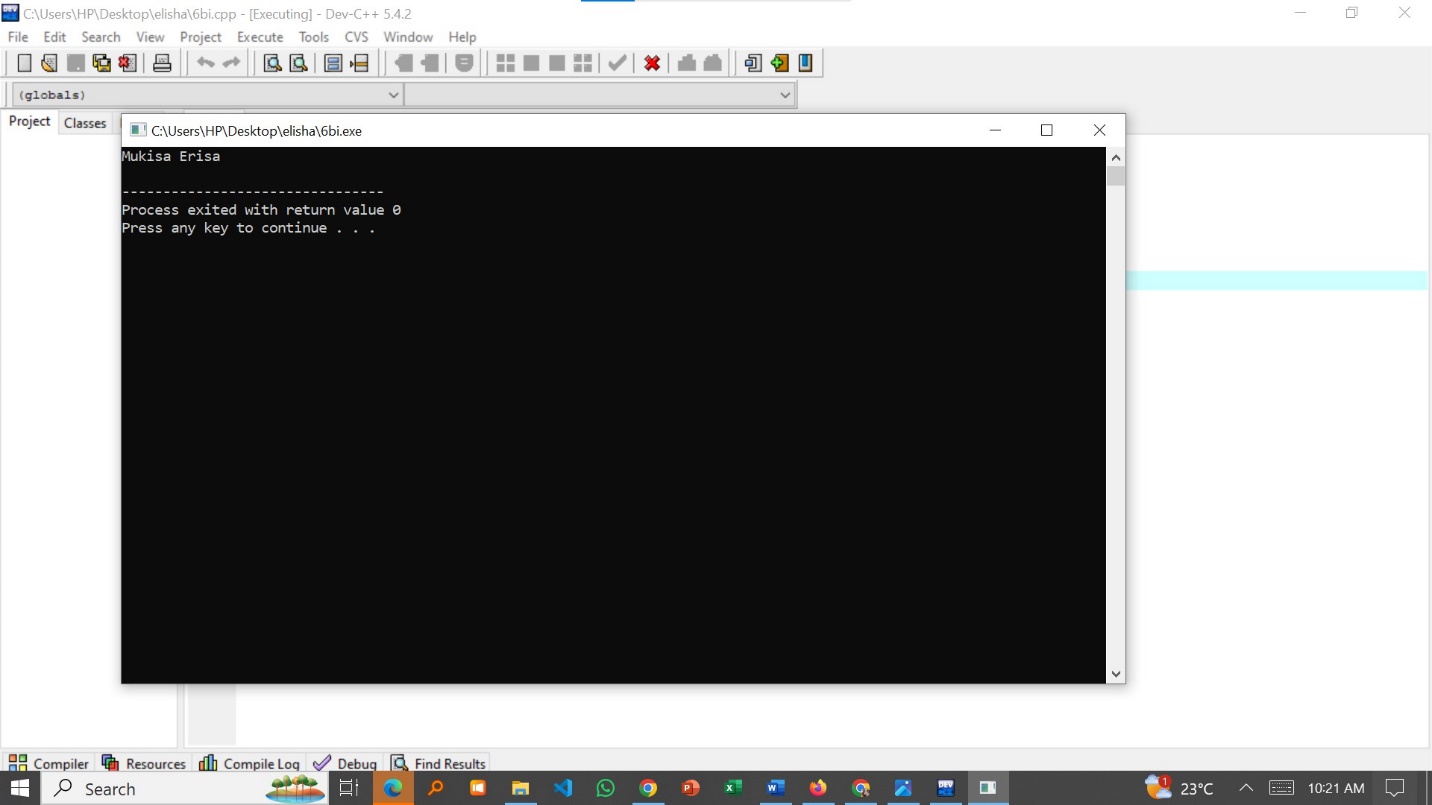
int main() {

// Outputting the name

cout << "Mukisa Erisa" << endl;

return 0;

