PLAYER MANAGEMENT SYSTEM

Project Documentation (C Language)

---

1. Introduction

The **Player Management System** is a **menu-driven, console-based application** developed using the **C programming language**.
The system is designed to efficiently manage player records by implementing **real-world data management principles**, including:

- Structured data storage
- Dynamic memory allocation
- Controlled data visibility
- Automatic system auditing

The application supports full **CRUD operations**, advanced **search and sorting**, and **user-friendly data presentation**.

---

2. Objectives of the Project

- To manage player records efficiently using C
- To implement CRUD operations using structures and pointers
- To apply dynamic memory allocation using `malloc` and `realloc`
- To provide partial and case-insensitive search functionality
- To implement automatic system date tracking
- To improve user experience by hiding unnecessary system data
- To demonstrate real-world console-based application design

---

3. Technologies Used

Component	Description
Programming Language	C
Compiler	Dev C++
Platform	Console-based
Header Files	stdio.h, stdlib.h, string.h, time.h

4. Key Features of the System

4.1 Core Functionalities

- Add multiple player records
- Add a single player dynamically
- Display all players
- Display a specific player
- Update player details
- Delete player records
- Sort players based on performance

4.2 Advanced Enhancements

- Case-insensitive and partial name search
- Automatic system date assignment
- Automatic last-updated tracking
- User-friendly display (important data only)
- System-level data hidden from users
- Safe dynamic memory management

---

5. Data Structures Used

5.1 Date Structure

Stores date-related information:

- Day
- Month
- Year

5.2 Team Structure

Stores team-related details:

- Team ID
- Team Name
- Player Role
- Captain Status
- Active Status

5.3 PerformanceStats Structure

Stores performance-related data:

- Matches Played
- Total Score
- Best Performance Score

5.4 System Structure

Stores system-level metadata:

- Data upload date
- Last updated date
- Remarks
- Contract / contact value

5.5 Player Structure

Central structure that combines:

- Personal information
- Contact details
- Team details
- Performance statistics
- System metadata

---

6. Functional Modules

6.1 Add Players

Allows the user to add multiple players at once.

- Player ID is auto-generated
- System date and last updated date are set automatically

6.2 Add Single Player

- Dynamically increases memory using `realloc`
- Automatically assigns:
    - Player ID
    - System upload date
    - Last updated date

6.3 Display Player (User View)

The system **only displays important information** to users:

- Player ID
- Name
- Age
- Gender
- Jersey Number
- Team Name
- Role
- Captain Status
- Active Status
- Match statistics
- Last updated date

🔒 **System data remains hidden** to improve clarity and security.

6.4 Display Player (Admin/System View)

Internally, the system maintains:

- System upload date
- Last updated date
- Remarks
- Contract value

(This data is not shown to normal users.)

6.5 Search Player

Search can be performed using:

1. **Player ID**
2. **Player Name**
   - Case-insensitive
   - Partial name matching
   - Example: searching `Virat` finds `Virat Kohli`

6.6 Update Player

- User selects a **specific player record**
- Can update:
  - Personal information
  - Team information
  - Performance details
  - System remarks and contract value
- **Last updated date is automatically refreshed** using system time

6.7 Delete Player

- Deletes a selected player safely
- Remaining records are shifted
- Memory size is adjusted logically
- Handles empty data scenarios

6.8 Sort Player Data

Sorting options:

- By total score
- By matches played
- By best performance score

Each sort supports:

- Ascending order
- Descending order

---

7. System Date Management

The system uses `<time.h>` to:

- Automatically set **system upload date**
- Automatically update **last modified date** whenever data changes

This ensures **accurate auditing and tracking**.

---

8. Memory Management

- Dynamic memory allocation using `malloc`
- Memory resizing using `realloc`
- Safe deletion without memory leaks
- Efficient heap usage

---

9. Error Handling

- Handles invalid menu selections
- Displays appropriate messages if data is unavailable
- Prevents update/delete when record is not found
- Maintains system stability

10. User Experience Improvements

- Clean and readable output
- Hides unnecessary system data
- Displays only meaningful information to users
- Smooth menu-driven navigation

---

11. Limitations

- Data is stored in memory only (no file storage)
- Console-based interface
- Uses `gets()` (not recommended for production systems)

---

12. Future Enhancements

- File handling for permanent data storage
- Admin/User login system
- Input validation improvements
- GUI-based interface
- Role-based access control

---

13. Conclusion

The **Player Management System** successfully demonstrates:

- Practical use of C programming concepts
- Real-world data abstraction
- Dynamic memory management
- Clean user experience design

This project reflects **strong logical thinking**, **system design awareness**, and **industry-relevant practices**, making it an excellent academic and portfolio project.

---

14. Developer Details

Developer Name: Pritish Ramesh Pawar
Course: Java Full Stack
Project Type: Console Application Project (C Language)
Year: 2026