Projecto de PPP 2011/2012
Manual do Programador

Professor: Fernando José Barros Rodrigues da Silva
Alunos: Fábio Mestre Nº 2011144794
Bruno Madureira Nº 2011161942

Índice

Introdução

Neste documento iremos descrever como funciona o código do projecto final de Principios de Programação Procedimental. Para além de descrevermos como funciona o código desenvolvido iremos também quando for necessário explicar o porquê de o código estar implementado de determinada forma, ou seja, as decisões que foram tomádas ao longo do desenvolvimento do projecto.

Estrutura do Projecto

O projecto está dividido em vários ficheiros de forma a facilitar a reutilização do código quando necessário. Os ficheiros utilizados foram:

- main.c – o ficheiro main.c é o ficheiro onde a interface entre o programa e o utilizador é construida usando para isso um menu numérico, é também neste ficheiro que estão implementadas as funções que criam a "base" do programa como por exemplo a gestão de ficheiros, criação de jogos, eliminação de jogos etc.
- Team.h e Team.c – definem a estrutura de dados de uma equipa e respectivas funções.
- Game.h e Game.c – definem a estrutura de dados de um jogo e respectivas funções.
- Date.h e Date.c - definem a estrutura de dados de uma data e respectivas funções.
- LinkedGame.h e LinkedGame.c - definem a estrutura de dados e respectivas funções para uma lista ligada de jogos que é usada para gravar jogos ordenados por data.

Foram criados 2 ficheiros de texto para guardar permanentemente informações sobre equipas e sobre jogos:

- Games.txt – este ficheiro nunca é acedido directamente pelo utilizador. É neste ficheiro que são guardados todos os jogos que já foram inseridos para que da próxima vez que o programa iniciar o utilizador não tenha de realizar o trabalho todo novamente.
- Teams.txt – este ficheiro existe sempre com a mesma quantidade de equipas e apenas pode ser alterado pelo próprio programa. O utilizador apenas pode alterar este ficheiro se quiser alterar o campeonato, mas tal como na vida real não se pode alterar as equipas de um campeonato que está a decorrer, neste programa também não e se o utilizador resolver alterar as equipas, todos os jogos no ficheiro Games.txt têm de ser

apagados para que o programa funcione correctamente. Poder-se-ia eventualmente criar uma pequena aplicação que pedisse dados sobre novas equipas ao utilizador e altera-se o ficheiro de forma automática. No entanto esta opção não foi implementada pois, na nossa opinião, não faz sentido implementa-la no programa de gestão de jogos visto que apenas tornaria a aplicação mais confusa e menos fácil de utilizar.

Todos os tipos definidos no programa com excepção do tipo "Date" são ponteiros que apontam para a estrutura do "objecto". Por exemplo o tipo "Team" é um ponteiro que aponta para a estrutura "tm" sendo esta a verdadeira estrutura de uma equipa. Desta forma (através do uso de ponteiros) é possível usar a mesma equipa (mesmo endereço de memória) em várias estruturas de dados em simultâneo e quando se altera uma equipa numa das estruturas de dados, esta é alterada também nas outras facilitando muito a implementação do programa.

O programa baseia o seu funcionamento principalmente em 2 estruturas de dados (variáveis globais definidas no ficheiro main.c) que são inicializadas quando o programa inicia e apenas são destruídas no final do programa. Essas estruturas de dados são:

- Um array de ponteiros para equipas com um tamanho limitado (tamanho definido através de uma constante e que pode ser alterado se necessário). O uso de um array justifica-se pelo facto de após o programa ser inicializado e serem carregadas todas as equipas para o programa já não são inseridas mais equipas novas até ao final do programa, não sendo deste modo necessário acrescentar equipas ao array tornando este mais eficiente que uma lista ligada de equipas para fazer pesquisas.
- Uma lista ligada de jogos ordenados por data. Como a quantidade de jogos varia durante o programa, visto que podem ser introduzidos jogos novos ou eliminados jogos existentes, uma lista ligada é a estrutura de dados adequada pois permite a alteração do seu tamanho duma forma rápida e eficiente.

