

/\*Code about player’s entity\*/

#include <stdio.h>

#include <stdbool.h>

#include <string.h>

// Structure to represent a player

struct Player {

char name[50];

int age;

char gender[10];

char location[50];

bool international;

bool singlesParticipant;

bool doublesParticipant;

bool mixedDoublesParticipant;

};

// Function to validate player's age

bool validateAge(int age) {

return (age >= 18 && age <= 25);

}

// Function to validate player's eligibility based on location and international status

bool validateEligibility(const char \*location, bool international) {

// Implement logic to check if the player is from India or living in India for more than a year

// Return true if eligible, false otherwise

// Example: (logic to check location and international status)

return (strcmp(location, "India") == 0 || international);

}

// Function to handle player entries

void enterPlayers(struct Player players[], int numPlayers) {

for (int i = 0; i < numPlayers; ++i) {

printf("\nEnter details for Player %d:\n", i + 1);

printf("Name: ");

scanf("%s", players[i].name);

printf("Age: ");

scanf("%d", &players[i].age);

// Other details (gender, location, international status) input handling goes here

// Validate age and eligibility here

// Example:

printf("Gender: ");

scanf("%s", players[i].gender);

printf("Location: ");

scanf("%s", players[i].location);

printf("International status (1 for yes, 0 for no): ");

scanf("%d", &players[i].international);

if (!validateAge(players[i].age)) {

printf("Invalid age. Player must be between 18-25 years.\n");

--i; // Re-enter details for the same player

continue;

}

if (!validateEligibility(players[i].location, players[i].international)) {

printf("Invalid eligibility. Participants must be from India or living in India for more than a year.\n");

--i; // Re-enter details for the same player

continue;

}

// Additional logic to handle entry for singles, doubles, mixed doubles, fees, documents, etc.

players[i].singlesParticipant = true; // For example, assuming every player participates in singles

players[i].doublesParticipant = false;

players[i].mixedDoublesParticipant = false;

}

}

// Main function

int main() {

const int totalPlayers = 60;

struct Player allPlayers[totalPlayers];

printf("Welcome to the Badminton Tournament Administration Software!\n");

// Enter player details

enterPlayers(allPlayers, totalPlayers);

// Additional functionality for handling teams, sponsors, tournament scheduling, etc. can be added here

return 0;

}

/\*Code for administration work\*/

#include <stdio.h>

#include <stdbool.h>

#include <string.h>

#include <stdlib.h>

#include <time.h>

// Structure to represent a match

struct Match {

int courtNumber;

int player1Index;

int player2Index;

// Additional match details can be added here

};

// Function to generate fixture on a random basis

void generateFixture(struct Match allplayers[], int numPlayers, struct Match matches[], int numMatches) {

// Logic to generate random matches, ensuring no clashes for a player

// Example: (Generate matches randomly)

srand(time(0));

int i;

for (i = 0; i < numMatches; ++i) {

matches[i].courtNumber = (rand() % 10) + 1; // Randomly assign a court number

matches[i].player1Index = rand() % numPlayers;

matches[i].player2Index = rand() % numPlayers;

// Ensure players don't play more than one match in a day (for demonstration purposes)

// Actual implementation needs more sophisticated logic considering schedule and time

while (matches[i].player1Index == matches[i].player2Index) {

matches[i].player2Index = rand() % numPlayers;

}

}

}

// Function to calculate sports material requirement per match and overall total requirement

void calculateMaterialRequirement(struct Match matches[], int numMatches) {

// Logic to calculate sports material requirement per match and overall

// Example: (for demonstration purposes)

int shuttlecockPerMatch = 6; // Assuming 6 shuttlecocks per match

int totalShuttlecocks = shuttlecockPerMatch \* numMatches;

printf("Total shuttlecocks required for all matches: %d\n", totalShuttlecocks);

// Similar calculations for other sports materials can be added

}

// Function to fix dining for players with a menu for 20 days

void fixDining(struct Match allplayers[], int numPlayers) {

// Logic to fix dining arrangements and menu for 20 days

// Example: (for demonstration purposes)

printf("Dining arrangements and menu for players for 20 days:\n");

// Display menu or set up dining schedule for 20 days

}

// Main function

int main() {

// Assuming player details are already entered and available in the 'allPlayers' array

const int totalplayers=60;

const int totalMatches = 40; // Assuming 40 matches over 20 days (2 matches per day)

struct Match Matches[totalMatches];

struct Match allPlayers[totalplayers];

