**Author:** Joshua Nguyen  
**Date:** June 15, 2025  
**Course:** CIS-5 42635

This project is a feature-complete version of the *"Higher or Lower"* card game done previously in project 1. The player guesses if the next randomly generated number will be higher or lower than the current one. The program keeps track of scores, maintains a leaderboard, and demonstrates advanced programming concepts like arrays, file I/O, sorting, searching, vectors, 2D arrays, and function overloading.

It improves on Project 1 by organizing the code into multiple functions, passing arrays and references properly, validating user input, and avoiding global variables (except for necessary constants).

I referenced the example structure from Project 1 and tried to meet every checklist item, especially avoiding global variables, using local constants, commenting 20% of the code, and formatting cleanly with functions and loops. I learned how to sort parallel arrays, validate input, and how to overload functions in C++. The biggest challenge was keeping everything modular while still tracking players and scores across the program.

Variables.

| Variable | Type | Scope | Purpose |
| --- | --- | --- | --- |
| pNames | string[] | main() | Stores player names |
| pScores | int[] | main() | Stores corresponding player scores |
| numPlyr | int | main() | Tracks current number of players |
| choice | char | main() | Stores menu input from user |
| score | int | play() | Player’s score during a game round |
| name | string | play(), main() | Stores player name |
| current | int | play() | Current number shown to player |
| next | int | play() | Next randomly generated number |
| gCount | static int | play() | Counts how many games have been played this session |
| guess | char | getGues() | Stores user’s guess input ('H' or 'L') |
| matrix | int[TBL\_ROWS][TBL\_COLS] | dem2DAr() | 2D array for multiplication table demo |
| items | vector<string> | demVctr() | Stores example items for vector demo |
| avg | float | prntBrd() | Holds the average score of all players on leaderboard |
| total | float | prntBrd() | Accumulates total score for averaging |
| pIndex | int | addPlyr(), findPlr() | Index position of player found in array |
| maxIdx | int | srtBord() | Index of highest score during sorting |
|  |  |  |  |

/\* \* File: main.cpp

\* Author: Joshua Nguyen

\* Created on June 15, 2025, 2 am~~

\* Purpose: Project 2 - High or Low Game with Leaderboard

\* Version: 2.0 (Feature Complete)

\*/

//System Libraries

#include <iostream> //Input/Output Library

#include <fstream> //File Input/Output Library

#include <string> //String Objects Library

#include <vector> //STL Vector Library

#include <iomanip> //Formatting Library (setw, setprecision)

#include <cstdlib> //Standard Library for rand(), srand(), exit()

#include <ctime> //Time Library for seeding rand()

#include <cctype> //Character handling functions (toupper)

**using** **namespace** std;

//Global Constants - Physical or Mathematical Conversions

//No Global Variables are used in this program.

**const** **int** MAX\_PLR = 10; //Maximum number of players on the leaderboard

**const** **int** TBL\_ROWS = 3; //Number of rows for the 2D array demonstration

**const** **int** TBL\_COLS = 4; //Number of columns for the 2D array demonstration

//Function Prototypes

//---------------------

//Core Game Logic

**void** play(string[], **int**[], **int**&);

**bool** getGues(**int**);

**int** getRnd(**int** min = 1, **int** max = 100);

//File I/O Functions

**void** filRead(string[], **int**[], **int**&);

**void** filSave(**const** string[], **const** **int**[], **int**);

//Data Management Functions

**void** addPlyr(string[], **int**[], **int**&, string, **int**);

**void** srtBord(string[], **int**[], **int**);

**int** findPlr(**const** string[], **int**, string);

//Display & Demonstration Functions

**void** prntBrd(**const** string[], **const** **int**[], **int**);

**void** prntBrd(**const** string[], **const** **int**[], **int**, string); //Overloaded version

**void** dem2DAr(); //2D Array Demo

**void** demVctr(); //Vector Demo

//Execution Begins Here

//---------------------

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

//Set the random number seed

srand(**static\_cast**<**unsigned** **int**>(time(0)));

//Declare local variables for main

string pNames[MAX\_PLR]; //1D Array for player names

**int** pScores[MAX\_PLR]; //Parallel 1D array for player scores

**int** numPlyr = 0; //Number of current players stored

**char** choice; //User menu choice

//Read previous leaderboard data from file into arrays

filRead(pNames, pScores, numPlyr);

//Main menu loop

**do** {

//Display menu options to the user

cout << "\n------ HIGHER OR LOWER ------" << endl;

cout << "1. Play Game" << endl;

cout << "2. View Leaderboard" << endl;

cout << "3. Search for Player" << endl;

cout << "4. 2D Array Demo" << endl;

cout << "5. Vector Demo" << endl;

cout << "6. Quit" << endl;

cout << "---------------------------" << endl;

cout << "Enter your choice: ";

cin >> choice;