De forma a aumentar o desempenho do programa foram criadas várias versões das 2 estruturas da seguinte forma:

- Existem 2 arrays de ponteiros para equipas. Um está ordenado pelo nome e o outro está ordenado pela classificação da equipa apontada. Deste modo quando é necessário imprimir equipas ordenadas por nome usa-se o primeiro array e quando é necessário imprimir equipas ordenadas por classificação usa-se o segundo não sendo necessário ter de ordenar novamente as equipas.
- Para além da lista ligada de jogos principal (que contem todos os jogos existentes) existe uma lista ligada de jogos por cada equipa que contém os jogos realizados por essa equipa. Quando é necessário imprimir apenas os jogos realizados por determinada equipa basta imprimir a lista ligada de jogos respectiva não sendo necessário pesquisar na lista principal quais são os jogos em que essa equipa participa.

Código Implementado

Date – Nestes ficheiros é definida a estrutura de uma data. Uma data é constituída por 3 inteiros (dia, mês e ano). Como uma data é um atributo específico e único para determinado jogo não é necessário neste caso o uso de um ponteiro a definir o tipo Date pois em princípio uma data geralmente não é reutilizada por outro jogo.

```
typedef struct dt{
   int year;
   int month;
   int day;
}Date;
```

- ***Date newDate(int day, int month, int year);***

  Esta função devolve uma estrutura "Date" inicializada com os valores indicados como parâmetros.

- ***void printDate(Date* date);***

  Imprime a data recebida como parâmetro na consola ("dia-mês-ano").

- ***int leapYear (int year);***

  Indica se o ano que foi recebido como parâmetro é bissexto ou não. Devolve 1 se o ano for bissexto e 0 caso contrário.

  Nota: Um ano é bissexto se não for divisível por 100 e for divisível por 4.

- ***int isValid (Date* date);***

  Indica se a data recebida como parâmetro é uma data valida ou não. Devolve 1 se a data for valida e 0 caso contrário.

  Nota: Ao implementar as 2 funções anteriores o grupo teve como objectivo tornar o programa mais "real" pois usando estas funções é possível não aceitar datas que obviamente são inválidas, por exemplo:

  29-2-2011 – Como 2011 não é ano bissexto a data é inválida

  31-4-2010 – O mês de Abril não tem 31 dias

- ***int compareDate(Date* date1, Date* date2);***

  Compara 2 datas. Devolve:

  - -1 se “date1” < “date2”
  - 0 se “date1” = “date2”
  - 1 se “date1” > “date2”

  Esta função é usada para comparar jogos entre si e coloca-los na lista ligada de jogos ordenados por data.

Team – É nestes ficheiros que estão definidas as equipas. Uma equipa consiste num nome, uma localidade e a pontuação que esta possui no campeonato. O tipo “Team” é um ponteiro que aponta para a verdadeira estrutura “tm”.

- ***typedef struct game_lnode *TeamGames;***

  Ao tentar implementar uma lista ligada de jogos dentro da estrutura das equipas o grupo deparou-se com um problema de dependência cíclica ao fazer o “include” do ficheiro “LinkedGame.h” uma vez que este último indirectamente está também a fazer “include” do ficheiro “Team.h”. Para resolver este problema recorreu-se a "Forward Declaration" desta forma o compilador sabe que o tipo "TeamGames" é um ponteiro para “game_lnode” ou seja é um ponteiro para uma lista de jogos.

```
typedef struct tm *Team;
typedef struct tm{
        char* name;
        char* town;
        int leagueScore;
        TeamGames teamGames;
}teamStruct;
```

- ***Team newTeam(char* name, char* town, int leagueScore);***

  Reserva espaço na heap para guardar a estrutura usando a função “malloc()”, guarda as variáveis recebidas como parâmetros na estrutura e por fim devolve um ponteiro para a estrutura criada. Se ocorrer algum erro durante a alocação de memória é devolvido “NULL”.

- ***void freeTeam(Team team);***

  Liberta a memória alocada para a equipa recebida como parâmetro e para o respectivo nome, localidade e lista ligada de jogos.

