

Basket Scores



Unidade Curricular : Algoritmos e Tipos Abstratos de Dados

Docente: Aníbal Ponte

Aluno: Miguel Pires Nº150221028

Aluno: João Gomes Nº150221001

Ano Letivo 2017/2018

Índice

Introdução	3
Descrição das Estruturas de Dados	4
Comandos e Resultados	9
1) LOAD Jogadores e Jogos.....	9
2) Informação sobre os Jogadores	11
2.1) Command SHOW.....	11
2.2) Command SORT	12
2.3) Command AVG	13
2.4) Command TYPE	14
2.5) Command CHECKTYPE	15
Limitações.....	16
Conclusão	17

Introdução

No âmbito da Unidade Curricular de Algoritmos e Tipos Abstratos de Dados foi proposto a elaboração de um programa em linguagem C para extrair informação útil de ficheiros com dados sobre os jogadores e os jogos de basquetebol. Este programa consiste num interpretador de comandos que o utilizador irá usar para obter os diversos tipos de informação, principalmente informação sobre a parte das estatísticas.

Na realização deste projeto foi aplicada apenas a matéria lecionada nas aulas Teórico-práticas e Laboratoriais.

Descrição das Estruturas de Dados

Neste projeto foram implementadas as seguintes estruturas de dados:

```
#pragma once

typedef struct date {
    unsigned int day;
    unsigned int month;
    unsigned int year;
} Date;

typedef Date* PtDate;

Date createDate(unsigned int day, unsigned int month, unsigned int year);
void datePrint(PtDate _this);
```

Estrutura Date utilizada posteriormente na criação de um Player.

```
typedef struct statistics {
    float twoPoints; /* total ou media de cestos de dois pontos */
    float threePoints; /* total ou media de cestos de três pontos */
    float assists; /* total ou media de assistências */
    float fouls; /* total ou media de faltas */
    float blocks; /* total ou media de blocos */
    int gamesPlayed; /* total de jogos disputados */
    /* pode acrescentar algum atributo/campo se achar relevante */
} Statistics;

typedef Statistics* PtStatistics;

Statistics createStatistics();
void printStatistic(PtStatistics _this);
```

Estrutura de Estatísticas, onde irá guardar as estatísticas dos todos os jogos, bem como o número de jogos.

```
#pragma once

#include "Date.h"
#include "Statistics.h"

typedef struct player {
    int id;
    char name[50];
    char team[50];
    Date birthDate;
    char gender;
    Statistics statistic;
} Player;

typedef Player* PtPlayer;

Player createPlayer(unsigned int id, char name[], char team[], Date birthDate, char gender);
void printPlayer(PtPlayer _this);
```

Estrutura de Player, onde se criará um jogador já com uma data de nascimento e estatísticas desse Jogador

```
#pragma once
#include "Player.h"

typedef Player ListElem;

void listElemPrint(ListElem elem);
```

Estrutura de ListElem, onde se define o tipo de dados a guardar no TAD List.

```
#include "ListElem.h"

struct listImpl;
typedef struct listImpl *PtList;

PtList listCreate(unsigned int initialCapacity);
int listDestroy(PtList *ptList);

int listAdd(PtList list, int rank, ListElem elem);
int listRemove(PtList list, int rank, ListElem *ptElem);
int listGet(PtList list, int rank, ListElem *ptElem);
int listSet(PtList list, int rank, ListElem elem, ListElem *ptOldElem);
int listSize(PtList list, int *ptSize);
int listIsEmpty(PtList list);
int listClear(PtList list);
void listPrint(PtList list);
```

Estrutura de TadList

```
#pragma once

#include "playerType.h"

/* definicao do tipo da chave */
typedef int MapKey;
/* definicao do tipo do valor */
typedef PlayerType MapValue;

void mapKeyPrint(MapKey key);
void mapValuePrint(MapValue value);

/* funcao de comparacao de chaves */
int mapKeyEquals(MapKey key1, MapKey key2);
```

Estrutura do tipo de dados a guardar no TAD Map.

```

#pragma once

#define MAP_OK            0
#define MAP_NULL         1
#define MAP_NO_MEMORY    2
#define MAP_EMPTY        3
#define MAP_FULL         4
#define MAP_UNKNOWN_KEY  5

#include "mapElem.h"

struct mapImpl;
typedef struct mapImpl *PtMap;

PtMap mapCreate(unsigned int initialCapacity);
int mapDestroy(PtMap *ptMap);
int mapPut(PtMap map, MapKey key, MapValue value);
int mapRemove(PtMap map, MapKey key, MapValue *ptValue);
int mapGet(PtMap map, MapKey key, MapValue *ptValue);
int mapContains(PtMap map, MapKey key);
int mapSize(PtMap map, int *ptSize);
int mapIsEmpty(PtMap map);
int mapClear(PtMap map);
void mapPrint(PtMap map);