//Process menu choice using a switch statement

**switch** (choice) {

**case** '1': {

play(pNames, pScores, numPlyr);

**break**;

}

**case** '2': {

prntBrd(pNames, pScores, numPlyr);

**break**;

}

**case** '3': {

//Create a local scope for the name variable

string name;

cout << "Enter name to search for: ";

cin >> name;

//Call the overloaded print function for a single player

prntBrd(pNames, pScores, numPlyr, name);

**break**;

}

**case** '4': {

dem2DAr();

**break**;

}

**case** '5': {

demVctr();

**break**;

}

**case** '6': {

cout << "Thanks for playing! Saving data..." << endl;

//Save the leaderboard to a file before quitting

filSave(pNames, pScores, numPlyr);

//Demonstrates the exit() function

exit(0);

}

**default**: {

cout << "Invalid choice. Please try again." << endl;

}

}

} **while** (choice != '6');

//Exit the Program

**return** 0;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Definition of function play

// Purpose: Manages a full game round for a new player.

// Params: string array for names, int array for scores,

// int reference for number of players.

// Includes a static variable to count games played this session.

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**void** play(string names[], **int** scores[], **int**& num) {

//Static variable to track total games played this session

**static** **int** gCount = 0;

gCount++;

cout << "--- (Game Session #" << gCount << ") ---" << endl;

//Declare function-level variables

string name;

**int** score = 0;

**int** current, next;

**const** **int** ROUNDS = 10;

//Get player name

cout << "\nEnter your name (<= 7 chars): ";

cin >> name;

//Initialize first number

current = getRnd();

cout << "\nStarting game for " << name << "!" << endl;

//Loop for the number of rounds

**for** (**int** round = 0; round < ROUNDS; round++) {

cout << "\nRound " << round + 1 << ": Current is " << current << endl;

//Get and validate user's guess

**bool** isHigh = getGues(current);

//Get the next random number

next = getRnd();

cout << "The next number was: " << next << endl;

//Check for a correct guess using relational and logical operators

**bool** correct = (isHigh && next > current) || (!isHigh && next < current);

//Use conditional operator to display result

cout << (correct ? "Correct!\n" : "Incorrect.\n");

//Increment score if correct

**if** (correct) {

score++;

}

//Current number becomes the next for the following round

current = next;

}

//Display final score for the round

cout << "\n" << name << "'s final score: " << score << "/" << ROUNDS << endl;

//Add or update the player's score on the leaderboard

addPlyr(names, scores, num, name, score);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Definition of function getGues

// Purpose: Prompts user for Higher/Lower and validates input.

// Params: int for the current number (for context).

// Returns: A boolean, true if the guess was 'H'.

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**bool** getGues(**int** cur) {

**char** guess;

//Input validation using a do-while loop

**do** {

cout << "Will the next be (H)igher or (L)ower? ";

cin >> guess;

guess = toupper(guess);

//Check for invalid characters

**if** (guess != 'H' && guess != 'L') {

cout << "Invalid input. Please enter 'H' or 'L'." << endl;

}

} **while** (guess != 'H' && guess != 'L');

//Return true for 'Higher', false for 'Lower'

**return** (guess == 'H');

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Definition of function getRnd

// Purpose: Returns a random integer between min and max.

// Params: int min and int max (uses defaulted arguments).

// Returns: A random integer.

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**int** getRnd(**int** min, **int** max) {

**return** rand() % (max - min + 1) + min;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Definition of function filRead

// Purpose: Reads player names and scores from "leaderboard.txt".

// Params: string/int arrays and int reference for player count.

// Uses pass by reference to modify arrays and the count in main.

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**void** filRead(string names[], **int** scores[], **int**& num) {

//Input file stream object

ifstream inFile("leaderboard.txt");

//Check if the file was successfully opened

**if** (inFile) {

//Read data from file until end of file or array is full

**while** (num < MAX\_PLR && inFile >> names[num] >> scores[num]) {

num++;

}

inFile.close();

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Definition of function filSave

// Purpose: Saves the current leaderboard to "leaderboard.txt".

// Params: const string/int arrays and int for player count.

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**void** filSave(**const** string names[], **const** **int** scores[], **int** num) {

//Output file stream object

ofstream outFile("leaderboard.txt");

//Write each player's data to the file on a new line

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

outFile << names[i] << " " << scores[i] << endl;

}

outFile.close();

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Definition of function addPlyr

// Purpose: Adds a new player or updates an existing one's score.

// Params: string/int arrays, int reference for player count,

// string for name, and int for score.

