**Upcountry Warriors Player Management System (C++)**

Contents

[**List of Figures** 5](#_Toc202732498)

[**List of Tables** 5](#_Toc202732499)

[1. Task 1 – System Design 6](#_Toc202732500)

[1.1. System Requirements Specification (SRS) 6](#_Toc202732501)

[1.2. System Design Diagrams 8](#_Toc202732502)

[1.2.1. Flowchart: Add New Player 9](#_Toc202732503)

[1.2.2. Flowchart: Search Player 10](#_Toc202732504)

[1.2.3. Flowchart: Display All Players 11](#_Toc202732505)

[1.2.4. System Architecture 12](#_Toc202732506)

[2. Task 2 – System Implementation 12](#_Toc202732507)

[2.1. Use of Control Structures 13](#_Toc202732508)

[2.2. Modularization and Data Passing 14](#_Toc202732509)

[2.3. File Handling and Data Structures 15](#_Toc202732510)

[2.4. Input Validation and User Guidelines 17](#_Toc202732511)

[2.5. Navigation, Completeness and Creativity 18](#_Toc202732512)

[3. Task 3 – Software Testing and Evaluation 19](#_Toc202732513)

[3.1. Test Plan 19](#_Toc202732514)

[3.2. Test Cases 19](#_Toc202732515)

[3.3. User Acceptance Testing 20](#_Toc202732516)

[3.4. Testing Methodology 21](#_Toc202732517)

[**Conclusion** 21](#_Toc202732518)

[**References** 21](#_Toc202732519)

## **List of Figures**

[Figure 1 : Flowchart-Add New Player 9](#_Toc202724071)

[Figure 2 : Flowchart-Search Player 10](#_Toc202724072)

[Figure 3 : Flowchart-Display All Players 11](#_Toc202724073)

[Figure 4 : Sequence Structure 13](#_Toc202724074)

[Figure 5 : Selection Structure 13](#_Toc202724075)

[Figure 6 : Repetition Structure 14](#_Toc202724076)

[Figure 7 : Modularization and Data Passing 14](#_Toc202724077)

[Figure 8 : Example of data passing 15](#_Toc202724078)

[Figure 9 : Writing to File 16](#_Toc202724079)

[Figure 10 : Reading from File 16](#_Toc202724080)

[Figure 11 : Struct for player data 16](#_Toc202724081)

[Figure 12 : Application Menu 18](#_Toc202724082)

## **List of Tables**

[Table 1 : Test Cases 19](#_Toc202724094)

## Task 1 – System Design

## System Requirements Specification (SRS)

* System Overview

The "Upcountry Warriors" baseball club, a well-known organization in the city, would like to change from manual record keeping to a computerized system for keeping track of players and teams. The proposed C++ console application tracks the storage and retrieval of player details including registration number, first name, last name, date-of-birth, and run scored. Additionally, it tracks team management (note: each player can only be part of a maximum of two teams). The application user can be restricted to one administrator user. An administrator can utilize menus to facilitate operations and workflows for core activities.

* Purpose

The primary objective of this application is to optimize the club's administration and replace the current paper-based record system. Following the principles of structured programming has enhanced the data, facilitating persistence, checking input, and a logical interface to minimize errors and improve efficiency in managing player and team data.

* Scope

The application encompasses the following functionalities:

* Adding new players with validated inputs to ensure data integrity.
* Searching for players using their unique registration number.
* Displaying all registered players’ details.
* Managing team assignments, including adding players to teams and viewing team rosters.
* Implementing secure login and logout mechanisms for admin access.
* Providing an option to exit the application safely.

The system operates through a console interface, stores data in plain text files (players.txt for player details and teams.txt for team assignments), and is designed for simplicity and reliability.

