IPL
instituto politécnico
de leiria

ESTG Air Race

Trabalho prático para a unidade
curricular de Programação I
1º Ano 1º Semestre

Realizado pelos alunos

Bruno Tereso - 2091066
Marco Ferreira - 2092016

Docentes

Filipe Lopes
Vítor Noronha e Távora

Trabalho prático para a unidade curricular de Programação I
1º Ano 1º Semestre | 2009/2010

Realizado pelos alunos
Bruno Tereso – 2091066@student.estg.ipleiria.pt
Marco Ferreira – 2092016@student.estg.ipleiria.pt

Docentes
Filipe Lopes – filipe.lopes@estg.ipleiria.pt
Vítor Noronha e Távora - vntavora@estg.ipleiria.pt

Bruno Tereso, 2091066@student.estg.ipleiria.pt
Marco Ferreira, 2092016@student.estg.ipleiria.pt

Índice

Bruno Tereso, 2091066@student.estg.ipleiria.pt
Marco Ferreira, 2092016@student.estg.ipleiria.pt

Bruno Tereso, 2091066@student.estg.ipleiria.pt
Marco Ferreira, 2092016@student.estg.ipleiria.pt

Introdução

Conforme pedido, pretende-se elaborar um programa em linguagem C que auxilie na gestão de um campeonato de corridas aéreas do tipo Red Bull Air Race – o ESTG Air Race 2010.

O programa vai gerir um campeonato composto por 10 etapas, podendo-se inscrever em cada etapa um máximo de 20 pilotos.

Para tal, implementamos as seguintes funcionalidades:

- **Pilotos**, onde podemos fazer a gestão dos pilotos.
- **Etapas**, onde será feita a gestão de etapas.
- **Resultados**, onde inserimos e consultamos resultados de etapas.
- **Estatísticas**, onde oferecemos um conjunto de dados estatísticos.
- **Gravar**, que permite gravar a informação em formato digital.

As quais passamos a descrever mais detalhadamente.

Pilotos

A opção Pilotos dá acesso a um menu para uma gestão completa dos pilotos do campeonato.

A primeira opção do menu, **Inserir Piloto**, permite a inserção de um piloto no campeonato, o utilizador devera inserir o Nome, Apelido, Nº do Avião com que vai participar e Idade. Para a inscrição ser válida, o nome completo deverá ter um comprimento máximo de trinta caracteres, o avião deve ser um número valido entre 1 e 999, e, a idade deve estar compreendida entre 18 e 80 anos.

A opção, **Eliminar Piloto**, permite ao utilizador apagar um piloto registado no programa, deve ter em conta que se o piloto participou numa etapa já realizada, não pode ser apagado.

Para **Consultar Dados Piloto**, o utilizado usa a terceira opção do menu, que mostra todos os dados relativos a um piloto, incluindo inscrições, pontuações e posições em cada etapa já realizada.

De uma forma mais generalizada, na última opção do menu Pilotos é possível **Listar todos os pilotos** mostrando o nome, idade e avião.

Etapas

O menu secundário Etapas, permite gerir toda a documentação das etapas envolvidas no campeonato.

Tal como no menu pilotos, a primeira opção permite ao utilizador, **Inserir Etapa**, se já existirem etapas inseridas o programa mostra uma pequena lista das etapas anteriores para facilitar a inserção da nova etapa. O programa permite etapas com nomes idênticos desde que em ordens e datas diferentes, o nome da etapa deverá ter no mínimo 3 caracteres. Ao inserir uma etapa com ordem entre duas etapas já existentes, terá que escolher uma data válida para a sua realização, se não souber as datas das outras etapas, ao inserir uma etapa inválida o programa vai mostrar-lhe os limites válidos para a data da etapa. Após o processo de validação da data o utilizador escolhe o número máximo de inscritos para a etapa, obedecendo ao valor máximo de 20 imposto.

Este menu permite também, **Editar Etapa**, onde o utilizador pode modificar todos os dados de uma etapa, as validações são impostas tal e qual como na inserção. Ao editar uma etapa tem a hipótese de manter, apagar ou acrescentar inscrições.

Bruno Tereso, 2091066@student.estg.ipleiria.pt
Marco Ferreira, 2092016@student.estg.ipleiria.pt

Para **Inscrever Piloto em Etapa**, o utilizador deverá usar a terceira opção do submenu. Nesta opção, para facilitar o utilizador será em primeiro lugar mostrada uma lista das etapas, dentro das quais as realizadas possuem a letra R ao lado da data, é por isso impossível inscrever um piloto nesses casos. Após a escolha da etapa onde quer inscrever o piloto, é mostrada uma lista de pilotos onde pode escolher qual inscrever, usando o nome ou o número do avião, no caso de existirem nomes iguais, o utilizador é avisado e deverá escolher o piloto pelo número de avião.

Na opção **Dados de Etapa**, o utilizador pode consultar os dados completos de uma etapa, descrição, data, se está realizada ou não, e, a lista de pilotos inscritos com respectivo avião, tempos e pontos (no caso de estar realizada).

Na última opção, **Listar Inscrições de Piloto**, o programa mostra ao utilizador uma versão básica dos dados de piloto, onde é possível ver o nome e data das etapas onde está inscrito.

Resultados

O submenu Resultados, recolhe os tempos dos pilotos, classifica e ordena as pontuações de modo a apresentar as classificações gerais. Este submenu apresenta de inicio a etapa corrente (se existir), esta etapa é sempre a etapa mais próxima de realizar. Se forem inseridas etapas em ordens anteriores posteriormente, a etapa corrente será actualizada de modo a ler os valores precisos para a etapa.

A primeira opção do menu, **Resultados da Etapa Corrente**, verifica qual a etapa corrente, mostra os seus detalhes e recolhe, um a um, os tempos de todos os pilotos inscritos. Este tempos devem ser inseridos em forma universal (minuto:segundo.milesimos), de outra forma não será validado.

No caso de existem erros nos tempos, o utilizador pode utilizar a segunda opção do menu, **Alterar Tempos**, onde o programa volta a recolher os tempos para todos os pilotos inscritos. Não é possível alterar tempos em etapas ainda não realizadas, o utilizador terá que escolher uma das etapas da lista apresentada.

Para **Consultar Dados de uma Etapa** o utilizador deverá escolher a terceira opção do menu. Nesta opção serão listadas todas as etapas realizadas, e, ao escolher uma são apresentados todos os pilotos, tempos e pontos resultantes.

Na última opção, **Classificação Geral**, é possível visualizar a classificação actual do campeonato, ordenado por pontos.

Estatísticas

O submenu Estatísticas permite, para além de visualizar um leque diverso de estatísticas nos **Dados Estatísticos**, aceder a uma tabela que mostra a informação de todos os resultados do campeonato num só ecrã, o **Histórico por Piloto**. De forma a simplificar a visualização quando existem listas grandes de pilotos, o utilizador pode ainda indexar a **Classificação Geral** pelo número de avião.