```

Estrutura de TadMap para a execução do comando CHECKTYPE.

```

#pragma once
#include "List.h"

void bubbleSortName(PtList list);
void bubbleSortDate(PtList list);
void bubbleSortGames(PtList list);
void sortByName(PtList originalList);
void sortByDate(PtList originalList);
int compareDates(Date date1, Date date2);
void sortByGamesPlayed(PtList list);
PtList averageStatistics(PtList players);
PtList normalizeStatistics(PtList players);
void playerType(PtList list);
void checktype(PtList _this);

```

Modulo Functions onde colocamos as funções auxiliares para os comandos solicitados.

```

#pragma once
#include "List.h"

int equalsStringIgnoreCase(char str1[], char str2[]);
char** split(char *str, int nFields, const char *delim);

//COMMAND LOAD
void importPlayers(PtList list);

//COMMAND CLEAR
void CLEAR(PtList list);

//COMMAND SHOW
void SHOW(PtList list);

//COMMAND SORT
void SORT(PtList list);

//COMMAND AVG
void AVG(PtList list);

//COMMAND NORM
void NORM(PtList list);

//COMMAND TYPE
void TYPE(PtList list);

//COMMAND CHECKTYPE
void CHECKTYPE(PtList list);

```

Módulo Command onde são chamadas as funções auxiliares que fazem com que o comando execute o pretendido.

```

#pragma once
#define STRINGKEY_MAX_LENGTH 15

typedef struct playerType {
    char text[STRINGKEY_MAX_LENGTH];
} PlayerType;

PlayerType createPlayerType(char *initialText);
int stringKeyEqualsIsCaseInsensitive(PlayerType sk1, PlayerType sk2);

```

```
#include "List.h"
```

```
typedef struct cluster  
{  
    float meanTwoPoints;  
    float meanThreePoints;  
    float meanAssists;  
    float meanFouls;  
    float meanBlocks;  
    PtList members;  
}Cluster;
```

```
void Kmeans(PtList list,int k,int maxIterations,float deltaError);
```

NOTA: ALGUNS DOS RESULTADOS EXEMPLIFICATIVOS NÃO SÃO COMPLETOS DEVIDO Á EXTENSIVIDADE DOS MESMOS.

1) LOAD Jogadores e Jogos

//COMMAND LOAD

void importPlayers(PtList list);

```
void importPlayers(PtList list) {
    FILE *fd;
    int err = fopen_s(&fd, "players.csv", "r");
    if (err != 0) {
        printf("\nNao foi possivel abrir o ficheiro %s ... \n", "players.csv");
    }
    char nextLinee[1024];
    //contagem de jogadores lidos
    int countPlayers = 0;
    while (fgets(nextLinee, sizeof(nextLinee), fd))
    {
        if (strlen(nextLinee) < 1) {
            continue;
        }
        char **tokens = split(nextLinee, 5, ";");
        //o array neste momento contém as seguintes "strings":
        //tokens[0] - id
        //tokens[1] - nome
        //tokens[2] - clube
        //tokens[3] - data de nascimento
        //tokens[4] - genero

        int playerID = atoi(tokens[0]);
        int day, month, year;
        sscanf_s(tokens[3], "%d/%d/%d", &day, &month, &year);
        char playerGender = tokens[4][0]; //primeiro carater de tokens[4]
        Date date = createDate(day, month, year);
        Player player = createPlayer(playerID, tokens[1], tokens[2], date, playerGender);
        listAdd(list, countPlayers, player);
        free(tokens);
        countPlayers++;
    }
}
```

```

FILE *games;
int file = fopen_s(&games, "games.csv", "r");
if (file != 0) {
    printf("\nNao foi possivel abrir o ficheiro %s ... \n", "games.csv");
}
char nextLine[1024];
//contagem de jogadores lidos
int countGames = 0;
while (fgets(nextLine, sizeof(nextLine), games))
{
    if (strlen(nextLine) < 1) {
        continue;
    }
    char **tokens = split(nextLine, 8, ";");
    int playerID = atoi(tokens[0]);
    float twoPoints = atof(tokens[2]);
    float threePoints = atof(tokens[3]);
    float assists = atof(tokens[4]);
    float fouls = atof(tokens[5]);
    float blocks = atof(tokens[6]);
    int size;
    listSize(list, &size);
    for (int i = 0 ; i < size ; i++)
    {
        ListElem player, aux;
        listGet(list, i, &player);
        if (player.id == playerID) {
            player.statistic.twoPoints += twoPoints;
            player.statistic.threePoints += threePoints;
            player.statistic.assists += assists;
            player.statistic.fouls += fouls;
            player.statistic.blocks += blocks;
            player.statistic.gamesPlayed++;
            listSet(list, i, player, &aux);
        }
    }
    free(tokens);
    countGames++;
}
printf("\nForam lidos %d jogadores e informacao sobre %d jogos\n", countPlayers, countGames);