// Generate fixture

generateFixture(allPlayers,totalplayers, Matches, totalMatches);

// Calculate sports material requirement

calculateMaterialRequirement(Matches, totalMatches);

// Fix dining arrangements for players

fixDining(allPlayers, totalplayers);

return 0;

}

/\* code for sponsors\*/

#include <stdio.h>

#include <stdbool.h>

#include <stdlib.h>

// Structure to represent a sponsor

struct Sponsor {

char name[50];

float contribution; // Amount contributed by the sponsor

int publicityPoints;

int courtNumber;

int player1Index;

int player2Index;

};

// Function to handle sponsor entries

void sponsorEntries(struct Sponsor sponsors[], int numSponsors)

{

int i;

for (i = 0; i < numSponsors; ++i) {

printf("\nEnter details for Sponsor %d:\n", i + 1);

printf("Name: ");

scanf("%s", sponsors[i].name);

printf("Contribution amount (in Rs.): ");

scanf("%f", &sponsors[i].contribution);

// Assigning publicity points based on the contribution (for demonstration purposes)

// Actual logic for assigning points can be based on specific criteria

sponsors[i].publicityPoints = sponsors[i].contribution / 1000; // Assuming 1000 Rs. = 1 publicity point

}

}

// Function to save sponsor details and generate a detailed report

void saveSponsorDetails(struct Sponsor sponsors[], int numSponsors, struct Sponsor matches[], int numMatches) {

// Logic to save sponsor details and generate a detailed report

// Example: (for demonstration purposes)

printf("\nDetailed Report:\n");

printf("Sponsor Entries and Publicity Points:\n");

int i;

for (i = 0; i < numSponsors; ++i) {

printf("Sponsor Name: %s, Contribution: Rs. %.2f, Publicity Points: %d\n",

sponsors[i].name, sponsors[i].contribution, sponsors[i].publicityPoints);

}

printf("\nSchedule with Match Details:\n");

for (i = 0; i < numMatches; ++i) {

printf("Match %d - Court: %d, Player 1: %d, Player 2: %d\n",

i + 1, matches[i].courtNumber, matches[i].player1Index, matches[i].player2Index);

// Additional match details can be added here

}

// Calculate prize distribution based on the total sponsors' money

float totalSponsorsMoney = 0;

for (i = 0; i < numSponsors; ++i) {

totalSponsorsMoney += sponsors[i].contribution;

}

float totalPrizeMoney = totalSponsorsMoney / 2;

printf("\nPrize Distribution:\n");

printf("Total Prize Money: Rs. %.2f\n", totalPrizeMoney);

printf("Winner: Rs. %.2f\n", totalPrizeMoney \* 0.6); // Assuming 60% for the winner

printf("Silver: Rs. %.2f\n", totalPrizeMoney \* 0.3); // Assuming 30% for the silver

printf("Bronze: Rs. %.2f\n", totalPrizeMoney \* 0.1); // Assuming 10% for the bronze

// Similarly, calculate and print prizes for doubles and other categories

}

// Main function

int main() {

// Assuming sponsors' details are already entered and available in the 'allSponsors' array

const int totalSponsors = 5; // Assuming 5 sponsors

struct Sponsor allSponsors[totalSponsors];

// Assuming match details are available in the 'allMatches' array

const int totalMatches = 40; // Assuming 40 matches

struct Sponsor allMatches[totalMatches];

// Assuming sponsors' entries are invited and collected

sponsorEntries(allSponsors, totalSponsors);

// Saving sponsor details and generating a detailed report

saveSponsorDetails(allSponsors, totalSponsors, allMatches, totalMatches);

return 0;

}

/\* Code for final selection team\*/

#include <stdio.h>

#include <stdbool.h>

#include <stdlib.h>

// Structure to represent a team

struct Team {

int player1Index;

int player2Index;

int courtNumber

// Additional team details can be added here

};

// Function to update fixtures regularly by the administrator

void updateFixtures(struct Team matches[], int numMatches) {

// Logic to update fixtures based on the administrator's input

// Example: (for demonstration purposes, assuming new matches are entered manually)

printf("Please enter updated fixtures:\n");

int i;

for (i = 0; i < numMatches; ++i) {

printf("Enter details for Match %d - Court: ", i + 1);

scanf("%d", &matches[i].courtNumber);

printf("Enter Player 1 Index: ");

scanf("%d", &matches[i].player1Index);

printf("Enter Player 2 Index: ");

scanf("%d", &matches[i].player2Index);

