IPS Instituto Politécnico de Setúbal
Escola Superior de Tecnologia de Setúbal

Mini-Projecto ATAD (2017/2018)

Alunos:

- Gabriel Ambrósio, nº 160221013
- Hugo Ferreira, nº 160221039

Turma: Lab 6ª 16H30

Docente: Prof. Aníbal Ponte

Índice

a) Descrição estruturas de dados usadas e respetivas implementações:

Existem neste projecto 6 estruturas de dados, sendo elas:

A estrutura de dados Date é composta por 3 dados de tipo inteiro que vão guardar o dia, o mês e o ano:

```
1    typedef struct date {
2        unsigned int day, month, year;
3    }Date;
```

A estrutura de dados Player é composta por 3 dados do tipo char que vão guardar o nome, a equipa e o género, é também composta por um tipo de dados inteiro que vai guardar o id e ainda um do tipo Date que vai guardar uma data de nascimento :

```
5    typedef struct player {
6        unsigned int id;
7        char name[50];
8        char team[50];
9        Date birthDate;
10       char gender;
11   }Player;
```

A estrutura de dados Statistics é composta por 5 dados do tipo inteiro que vão guardar o número de ponto duplos, o número de pontos triplos, o número de assistências, o número de faltas e o número de bloqueios:

```
1    typedef struct statistics {
2        unsigned int twoPoints, threePoints, assists, fouls, blocks;
3    }Statistics;
```

A estrutura de dados PlayerGameStatistics é composta por 2 dados do tipo inteiro que vão guardar o id de um jogador e o id de um jogo, e ainda um do tipo Statistics :

```
7    typedef struct playerGameStatistics {
8        unsigned int idPlayer, idGame;
9        Statistics statistics;
10   }PlayerGameStatistics;
```

E os ponteiros de Player e de Statistics:

```
5    typedef Statistics* PtStatistics;
```

```
13   typedef Player* PtPlayer;
```

No que diz respeitos às suas implementações, existem também 6:

```
15    Player createPlayer(unsigned int id, char name[], char team[], Date birthDate, char gender);
16    Date createDate(unsigned int day, unsigned int month, unsigned int year);
17    void printPlayer(PtPlayer _this);
18    Statistics createStatistics(unsigned int twoPoints, unsigned int threePoints, unsigned int assists, unsigned int fouls, unsigned int blocks);
19    PlayerGameStatistics createplayerGameStatistics(unsigned int idPlayer, unsigned int idGame, Statistics statistics);
20    void printStatistics(PtStatistics _this);
```

A implementação createPlayer é do tipo Player e é utilizada para criar um Player onde recebe como argumentos o id, o nome, a equipa, uma Data e um género;

A implementação createDate é do tipo Date e é utilizada para criar uma Date onde recebe como argumentos o dia, o mês e o ano;

A implementação printPlayer é do tipo void e é utilizada para imprimir as informações de um Player;

A implementação createStatistics é do tipo Statistics e é utilizada para criar um Statistics onde recebe como argumentos o número de ponto duplos, o número de pontos triplos, o número de assistências, o número de faltas e o número de bloqueios.

A implementação createplayerGameStatistics é do tipo PlayerGameStatistics e é utilizada para criar um PlayerGameStatistics onde recebe como argumentos um id de um Player, o id de um jogo e um Statistics;

A implementação printStatistics é do tipo void e é utilizada para imprimir as informações de um Statistics;

b) O código/função respeitante e complexidade algorítmica

COMANDO SHOWP

```
403  void showP(Player players[]) {            //FUNCIONA
404      printf("Chose order:\n1 - Sort [A-Z]\n2 - Sort [Z-A]\n");
405      int order;
406      printf("Option> ");
407      scanf_s("%d", &order);
408      //fgets(order, sizeof(order), stdin);
409
410      if (order == 1) {
411          sortPlayersAZ(players);
412      } else if (order == 2) {
413          sortPlayersZA(players);
414      } else {
415          printf("Invalid Input...");
416          return;
417      }
418      printf("Player\t| Name\t\t\t| Team\t\t\t| BirthDate\t| Gender |\n");
419      printf("=================================================================================\n");
420      printArray(players);
421  }
```