Bruno Tereso, 2091066@student.estg.ipleiria.pt
Marco Ferreira, 2092016@student.estg.ipleiria.pt

Constantes Utilizadas

```
#define MAX_PILOTOS 200
```
Constante que guarda o número máximo de pilotos que o campeonato deve ter.

```
#define MAX_PILOTOS_ETAPA 20
```
Constante que guarda o número máximo de pilotos a inscrever numa etapa.

```
#define MAX_ETAPAS 10
```
Constante que guarda o número máximo de etapas.

```
#define PROGRAMA "\n\n\t\t\t\tESTG Air Race 2010\n"
```
Constante que guarda o título do programa, para facil alteração do ano posteriormente.

```
#define ERRO_NOME 500
```
Constante usada para representar um erro no nome, seja curto demais ou nome inválido, usada para retorno em diversas funções.

```
#define ERRO_DUPLICADO 998
```
Constante usada para representar um erro, mais concretamente erro de nomes duplicados, usada para retorno em diversas funções.

```
#define ERRO_NAOEXISTE 999
```
Constante usada para representar um erro, representa um nome de piloto ou etapa que não existe.

```
#define SAIR '.'
```
Constante usada permitir a saida do programa principal depois de os dados serem gravados.

```
#define NAOSAIR ','
```
Usada na função sair, quando o utilizador pretende cancelar o fim do programa.

```
#define NUMTOPS 3
```
Numtops representa o número total de registos a apresentar nos diversos tops das estatisticas. De forma a minimizar a informação apresentada este valor foi reduzido para dois. Em campeonatos com muitos pilotos faz sentido aumentar os tops.

```
#define ETAPA_EXTREMO 501
```
Usada em comparações de datas, representa o extremo quando o utilizador insere uma etapa que está em primeiro ou ultimo lugar da lista.

```
#define DIAS_MAX 99999999
```
Representa o número máximo de dias para a próxima etapa, valor ficticio infinito.

```
#define DIAS_MIN 1
```
Representa o número mínimo de dias para a etapa anteior, valor ficticio.

```
#define CHARS_NOME 31
```
Representa o limite de caracteres que podem ser usados no nome.

Bruno Tereso, 2091066@student.estg.ipleiria.pt
Marco Ferreira, 2092016@student.estg.ipleiria.pt

Estruturas

Passaremos a enumerar e explicar uma por uma todas as estruturas presentes no código. Todas as estruturas definidas terminam em _t.

Estruturas | Principais

```
typedef struct {
      char nome[CHARS_NOME];
      int idade;
      int aviao;
      int pontos;
      } ficha_t;
```

A estrutura **ficha_t** guarda todos os dados referentes à ficha de um piloto, para o correcto funcionamento deste programa, foram definidos os seguintes tipos/variáveis:

nome, vector de caracteres, guarda o nome e apelido do piloto.
idade, inteiro que guarda a idade do piloto.
aviao, inteiro que guarda o número de avião.
pontos, inteiro que guarda os pontos do piloto no campeonato.

```
typedef struct {
      char nome[CHARS_NOME];
      int ordem;
      data_t data;
      int participantes;
      int totalinscritos;
      int inscritos[MAX_PILOTOS_ETAPA];
      int tempos[MAX_PILOTOS_ETAPA];
      int pontos[MAX_PILOTOS_ETAPA];
      int realizada;
      } etapa_t;
```

A estrutura **etapa_t** guarda todos os dados referentes a uma etapa, para o correcto funcionamento deste programa, foram definidos os seguintes tipos/variáveis:

nome, vector de caracteres, guarda a designação da etapa.
ordem, inteiro que guarda a ordem da etapa, inicialmente representa sempre o valor do indice da etapa +1.
data, estrutura do tipo data_t, guarda a data em que se realiza a etapa.
participantes, inteiro que guarda o número máximo de pilotos da etapa.
totalinscritos, inteiro que guarda o número actual de inscritos na etapa.
inscritos[], vector de inteiros que garda o número de avião dos inscritos na etapa.
tempos[], vector de inteiros que guarda o tempo (em milesimos de segundos) de cada piloto inscrito na etapa.
pontos[], vector de inteiros que guarda os pontos atribuidos com base no tempos[].
realizada, inteiro com valor logico que toma o valor de 1 se a etapa se encontra realizada, ou 0 para o caso de estar por realizar.

```
typedef struct {
      int pilotos;
      int etapas;
      int realizadas;
      int etapa_corrente;
      int grava;
      } conta_t;
```

A estrutura **conta_t** guarda diversos contadores usados globalmente no programa, de entre os quais:

pilotos, inteiro que conta o número de pilotos inseridos no programa.
etapas, inteiro que conta o número de etapas inseridas no programa.
realizadas, inteiro que guarda o total de etapas realizadas.
etapa_corrente, inteiro que guarda o índice da etapa corrente.
grava, inteiro lógico usado para verificar se é preciso gravar o programa.

Bruno Tereso, 2091066@student.estg.ipleiria.pt
Marco Ferreira, 2092016@student.estg.ipleiria.pt

Auxiliares ou Secundárias

```
typedef struct {
      int dia, mes, ano;
      }data_t;
```

A estrutura **data_t** guarda os valores inteiros para o **dia**, **mes** e **ano**.

```
typedef struct {
      int min, seg, mseg;
      }tmp_t;
```

A estrutura **tmp_t** guarda os valores inteiros para minutos, segundos e milesimos de segundos.

Variáveis principais

Declarações principais para o inicio do programa:

```
ficha_t *ptr_piloto;
```
Ponteiro que aposta para o inicio da estrutura ficha_t, definido para guardar os pilotos dinamicamente.

```
etapa_t ptr_etapa[MAX_ETAPAS];
```
Vector de estrutura que guarda no total as etapas definidas em MAX_ETAPAS.

```
conta_t ptr_contador;
```
Estrutura simples que guarda todos os valores de contadores.

Bruno Tereso, 2091066@student.estg.ipleiria.pt
Marco Ferreira, 2092016@student.estg.ipleiria.pt

Funções

Passaremos agora a explicar as principais características das funções utilizadas, bem como uma breve descrição e razões que nos levaram a escolher a metodologia aplicada. Colocando apenas os respectivos protótipos uma vez que a função completa pode ser consultada no *Anexo I – Código Fonte*.

As variáveis **ptr_contador**, **ptr_piloto** e **ptr_etapa**, não vão ser aqui referidas uma vez que são comuns a grande parte das funções e já foram descritas anteriormente.

Imediatamente antes do protótipo da função está o número da linha onde ela inicia, para mais fácil consulta no código fonte.

Menus

```
[250] char menuPrincipal(conta_t *ptr_contador)
```

Esta função mostra o menu principal do programa. É chamada directamente no programa principal e devolve o caracter correspondente à escolha do utilizador, após ser validado.

```
[283] char pilMenu(conta_t *ptr_contador)
[554] char etaMenu(conta_t *ptr_contador)
[1548] char resMenu(conta_t *ptr_contador, etapa_t
*ptr_etapa)
[2019] char estMenu(conta_t *ptr_contador)
```

Identicas às anteriores, estas funções mostram os menus secundários de pilotos, etapas, resultados e estatísticas. Devolvem o caracter correspondente à escolha do utilizador, após ser validado.

```
[925] void menuCabecalho(int tipo, conta_t *ptr_contador)
```

Função que mostra os diversos cabeçalhos para as subfunções.