  Nota: Para libertar memória recorreu-se à função “free()”.

- ***int compareTeamByName(Team team1, Team team2) ;***

  Compara 2 jogos usando o nome da equipa como forma de comparação. Devolve:

  - -1 se nome de “team1” < nome de “team2”
  - 0 se nome de “team1” = nome de “team2”
  - 1 se nome de “team1” > nome de “team2”

  Nota: Para fazer a comparação de “strings” foi usada a função “strcmp()” que está definida em “string.h”.

- ***void printTeamName(Team team);***

  Imprime o nome da equipa recebida como parâmetro na consola.

- ***void printTeam(Team team);***

  Imprime a equipa recebida como parâmetro na consola.

Game – Nestes ficheiros está definida a estrutura de um jogo. Um jogo consiste em 2 equipas (a local e a visitante), a data do jogo e 2 inteiros que representam os golos marcados por cada uma das equipas. O tipo "Game" é um ponteiro para a estrutura "gm".

```
typedef struct gm* Game;
typedef struct gm {
        Team local;
        Team visitor;
        Date date;
        int lScore;
        int vScore;
} gameStruct;
```

- ***Game newGame(Team local, Team visitor, Date date, int lScore, int vScore);***
  Reserva espaço na heap para guardar a estrutura usando a função "malloc()", guarda as variáveis recebidas como parâmetros na estrutura e por fim devolve um ponteiro para a estrutura criada. Se ocorrer algum erro durante a alocação de memória é devolvido "NULL".
- ***void freeGame(Game game);***
  Liberta a memória aloca para o jogo recebido como parâmetro.
  Nota: Para libertar memória recorreu-se à função "free()".
- ***int compareGameByDate(Game game1, Game game2);***
  Compara 2 jogos usando a data do jogo como forma de comparação. Chama a função "compareDate()" (definida no ficheiro "Date.h") usando como parâmetros para essa função a data dos jogos que se quer compararar.
  Devolve:
  - -1 se a data de "game1" < data de "game2"
  - 0 se a data de "game1" = data de "game2"
  - 1 se a data de "game1" > data de "game2"
- ***void printGame(Game game);***
  Imprime o jogo recebido como parâmetro na consola.

LinkedGame – Define a estrutura de uma lista ligada de jogos e declara as funções que permitem a sua utilização. Uma lista ligada de jogos consiste numa série de "nós" contendo cada um desses nós um ponteiro para o próximo "nó" e um ponteiro para um jogo. O tipo "GameList" é um ponteiro para a estrutura "game_lnode". O grupo foi obrigado a alterar a implementação fornecida nos slides da disciplina pois essa implementação considerava 2 jogos com a mesma data como sendo iguais quando na verdade são jogos completamente diferentes um do outro.

```
typedef struct game_lnode *GameList;
typedef struct game_lnode {
        Game info;
        GameList next;
} ListGame_node;
```

- ***GameList createGameList ();***

  Cria uma lista ligada de jogos vazia alocando espaço para um "nó" especial que não guarda informação nenhuma ("header"). Se ocorrer algum erro durante a alocação de memória é devolvido "NULL".

- ***GameList destroyGameList (GameList list);***

  Destrói a lista ligada de jogos recebida como parâmetro libertando a memória que estava alocada para cada um dos seus "nós". É utilizado um algoritmo recursivo que percorre a lista até ao fim, libertando memória do fim para o início.

  Nota: Para libertar memória recorreu-se à função "free()".

- ***int gameListIsEmpty (GameList list);***

  Verifica se a lista ligada de jogos está vazia.

- ***int gameListIsFull (GameList list);***

  Verifica se a lista ligada de jogos está cheia (devolve sempre falso).

- ***int insertGame (GameList list, Game item);***

  Insere um jogo na lista ligada de jogos recebida como parâmetro. O jogo é inserido de forma ordenada recorrendo para isso a uma função auxiliar “searchPositionToInsert()”.

  Devolve:

  - 0 se ocorreu um erro
  - 1 se a função executou normalmente

- ***GameList searchPositionToInsert (GameList list, Game key);***

  Pesquisa na lista ligada de jogos a posição em que se pode inserir o jogo "key". Devolve o nó que contém o jogo com a data mais próxima da data de “key” mas contudo inferior a esta.