Complexidade algorítmica: linear

Como funções auxiliares temos:

```
184  void sortPlayersAZ(Player players[]) {//selectionS
185      for (int i = 0; i < 300; i++) {
186          Player minP = players[i];
187          int minPindex = i;
188          for (int j = i; j < 300; j++) {
189              if (strcmp(players[j].name, minP.name) < 0) {
190                  minP = players[j];
191                  minPindex = j;
192              }
193          }
194          swap(&players[i], &players[minPindex]);
195      }
196  }
197
198  void sortPlayersZA(Player players[]) {//selectionS
199      for (int i = 0; i < 300; i++) {
200          Player minP = players[i];
201          int minPindex = i;
202          for (int j = i; j < 300; j++) {
203              if (strcmp(players[j].name, minP.name) > 0) {
204                  minP = players[j];
205                  minPindex = j;
206              }
207          }
208          swap(&players[i], &players[minPindex]);
209      }
210  }
211
```

Que organiza o array de players como o utilizador escolher (A-Z/Z-A) usando o selection sort dado nas aulas. Estes usam o método swap:

```
178  void swap(Player *x, Player *y) {//swaps two elements in array
179      Player aux = *x;
180      *x = *y;
181      *y = aux;
182  }
```

COMANDO TABLE

```c
void table(Player players[]){         //FUNCIONA
    int matrix[4][2];
    int sub14M=0, sub16M=0, sub18M=0, senioresM=0, sub14F=0, sub16F=0, sub18F=0, senioresF=0;

    countPlayers(players, &sub14M, &sub16M, &sub18M, &senioresM, &sub14F, &sub16F, &sub18F, &senioresF);

    matrix[0][0] = sub14F;
    matrix[0][1] = sub14M;
    matrix[1][0] = sub16F;
    matrix[1][1] = sub16M;
    matrix[2][0] = sub18F;
    matrix[2][1] = sub18M;
    matrix[3][0] = senioresF;
    matrix[3][1] = senioresM;

    char *escalao[4] = {"Sub 14","Sub 16","Sub18","Seniores"};
    printf("Level/Genre|\tFemale\tMale\n");
    printf("============================\n");
    for (int i = 0; i < 4; i++) {
        printf("%-11s|\t", escalao[i]);
        for (int j = 0; j < 2; j++) {
            printf("%d\t", matrix[i][j]);
        }
        printf("\n----------------------------\n");
    }
}
```

Complexidade algorítmica: constante

Como funções auxiliares temos:

```c
void countPlayers(Player players[], int *sub14M, int *sub16M, int *sub18M, int *senioresM, int *sub14F, int *sub16F, int *sub18F, int *senioresF) {
    for (int i = 0; i < 300; i++) {//conta os players pelas idades e divide-os por genero
        if (players[i].birthDate.year >= 2004) {
            if (players[i].gender == 'M') {
                (*sub14M)++;
            }
            else {
                (*sub14F)++;
            }
        }
        else if (players[i].birthDate.year >= 2002) {
            if (players[i].gender == 'M') {
                (*sub16M)++;
            }
            else {
                (*sub16F)++;
            }
        }
        else if (players[i].birthDate.year >= 2000) {
            if (players[i].gender == 'M') {
                (*sub18M)++;
            }
            else {
                (*sub18F)++;
            }
        }
        else {
            if (players[i].gender == 'M') {
                (*senioresM)++;
            }
            else {
                (*senioresF)++;
            }
        }
    }
}
```


Que faz o cálculo do número de jogadores por escalão e género, retorna esse valor por referência.

COMANDO SEARCH

```
450  void search(Player players[]) {
451      printf("Name of the team?:");
452      char team[50];
453      fgets(team, sizeof(team), stdin);
454      int aux = 0;
455      for (int i = 0; i < 300; i++) {
456          if (strcmp("Imortal", players[i].team) == 0) {//nao funciona com o input do utilizador
457              aux = 1;
458              printPlayer(&players[i]);
459          }
460      }
461      if (aux == 0) {
462          printf("NONEXISTENG TEAM\n");
463      }
464  }
```