* Functional Requirements

1. **FR1: Add New Player -** The system prompts the admin to enter player details, validates the inputs, and stores them in a text file for persistence.
2. **FR2: Search Player by Registration Number -** The system retrieves and displays player details based on a provided unique ID, ensuring quick access to information.
3. **FR3: Display All Players -** The system lists all player records stored in the file, presenting them in a readable format.
4. **FR4: Manage Team Details -** The system allows the admin to assign players to teams (up to two per player) and view team rosters.
5. **FR5: Login and Logout Functionality -** The system requires admin authentication at startup and provides a logout option to secure the session.
6. **FR6: Exit Application -** The system terminates safely, ensuring all files are closed properly.

* Non-Functional Requirements

1. **NFR1: File Handling -** Data is stored in plain text files to ensure persistence across sessions, with robust error handling for file operations.
2. **NFR2: Input Validation -** All user inputs are validated to prevent incorrect or malicious data entries, such as invalid dates or duplicate IDs.
3. **NFR3: User Interface -** The interface provides clear prompts, feedback messages, and a menu-driven navigation system to enhance usability.
4. **NFR4: Code Modularization -** The system is divided into functions to separate concerns, improving maintainability and readability.
5. **NFR5: Performance -** The system ensures quick response times for user interactions and file operations, even with large datasets.
6. **NFR6: Reliability -** The system handles errors gracefully, such as file access failures or invalid inputs, to ensure robust operation.

* Assumptions and Constraints
  + Assumptions
    - The system is operated by a single admin user with basic computer literacy.
    - Each player has a unique registration number assigned by the club.
    - The system runs on a standard desktop environment with C++ compiler support.
  + Constraints
    - The interface is console-based, without graphical user interface components.
    - Players are limited to membership in a maximum of two teams.
    - Data storage is limited to plain text files, without database integration.
    - No advanced security measures (e.g., encryption) are implemented for file storage.

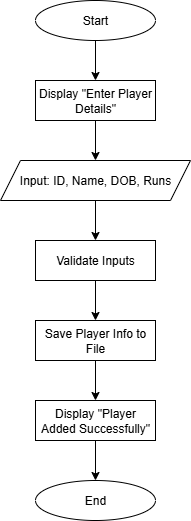
## System Design Diagrams

The system design depicts flowcharts for the typical functions of the system: add a new player, search for a player, and display all players. The flowcharts have maintained standard notations within the flowcharts (ovals for start/end, rectangles for processes, diamonds for decisions) to clarify the understanding and simplify the process. Below you will see a detailed textual description of the flowcharts, using structured English pseudocode to demonstrate the logic and modularization.

## Flowchart: Add New Player

This process allows the admin to input player details, validate them, and append them to the player file. The flowchart begins with a start symbol, followed by input prompts for player ID, first name, last name, date of birth, and run scored. Decision points validate each input (e.g., checking for duplicate IDs or valid date formats). If validation passes, the data is written to the file; otherwise, an error message is displayed, and the process loops back to the input stage.

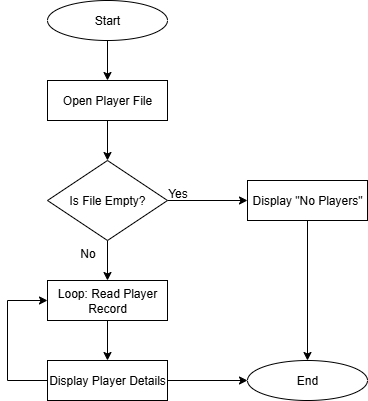
Figure : Flowchart-Add New Player



## Flowchart: Search Player

In this process, players can retrieve their details using their registration number. The flowchart begins with a prompt to enter the player ID, then shows a decision to check if the ID is a valid ID. If this is valid, the system opens the player file, reads the file line by line, and checks each ID until it finds a match. It then displays the names and details of the player. If the ID is not valid, it displays an error message. The flowchart indicates a clear logical flow and decisions.

