**LAB PROGRAMS 0F C++**

**NAYUNIPATI CHARAN SAI**

**192111039**

**DAY - 1**

**QUESTION NO 5.**

**Imagine you're developing a game where characters collect items that affect their stats and make decisions based on those stats. In this scenario, a character's health, experience, and inventory items will be influenced by various actions, demonstrating the use of different operators.**

**Objective: Update the character's status based on the following actions:**

**Collect experience points and items.**

**Encounter a trap that affects health.**

**Use an item from the inventory that boosts health.**

**Make a decision based on health and experience levels.**

**PROGRAM:**

**#include <iostream>**

**using namespace std;**

**int main() {**

**int health = 100;**

**int experience = 0;**

**int inventoryItems = 2;**

**experience << 50;**

**inventoryItems += 2;**

**int healthBoost = 0;**

**if (inventoryItems > 0) {**

**health += healthBoost;**

**inventoryItems--;**

**}**

**if (health <= 0) {**

**cout << "Game over! Your character has died." << endl;**

**} else if (experience >= 100) {**

**cout << "Congratulations! Your character leveled up!" << endl;**

**} else {**

**cout << "Keep playing! Your character is still alive and gaining experience." << endl;**

**}**

**cout << "Final status:" << endl;**

**cout << "Health: " << health << endl;**

**cout << "Experience: " << experience << endl;**

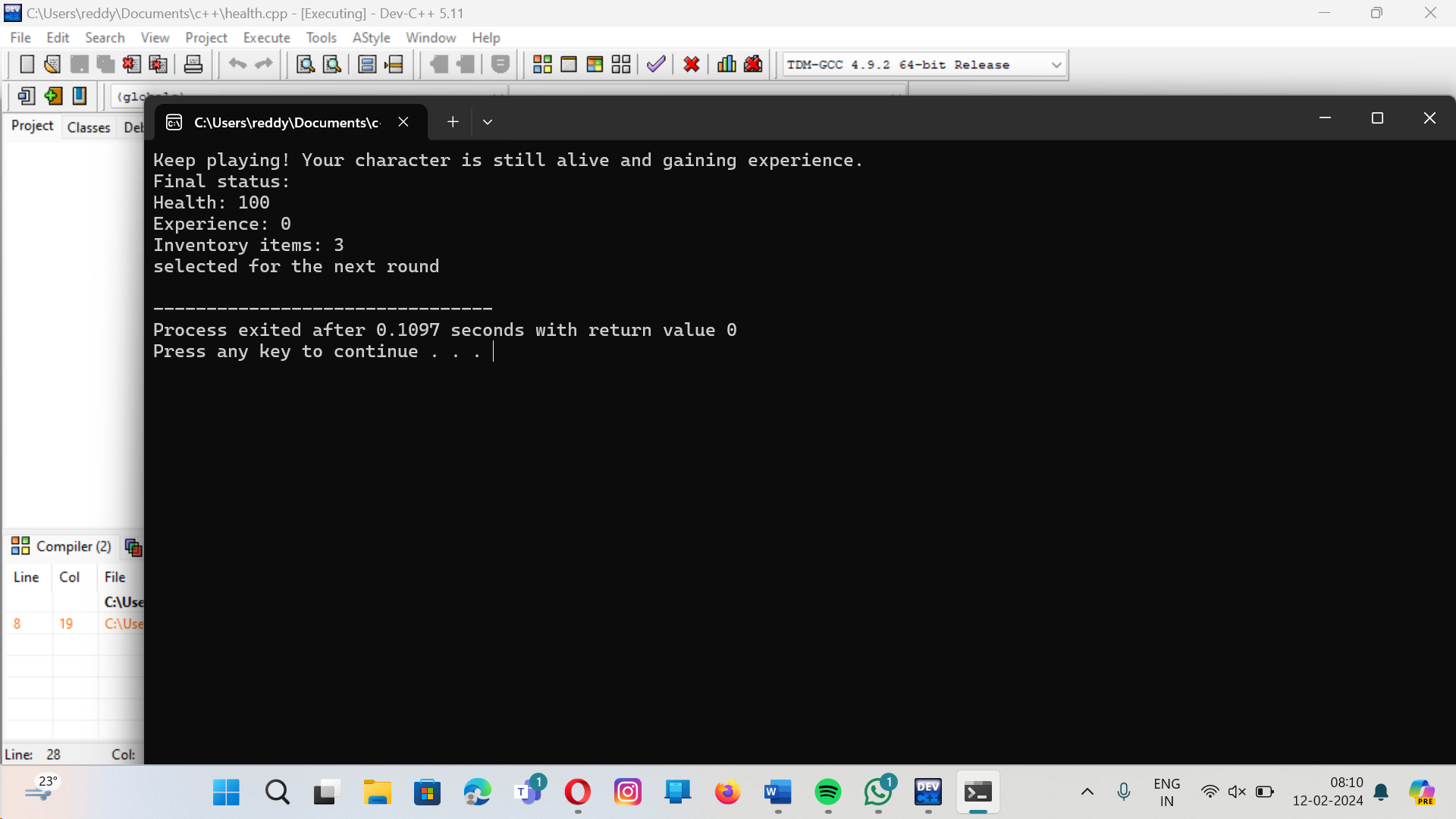
**cout << "Inventory items: " << inventoryItems << endl;**