Recebe:

tipo, inteiro para escolha do tipo de menu.
- **0,** menu principal
- **1**, menu pilotos
- **2**, menu etapas
- **3**, menu resultado

Pilotos

```
[309] ficha_t *pilFuncoes(ficha_t *ptr_piloto, etapa_t
*ptr_etapa, char op, conta_t *ptr_contador)
```

Função que direcciona o programa para as funções do menu Pilotos, inclui validações simples para casos como o limite máximo de pilotos atingido, inexistência de pilotos para eliminar, consultar ou listar.

Recebe:

op, variável do tipo char que guarda a escolha do utilizador para o submenu pilotos.

Devolve:

ptr_piloto, endereço para o inicio da estrutura piloto.

```
[361] ficha_t *pilInserir(ficha_t *ptr_piloto, conta_t
*ptr_contador)
```

Função para inserir um piloto dinâmicamente. Lê o valor do apelido e nome, posteriormente junta num vector único para guardar. Após a validação do

Bruno Tereso, 2091066@student.estg.ipleiria.pt
Marco Ferreira, 2092016@student.estg.ipleiria.pt

nome, lê o valor do avião e verifica se é um valor único, e, a idade compreendida entre os valores definidos.
Após a validação de todos os elementos, a função prepara a alocação de memória para guardar os dados do novo piloto.
Nesta função é tambem atribuido o valor 0 aos pontos do piloto e o valor 1 ao ptr_contador.grava, uma vez que depois de inserido um piloto a opção gravar passa a activa.

Devolve:

ptr_piloto, endereço para o inicio da estrutura piloto.

```
[416] void pilConsultar(ficha_t *ptr_piloto, etapa_t *ptr_etapa, conta_t *ptr_contador)
```

Função para consultar os dados de um piloto. Após listar os pilotos existentes de uma forma reduzida, esta função recebe o nome de um piloto ou o número de avião. A função verifica qual o valor introduzido automaticamente e mostra os dados pretendidos no ecrã. Além do nome, idade, pontos e número de avião, esta função mostra uma lista com todas as inscrições do piloto que inclui a posição em que ficou e os pontos atribuidos.

```
[456] void pilListagem(int tipo_lista, ficha_t *ptr_piloto, conta_t *ptr_contador)
```

Função que apresenta uma lista dos pilotos de forma completa, ou, de forma reduzida dividindo a listagem em dois para facilitar o aspecto visual do programa e escolha do utilizador.

Recebe:

tipo_lista, variável inteira com dois valores possíveis, 1 para uma lista completa e diferente de 1 para uma lista reduzida.

```
[488] ficha_t *pilEliminar(ficha_t *ptr_piloto, etapa_t *ptr_etapa, conta_t *ptr_contador)
```

Função ponteiro do tipo ficha_t que permite elimitar um piloto da lista de registos. Após a trivial validação de todos os elementos, a função verifica se o piloto está inscrito em alguma etapa, desta forma, impede que seja apagado no caso deste estar inscrito numa etapa já realizada, ou, no caso se estar inscrito numa outra etapa, pergunta a confirmação ao utilizado confrontando-o os novos dados. Se o utilizador optar por apagar o piloto, a função vai limpar a inscrição em todas as etapas por realizar onde este se encontra.
No caso de se confirmar a remoção a função prepara a realocação de memória para guardar menos um piloto.
Nesta função é tambem atribuido o valor 1 ao ptr_contador.grava, uma vez que depois de eliminado um piloto a opção gravar passa a activa.

```
[873] void pilInscricoes(etapa_t *ptr_etapa, ficha_t *ptr_piloto, conta_t *ptr_contador)
```

Função que mostra uma listagem simples das etapas em que um determinado piloto se inscreveu, usada no menu etapas.

```
[986] int pilProcurar(char nomenumero[], ficha_t *ptr_piloto, conta_t *ptr_contador)
```

Procura o indice de um piloto através do nome ou número de avião.

Recebe:

nomenumero[], vector de caracteres inserido pelo utilizador, é analisado na função para verificar se se trata de um número ou um nome.

Devolve: (um inteiro)

i, posição onde encontrou o piloto.
ERRO_NOME, quando foram introduzidos dados inválidos.
ERRO_DUPLICADO, quando existem dois ou mais nomes iguais.
ERRO_NAOEXISTE, se o nome de piloto ou número de avião não existe.

Bruno Tereso, 2091066@student.estg.ipleiria.pt
Marco Ferreira, 2092016@student.estg.ipleiria.pt

[1290] `int pilIndiceAviao(int numero_aviao, ficha_t *ptr_piloto, conta_t *ptr_contador)`

Procura o indice de um piloto correspondente a um avião.

Recebe:

numero_aviao, inteiro com o número do avião.

Devolve:

i, valor inteiro com o índice para o piloto do aviao 'numero_aviao'.

[1350] `int pilProcuraDuplicado(char nomenumero[], ficha_t *ptr_piloto, conta_t *ptr_contador)`

Função que recebe um nome e verifica se existem pilotos com nome igual.

Recebe:

nomenumero, vector de caracteres com o nome a comparar.

Devolve:

0 , se o nome é único.
1 , se o nome existe.

[1467] `void pilMostraInscricoes(ficha_t *ptr_piloto, etapa_t *ptr_etapa, int indice, int tipo)`

Função auxiliar de pilotos, mostra a lista de inscrições de um determinado piloto, consoante o tipo escolhido.

Recebe:

indice, inteiro com o indice do piloto a listar.
tipo, inteiro com o tipo de lista a usar, pode assumir o valor:
1, lista com etapa e pontuações
2, lista com nome e data

[1504] `void pilLimpaInscricoes(ficha_t *ptr_piloto, etapa_t *ptr_etapa, int indice)`

Função para auxiliar a pilEliminar, limpa todas as inscrições de um determinado piloto.

Recebe:

indice, indice do piloto a mostras

[1525] `ficha_t *pilApaga(ficha_t *ptr_piloto, conta_t *ptr_contador, int indice)`

Função ponteiro do tipo ficha_t para auxilio à função pilEliminar. Recebe o indice do piloto a eliminar, apaga a sua posição no vector e recua a posição de todos os vectores seguintes. Quando termina, realoca a memoria para ajustar ao número actualizado de pilotos.

Etapas

[581] `void etaFuncoes(etapa_t *ptr_etapa, ficha_t *ptr_piloto, char op, conta_t *ptr_contador)`

Função que direcciona o programa para as funções do menu Etapas, inclui validações simples para casos como o limite máximo de etapas atingido, inexistência de etapas para eliminar, consultar ou listar.