Figure : Flowchart-Search Player



## Flowchart: Display All Players

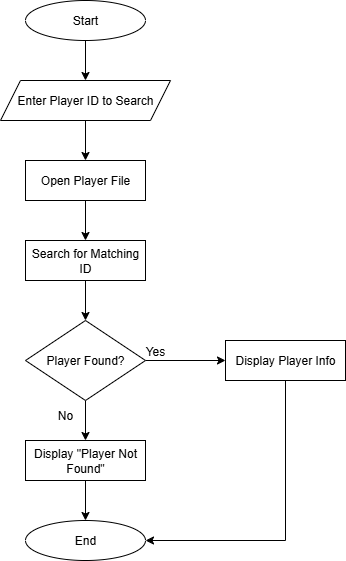
This process lists all player records from the file. The flowchart begins by opening the player file, followed by a loop to read and display each line until the end of the file. Error handling is included for file access failures, ensuring the system remains robust.

Figure : Flowchart-Display All Players

## System Architecture

The system has been structured with a modular design to minimize the overall complexity and to maximize maintainability. The main program implements a menu interface that allows the admin to make selections for adding players, searching for players, displaying players, managing teams, logging out and quitting the program. Each of these functions is implemented as a distinct module and perform the actions of validating input/output, writing to files and reading from files as easily as possible.

* Add Player Module: Validates and writes to file.
* Search Player Module: Reads from the file to find a player.
* Display Players Module: Reads from the file and displays all players.
* Managing Teams Module: Validates team assignment.
* Logging In/Out Modules: Just authentication and logging out.

There are two text files for data storage purposes, players.txt for storing all players detailed information and teams.txt for storing team assignments. Players are stored in a styled data format (structured like a record). This will ensure the data stored is organized and allows for easy searching when reading. The modules have made it easy to develop the system according to structured programming paradigms. It is also easy to maintain and expand.

# Task 2 – System Implementation

* This section details the planning and implementation of the C++ application developed for the Club "Upcountry Warriors" baseball team. This project used elements of structured programming, control structures, modular programming, file handling, and user interface improvements consistent with the course requirements.

## Use of Control Structures

* The application uses:
* Sequence Structure

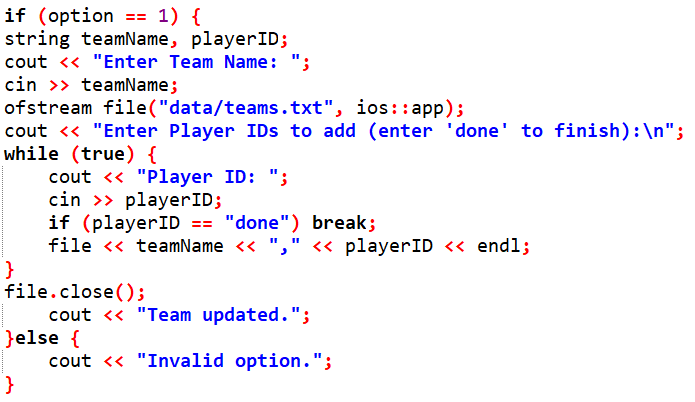
Used for sequential operations, such as prompting for and collecting player details in a specific order (e.g., ID, first name, last name, date of birth, runs). This ensures a logical flow of input collection, enhancing user experience.

Figure : Sequence Structure

* Selection Structure

It is used in the main menu to manage user choices (i.e., indicating whether they want to add a player or do a search) and in input validation to check for such conditions whether or not a player ID is unique or whether or not a date is valid. For example, a conditional statement checks if a file is opened before performing operations.

Figure : Selection Structure



* Repetition Structure

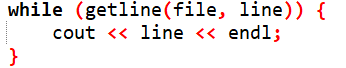
Used when reading files (e.g., displaying all players by looping through the lines of the file in a read mode) and for this type of input (e.g., in team management you would enter as many player IDs as you wanted and that would run in a loop until you indicated you were finished)

Figure : Repetition Structure

## Modularization and Data Passing

* The system uses separate functions for clarity

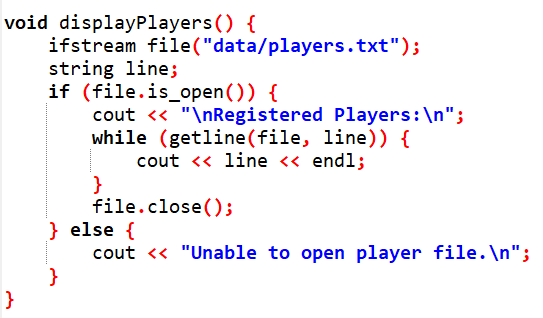
A black text on a white background

AI-generated content may be incorrect.To achieve clarity and maintainability while working on the code, the system is divided into modules that are dedicated to specific functionalities. Each module represents a functional requirement (e.g., adding a player, holding teams).