Complexidade algorítmica: linear

COMANDO SEARCHG

```
512  void searchG(PlayerGameStatistics pgs[]) {                    //FUNCIONA
513      int idGame;
514      printf("ID of the game?>");
515      scanf_s("%d", &idGame);
516
517      int sumOfPoints=0, sumOfPlayers=0, sumOfBlocks=0;
518      sumOfPointsOfGame(pgs, idGame, &sumOfPoints);
519      sumOfPlayersInAGame(pgs, idGame, &sumOfPlayers);
520      sumOfBlocksInAGame(pgs, idGame, &sumOfBlocks);
521      if (sumOfBlocks > 0 && sumOfPlayers > 0 && sumOfPoints > 0) {
522          printf("-----------------------------------------------------\n");
523          printf("Total number of points %d\n", sumOfPoints);
524          printf("-----------------------------------------------------\n");
525          printf("Total number of blocks %d\n", sumOfBlocks);
526          printf("-----------------------------------------------------\n");
527          printf("Total number of used players %d\n", sumOfPlayers);
528          printf("-----------------------------------------------------\n");
529      }else{
530          printf("Invalid Game");
531      }
532  }
```

Complexidade algorítmica: constante

Como funções auxiliares temos:

```c
void sumOfPointsOfGame(PlayerGameStatistics pgs[], int idGameToSum, int *sum) {//soma dos pontos de um determinado jogo
    for (int i = 0; i < 517; i++) {
        if (pgs[i].idGame == idGameToSum) {
            (*sum) += (pgs[i].statistics.twoPoints * 2);
            (*sum) += (pgs[i].statistics.threePoints * 3);
        }
    }
}
void sumOfPlayersInAGame(PlayerGameStatistics pgs[], int idGameToSum, int *sum) {//soma de jogadores num determinado jogo
    for (int i = 0; i < 517; i++) {
        if (pgs[i].idGame == idGameToSum) {
            (*sum)++;
        }
    }
}
void sumOfBlocksInAGame(PlayerGameStatistics pgs[], int idGameToSum, int *sum) {//soma de blocos num determinado jogo
    for (int i = 0; i < 517; i++) {
        if (pgs[i].idGame == idGameToSum) {
            (*sum) += pgs[i].statistics.blocks;
        }
    }
}
```

Esta três funções calculam o número de pontos, número de jogadores e número de blocos de um determinado jogo, respetivamente. Todos os valores finais são passados por referência.

COMANDO MVP

```c
void mvp(PlayerGameStatistics pgs[], Player p[]) {                  //FUNCIONA
    int idGame;
    printf("ID of the game?:");
    scanf_s("%d", &idGame);
    printf("O melhor jogador em campo tem o ID %d e um MVP = %d", calculateMPVId(pgs, idGame), calculateMPV(pgs, idGame));
}
```

Complexidade algorítmica: constante

Como funções auxiliares temos:

```c
int calculateMvpValue(PlayerGameStatistics pgs) {//calcular o valor individual mvp de um jogador num jogo
    return (3 * pgs.statistics.threePoints) + (2 * pgs.statistics.twoPoints) + pgs.statistics.assists + (2 * pgs.statistics.blo
}

int calculateMPV(PlayerGameStatistics pgs[], int idGame) {//calcula o mvp de um jogo e retorna o valor mvp

    int classification = 0;
    for (int i = 0; i < 517; i++) {
        if (pgs[i].idGame == idGame) {
            if (calculateMvpValue(pgs[i]) > classification) {
                classification = calculateMvpValue(pgs[i]);
            }
        }
    }
    return classification;
}

int calculateMPVId(PlayerGameStatistics pgs[], int idGame) {//calcula o mvp e retorna o seu id

    int classification = 0;
    int mvp = 0;
    for (int i = 0; i < 517; i++) {
        if (pgs[i].idGame == idGame) {
            if (calculateMvpValue(pgs[i]) > classification) {
                classification = calculateMvpValue(pgs[i]);
                mvp = pgs[i].idPlayer;
            }
        }
    }

    return mvp;
}
```

Para a função MVP, decidimos dividir as tarefas, uma simples calculateMvpValue que usa a formula dada do enunciado para calcular o valor individual de um jogador, esta e usada nas outras duas. O calculateMVP que calcula qual e o maior valor individual (MVP), e retorna-o. A ultima função usa o mesmo método anterior, mas retorna o ID do MVP.