}

****

**ii**

#include <iostream>

using namespace std;

int main() {

// Looping through numbers from 1 to 20

for (int i = 1; i <= 20; i++) {

if (i % 3 == 0) {

// Print if the number is a multiple of 3

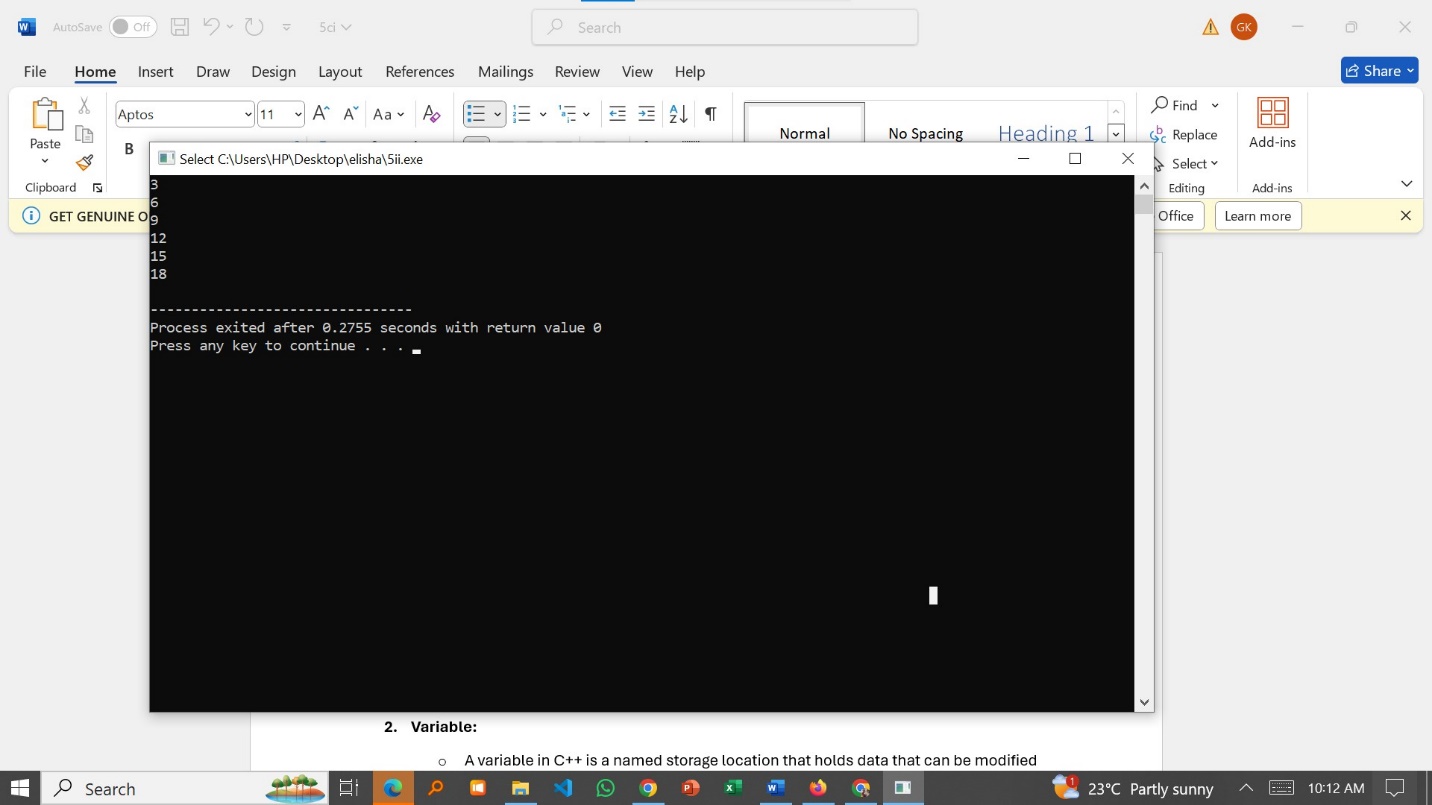
cout << i << endl;

}

}

return 0;

}

****

**5.C(i)**

**Single-line comments**:

* **Why**: To explain or annotate specific parts of code in a concise manner.
* **How**: Use two forward slashes (//). Everything after // on the same line is treated as a comment and ignored by the compiler.
* **When**: Use when you want to explain a small part of your code (e.g., a single line or variable).

For example

// This is a single-line comment

int x = 5; // Declaring a variable

ii

**Multi-line comments**:

* **Why**: To explain larger sections of code or provide more detailed descriptions.
* **How**: Use /\* to start and \*/ to end the comment block. Everything between is ignored by the compiler.
* **When**: Use for providing detailed explanations or commenting out multiple lines of code.