**cout << "selected for the next round" << endl;**

**return 0;**

**}**

**OTPUT:**



**QUESTION NO 6.**

**In a high-performance computing application, you need to frequently calculate the square of numbers. To optimize these calculations, you decide to use an inline function.**

**PROGRAM:**

**#include<iostream>**

**inline double square(double x){**

**return x\*x;**

**}**

**int main(){**

**double result=square(8.0);**

**std::cout<<"square od 8 is"<<result<<std::endl;**

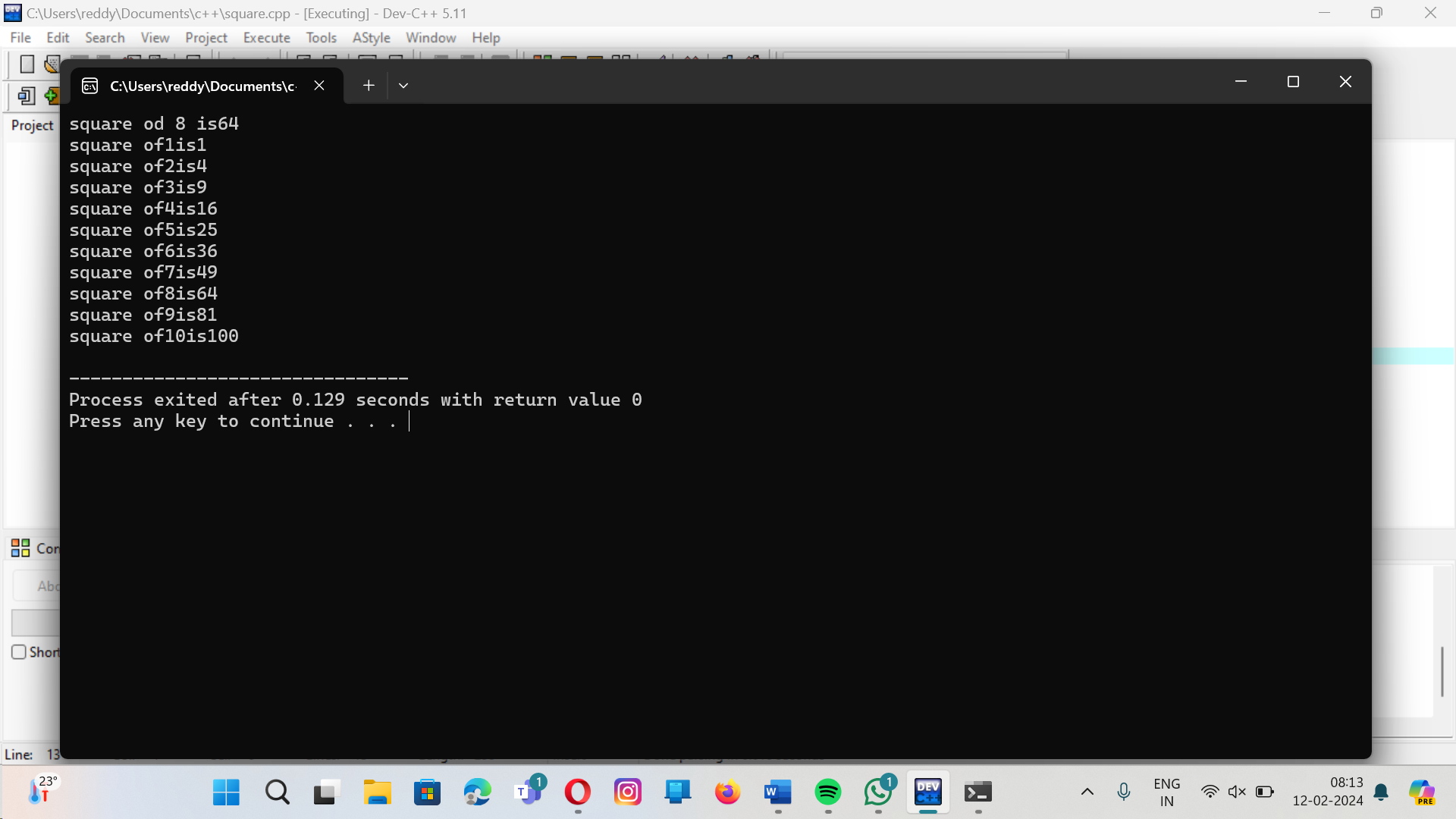
**for(int i=1;i<=10;i++){**

**std::cout<<"square of"<<i<<"is"<<square(i)<<std::endl;**

**}**

**}**

**OUTPUT:**



**QUESTION NO 7.**

**You are implementing a configuration function for a game engine that initializes game settings like screen resolution, difficulty level, and whether to enable fullscreen mode.**

**Question:**

**Design a configureGame function that includes default arguments, allowing users to specify only the settings they want to change from the defaults.