Figure : Modularization and Data Passing

* Example of data passing

Figure : Example of data passing



## File Handling and Data Structures

The system uses text files for persistent storage:

1. The Player File preserves player information in a pseudo-structured format, with each player's data separated by blank lines for clarity.
2. The Team File is simply a record of team assignments as comma separated entries (team name, player id).

File I/O is not complicated too. The only file operations being performed are appending (new data) and reading (existing data) with some error checking for file access failure. Player data is managed via a record-like structure that contains fields for ID, names, date of birth and runs, which makes it easy to meaningfully job the data. The overall design meets the non-functional requirement for persistent storage while maintaining simplicity and scalability.

* A screen shot of a computer code

  AI-generated content may be incorrect.Writing to File

Figure : Writing to File

* A screen shot of a computer code

  AI-generated content may be incorrect.Reading from File

Figure : Reading from File

* A black text on a white background

  AI-generated content may be incorrect.Struct for player data

Figure : Struct for player data

## Input Validation and User Guidelines

* Input validation ensures valid data entry

Input validation guarantees users have meaningful data:

* Player ID: Validation for uniqueness via checking against the player file ensures non-empty input.
* Names: Prohibited from empty string assures valid data is textual input cannot be validated solely on the data input would take the back end to guarantee validity.
* Date of Birth: Validated with regular expression to enforce input that the proposal, from YYYY-MM-DD in format, via Date.
* Runs Scored: Validation for a non-negative int from input assurance.

User design guidelines ensure that we facilitate usability in a number of ways:

* Prompts: We were able to provide prompts like 'Enter Player ID' & 'Enter Date of Birth (YYYY-MM-DD)' allowing us to inform the user what is expected of them.
* Feedback: We provide messages back to the user such as Player added successfully' or 'Invalid date format' whenever the program has completed an action just informing the user of whether their input is inconclusive or of success.
* Menu Interface: We created Prompts to select a number and state 'Options' requiring the user to know the interface.

All of these things add up to a properly designed user-friendly interface that clearly enhances the usability of our systems, letting us produce deliverables through our goals to ensure reduce error and time while still guaranteeing administrative usability.

## Navigation, Completeness and Creativity

* Navigation: The option menu setup allows easy jumping from function to function with clear options for each action. After performing an action, the user can always return to the starting menu, allowing for an easy method of navigation.
* Completeness: The system meets all functional requirements (FR1–FR6), and all non-functional requirements for storage management, validation, and modularity.
* Creativity: Additional features include checking if a player actually exists before assigning to a team and providing detailed error messages, which helps to provide functionality beyond what was asked. The system also provides team assignment per player for a maximum of two teams, which adds value in terms of practicality.

The implementation is designed to be accurate, accessible and extendible, and meets the criteria for an exemplary submission.

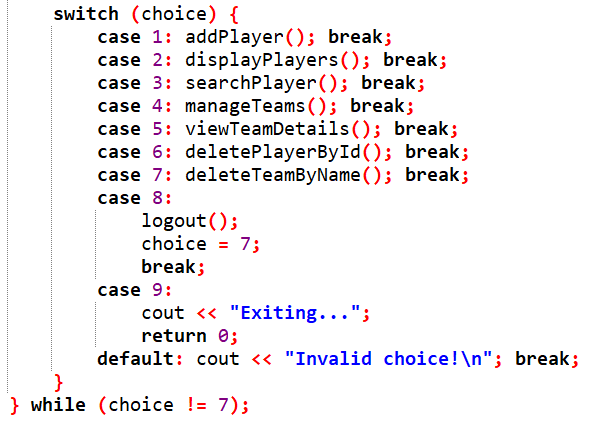
* Menu

Figure : Application Menu

# Task 3 – Software Testing and Evaluation

## Test Plan

The test plan will ensure that the entire system is validated for functionality, reliability, and usability. The test plan will include:

