PlayerLinkedList.h

CODE:

```
#ifndef PLAYER_LINKED_LIST_H
#define PLAYER_LINKED_LIST_H

#include <iostream>
using namespace std;

struct Player {
    int id;
    string name;
    Player* next;
};

class PlayerLinkedList {
    Player* head;
public:
    PlayerLinkedList() { head = NULL; }

    void addPlayer(int id, string name) {
        Player* newNode = new Player{id, name, head};
        head = newNode;
    }

    void displayPlayers() {
        Player* temp = head;
        while (temp) {
            cout << "ID: " << temp->id << " Name: " << temp->name << endl;
```

```
            temp = temp->next;
        }
    }
};

#endif
```

MatchQueue.h

CODE:

```
#ifndef MATCH_QUEUE_H
#define MATCH_QUEUE_H

#include <iostream>
#include <queue>
using namespace std;

class MatchQueue {
   queue<string> matches;
public:
   void addMatch(string match) {
      matches.push(match);
   }

   void playMatch() {
      if (!matches.empty()) {
         cout << "Playing Match: " << matches.front() << endl;
         matches.pop();
      }
   }
};

#endif
```

ScoreStack.h

CODE:

```
#ifndef SCORE_STACK_H
#define SCORE_STACK_H

#include <iostream>
#include <stack>
using namespace std;

class ScoreStack {
   stack<int> scores;
public:
   void addScore(int score) {
      scores.push(score);
   }

   void undoScore() {
      if (!scores.empty()) {
         cout << "Undo Score: " << scores.top() << endl;
         scores.pop();
      }
   }
};

#endif
```

TournamentTree.h

CODE :

```
#ifndef TOURNAMENT_TREE_H
#define TOURNAMENT_TREE_H

#include <iostream>
using namespace std;

struct TreeNode {
   string match;
   TreeNode* left;
   TreeNode* right;
};

class TournamentTree {
public:
   TreeNode* createNode(string match) {
      TreeNode* node = new TreeNode{match, NULL, NULL};
      return node;
   }

   void display(TreeNode* root) {
      if (!root) return;
      cout << root->match << endl;
      display(root->left);
      display(root->right);
   }
};
```

```
#endif
```

MatchGraph.h

CODE:

```
#ifndef MATCH_GRAPH_H
#define MATCH_GRAPH_H

#include <iostream>
#include <map>
#include <vector>
using namespace std;

class MatchGraph {
   map<string, vector<string>> graph;
public:
   void addMatch(string p1, string p2) {
      graph[p1].push_back(p2);
      graph[p2].push_back(p1);
   }

   void display() {
      for (auto g : graph) {
         cout << g.first << " played with: ";
         for (string p : g.second)
            cout << p << " ";
         cout << endl;
      }
   }
};
```

```
#endif
```

PlayerHash.h

CODE:

```
#ifndef PLAYER_HASH_H
#define PLAYER_HASH_H

#include <iostream>
#include <unordered_map>
using namespace std;

class PlayerHash {
   unordered_map<int, int> stats;
public:
   void addStat(int id, int score) {
      stats[id] += score;
   }

   void displayStats() {
      for (auto s : stats) {
         cout << "Player ID: " << s.first
             << " Total Score: " << s.second << endl;
      }
   }
};

#endif
```

main.cpp

CODE:

```
#include "PlayerLinkedList.h"
#include "MatchQueue.h"
#include "ScoreStack.h"
#include "TournamentTree.h"
#include "MatchGraph.h"
#include "PlayerHash.h"

int main() {
    PlayerLinkedList players;
    players.addPlayer(1, "Alex");
    players.addPlayer(2, "John");
    players.displayPlayers();

    MatchQueue mq;
    mq.addMatch("Alex vs John");
    mq.playMatch();

    ScoreStack ss;
    ss.addScore(10);
    ss.undoScore();

    TournamentTree tree;
    TreeNode* root = tree.createNode("Final Match");
    tree.display(root);

    MatchGraph graph;
```

```
    graph.addMatch("Alex", "John");
    graph.display();

    PlayerHash stats;
    stats.addStat(1, 50);
    stats.displayStats();

    return 0;
}
```

README.md

CODE:

Gaming Tournament Management System

Mini Project using Data Structures in C++

Data Structures Used

- Linked List (Players)
- Queue (Match Scheduling)
- Stack (Score Undo)
- Tree (Tournament Structure)
- Graph (Match Relationships)
- Hashing (Player Statistics)

How to Run

g++ main.cpp -o tournament

./tournament