**

**PROGRAM:**

**#include <iostream>**

**#include <string>**

**struct GameSettings {**

**int screenWidth;**

**int screenHeight;**

**int difficultyLevel;**

**bool fullscreen;**

**GameSettings(int width = 1920, int height = 1080, int difficulty = 1, bool full = false)**

**: screenWidth(width), screenHeight(height), difficultyLevel(difficulty), fullscreen(full) {}**

**};**

**void configureGame(const GameSettings& settings) {**

**std::cout << "Configuring game settings...\n";**

**std::cout << "Screen Resolution: " << settings.screenWidth << "x" << settings.screenHeight << std::endl;**

**std::cout << "Difficulty Level: " << settings.difficultyLevel << std::endl;**

**std::cout << "Fullscreen: " << (settings.fullscreen ? "Enabled" : "Disabled") << std::endl;**

**}**

**int main() {**

**configureGame(GameSettings());**

**configureGame(GameSettings(2560, 1440));**

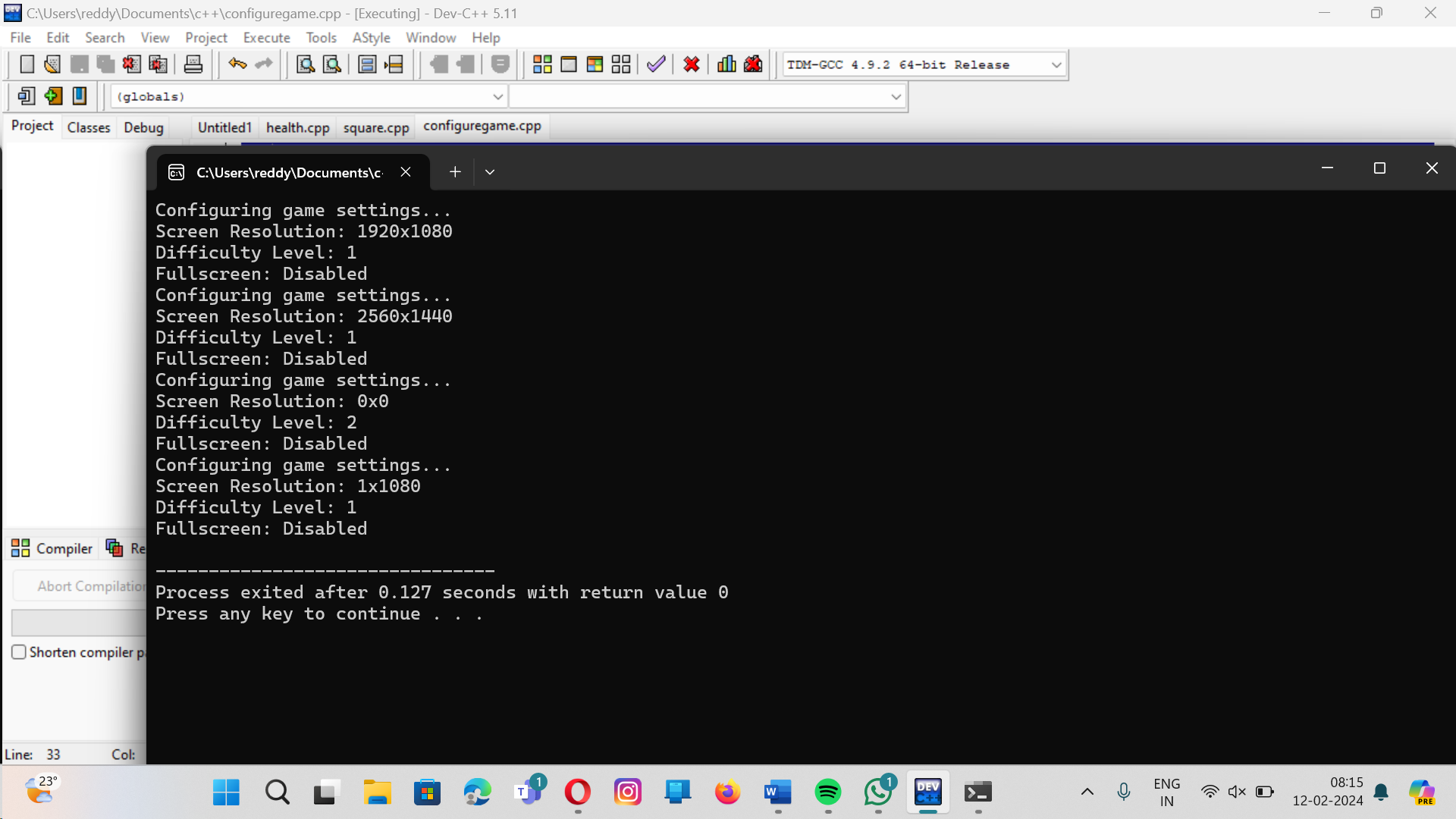
**configureGame(GameSettings(0, 0, 2));**