}

}

// Function to select teams for quarter-finals, semi-finals, and finals based on updated fixtures

void selectTeams(struct Team matches[], int numMatches, struct Team players[], int numPlayers) {

// Logic to select teams for quarter-finals, semi-finals, and finals based on updated fixtures

// Example: (for demonstration purposes, assuming simple selection based on match results)

printf("\nSelecting teams for quarter-finals, semi-finals, and finals:\n");

// Assume winners from matches progress to the next round

// (In a real scenario, results would be evaluated)

struct Team quarterFinalsTeams[4];

struct Team semiFinalsTeams[2];

struct Team finalsTeam;

// For demonstration purposes, assume the first 4 matches are for quarter-finals

int i;

for (i = 0; i < 4; ++i) {

quarterFinalsTeams[i].player1Index = matches[i].player1Index;

quarterFinalsTeams[i].player2Index = matches[i].player2Index;

}

// Assume winners from quarter-finals move to semi-finals

// (In a real scenario, results would be evaluated)

for (i = 0; i < 2; ++i) {

semiFinalsTeams[i].player1Index = quarterFinalsTeams[i].player1Index;

semiFinalsTeams[i].player2Index = quarterFinalsTeams[i].player2Index;

}

// Assume winners from semi-finals move to the finals

// (In a real scenario, results would be evaluated)

finalsTeam.player1Index = semiFinalsTeams[0].player1Index;

finalsTeam.player2Index = semiFinalsTeams[0].player2Index;

// Display selected teams

printf("\nTeams for Quarter-Finals:\n");

for (i = 0; i < 4; ++i) {

printf("Team %d: Player %d and Player %d\n", i + 1, quarterFinalsTeams[i].player1Index, quarterFinalsTeams[i].player2Index);

}

printf("\nTeams for Semi-Finals:\n");

for (i = 0; i < 2; ++i) {

printf("Team %d: Player %d and Player %d\n", i + 1, semiFinalsTeams[i].player1Index, semiFinalsTeams[i].player2Index);

}

printf("\nTeams for Finals:\n");

printf("Finals Team: Player %d and Player %d\n", finalsTeam.player1Index, finalsTeam.player2Index);

}

// Main function

int main() {

const int totalMatches = 40; // Assuming 40 matches

struct Team allMatches[totalMatches];

// Assume player details are already entered and available in the 'allPlayers' array

const int totalPlayers = 60; // Assuming 60 players

struct Team allPlayers[totalPlayers];

// Update fixtures by the administrator

updateFixtures(allMatches, totalMatches);

// Select teams for quarter-finals, semi-finals, and finals based on updated fixtures

selectTeams(allMatches, totalMatches, allPlayers, totalPlayers);

return 0;

}

/\* Code for badminton tournament\*/

#include <stdio.h>

#include <stdbool.h>

#include <string.h>

// Structure to represent a player

struct Player {

char name[50];

int age;

char gender[10];

char location[50];

bool international;

bool singlesParticipant;

bool doublesParticipant;

bool mixedDoublesParticipant;

};

// Structure to represent a match

struct Match {

int courtNumber;

int player1Index;

int player2Index;

// Additional match details can be added here

};

// Structure to represent a sponsor

struct Sponsor {

char name[50];

float contribution; // Amount contributed by the sponsor

int publicityPoints;

};

// Structure to represent a team

struct Team {

int player1Index;

int player2Index;

// Additional team details can be added here

};

// Function to validate player's age

bool validateAge(int age) {

return (age >= 18 && age <= 25);

}

// Function to validate player's eligibility based on location and international status

bool validateEligibility(const char \*location, bool international) {

// Implement logic to check if the player is from India or living in India for more than a year

// Return true if eligible, false otherwise

// Example: (logic to check location and international status)

return (strcmp(location, "India") == 0 || international);

}

// Function to handle player entries

void enterPlayers(struct Player players[], int numPlayers) {

int i;

for ( i = 0; i < numPlayers; ++i) {

printf("\nEnter details for Player %d:\n", i + 1);

printf("Name: ");

scanf("%s", players[i].name);

printf("Age: ");

scanf("%d", &players[i].age);

// Other details (gender, location, international status) input handling goes here

// Validate age and eligibility here

// Example:

printf("Gender: ");

scanf("%s", players[i].gender);

printf("Location: ");

scanf("%s", players[i].location);

printf("International status (1 for yes, 0 for no): ");

scanf("%d", &players[i].international);

if (!validateAge(players[i].age)) {

printf("Invalid age. Player must be between 18-25 years.\n");