Recebe:

op, variável do tipo char que guarda a escolha do utilizador para o submenu etapas.

[649] `void etaInserir(etapa_t *ptr_etapa, conta_t *ptr_contador)`

Função para inserir uma etapa. Lê a designação, ordem, data e número máximo de participantes, após validação de todas as variaveis, insere o registo no vector de estruturas ptr_etapa. Quando guarda um registo com sucesso, preenche os vectores, tempos, pontos e inscritos a zero, muda o estado do etapa.realizada para zero e inicia o seu número de inscritos a zero.
Nesta função é tambem atribuido o valor 1 ao ptr_contador.grava, uma vez que depois de inserida uma etapa a opção gravar passa a activa.

```
[696] void etaActCorrente(etapa_t *ptr_etapa, conta_t
*ptr_contador)
```

Função para actualizar a etapa corrente (ptr_contador . etapa_corrente),usada depois de editar uma etapa, para certificar que o valor da etapa se mantem certo.

```
[713] void etaEditar(etapa_t *ptr_etapa, conta_t
*ptr_contador)
```

Função que permite editar uma etapa. Após listar as etapas existentes, esta função recebe e valida a ordem da etapa a editar, a letra R ao lado da data significa que a etapa já foi realizada e é por isso impossivel de editar. Após a escolha da etapa, o programa pergunta utilizador os novos dados da etapa, este tem tambem a hipotese de manter, apagar ou acrescentar inscrições. Após uma edição bem sucedida é tambem atribuido o valor 1 ao ptr_contador.grava, uma vez que depois de editada a etapa a opção gravar passa a activa.

```
[773] void etaInscreverPiloto(etapa_t *ptr_etapa, ficha_t
*ptr_piloto, conta_t *ptr_contador)
```

Função que inscreve um piloto numa etapa. Inicialmente a função mostra a lista de etapas para uma escolha mais facil do utilizador, depois de escolhida e validada uma etapa é apresentada uma lista reduzida dos pilotos onde é possivel escolher o piloto a inscrever pelo nome ou número de avião.

Depois de todas as escolhas validades, e verificado se o piloto já se encontra inscrito na estapa escolhida, a função guarda o avião do piloto escolhido no vector de inscritos, no primeiro espaço disponivel, e, inscrementa o número de inscritos.
Após uma inscrição bem sucedida é atribuido o valor 1 ao ptr_contador.grava, uma vez que a opção gravar passa a activa.

```
[830] void etaDados(etapa_t *ptr_etapa, ficha_t
*ptr_piloto, conta_t *ptr_contador)
```

Função para consultar os dados de uma etapa. Após listar as etapas existentes, esta função recebe e valida a ordem da etapa a consultar e mostra a informação no ecrã. Além de a ordem, data e local, mostra uma lista detalhada dos inscritos, incluido pontuações e tempos no caso de ser uma etapa realizada.

```
[1091] void etaListar(int tipo, etapa_t *ptr_etapa, conta_t
*ptr_contador)
```

Função que apresenta uma lista das etapas de forma completa, ou, de forma reduzida dividindo a listagem em dois para facilitar o aspecto visual do programa e escolha do utilizador.

Recebe:

tipo, variável inteira com dois valores possíveis, 1 para uma lista completa das etapas, 2 para uma lista que mostra apenas as etapas realizadas.

```
[1134] int etaProcurar(char nomenumero[40], etapa_t
*ptr_etapa, conta_t *ptr_contador)
```

Procura o indice de uma etapa através de um número ou designação.

Recebe:

Bruno Tereso, 2091066@student.estg.ipleiria.pt
Marco Ferreira, 2092016@student.estg.ipleiria.pt

nomenumero[], vector de caracteres inserido pelo utilizador, é analisado na função para verificar se este se trata de um número ou um nome.

Devolve: (um inteiro)
i, posição onde encontrou a etapa.
ERRO_NOME, quando foram introduzidos dados inválidos.
ERRO_DUPLICADO, quando existem dois ou mais nomes iguais.
ERRO_NAOEXISTE, se o nome ou número da etapa não existem.

```
[1315] int etaVerificaInscricao(ficha_t *ptr_piloto, etapa_t *ptr_etapa, int aviao_piloto, int indice_etapa, conta_t *ptr_contador)
```

Função que ao receber o numero de um avião, verifica se este está inscrito em alguma ou na etapa recebida.

Recebe:
aviao_piloto, número do avião que pretende procurar.
indice_etapa, índice da etapa onde verifica a inscrição.

Devolve:
i, inteiro com o resultado da pesquisa, de valor:
0 , se o piloto não tem qualquer inscrição.
1 , se o piloto está inscrito na etapa em teste.
-1, se o piloto está inscrito em qualquer outra etapa.
-2, se o piloto está inscrito numa etapa realizada.

```
[1373] int etaProcuraDuplicada(char nomenumero[], etapa_t *ptr_etapa)
```

Função que recebe um nome e verifica se existem etapas com nome igual.

Recebe:
nomenumero, vector de caracteres com o nome a comparar.

Devolve:
0 , se o nome é único.
1 , se o nome existe.

```
[1395] int etaInscricoesVagas(etapa_t *ptr_etapa, conta_t *ptr_contador)
```

Função do tipo inteiro que conta o número de vagas actuais no campeonato.

Devolve:
contador , número de inscrições actuais.

```
[1414] void etaLimpaInscricoes(int indice_etapa, etapa_t *ptr_etapa)
```

Função usada para limpar todas as inscrições de uma determinada etapa.

Recebe:
indice_etapa, índice da etapa onde apaga as inscrições.

```
[1428] int etaNumRealizadas(etapa_t *ptr_etapa)
```

Função que percorre a estrutura etapas e conta a quantidade de realizadas para apresentar no ecrã.

Devolve:
cont, número de etapas realizadas.

```
[1446] int etaProcuraPiloto (int etapa, int aviao, etapa_t *ptr_etapa)
```

Função que recebe um número de avião e pesquisa se o piloto está inscrito numa determinada etapa.

Bruno Tereso, 2091066@student.estg.ipleiria.pt
Marco Ferreira, 2092016@student.estg.ipleiria.pt

Recebe:

etapa, inteiro com o índice da etapa onde vai procurar o piloto.
aviao, inteiro com o número de avião a pesquisar.

Devolve:

i, índice do vector inscritos, onde se encontra o avião pesquisado.
ERRO_NAOEXISTE, se o piloto não está inscritos na etapa.

Resultados

```
[967] void resCabecalho(etapa_t *ptr_etapa, conta_t *ptr_contador)
```

Função que mostra o cabeçalho das funções dos resultados.

```
[1574] void resFuncoes(ficha_t *ptr_piloto, etapa_t *ptr_etapa, char op, conta_t *ptr_contador)
```

Função que direcciona o programa para as funções do menu Resultados, inclui validações simples para casos como a falta de etapas realizadas ou falta de etapas para editar.