  Nota: Para fazer a pesquisa da posição onde se pode inserir o jogo é utilizado o seguinte ciclo:

```
GameList aux = list;
while ((aux->next) != NULL && (compareGameByDate((aux->next->info), key) == -1)) {
    aux = aux->next;
}
```

  Este ciclo percorre a lista enquanto a data de “aux” for menor que a data de “key”.

- **void searchGame (GameList list, Game key, GameList *previous, GameList *node);**

  Pesquisa na lista ligada de jogos o jogo "key" e devolve o "nó" onde o jogo está guardado e o "nó" imediatamente anterior.

- ***int gameListSize (GameList list);***

  Devolve o tamanho da lista ligada de jogos recebida como parâmetro

- ***void deleteGame (GameList list, Game item);***

  Apaga o jogo recebido como parâmetro da lista ligada de jogos libertando o espaço alocado para o respectivo “nó”.

  Nota: Para fazer a pesquisa do nó onde o jogo está guardado recorreu-se à função “searchGame()”. Para libertar memória recorreu-se à função “free()”.

- ***void printGameList (GameList list);***

  Imprime a lista ligada de jogos recebida como parâmetro.

main – As funções deste ficheiro criam os meios necessários para implementar as 7 opções que o utilizador pode escolher. Foi utilizada (com autorização do professor) a biblioteca “conio.h” de forma a ser possível limpar a consola quando necessário criando deste modo um programa mais fácil de usar e menos confuso. Foram também definidas várias constantes que definem limites “virtuais” para por exemplo a quantidade máxima de equipas num campeonato ou um limite máximo para os anos que o programa aceita como válidos. Estas constantes podem obviamente ser redefinidas se o programador achar necessário não afectando de modo algum o código previamente implementado.

- ***#define TEAMS_MAX 40***

  Define uma constante que representa a quantidade máxima de equipas num campeonato.
- ***#define MAX 200***

  Define uma constante que é usada como tamanho máximo para “arrays” se não se souber a quantidade de informação que estes irão guardar.
- ***#define DAYS_MAX 31***

  Define o número máximo de dias num mês.
- ***#define MONTHS_MAX 12***

  Define o número máximo de meses num ano.
- ***#define YEARS_MAX 2100***

  Define um limite máximo para os anos.
- ***#define SCORE_MAX 200***

  Define um limite máximo para a quantidade de golos que uma equipa pode marcar por jogo

- ***Team* NTeams;***

  Variável global que guarda um ponteiro para um "array" de equipas ordenadas por nome.

- **Team* STeams;**

  Variável global que guarda um ponteiro para um "array" de equipas ordenadas por pontuação.

- **GameList gameList;**

  Variável global que representa uma lista ligada de jogos ordenados por data.

- ***void error();***

  Informa o utilizador que ocorreu um erro ao alocar espaço na "heap" ou ao abrir/escrever num ficheiro e termina o programa.

- ***int size (Team* teams);***

  Recebe um ponteiro para um "array" de equipas e devolve o tamanho do "array".

  Nota: Considera-se que o "array" acaba quando um elemento for NULL

- ***void printTeams(Team* teams);***

  Recebe um ponteiro para um "array" de equipas e imprime o "array" na consola.

  Nota: Considera-se que o "array" acaba quando um elemento for NULL

- ***void sortByName (Team *teams, int n);***

  Recebe um ponteiro para um "array" de equipas e o tamanho do "array" como parâmetro e ordena o "array" por nome usando o algoritmo de ordenação "BubbleSort".

- ***void sortByName (Team *teams, int n);***

  Recebe um ponteiro para um "array" de equipas e o tamanho do "array" como parâmetro e ordena o "array" por pontuação usando o algoritmo de ordenação "BubbleSort".

- ***int countDigits(int number);***

  Conta o número de dígitos de um inteiro.