```

Na Consola:

```

COMMAND> load
[|||||]
Foram lidos 300 jogadores e informacao sobre 517 jogos

```

2) Informação sobre os Jogadores

2.1) Command SHOW

```
//COMMAND SHOW
void SHOW(PtList list) {
    listPrint(list);
}

void listPrint(PtList list) {
    if (list == NULL) {
        printf("LIST NULL \n");
    }
    else if (listIsEmpty(list)) {
        printf("LIST EMPTY \n");
    }
    else {
        //mostra ranks e elementos
        printf("LIST elements: \n");
        for (int i = 0; i < list->size; i++) {
            SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE), FOREGROUND_INTENSITY | 3);
            printf("\nAt rank %d: ", i);
            listElemPrint(list->elements[i]);
        }
        printf("----- \n");
    }
}
```

Na consola:

```
At rank 281: Jogador 282 : Leila Mclean | Sampaio | 08/03/1995 | F

Two Points: 1.00
Three Points: 2.00
Assists: 4.00
Fouls: 16.00
Blocks: 18.00
Games Played: 2

At rank 282: Jogador 283 : Melina Livingston | Sampaio | 12/07/2005 | F

Two Points: 17.00
Three Points: 14.00
Assists: 16.00
Fouls: 25.00
Blocks: 24.00
Games Played: 3
```

2.2) Command SORT

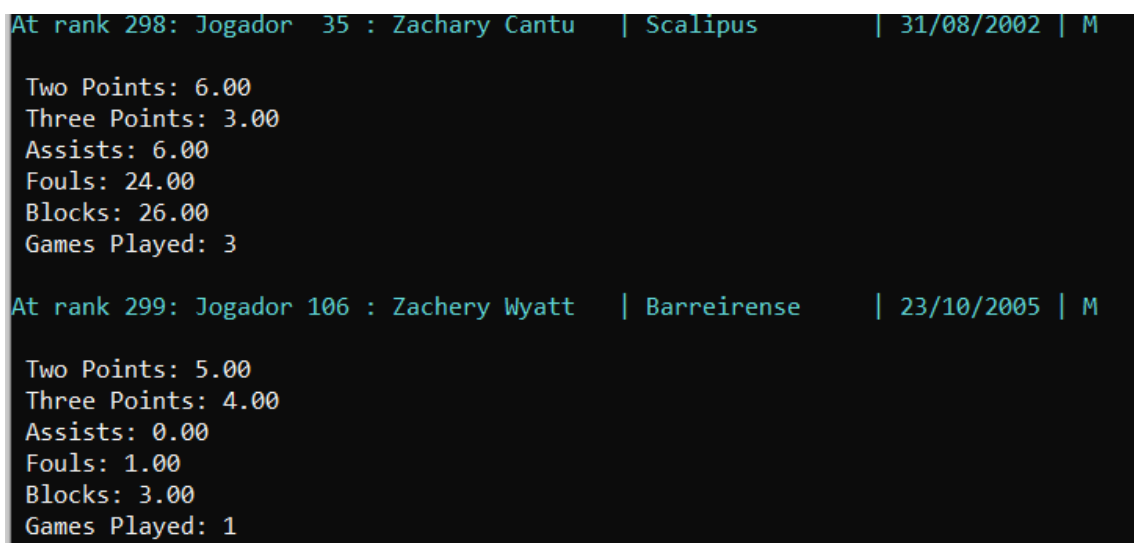
```
//COMMAND SORT
void SORT(PtList list) {
    int option;
    printf("Sort by:\n");
    printf("\t1-Name\n\t2-BirthDay\n\t3-Games Played\n");
    printf("Option> ");
    scanf_s("%d", &option);

    if (option == 1)
    {
        sortByName(list);
    }
    else if (option == 2)
    {
        sortByDate(list);
    }
    else if (option == 3)
    {
        sortByGamesPlayed(list);
    }
    else {
        printf("Insira uma opcao valida!");
    }
}
```

Este método ordena a lista de jogadores por NOME, BIRTHDAY e GAMES PLAYED pedindo ao utilizador se quer que seja ordenada de NOME (opção 1) ,BIRTHDAY (opção 2) , GAMES PLAYED (opção 3).