Recebe:

op, variável do tipo char que guarda a escolha do utilizador para o submenu resultados.

```
[1623] void resLeCorrente(etapa_t *ptr_etapa, ficha_t *ptr_piloto, conta_t *ptr_contador)
```

Função usada para inicio da leitura dos tempos correspondentes à etapa corrente. Depois de ler os tempos com sucesso, procura um indice para a proxima etapa válida e passa a actual ao estado 'realizada'.
Após uma inscrição bem sucedida é atribuido o valor 1 ao ptr_contador.grava, uma vez que a opção gravar passa a activa.

```
[1656] void resLeTempos(int etapa, etapa_t *ptr_etapa, ficha_t *ptr_piloto, conta_t *ptr_contador)
```

Função usada em auxilio à resLeCorrente, lista e recebe todos os valores de tempo para um determinada etapa, validando o tempo inserido pela norma minutos:segundos.milesimos. A função é auxiliada pela função resLeSegundos, que converte os valores inseridos em milesimos de segundo de forma a guardar no vector tempos e ordenar mais facilmente.

Recebe:

etapa, índice da etapa onde vai guardar os tempos.

```
[1684] int resLeSegundos()
```

Função do tipo inteiro para leitura de um valor de tempo no formato mm:ss.ms. e transforma em milesimos de segundos, função para auxilio à resLeTempos.

Devolve:

total, inteiro com o tempo convertido em milesimos de segundo.

```
[1710] void resAlterarResultados (etapa_t *ptr_etapa, ficha_t *ptr_piloto, conta_t *ptr_contador)
```

Função que lista as etapas já realizadas e permite alterar os resultados de uma delas à escolha do utilizador. Depois de um ordem escolhida e validade, usa a função resRemPontuação para limpar a pontuação da etapa a alterar e a função resLeTempos para ler os novos tempos. Após uma alteração bem sucedida é atribuido o valor 1 ao ptr_contador.grava, uma vez que a opção gravar passa a activa.

```
[1739] void resOrdenaTempos (int etapa, etapa_t *ptr_etapa)
```

Bruno Tereso, 2091066@student.estg.ipleiria.pt
Marco Ferreira, 2092016@student.estg.ipleiria.pt

Função auxiliar para ordenar todos os tempos recebidos crescentemente e actualizar as posições no vector de inscritos, em função de uma determinada etapa. Para não perder rastreabilidade, troca tambem o aviao e o pontos de forma a usar correspondencia entre indices.

Recebe:

etapa, inteiro com o indice da etapa ordenar.

```
[1773] void resPontuacao (int etapa, ficha_t *ptr_piloto, etapa_t *ptr_etapa, conta_t *ptr_contador)
```

Função auxiliar desenvolvida para preencher automaticamente a pontuação em vectores do tipo etapa.pontos, para uma determinada etapa. Automaticamente, incrementa também a pontuação do piloto na estrutura pilotos.

Recebe:

etapa, inteiro com o indice da etapa onde vai ler os tempos e preencher os pontos.

```
[1801] void resRemPontuacao (int etapa, ficha_t *ptr_piloto, etapa_t *ptr_etapa, conta_t *ptr_contador)
```

Função auxiliar usada em conjunto com a etaAlterar, permite (ao contrário da resPontuacao) remover todos os pontos atribuidos numa determinada etapa e decrementar os valores creditados na estrutura pilotos.

Recebe:

etapa, inteiro com o indice da etapa onde vai limpar as pontuações.

```
[1828] void resConsultar (etapa_t *ptr_etapa, ficha_t *ptr_piloto, conta_t *ptr_contador)
```

Função para consultar os resultados de uma etapa. Após listar as etapas realizadas, esta função recebe e valida a ordem da etapa a consultar, e, chama a função resEtapa para apresentar a lista de pilotos detalhada.

```
[1855] void resEtapa (int etapa, etapa_t *ptr_etapa, ficha_t *ptr_piloto, conta_t *ptr_contador) {
```

Função desenvolvida para apoio à resConsultar. Para uma etapa especifica, mostra uma lista de pilotos ordenada pela classificação, com orespectivo nome, tempo e número de avião.

Recebe:

etapa, inteiro com o indice da etapa onde vai ler os dados a apresentar.

```
[1903] void resOrdenaClass (ficha_t *ptr_piloto, conta_t *ptr_contador)
```

Função que permite ordenar todos os registos da estrutura pilotos em ordem ao número de pontos, decrescentemente.

```
[1937] void resOrdenaClassAviao (ficha_t *ptr_piloto, conta_t *ptr_contador)
```

Necessária para a classificação mostrada no menu de estatísticas, esta função permite ordenar os pilotos de uma etapa em ordem ao número do avião.

Auxiliares

```
[1024] void auxLeString(int tipo, char texto[], char pergunta[30], char erro[50], int inferior, int superior)
```

Função auxiliar que lê e valida um vector de caracteres.

Bruno Tereso, 2091066@student.estg.ipleiria.pt
Marco Ferreira, 2092016@student.estg.ipleiria.pt

Recebe:

tipo, inteiro do tipo 1 (só texto) ou 2 (texto e números).
texto, vector de caracteres que passa por referencia, para guardar os valores obtidos na função.
pergunta[], questão a colocar ao utilizador.
erro[], mensagem de erro a apresentar no caso de leitura inválida.
inferior, valor mínimo de caracteres da frase.
superior, valor máximo de caracteres da frase.

```
[1058] int auxLeInteiro (char pergunta[40], int menor, int maior)
```

Função auxiliar que lê e valida um número inteiro.

Recebe:

menor, inteiro que guarda o limite inferior do valor a ler.
maior, inteiro que guarda o limite superior do valor a ler.

```
[1172] void auxLeData(int numero_etapa, etapa_t *ptr_etapa)
```

Função auxiliar para ler a data de uma etapa, o valor é lido na altura quem que é inserida ou alterada uma etapa.
Esta função converte os valores introduzidos em dias, contado os dias desde 01.01.1900 até à data inserida. Para verificar se a data inserida é valida, esta função vai procurar e converter as datas das etapas imediatamente antes e depois da inserida.

Esta função tem erros conhecidos, não está a considerar anos bisextos e em determinada altura, contabiliza todos os meses com 30 dias. A sua complexidade e a falta de tempo não nos permitiu reescrever e repensar o codigo escrito com a intensão de posteriormente aperfeiçoar.