- ***int option(int args, int zero);***

  Função onde é realizada toda a interacção com o utilizador. O objectivo da função é aceitar apenas como válidas as opções que respeitem os seguintes critérios:

  - O texto introduzido pelo utilizador apenas pode conter dígitos
  - Se for introduzido algo que não seja um dígito (ex.: espaço, tab, letra) a opção é considerada inválida
  - O número máximo que pode ser introduzido é especificado pelo parâmetro "args". Se for introduzido um número superior a “args” a opção é considerada inválida
  - Não são aceites números negativos

  Parâmetros:

  - args - Número máximo que o utilizador pode introduzir
  - zero - Variável booleana. Se zero = 1 o número 0 é considerado uma opção válida. Se zero = 0 o número 0 é uma opção inválida

  Devolve:

  - A opção escolhida pelo utilizador ou -1 se o utilizador escrever uma opção inválida

- ***void initializeTeamList(void);***

  Aloca espaço na “heap” para os “arrays” de equipas "NTeams e STeams" e inicializa-os com as equipas guardadas no ficheiro "Teams.txt". No final ordena-os por nome e por pontuação respectivamente.

- ***void initializeGameList(void);***

  Inicializa a lista ligada "gameList" com os jogos guardados no ficheiro “Games.txt” e insere os jogos específicos de cada equipa na lista da equipa.

- ***void destroy();***

  Função usada para libertar toda a memória que foi alocada na “heap” durante a execução do programa.

  Liberta o espaço alocado para:

  - Todas as equipas e respectivos nomes, localidades e lista de jogos.
  - Todos os jogos e respectivas datas.
  - Os arrays "NTeams" e "STeams".
  - A lista ligada "gameList".

- ***void saveGameList();***

  Guarda no ficheiro "Games.txt" todos os jogos que estão guardados na lista ligada "gameList"

- ***void saveTeamList();***

  Guarda no ficheiro "Teams.txt" todas as equipas guardadas no “array” "STeams" de forma a actualizar as pontuações do campeonato que estão gravadas no ficheiro

- ***Team selectTeam(char* toPrint);***

  Pede ao utilizador para escolher uma equipa das que estão guardadas em "NTeams". Devolve a equipa escolhida

- ***Game selectGame();***

  Pede ao utilizador para escolher um jogo dos que estão guardados em "gameList". Devolve o jogo escolhido.

- ***Date selectDate();***

  Pede ao utilizador para introduzir uma data. Devolve a data introduzida.

- ***Game createGame();***

  Cria um novo jogo pedindo ao utilizador informações sobre as equipas que jogaram, a data do jogo e o resultado do encontro. Devolve o jogo criado ou “NULL” caso o utilizador escolha a opção retroceder.

- ***void printGlobalRanking();***

  Imprime a classificação geral das equipas do campeonato recorrendo ao “array” “STeams”.

- ***void removeGame(Game game);***

  Remove um jogo da lista ligada "gameList" e das listas ligadas de jogos específicas para cada equipa interveniente e por fim altera a pontuação geral das equipas de forma a anular as pontuações que tinham sido obtidas com o jogo que irá ser apagado.

Implementação do menu principal e respectivas opções – Para implementar o menu principal, o grupo decidiu criar um ciclo “infinito” que imprime as várias opções disponíveis e pede ao utilizador para escolher uma opção. A escolha da opção é feita recorrendo à função option(). Após a opção ter sido escolhida é utilizado um “switch” de forma a executar as instruções necessárias para executar essa escolha.

```
int main(void) {

   initializeTeamList();
   initializeGameList();
   Game game;

   while (1) {

      int choice;
      do {
         printf("Menu:\n\n");
         printf("1 - Introduzir Jogo\n");
         printf("2 - Retirar Jogo\n");
         printf("3 - Mostrar classificacao geral\n");
         printf("4 - Listar equipas\n");
         printf("5 - Mostrar jogos de equipa\n");
         printf("6 - Mostrar todos os jogos do campeonato\n");
         printf("7 - Exit\n\n");
         printf("Opcao: ");
         choice = option(7, 0);

         system("cls");

      } while(choice == -1);
```