Na consola:

Opção 1



```
At rank 298: Jogador 35 : Zachary Cantu | Scalipus | 31/08/2002 | M
Two Points: 6.00
Three Points: 3.00
Assists: 6.00
Fouls: 24.00
Blocks: 26.00
Games Played: 3

At rank 299: Jogador 106 : Zachery Wyatt | Barreirense | 23/10/2005 | M
Two Points: 5.00
Three Points: 4.00
Assists: 0.00
Fouls: 1.00
Blocks: 3.00
Games Played: 1
```

Opção 2

```
At rank 298: Jogador 56 : Tobias Church | Scalipus | 28/12/2005 | M

Two Points: 0.00
Three Points: 0.00
Assists: 0.00
Fouls: 0.00
Blocks: 0.00
Games Played: 0

At rank 299: Jogador 23 : Dahlia Williams | Scalipus | 30/12/2005 | F

Two Points: 21.00
Three Points: 19.00
Assists: 25.00
Fouls: 36.00
Blocks: 38.00
Games Played: 4
```

Opção 3

```
At rank 282: Jogador 30 : Serena Knowles | Scalipus | 25/09/2005 | F

Two Points: 20.00
Three Points: 21.00
Assists: 10.00
Fouls: 7.00
Blocks: 4.00
Games Played: 4

At rank 283: Jogador 227 : Alisa Medina | Barreirense | 12/05/2005 | F

Two Points: 21.00
Three Points: 23.00
Assists: 40.00
Fouls: 6.00
Blocks: 7.00
Games Played: 5
```

2.3) Command AVG

```
//COMMAND AVG
void AVG(PtList list) {
    PtList average = averageStatistics(list);
    listPrint(average);
    listClear(average);
    listDestroy(&average);
}
```

```

PtList averageStatistics(PtList players) {
    int inicialCapacity;
    listSize(players, &inicialCapacity);

    PtList newList = listCreate(inicialCapacity);
    int posicao = 0;

    for (int i = 0; i < inicialCapacity; i++) {
        ListElem oldElem;
        ListElem elem;
        listGet(players, i, &elem);

        if (elem.statistic.gamesPlayed != 0) {
            listAdd(newList, posicao, elem);

            elem.statistic.twoPoints = elem.statistic.twoPoints / (float)elem.statistic.gamesPlayed;
            elem.statistic.threePoints = elem.statistic.threePoints / (float)elem.statistic.gamesPlayed;
            elem.statistic.assists = elem.statistic.assists / (float)elem.statistic.gamesPlayed;
            elem.statistic.blocks = elem.statistic.blocks / (float)elem.statistic.gamesPlayed;
            elem.statistic.fouls = elem.statistic.fouls / (float)elem.statistic.gamesPlayed;

            listSet(newList, posicao, elem, &oldElem);
            posicao++;
        }
    }

    return newList;
}

```

Este método retorna lista de jogadores com as suas estatísticas calculadas e que tenham jogado pelo menos um jogo. Como nos é mandado no Enunciado:

O comando, posteriormente, mostra a listagem dos jogadores ordenada decrescentemente pelo índice MVP médio (*avgMVP*) de cada jogador:

$\text{avgMVP} = 3 \times \text{<avg treePoints>} + 2 \times \text{<avg twoPoints>} + \text{<avg assists>} + 2 \times \text{<avg blocks>} - 3 \times \text{<avg fouls>};$

Só deve considerar os jogadores que realizaram pelo menos um jogo.

2.4) Command TYPE

```

//COMMAND TYPE
void TYPE(PtList list) {
    playerType(list);
}

```

A partir da `averageList` este método define-nos o tipo de jogador dependendo das estatísticas do jogador em comparação das estatísticas globais de todos os jogadores.

Sendo os tipos definidos pelas seguintes métricas:

- Tipo **Shooting-Guard**: Todos os jogadores cujos valores médios *twoPoints* e *threePoints* sejam superiores à média geral, e cujos valores médios *assists* e *blocks* sejam inferiores à sua respetiva média geral.

- Tipo **Point-Guard**: Todos os jogadores cujos valores médios *twoPoints* e *threePoints* sejam inferiores à média geral, e cujos valores médios *assists* e *blocks* sejam superiores à média geral.

- Tipo **All-Star**: Todos os jogadores cujos valores médios *twoPoints*, *threePoints*, *assists* e *blocks* sejam superiores à média geral.

2.5) Command CHECKTYPE

```
//COMMAND CHECKTYPE
void CHECKTYPE(PtList list) {
    checktype(list);
}
```

Neste método primeiramente temos de definir o tipo de jogador numa estrutura igual a está.

Limitações

Neste projeto dos comandos pedidos apenas um não foi implementado, o comando Kmeans.

Conclusão

Após concluirmos o projeto podemos fazer um balanço positivo, as dificuldades que surgiram foram ultrapassadas com sucesso, umas com mais facilidade que outras. A única dificuldade que nos deparamos, e não tendo conseguido resolver, foi o comando Kmeans.

Em suma, podemos dizer que consolidamos melhor a matéria lecionada que foi necessária para a elaboração deste Mini Projeto2, e que alcançamos todos os objetivos traçados para o mesmo.