Recebe:

numero_etapa, índice da etapa onde vai ler a data.

```
[1307] void auxPausa()
```

Muda de linha e pausa o sistema.

```
[2581] void auxIniciaValores(ficha_t *ptr_piloto, etapa_t *ptr_etapa, conta_t *ptr_contador)
```

Função auxiliar para inicializar valores de estruturas a zero, contador e etapas. O programa usa esta função sempre que o utilizador nega a leitura do ficheiro de dados.

```
[2542] ficha_t *auxLer(conta_t *ptr_contador, ficha_t *ptr_piloto, etapa_t *ptr_etapa)
```

Função ponteiro que permite ao utilizador ler o ficheiros de dados ou iniciar o programa com os valores limpos. Se não existir um ficheiro ignora o menu com a pergunta e entra directamente no programa

Devolve:

ptr_piloto, endereço para o inicio da estrutura piloto.

```
[2506] char auxSair(conta_t *ptr_contador, ficha_t *ptr_piloto, etapa_t *ptr_etapa)
```

Função auxiliar que mostra um pequeno menu a confirmar a saida no caso dos dados não estarem gravados. Permite discartar todos os dados ou gravar antes de sair.

```
[1879] tmp_t auxConverteMseg (int tempo_mseg)
```

Função auxiliar que ao receber um tempo em milésimos de segundos o converte para um estrutura do tipo tmp_t com valores diferenciados para minutos, segundos e milesimos. Usada para restaurar os tempos num formato mais proprio para apresentação em listas ou impressão.

Bruno Tereso, 2091066@student.estg.ipleiria.pt
Marco Ferreira, 2092016@student.estg.ipleiria.pt

Recebe:

tempo_mseg, inteiro com o tempo em msegundos, guardado na estrutura etapas.

Devolve:

tempo, estrutura tempo com os minutos, segundos e milesimos de segundos separados.

```
[2378] void auxOrdenaMatriz (int vector[][2], int max, char tipo)
```

Função auxiliar utilizada no calculo de estatísticas, ordena uma matriz sempre pelos valores contidos na segunda linha. Ao trocar os valores na segunda linha, troca igualmente na primeira. Permite uma ordenação ascendente ou descendente.

Recebe:

vector[][], matriz de 2 linhas com valores a ordenar.

max, inteiro com o valor máximo de ordenações a usar na indexação por borbulhamento.

Estatística

```
[2045] void estFuncoes(ficha_t *ptr_piloto, etapa_t *ptr_etapa, char op, conta_t *ptr_contador)
```

Função que direcciona o programa para as funções do menu Estatísticas, inclui validações simples para casos como o limite mínimo de etapas realizadas ou inexistência de etapas para mostrar.

Recebe:

op, variável do tipo char que guarda a escolha do utilizador para o submenu estatísticas.

```
[2090] void estHistorico (etapa_t *ptr_etapa, ficha_t *ptr_piloto, conta_t *ptr_contador)
```

Função que mostra ao utilizador uma tabela com todos os resultados do campeonato num só ecrã, ordenando os pilotos nas linhas e as etapas nas colunas, permite simplificar a visualização directa dos resultados ou possível impressão.

```
[2124] void estCabecalhoHistorico (etapa_t *ptr_etapa, conta_t *ptr_contador)
```

Função usada em auxilio à estHistorico, mostra apenas o cabeçalho do historial para piloto de forma a permitir uma troca facil de dados quando esta é ordenada por aviões/pontos.

```
[1973] void estMostraClassificacao (int simples, ficha_t *ptr_piloto, conta_t *ptr_contador)
```

Função usada para listar a classificação geral do campeonato. Permite uma listagem simples ou uma listagem mais complexa onde é possivel alternar entre ordenação por pontos ou número de avião. Desta forma, permite a usabilidade nos menus de resultados e estatísticas.

Recebe:

simples, variável inteira que define o tipo de lista a apresentar.

1, listagem normal sem opção de alternação.

0, listagem completa com opção para ordenar por pontos ou avião.

```
[2157] void estDados (conta_t *ptr_contador, etapa_t *ptr_etapa, ficha_t *ptr_piloto)
```

Função principal e mais completa do menu de estatisticas, possui declarações para um vector temporario (vencedores) que guarda o número do avião vendedor de cada etapa, e, matrizes temporarias para pilotos e etapas,

usadas ao longo de toda a função e subfunções para permitir ordenações sem perca de rastreabilidades. O corpo da função serve quase para apenas chamar outras funções que calculam e imprimem os valores das várias estatisticas pedidas. Talvez por fazer muitos calculos, foi necessário limpar o buffer no decorrer da função pois de outra forma ela apresentava valores errados.

```
[2202] void auxVencedores ( int vencedores[], etapa_t *ptr_etapa)
```

Função usada em auxilio à estDados, recebe um vector por referencia e vai percorrer todas as etapas, verificar qual foi o avião vencedor de cada um e guardar o valor no vector vencedores, este vector é utilizado em outras funções de apoio para facilitar os calculos e percepção do codigo.

Recebe:

vencedores, vector de inteiros que guarda o avião dos pilotos vencedores.

```
[2223] void estNumVitorias ( int pilotos[][2], int vencedores[], ficha_t *ptr_piloto, conta_t *ptr_contador, char tipo)
```

Função usada em auxilio à estDados, preenche a matriz pilotos com o número do avião na primeira linha e o número de vitórias na segunda, posteriormente ordena a matriz tento em conta a segunda linha mas actualizando o valor da primeira. Desta forma ordena pelo piloto com mais ou menos victorias, consoante o tipo de ordenação escolhida.

Recebe:

pilotos, matriz auxiliar temporária para calculos e ordenações.
vencedores, vector que tem o vendedor de cada etapa.
tipo, tipo de ordenação, 'A'scendente ou 'D'escendente.

```
[2266] void estPctVictorias ( int pilotos[][2], ficha_t *ptr_piloto, conta_t *ptr_contador)
```

Função usada em auxilio à estDados, atravéz da matriz, conta o número total de vitórias e cálcula a média com o número total de pilotos.

Recebe:

pilotos, matriz auxiliar temporária para calculos e ordenações.

```
[2288] void estEtapas ( int etapas[][2], etapa_t *ptr_etapa, conta_t *ptr_contador, char tipo)
```

Função usada em auxilio à estDados, preenche a matriz etapas com o indice da etapa na primeira linha e o número de inscritos na segunda, posteriormente ordena a matriz tento em conta a segunda linha mas actualizando o valor da primeira. Desta forma ordena pelo número de inscritos permitindo uma ordenação ascendente ou descendente.

Recebe:

etapas, matriz auxiliar temporária para calculos e ordenações.
tipo, tipo de ordenação, 'A'scendente ou 'D'escendente.

```
[2323] void estMediaParticipantes (etapa_t *ptr_etapa, conta_t *ptr_contador)
```

Função usada em auxilio à estDados, conta o número de etapas inseridas e os participantes de cada uma e mostra ao utilizador a média de pilotos por etapa.

```
[2343] void estVitoriasConsecutivas (int vencedores[], etapa_t *ptr_etapa, ficha_t *ptr_piloto, conta_t *ptr_contador)
```

Bruno Tereso, 2091066@student.estg.ipleiria.pt
Marco Ferreira, 2092016@student.estg.ipleiria.pt

Função usada em auxilio à estDados, recebe e analiza o vector vencedores, procurando pilotoscom duas ou mais vitórias consecutivas.

Recebe:

vencedores, vector que tem o vendedor de cada etapa.

Ficheiros

```
[2464] ficha_t *fichLer(conta_t *ptr_contador, etapa_t
*ptr_etapa)
```

Função ponteiro usada para ler os valores do ficheiro dados.dat. Verifica existência de erros por inexistência do ficheiro ou falhas de leitura.