switch (choice) {

Opção 1: Introduzir Jogo

1. Cria-se um jogo novo chamando a função “createGame()” que vai pedir ao utilizador os dados necessários para a criação do jogo e devolve-o no fim.
2. Se a função “createGame()” foi bem sucedida insere-se o jogo recentemente criado na lista ligada de jogos “gameList” e nas listas de jogos especificas para cada equipa interveniente usando para isso a função “insertGame()” .
3. Guarda-se a lista ligada recentemente alterada no ficheiro “Games.txt” usando a função “saveGameList()”.
4. Como a pontuação geral do campeonato foi alterada é necessário ordenar novamente o “array” “STeams”, para isso recorre-se à função “sortByScore()”.
5. Guarda-se as equipas contidas no “array” “STeams” recentemente reordenado e com pontuações das equipas alteradas no ficheiro “Teams.txt”.

```
case 1:
   game = createGame();
   if(game != NULL) {
            if (insertGame(gameList, game) == 0) error();
            if (insertGame(game->local->teamGames, game) == 0) error();
            if (insertGame(game->visitor->teamGames, game) == 0) error();
   }
   saveGameList();
   sortByScore(STeams, size(STeams));
   saveTeamList();
   break;
```

Opção 2: Retirar Jogo

1. Verifica-se se “gameList” está vazio pois se a lista estiver vazia não há jogos no campeonato, logo é impossível retirar um jogo do campeonato. Caso a lista esteja vazia avisa-se o utilizador que não existem jogos na base de dados.
2. Chama-se a função “removeGame()” que retira o jogo de “gameList” e das listas ligadas de jogos especificas de cada equipa interveniente e por fim altera-se a pontuação das equipas.
3. Guarda-se a lista ligada recentemente alterada no ficheiro “Games.txt” usando a função “saveGameList()”.
4. Como a pontuação geral do campeonato foi alterada é necessário ordenar novamente o “array” “STeams”, para isso recorre-se à função “sortByScore()”.
5. Guarda-se as equipas contidas no “array” “STeams” recentemente reordenado e com pontuações das equipas alteradas no ficheiro “Teams.txt”.

```
case 2:
  if(!gameListIsEmpty(gameList)) {

    game = selectGame();
    removeGame(game);
    saveGameList();
    sortByScore(STeams, size(STeams));
    saveTeamList();
  } else {
    printf("Nao existem jogos na base de dados\n");
    getch();
    system("cls");
  }
  break;
```

Opção 3: Mostrar Classificação Geral

1. Usa-se a função “printGlobalRanking()” que vai imprimir as equipas guardadas no “array” “STeams” ordenadas por classificação.

```
case 3:
  printf("Classificacao geral:\n\n");
  printGlobalRanking();
  getch();
  system("cls");
  break;
```

Opção 4: Listar Equipas

1. Usa-se a função “printTeams()” que vai imprimir as equipas guardadas no “array” “Nteams” ordenadas por nome.

```
case 4:
  printf("Lista de Clubes:\n\n");
  printTeams(NTeams);
  getch();
  system("cls");
  break;
```

Opção 5: Mostrar jogos de equipa

1. Usa-se a função “selectTeam()” que permite pedir ao utilizador para seleccionar uma equipa das que participam no campeonato.
2. Imprime-se a lista ligada de jogos correspondente à equipa seleccionada usando a função “printGameList()”.

```
        case 5:
            printGameList(selectTeam(NULL)->teamGames);
            getch();
            system("cls");
            break;
```

Opção 6: Mostrar todos os jogos do Campeonato

1. Usa-se a função “printGameList()” para imprimir a lista ligada de jogos “gameList” que contém todos os jogos do campeonato.

```
        case 6:
            printf("Lista de Jogos do Campeonato:\n\n");
            printGameList(gameList);
            getch();
            system("cls");
            break;
```

Opção 7: Exit

1. Liberta-se toda a memória alocada durante a execução do programa recorrendo para isso à função “destroy()”.
2. Retorna-se o valor 0 que permite fazer um “break” ao ciclo infinito e indica que o programa terminou sem erros.

```
        case 7:
            destroy();
            return 0;
        }
    }
    return 0;
}
```