COMANDO MFOULP

```
543  void mFoulP(PlayerGameStatistics pgs[], Player p[]) {                  //FUNCIONA
544      printf("FOULS AVERAGE PER PLAYER\n\n");
545      printf("Player Name\t    | #Played Games | Av. Fouls\n");
546      printf("================================================\n");
547
548      for (int i = 0; i < 300; i++) {
549          if (numberOfGamesOfPlayer(pgs, p[i]) > 0) {
550              printf("%-20s|       %d\t    | %.2f\n", p[i].name, numberOfGamesOfPlayer(pgs, p[i]), calculateAverageFoulsByPlayerByGame(pgs, p[i]));
551          }
552      }
553  }
```

Complexidade algorítmica: linear

Como funções auxiliares temos:

```
320  double calculateAverageFoulsByPlayerByGame(PlayerGameStatistics pgs[], Player p) {//media de faltas de um jogador por jogo
321      int sum = 0;
322      int j = 0;
323      for (int i = 0; i < 517; i++) {
324          if (pgs[i].idPlayer == p.id) {
325              sum += pgs[i].statistics.fouls;
326              j++;
327          }
328      }
329      return sum / (double)j;
330  }
331
332  int numberOfGamesOfPlayer(PlayerGameStatistics pgs[], Player p) {//numero de jogos de um jogador
333      int j = 0;
334      for (int i = 0; i < 517; i++) {
335          if (pgs[i].idPlayer == p.id) {
336              j++;
337          }
338      }
339      return j;
340  }
```

A primeira usada (numberOfGamesOfPlayer) calcula o número de jogos em que um jogador participou, para depois ser apresentado. A outra função auxiliar calcula a média de faltas de um determinado jogador por jogo.

COMANDO MFOULG

```
555  void mFoulG(PlayerGameStatistics pgs[]) {
556      int arr[517];
557      int arrSize = 0;
558      int aux = 1;
559      printf("FOULS AVERAGE PER GAME\n\n");
560      printf("#Game | Av.Fouls\n");
561      printf("================\n");
562      for (int i = 0; i < 517; i++) {
563          for (int a = 0; a < arrSize; a++) {
564              if (pgs[i].idGame == arr[a]) {
565                  aux = 0;
566              }
567          }
568          if (aux == 1) {
569              if (calculateAverageFoulsByGameByPlayer(pgs, pgs[i]) > 0) {
570                  printf("%-6d|   %.2f\n", pgs[i].idGame, calculateAverageFoulsByGameByPlayer(pgs, pgs[i]));
571                  arr[arrSize++] = pgs[i].idGame;
572              }
573              else {
574                  printf("%-6d|   %.2f\n", pgs[i].idGame, 0.0);
575                  arr[arrSize++] = pgs[i].idGame;
576              }
577
578          }
579          aux = 1;
580      }
581  }
```

Complexidade algorítmica: linear

Como funções auxiliares temos:

```
410 double calculateAverageFoulsByGameByPlayerById(PlayerGameStatistics pgs[], int id) {//media de faltas por jogo por jogador
411     int sum = 0;
412     int j = 0;
413
414     for (int i = 0; i < 517; i++) {
415         if (pgs[i].idGame == id) {
416             sum += pgs[i].statistics.fouls;
417             j++;
418         }
419     }
420     return (double)sum / j;
421 }
```

Esta função calcula a média de faltas num jogo de um determinador jogador (através do seu ID).

COMANDO FAIRP

```
716 void fairP(Player players[], PlayerGameStatistics pgs[]) {
717     printf("%.2f", calculateAverageFoulsByTeamByGame(players, "FC Porto", pgs));
718 }
```