Devolve:

ptr_piloto, endereço para o inicio da estrutura piloto.

```
[2429] void fichGravar(conta_t *ptr_contador, etapa_t
*ptr_etapa, ficha_t *ptr_piloto)
```

Função usada para escrever os valores do ficheiro dados.dat. Verifica existência de erros na criação do ficheiro ou falhas na escrita.

Bruno Tereso, 2091066@student.estg.ipleiria.pt
Marco Ferreira, 2092016@student.estg.ipleiria.pt

Anexo I – Código Fonte

Código fonte do programa .c é enviado em anexo juntamente com o relatório, o código aqui visivel foi transformado para mais fácil apresentação e leitura, tendo sido omitidos muitos dos comentários originais. Esta versão do código fonte não compila e os números de linha podem não corresponder aos do ficheiro de código.

Declaração de Livrarias

```
0000 |  /*  ###################################################################
0001 |                         DECLARAÇÃO DE LIVRARIAS
0002 |       ################################################################### */
0003 |
0004 |  #include <stdio.h>
0005 |  #include <math.h>
0006 |  #include <stdlib.h>
0007 |  #include <time.h>
0008 |
0009 |
```

Declaração de Constantes

```
0010 |  /*  ###################################################################
0011 |                         DECLARAÇÃO DE CONSTANTES
0012 |       ################################################################### */
0013 |
0014 |  #define MAX_PILOTOS 200 // 200 número máximo de pilotos
0015 |  #define MAX_PILOTOS_ETAPA 20 // 20 número máximo de pilotos por etapa
0016 |  #define MAX_ETAPAS 10 // 10 número máximo de etapas
0017 |
0018 |  #define PROGRAMA "\n\n\t\t\t\tESTG Air Race 2010\n" // titulo do programa
0019 |
0020 |  #define ERRO_NOME 500 // introduzidos dados inválidos, um nome curto demais
0021 |  #define ERRO_DUPLICADO 998 // erro, existem nomes repetidos
0022 |  #define ERRO_NAOEXISTE 999 // nome da etapa ou piloto não existe
0023 |  #define SAIR '.' // sai do programa depois de gravar
0024 |  #define NAOSAIR ','
0025 |  #define NUMTOPS 3 // numero de pilotos/etapas a apresentar nos tops ( estatísticas)
0026 |
0027 |  #define ETAPA_EXTREMO 501 // devolve extremos das etapas quando se compara a data
0028 |  #define DIAS_MAX 99999999 /* valor máximo em dias para a etapa seguinte,
0029 |  usado para validar data, significa que não existe um etapa com data posterior à inserida */
0030 |  #define DIAS_MIN 1 /* valor máximo em dias para a etapa anterior,
0031 |  usado para validar data, significa que não existe um etapa com data anterior à inserida*/
0032 |  #define CHARS_NOME 31
```

Bruno Tereso, 2091066@student.estg.ipleiria.pt
Marco Ferreira, 2092016@student.estg.ipleiria.pt

```
0033 |
0034 |
```

Declaração de Estruturas

```
0035 |  /*  ################################################################
0036 |                        DECLARAÇÃO DE ESTRUTURAS
0037 |       ################################################################ */
0038 |
0039 |  typedef struct { // estrutura para a data
0040 |      int dia, mes, ano;
0041 |  }data_t;
0042 |
0043 |  typedef struct { // estrutura para tempos
0044 |      int min, seg, mseg;
0045 |  }tmp_t;
0046 |
0047 |  typedef struct { // estrutura para ficha de piloto
0048 |      char nome[CHARS_NOME];  // último nome
0049 |      int idade;          // idade
0050 |      int aviao;      // numero aviao
0051 |      int pontos;
0052 |  } ficha_t;
0053 |
0054 |  typedef struct { // estrutura para as etapas
0055 |      char nome[CHARS_NOME];  // designação da etapa
0056 |      int ordem;          // ordem da etapa
0057 |      data_t data;      // estrutra data_t que guarda a data da etapa
0058 |      int participantes;      // numero de pilotos a participar (max20)
0059 |      int totalinscritos;
0060 |      int inscritos[MAX_PILOTOS_ETAPA]; // vector q vai guardar os numeros dos avioes inscritos
0061 |      int tempos[MAX_PILOTOS_ETAPA];
0062 |      int pontos[MAX_PILOTOS_ETAPA];
0063 |      int realizada; // variável logica 0 - por realizar 1 - realizada
0064 |  } etapa_t;
0065 |
0066 |  typedef struct { // estrutura que guarda os pilotos inscritos e etapas inseridas
0067 |      int pilotos;
0068 |      int etapas;
0069 |      int realizadas; // em testes
0070 |      int etapa_corrente;
0071 |      int grava; // 1 se houver modificações por gravar
0072 |      //int classgeral[MAX_PILOTOS];
0073 |  } conta_t;
0074 |
0075 |
0076 |
0077 |
```