--i; // Re-enter details for the same player

continue;

}

if (!validateEligibility(players[i].location, players[i].international)) {

printf("Invalid eligibility. Participants must be from India or living in India for more than a year.\n");

--i; // Re-enter details for the same player

continue;

}

// Additional logic to handle entry for singles, doubles, mixed doubles, fees, documents, etc.

players[i].singlesParticipant = true; // For example, assuming every player participates in singles

players[i].doublesParticipant = false;

players[i].mixedDoublesParticipant = false;

}

}

// Function to generate fixture on a random basis

void generateFixture(struct Player players[], int numPlayers, struct Match matches[], int numMatches) {

// Logic to generate random matches, ensuring no clashes for a player

// Example: (Generate matches randomly)

srand(time(0));

int i;

for (i = 0; i < numMatches; ++i) {

matches[i].courtNumber = (rand() % 10) + 1; // Randomly assign a court number

matches[i].player1Index = rand() % numPlayers;

matches[i].player2Index = rand() % numPlayers;

// Ensure players don't play more than one match in a day (for demonstration purposes)

// Actual implementation needs more sophisticated logic considering schedule and time

while (matches[i].player1Index == matches[i].player2Index) {

matches[i].player2Index = rand() % numPlayers;

}

}

}

// Function to calculate sports material requirement per match and overall total requirement

void calculateMaterialRequirement(struct Match matches[], int numMatches) {

// Logic to calculate sports material requirement per match and overall

// Example: (for demonstration purposes)

int shuttlecockPerMatch = 6; // Assuming 6 shuttlecocks per match

int totalShuttlecocks = shuttlecockPerMatch \* numMatches;

printf("Total shuttlecocks required for all matches: %d\n", totalShuttlecocks);

// Similar calculations for other sports materials can be added

}

// Function to fix dining for players with a menu for 20 days

void fixDining(struct Player players[], int numPlayers) {

// Logic to fix dining arrangements and menu for 20 days

// Example: (for demonstration purposes)

printf("Dining arrangements and menu for players for 20 days:\n");

// Display menu or set up dining schedule for 20 days

}

// Function to handle sponsor entries

void sponsorEntries(struct Sponsor sponsors[], int numSponsors) {

int i;

for (i = 0; i < numSponsors; ++i) {

printf("\nEnter details for Sponsor %d:\n", i + 1);

printf("Name: ");

scanf("%s", sponsors[i].name);

printf("Contribution amount (in Rs.): ");

scanf("%f", &sponsors[i].contribution);

// Assigning publicity points based on the contribution (for demonstration purposes)

// Actual logic for assigning points can be based on specific criteria

sponsors[i].publicityPoints = sponsors[i].contribution / 1000; // Assuming 1000 Rs. = 1 publicity point

}

}

// Function to save sponsor details and generate a detailed report

void saveSponsorDetails(struct Sponsor sponsors[], int numSponsors, struct Match matches[], int numMatches) {

// Logic to save sponsor details and generate a detailed report

// Example: (for demonstration purposes)

printf("\nDetailed Report:\n");

printf("Sponsor Entries and Publicity Points:\n");

int i;

for (i = 0; i < numSponsors; ++i) {

printf("Sponsor Name: %s, Contribution: Rs. %.2f, Publicity Points: %d\n",

sponsors[i].name, sponsors[i].contribution, sponsors[i].publicityPoints);

}

printf("\nSchedule with Match Details:\n");

for (i = 0; i < numMatches; ++i) {

printf("Match %d - Court: %d, Player 1: %d, Player 2: %d\n",

i + 1, matches[i].courtNumber, matches[i].player1Index, matches[i].player2Index);

// Additional match details can be added here

}

// Calculate prize distribution based on the total sponsors' money

float totalSponsorsMoney = 0;

for (i = 0; i < numSponsors; ++i) {

totalSponsorsMoney += sponsors[i].contribution;

}

float totalPrizeMoney = totalSponsorsMoney / 2;

printf("\nPrize Distribution:\n");

printf("Total Prize Money: Rs. %.2f\n", totalPrizeMoney);

printf("Winner: Rs. %.2f\n", totalPrizeMoney \* 0.6); // Assuming 60% for the winner

printf("Silver: Rs. %.2f\n", totalPrizeMoney \* 0.3); // Assuming 30% for the silver

printf("Bronze: Rs. %.2f\n", totalPrizeMoney \* 0.1); // Assuming 10% for the bronze