Complexidade algorítmica: linear

Como funções auxiliares temos:

```
342 double calculateAverageFoulsByTeamByGame(Player players[], char team[], PlayerGameStatistics pgs[]) {//media de faltas de uma equipa por jogo
343     int sum = 0;
344
345     int teamPlayers[100];
346     int arrSize = 0;
347
348
349     for (int i = 0; i < 300; i++) {//calcula o numero de jogadores numa equipa e coloca os seus id's no array
350         if (strcmp(players[i].team, team) == 0) {
351             teamPlayers[arrSize++] = players[i].id;
352         }
353     }
354
355     for (int a = 0; a < arrSize; a++) {//soma o numero de faltas dadas por cada jogador da equipa
356         for (int z = 0; z < 517; z++) {
357             if (teamPlayers[a] == pgs[z].idPlayer) {
358                 sum += pgs[z].statistics.fouls;
359             }
360         }
361     }
362
363     int mostPlayed = 0;
364     int max = 0;
365
366     for (int i = 0; i < arrSize; i++) {//jogador que jogou mais vezes, para saber quantos jogos a equipa tem
367         int aux = 0;
368         int j = 0;
369         for (int z = 0; z < 517; z++) {
370             if (teamPlayers[i] == pgs[z].idPlayer) {
371                 aux++;
372                 j = i;
373             }
374         }
375         if (aux > max) {
376             max = aux;
377             mostPlayed = pgs[j].idPlayer;
378         }
379     }
380
381     int numOfGames = 0;
382     for (int i = 0; i < 517; i++) {//numero de jogos de uma equipa
383         if (pgs[i].idPlayer == teamPlayers[mostPlayed]) {
384             numOfGames++;
385         }
386     }
387
388     return (double)sum / numOfGames;
389
390 }
```

Tivemos dificuldade nesta função, mas no fundo ela calcula a média de faltas de uma determinada equipa, através do seu nome. Primeiro é calculado o número de jogadores da equipa passada por parâmetro, e são colocados os seus Id's no array teamPlayers. Depois é calculada o número de

faltas dadas por esses jogadores. A seguir vamos descobrir qual foi o jogador que mais jogou, para nos ajudar a encontrar o numero de jogos (foi o único método que encontramos para resolver o problema). Por fim calculamos o numero de jogos que a equipa jogou e retornamos a media.

COMANDO FAIRP

```
727 void idealTeam(Player player[], PlayerGameStatistics pgs[]) {
728     printf("Chose Level:\n0 - sub14;1 - sub16; 2 - sub18; 3 - senior\n");
729     int order;
730     printf("Option> ");
731     scanf_s("%d", &order);
732
733     printf("Chose Gender:\nF-Feminino, M-Masculino\n");
734     //char order2;
735     printf("Option> ");
736     //scanf_s(" %c", &order2);
737
738     int max, min;
739     if (order == 1) {
740         max = 2005;
741         min = 2004;
742     }
743     else if (order == 2) {
744         max = 2003;
745         min = 2002;
746     }
747     else if (order == 3) {
748         max = 2001;
749         min = 2000;
750     }
751     else {
752         max = 3000;
753         min = 2000;
754     }
755     //if (numberOfPlayersByAgeAndGender(player, min, max, order2) < 5) {
756         //printf("NAO EXISTEM JOGADORES PARA A EQUIPA IDEAL");
757         //return;
758     //}
759
```

```
760     Player playersInRange[50];
761     int arrSize = 0;
762     //listOfPlayersByAgeAndGender(player, min, max, order2, &playersInRange, &arrSize);
763
764
765     printf("CENTER:\n");
766     printPlayerById(mostAssists(pgs, playersInRange), player);
767     printf("SHOOTY GUARD\n");
768     printPlayerById(mostPoints(pgs, playersInRange), player);
769     printf("POINT GUARD:\n");
770     printPlayerById(mostBlocks(pgs, playersInRange), player);
771 }
```

Complexidade algorítmica: linear

Esta função também nos deu trabalho, e acabou por não ficar totalmente funcional. Primeiramente pedimos ao utilizador o escalão e género que pretende a sua equipa. Depois dependendo das escolhas usamos uma função para saber se existem jogadores suficientes dentro dessa gama para fazer uma equipa, se existissem tentamos então coloca-los num array, para depois usarmos as outras funções auxiliares para descobrir os melhores jogadores nesse array.