**configureGame(GameSettings(true));**

**return 0;**

**}**

**OUTPUT:**



**QUESTION NO 8.**

**Problem Statement: You are given a dataset representing the daily average temperatures of a city over a month, stored in an array. Write a C++ program that includes functions to perform the following tasks:**

**Calculate the average temperature of the month.**

**Find the maximum and minimum temperatures recorded in the month.**

**Determine the number of days the temperature was above the monthly average.**

**PROGRAM:**

**#include <iostream>**

**double calculateAverageTemperature(const double temperatures[], int size) {**

**double total = 0;**

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

**total += temperatures[i];**

**}**

**return total / size;**

**}**

**double findMaxTemperature(const double temperatures[], int size) {**

**double maxTemp = temperatures[0];**

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

**if (temperatures[i] > maxTemp) {**

**maxTemp = temperatures[i];**

**}**

**}**

**return maxTemp;**

**}**

**double findMinTemperature(const double temperatures[], int size) {**

**double minTemp = temperatures[0];**

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

**if (temperatures[i] < minTemp) {**

**minTemp = temperatures[i];**

**}**

**}**

**return minTemp;**

**}**

**int countDaysAboveAverage(const double temperatures[], int size) {**

**double average = calculateAverageTemperature(temperatures, size);**

**int count = 0;**

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

**if (temperatures[i] > average) {**

**count++;**

**}**

**}**

**return count;**

**}**

**int main() {**

**double temperatures[] = {22.5, 25.3, 20.2, 18.6, 21.7, 24.5, 26.1, 27.8, 19.2, 23.4, 25.6, 24.1, 22.9, 19.8, 23.0, 24.7, 26.5, 27.3, 21.4, 20.3, 22.2, 23.9, 25.0, 26.2, 27.4, 28.0, 19.5, 18.4, 21.6, 23.7};**

**int size = sizeof(temperatures) / sizeof(temperatures[0]);**

**double average = calculateAverageTemperature(temperatures, size);**

**double maxTemp = findMaxTemperature(temperatures, size);**

**double minTemp = findMinTemperature(temperatures, size);**

**int daysAboveAverage = countDaysAboveAverage(temperatures, size);**

**std::cout << "Average Temperature of the month: " << average << std::endl;**

**std::cout << "Maximum Temperature recorded: " << maxTemp << std::endl;**

**std::cout << "Minimum Temperature recorded: " << minTemp << std::endl;**

**std::cout << "Number of days the temperature was above the monthly average: " << daysAboveAverage << std::endl;**

**return 0;**

**}**

**OUTPUT:**

**O/P : Average Temperature of the month: 23.36**

**Maximum Temperature recorded: 28**

**Minimum Temperature recorded: 18.4**

**Number of days the temperature was above the monthly average: 16**

**--------------------------------**

**Process exited after 0.1274 seconds with return value 0**

**Press any key to continue . . .**