// Similarly, calculate and print prizes for doubles and other categories

}

// Function to update fixtures regularly by the administrator

void updateFixtures(struct Match matches[], int numMatches) {

// Logic to update fixtures based on the administrator's input

// Example: (for demonstration purposes, assuming new matches are entered manually)

printf("Please enter updated fixtures:\n");

int i;

for (i = 0; i < numMatches; ++i) {

printf("Enter details for Match %d - Court: ", i + 1);

scanf("%d", &matches[i].courtNumber);

printf("Enter Player 1 Index: ");

scanf("%d", &matches[i].player1Index);

printf("Enter Player 2 Index: ");

scanf("%d", &matches[i].player2Index);

}

}

// Function to select teams for quarter-finals, semi-finals, and finals based on updated fixtures

void selectTeams(struct Match matches[], int numMatches, struct Player players[], int numPlayers) {

// Logic to select teams for quarter-finals, semi-finals, and finals based on updated fixtures

// Example: (for demonstration purposes, assuming simple selection based on match results)

printf("\nSelecting teams for quarter-finals, semi-finals, and finals:\n");

// Assume winners from matches progress to the next round

// (In a real scenario, results would be evaluated)

struct Team quarterFinalsTeams[4];

struct Team semiFinalsTeams[2];

struct Team finalsTeam;

int i;

// For demonstration purposes, assume the first 4 matches are for quarter-finals

for (i = 0; i < 4; ++i) {

quarterFinalsTeams[i].player1Index = matches[i].player1Index;

quarterFinalsTeams[i].player2Index = matches[i].player2Index;

}

// Assume winners from quarter-finals move to semi-finals

// (In a real scenario, results would be evaluated)

for (i = 0; i < 2; ++i) {

semiFinalsTeams[i].player1Index = quarterFinalsTeams[i].player1Index;

semiFinalsTeams[i].player2Index = quarterFinalsTeams[i].player2Index;

}

// Assume winners from semi-finals move to the finals

// (In a real scenario, results would be evaluated)

finalsTeam.player1Index = semiFinalsTeams[0].player1Index;

finalsTeam.player2Index = semiFinalsTeams[0].player2Index;

// Display selected teams

printf("\nTeams for Quarter-Finals:\n");

for (i = 0; i < 4; ++i) {

printf("Team %d: Player %d and Player %d\n", i + 1, quarterFinalsTeams[i].player1Index, quarterFinalsTeams[i].player2Index);

}

printf("\nTeams for Semi-Finals:\n");

for (i = 0; i < 2; ++i) {

printf("Team %d: Player %d and Player %d\n", i + 1, semiFinalsTeams[i].player1Index, semiFinalsTeams[i].player2Index);

}

printf("\nTeams for Finals:\n");

printf("Finals Team: Player %d and Player %d\n", finalsTeam.player1Index, finalsTeam.player2Index);

}

// Main function

int main()

{

const int totalPlayers = 60;

struct Player allPlayers[totalPlayers];

printf("Welcome to the Badminton Tournament Administration Software!\n");

// Enter player details

enterPlayers(allPlayers, totalPlayers);

// Additional functionality for handling teams, sponsors, tournament scheduling, etc. can be added here

// Assuming player details are already entered and available in the 'allPlayers' array

const int totalMatches = 40; // Assuming 40 matches over 20 days (2 matches per day)

struct Match allMatches[totalMatches];

// Generate fixture

generateFixture(allPlayers, 60, allMatches, totalMatches);

// Calculate sports material requirement

calculateMaterialRequirement(allMatches, totalMatches);

// Fix dining arrangements for players

fixDining(allPlayers, 60);

// Assuming sponsors' details are already entered and available in the 'allSponsors' array

const int totalSponsors = 5; // Assuming 5 sponsors

struct Sponsor allSponsors[totalSponsors];

// Assuming sponsors' entries are invited and collected

sponsorEntries(allSponsors, totalSponsors);

// Saving sponsor details and generating a detailed report

saveSponsorDetails(allSponsors, totalSponsors, allMatches, totalMatches);

// Update fixtures by the administrator

updateFixtures(allMatches, totalMatches);

// Select teams for quarter-finals, semi-finals, and finals based on updated fixtures

selectTeams(allMatches, totalMatches, allPlayers, totalPlayers);

return 0;

}