Como funções auxiliares temos:

```
int mostPoints(PlayerGameStatistics pgs[], Player players[]) {//jogador com mais pontos de todos
    int most = 0;
    int mostId = 0;
    int j = 0;
    for (int i = 0; i < 300; i++) {
        int aux = 0;
        for (int a = 0; a < 517; a++) {
            if (players[i].id == pgs[a].idPlayer) {
                aux += pgs[a].statistics.twoPoints;
                aux += pgs[a].statistics.threePoints;
                j = i;
            }
        }
        if (aux > most) {
            most = aux;
            mostId = j;
        }
    }
    return mostId;
}
```

Serve para descobrir qual o jogador (de todos) que fez mais pontos no total dos seus jogos.

```
int mostAssists(PlayerGameStatistics pgs[], Player players[]) {//jogador com mais assistencias de todos
    int most = 0;
    int mostId = 0;
    int j = 0;
    for (int i = 0; i < 300; i++) {
        int aux = 0;
        for (int a = 0; a < 517; a++) {
            if (players[i].id == pgs[a].idPlayer) {
                aux += pgs[a].statistics.assists;
                j = i;
            }
        }
        if (aux > most) {
            most = aux;
            mostId = j;
        }
    }
    return mostId;
}
```

Calcula o jogador com mais assistências.

```
int mostBlocks(PlayerGameStatistics pgs[], Player players[]) {//jogador com mais blocos de todos
    int most = 0;
    int mostId = 0;
    int j = 0;
    for (int i = 0; i < 300; i++) {
        int aux = 0;
        for (int a = 0; a < 517; a++) {
            if (players[i].id == pgs[a].idPlayer) {
                aux += pgs[a].statistics.blocks;
                j = i;
            }
        }
        if (aux > most) {
            most = aux;
            mostId = j;
        }
    }
    return mostId;
}
```

Calcula o jogador que fez mais blocos.

c) Limitações

Numa fase inicial do desenvolvimento do trabalho, não encontramos grandes problemas, sendo que o pedido já tinha sido feito em aula (criação de estruturas e as funções das mesmas).

Encontramos o primeiro obstáculo na leitura de ficheiros, apesar das nossas funções loadg e loadp serem praticamente idênticas às que foram disponibilizadas no laboratório de apoio ao projeto, estas não funcionam com o input do utilizador (recebido através de um scanf), mas funcionando quando é inserido o nome do ficheiro diretamente na função fopen.

Depois tentamos desenvolver a função equalsStringIgnoreCase já implementada, tentamos de quatro maneiras diferentes:

- Primeiramente usando uma função chamada strcasecmp() que descobrimos que comparava duas strings ignorando as maiúsculas e minúsculas. Esta dava um erro de compilação.

-Tentamos então a função srtcmpi() que é tambem usada na programção em C. Esta faz o mesmo que a strcmp() mas ignora as diferenças nos cases. Deu erro de compilação.

-Tentamos usar a função de C chamada de strlwr() que converte uma string mixed case numa string em lowercase. Erro de compilação…

-Por fim decidimos usar uma função que já tínhamos usado em anos anteriores, que manualmente alterava casa letra de uma string para minúscula (usando a função tolower())e depois comparava-as através do strcmp. Em projetos esta função funcionava por ser usada noutro IDE, no Visual Studio não é permitido o uso de uma variável para a inicialização de um array (`char primeira[strlen(str1)]`). Logo não conseguimos modificar esta função para permitir o uso de minúsculas e maiúsculas.

Depois só tivemos problemas nas últimas duas funções. Na fairP conseguimos calcular a média de cada equipa, apesar de que os valores obtidos através das funções auxiliares não estarem de acordo com os que aparecem no ficheiro dos resultados disponibilizado, apenas conseguimos calcular a média de uma equipa de cada vez, e acabamos por não conseguir ordenar as equipas.

Na última função tivemos logo dificuldade na gestão dos inputs do utilizador, da criação de um array com os jogadores que encaixam nas restrições do utilizador e depois de descobrir mais do que um jogador melhor para cada posição, ou seja, conseguimos descobrir um jogador (o melhor) para cada posição, mas não o segundo para quando são pedidos dois shooty guards e dois point guards.

d) Conclusões

Para concluir, há que referir que houve algumas dificuldades na implementação de algumas funções, tais como a fairP e idealTeam, mas tentamos implementa-las como achamos que ficaria melhor. Não tivemos dificuldade na criação das estruturas e na gestão de ficheiros e da sua informação, nem como na implementação de funções de cálculos e de ponteiros. Sentimos que se tivéssemos começado mais cedo, poderíamos ter melhorado o projeto. No final achamos que fizemos um bom trabalho, conseguindo implementar da totalidade quase todas as funções usando uma boa pratica de programação.