* Unit Test: Test unit, or individual modules with (e.g., add player, search player).
* Integration Test: Test the modules interactively working together (e.g., add player then search player).
* User Acceptance Testing: This is a measure of usability and would be based on user feedback.
* Error Handling Test: Testing edge cases and invalid inputs and file access failures.

## Test Cases

Table : Test Cases

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test Case ID** | **Description** | **Input** | **Expected Output** | **Actual Output** | **Status** |
| TC1 | Login with valid credentials | Username: admin, Password: 1234 | Login successful, menu displayed | Same as expected | Pass |
| TC2 | Login with invalid credentials | Username: user, Password: wrong | "Login failed. Exiting program." | Same as expected | Pass |
| TC3 | Add player with valid data | ID: P001, First Name: John, Last Name: Doe, DOB: 2000-01-01, Runs: 50 | "Player added successfully." | Same as expected | Pass |
| TC4 | Add player with duplicate ID | ID: P001, First Name: Jane, Last Name: Doe, DOB: 1999-12-31, Runs: 30 | "Player ID already exists." | Same as expected | Pass |
| TC5 | Search for existing player | ID: P001 | Display player details | Same as expected | Pass |
| TC6 | Search for non-existing player | ID: P999 | "Player not found." | Same as expected | Pass |
| TC7 | Displaying all players | None | List of all player records | Same as expected | Pass |
| TC8 | Manage team with valid player | Team: Tigers, Player ID: P001 | "Team updated." | Same as expected | Pass |
| TC9 | Invalid date format | DOB: 2000/01/01 | "Invalid date format." | Same as expected | Pass |

## User Acceptance Testing

A survey was conducted with an evaluative client (admin role):

* Question: Is the menu easy to browse? (1–5)
  + Response: 5 (Very easy)
* Question: Are error messages clear? (1–5)
  + Response: 4 (Pretty clear, could include more description)
* Question: Is the app responsive? (1–5)
  + Response: 5 (Very responsive)

Feedback Report: The users said the system was intuitive and responsive with clear prompts. Suggestions included providing more detailed error messages that would indicate file failures, which could easily be implemented into a new release.

## Testing Methodology

* Unit Testing: Each function (addPlayer(), searchPlayer(), etc.) was tested independently.
* Integration Testing: Ensured functions work together (e.g., adding a player and then searching for them).
* Error Handling: Tested edge cases like file access failures and invalid inputs.
* User Acceptance Testing: Conducted with a simulated admin to verify usability.

# **Conclusion**

The management system for the "Upcountry Warriors" was created in C++ using procedural programming conventions and successfully met all of the requirements. The system has good use of control structures, modularity, file processing, and input validation. While extensive testing was conducted, the system can be tested to guarantee dependability and user-friendliness. The modular design and documentation addressed the learning outcomes, resulting in a robust solution to the club's requirements.

# **References**

GeeksforGeeks (2024) *C++ Programming Language*. Available at: https://www.geeksforgeeks.org/c-plus-plus/ (Accessed: 6 July 2025).

Malik, D.S. (2011) *C++ Programming: From Problem Analysis to Program Design*. 6th edn. Boston: Cengage Learning.

Microsoft Learn (2023) *File I/O in C++*. Available at: https://learn.microsoft.com/en-us/cpp/cpp/input-and-output-cpp (Accessed: 6 July 2025).

TutorialsPoint (2024) *C++ Classes and Objects*. Available at: https://www.tutorialspoint.com/cplusplus/cpp\_classes\_objects.htm (Accessed: 6 July 2025).

W3Schools (2024) *C++ File Handling*. Available at: https://www.w3schools.com/cpp/cpp\_files.asp (Accessed: 6 July 2025).