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**void** addPlyr(string names[], **int** scores[], **int**& num, string name, **int** score) {

//Search if player already exists

**int** pIndex = findPlr(names, num, name);

//If-else-if to handle logic

**if** (pIndex != -1) { //Player was found

//Update score only if the new one is higher

**if** (score > scores[pIndex]) {

scores[pIndex] = score;

cout << name << "'s high score updated!" << endl;

}

} **else** **if** (num < MAX\_PLR) { //Player not found, add if space is available

names[num] = name;

scores[num] = score;

num++; //Increment player count, passed by reference

} **else** { //Leaderboard is full

cout << "Leaderboard is full. Cannot add new player." << endl;

}

//Sort the board after any potential change

srtBord(names, scores, num);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Definition of function srtBord

// Purpose: Sorts the parallel arrays by score (descending).

// Uses the Selection Sort algorithm.

// Params: string/int arrays and int for player count.

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**void** srtBord(string names[], **int** scores[], **int** num) {

//Outer loop for selection sort

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

**int** maxIdx = i;

//Inner loop to find the max element

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

**if** (scores[j] > scores[maxIdx]) {

maxIdx = j;

}

}

//Swap the elements

swap(scores[i], scores[maxIdx]);

swap(names[i], names[maxIdx]);

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Definition of function prntBrd (Overload 1)

// Purpose: Prints the formatted full leaderboard.

// Uses a float to calculate and display the average score.

// Params: const string/int arrays and int for player count.

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**void** prntBrd(**const** string names[], **const** **int** scores[], **int** num) {

cout << "\n--- LEADERBOARD ---" << endl;

**if** (num == 0) {

cout << "The leaderboard is empty." << endl;

**return**;

}

//Using a float variable for the total

**float** total = 0.0f;

//Set table headers

cout << left << setw(10) << "Player" << right << setw(5) << "Score" << endl;

cout << "-----------------" << endl;

//Loop through arrays and print data, summing the score

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

cout << left << setw(10) << names[i]

<< right << setw(5) << scores[i] << endl;

total += scores[i];

}

//Calculate and print average score using float division

**float** avg = total / num;

cout << "-----------------" << endl;

cout << left << setw(10) << "Average"

<< right << setw(5) << fixed << setprecision(1) << avg << endl;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Definition of function prntBrd (Overload 2)

// Purpose: This overloaded function finds and prints a single player's score.

// Params: const string/int arrays, int count, and string name to find.

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**void** prntBrd(**const** string names[], **const** **int** scores[], **int** num, string name) {

//Find the player's index using a linear search

**int** pIndex = findPlr(names, num, name);

//Check if the player was found

**if** (pIndex != -1) {

cout << "\n-- Player Found --" << endl;

cout << "Player: " << names[pIndex] << endl;

cout << "Score: " << scores[pIndex] << endl;

} **else** {

cout << "\nPlayer '" << name << "' not found on the leaderboard." << endl;

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Definition of function findPlr

// Purpose: Performs a linear search on the names array.

// Params: const string array, int count, string name to find.

// Returns: The integer index if found, or -1 if not found.

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**int** findPlr(**const** string names[], **int** num, string name) {

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

**if** (names[i] == name) {

**return** i; //Return index of the found player

}

}

**return** -1; //Return -1 if not found

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Definition of function dem2DAr

// Purpose: Demonstrates creating, populating, and printing a 2D array.

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**void** dem2DAr(){

cout << "\n--- 2D Array Demo (Multiplication Table) ---" << endl;

//Declare a 2D array

**int** matrix[TBL\_ROWS][TBL\_COLS];

//Populate the 2D array using nested for loops

**for**(**int** i=0; i < TBL\_ROWS; i++){

**for**(**int** j=0; j < TBL\_COLS; j++){

matrix[i][j] = (i+1)\*(j+1);

}

}

//Print the 2D array using nested for loops

**for**(**int** i=0; i < TBL\_ROWS; i++){

**for**(**int** j=0; j < TBL\_COLS; j++){

//Use setw for clean formatting

cout << setw(4) << matrix[i][j];

}

cout << endl;

}

cout << "------------------------------------------" << endl;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Definition of function demVctr

// Purpose: Demonstrates creating, using, and printing an STL Vector.

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**void** demVctr(){

cout << "\n--- STL Vector Demo ---" << endl;

//Create a vector of strings

vector<string> items;

//Add items to the vector using push\_back

items.push\_back("Apples");

items.push\_back("Pears");

items.push\_back("Oranges");

cout << "Vector contains " << items.size() << " items:" << endl;

//Print using a modern range-based for loop

**for**(**const** string &item : items){

cout << "- " << item << endl;

}

cout << "-----------------------" << endl;

}