**Number six**

#include <iostream>

using namespace std;

double calculate\_weight(double mass) {

return mass \* 9.8;

}

int main() {

double mass;

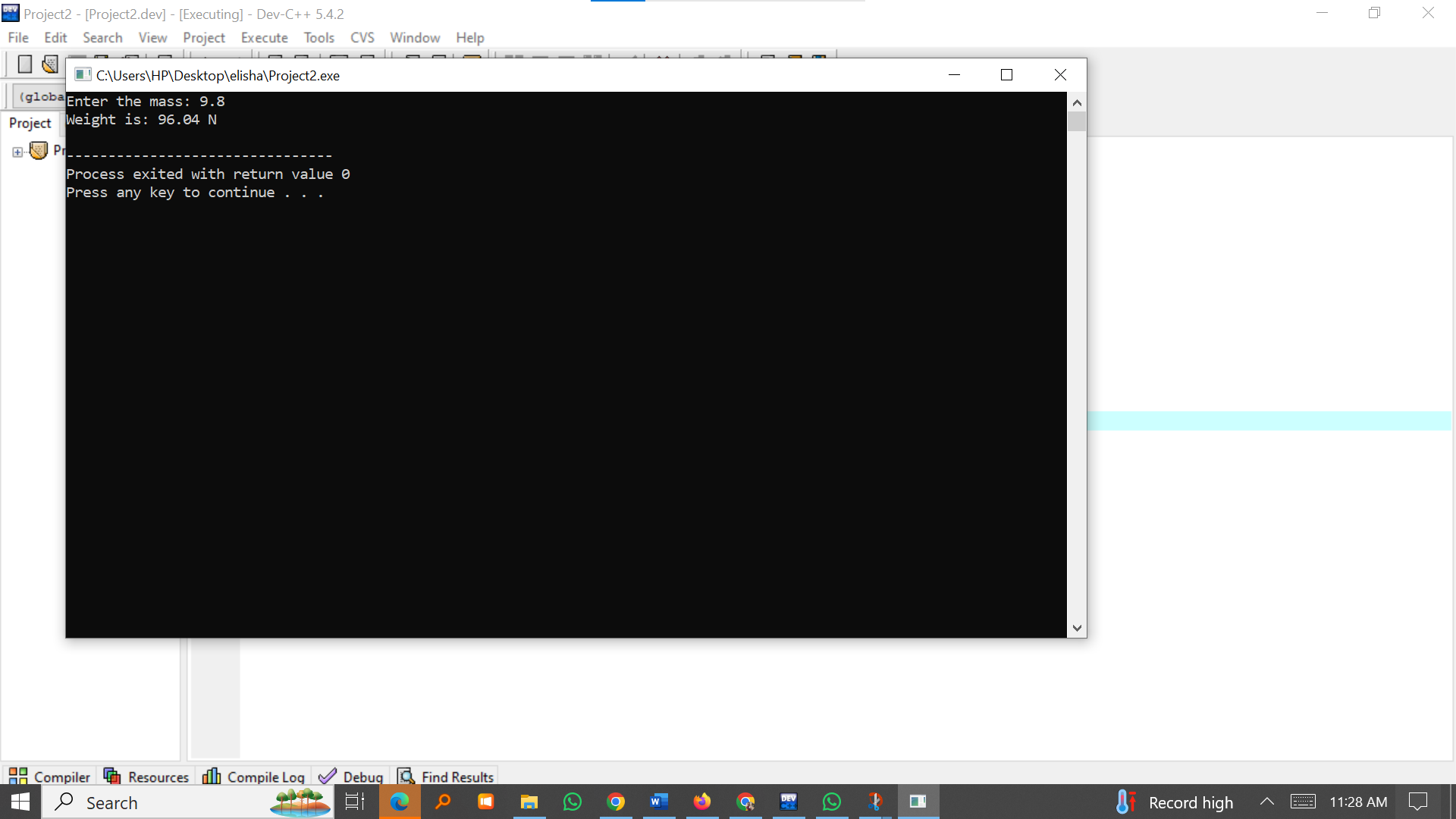
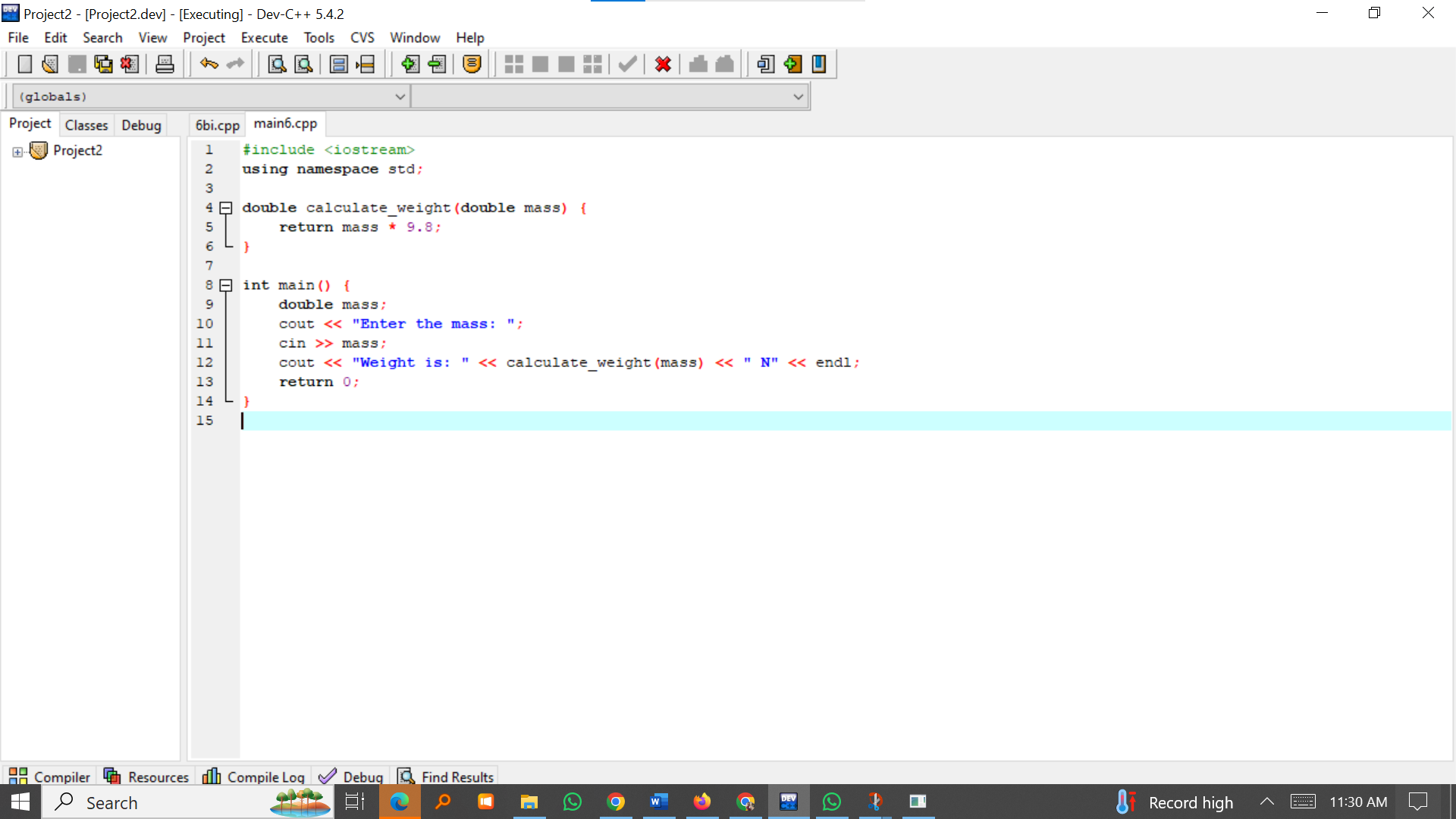
cout << "Enter the mass: ";

cin >> mass;

cout << "Weight is: " << calculate\_weight(mass) << " N" << endl;

return 0;

}

****

**Difference between pointer and reference in C++ (5 Marks)**

* **Pointer**: A variable that holds the memory address of another variable. Example: int\* ptr = &x;.
* **Reference**: An alias for another variable. Example: int &ref = x;.

**c) Evaluate the code (15 Marks)**

1. **Part 1**: The function prints out values of the array after multiplying each by 100.
2. **Part 2**: Returns and prints the value of y (16) after passing through the function foo.