Bruno Tereso, 2091066@student.estg.ipleiria.pt
Marco Ferreira, 2092016@student.estg.ipleiria.pt

```
0078 |
0079 |

Declaração de Prototipos
0080 |  /*  ################################################################
0081 |                          PROTOTIPOS DE FUNÇÕES
0082 |       ################################################################ */
0083 |
0084 |  // Funções principais
0085 |

Prototipos de Pilotos
0086 |  // Pilotos
0087 |  char pilMenu(conta_t *ptr_contador); // Mostra o menu de Pilotos no ecrã.
0088 |  ficha_t *pilFuncoes(ficha_t *ptr_piloto, etapa_t *ptr_etapa, char op, conta_t *ptr_contador);
0089 |  ficha_t *pilInserir(ficha_t *ptr_piloto, conta_t *ptr_contador); // Insere um novo registo nas fichas de pilotos
0090 |  void pilConsultar(ficha_t *ptr_piloto, etapa_t *ptr_etapa, conta_t *ptr_contador);// Consulta os dados de um piloto.
0091 |  ficha_t *pilEliminar(ficha_t *ptr_piloto, etapa_t *ptr_etapa, conta_t *ptr_contador); // Apaga um piloto.
0092 |  void pilListagem(int tipo_lista, ficha_t *ptr_piloto, conta_t *ptr_contador);
0093 |  void pilInscricoes(etapa_t *ptr_etapa, ficha_t *ptr_piloto, conta_t *ptr_contador); // Lista de incrições de um piloto
0094 |  int pilProcurar(char nomenumero[31], ficha_t *ptr_piloto, conta_t *ptr_contador); // Procura o indice de um
0095 |  int pilIndiceUltimo(ficha_t *ptr_piloto); // Devolve o próximo indice livre do vector fichas de pilotos
0096 |  int pilProcuraDuplicado(char nomenumero[], ficha_t *ptr_piloto, conta_t *ptr_contador); // Procura nomes repetidos
0097 |  void pilMostraInscricoes(ficha_t *ptr_piloto, etapa_t *ptr_etapa, int indice, int tipo); // Mostra inscrições de um piloto.
0098 |  void pilLimpaInscricoes(ficha_t *ptr_piloto, etapa_t *ptr_etapa, int indice); // Limpa as inscrições de um piloto
0099 |  ficha_t *pilApaga(ficha_t *ptr_piloto, conta_t *ptr_contador, int indice); // Apaga o registo de um piloto.
0100 |  int pilIndiceAviao(int numero_aviao, ficha_t *ptr_piloto, conta_t *ptr_contador); // Procura o avião indicado na lista de pilotos
0101 |

Prototipos de Etapas
0102 |  // Etapas
0103 |  char etaMenu(conta_t *ptr_contador); // Mostra o menu etapas
0104 |  void etaFuncoes(etapa_t *ptr_etapa, ficha_t *ptr_piloto, char op, conta_t *ptr_contador);
0105 |  void etaInserir(); // Insere um novo registo na lista de etapas
0106 |  void etaActCorrente(etapa_t *ptr_etapa, conta_t *ptr_contador);
0107 |  void etaEditar(etapa_t *ptr_etapa, conta_t *ptr_contador); // Lista as etapas e altera uma à escolha do utilizador
0108 |  void etaInscreverPiloto(etapa_t *ptr_etapa, ficha_t *ptr_piloto, conta_t *ptr_contador); // Mostra etapas e inscreve um piloto
0109 |  void etaDados(etapa_t *ptr_etapa, ficha_t *ptr_piloto, conta_t *ptr_contador); // Lista etapas e mostra os dados de uma delas
0110 |  void etaListar(int tipo, etapa_t *ptr_etapa, conta_t *ptr_contador); // Mostra uma lista de etapas para auxílio em funções
0111 |  int etaProcurar(char nomenumero[40], etapa_t *ptr_etapa, conta_t *ptr_contador); // Procura o índice de um etapa dado o nome
0112 |  int etaVerificaInscricao(ficha_t *ptr_piloto, etapa_t *ptr_etapa, int aviao_piloto, int indice_etapa, conta_t *ptr_contador);
0113 |  int etaProcuraDuplicada(char nomenumero[], etapa_t *ptr_etapa); // Verifica se existe mais do que uma etapa com o mesmo nome.
0114 |  int etaInscricoesVagas(etapa_t *ptr_etapa, conta_t *ptr_conta); // Conta o número de vagas do campeonato, em todas as etapas.
0115 |  void etaLimpaInscricoes(int indice_etapa, etapa_t *ptr_etapa); // Apaga todas as inscrições de uma etapa.
0116 |  int etaNumRealizadas(etapa_t *ptr_etapa); // Conta o número de etapa realizadas para apresentar no menu principal.
0117 |  int etaProcuraPiloto (int etapa, int aviao, etapa_t *ptr_etapa);
```

```
0118 |
```

Prototipos de Resultados

```
0119 |  // Resultados
0120 |  char resMenu(conta_t *ptr_contador, etapa_t *ptr_etapa);
0121 |  void resFuncoes(ficha_t *ptr_piloto, etapa_t *ptr_etapa, char op, conta_t *ptr_contador);
0122 |  void resLeCorrente(etapa_t *ptr_etapa, ficha_t *ptr_piloto, conta_t *ptr_contador);
0123 |  void resCabecalho(etapa_t *ptr_etapa, conta_t *ptr_contador);
0124 |  void resAlterarResultados (etapa_t *ptr_etapa, ficha_t *ptr_piloto, conta_t *ptr_contador);
0125 |  void resOrdenaTempos (int etapa, etapa_t *ptr_etapa);
0126 |  void resPontuacao (int etapa, ficha_t *ptr_piloto, etapa_t *ptr_etapa, conta_t *ptr_contador);
0127 |  void resRemPontuacao (int etapa, ficha_t *ptr_piloto, etapa_t *ptr_etapa, conta_t *ptr_contador);
0128 |  void resLeTempos(int etapa, etapa_t *ptr_etapa, ficha_t *ptr_piloto, conta_t *ptr_contador);
0129 |  void resConsultar (etapa_t *ptr_etapa, ficha_t *ptr_piloto, conta_t *ptr_contador);
0130 |  int resLeSegundos();
0131 |  void resActClassificacao (etapa_t *ptr_etapa, ficha_t *ptr_piloto, conta_t *ptr_contador);
0132 |  void resOrdenaClass (ficha_t *ptr_piloto, conta_t *ptr_contador);
0133 |  void resEtapa (int etapa, etapa_t *ptr_etapa, ficha_t *ptr_piloto, conta_t *ptr_contador);
0134 |  void resOrdenaClassAviao (ficha_t *ptr_piloto, conta_t *ptr_contador);
0135 |
```

Prototipos de Estatisticas

```
0136 |  // Estatisticas
0137 |  char estMenu(conta_t *ptr_contador);
0138 |  void estFuncoes(ficha_t *ptr_piloto, etapa_t *ptr_etapa, char op, conta_t *ptr_contador);
0139 |  void estHistorico (etapa_t *ptr_etapa, ficha_t *ptr_piloto, conta_t *ptr_contador);
0140 |  void estCabecalhoHistorico (etapa_t *ptr_etapa, conta_t *ptr_contador);
0141 |  void estMostraClassificacao (int simples, ficha_t *ptr_piloto, conta_t *ptr_contador);
0142 |  void estDados (conta_t *ptr_contador, etapa_t *ptr_etapa, ficha_t *ptr_piloto);
0143 |  void estNumVitorias ( int pilotos[][2], int vencedores[], ficha_t *ptr_piloto, conta_t *ptr_contador, char tipo);
0144 |  void estPctVictorias ( int pilotos[][2], ficha_t *ptr_piloto, conta_t *ptr_contador);
0145 |  void estEtapas ( int etapas[][2], etapa_t *ptr_etapa, conta_t *ptr_contador, char tipo);
0146 |  void estMediaParticipantes ( etapa_t *ptr_etapa, conta_t *ptr_contador);
0147 |  void estVitoriasConsecutivas (int vencedores[], etapa_t *ptr_etapa, ficha_t *ptr_piloto, conta_t *ptr_contador);
0148 |
```

Prototipos de Ficheiros

```
0149 |  // Ficheiros
0150 |  void fichGravar(conta_t *ptr_contador, etapa_t *ptr_etapa, ficha_t *ptr_piloto);
0151 |  ficha_t *fichLer(conta_t *ptr_contador, etapa_t *ptr_etapa);
0152 |
```

Prototipos de Funções Auxiliares

```
0153 |  // Funções diversas
0154 |  char menuPrincipal(conta_t *ptr_contador); // Mostra o menu principal no ecrã.
0155 |  void menuCabecalho(int tipo, conta_t *ptr_contador); // Mostra o cabeçalho do Programa.
```

Bruno Tereso, 2091066@student.estg.ipleiria.pt
Marco Ferreira, 2092016@student.estg.ipleiria.pt

```
0156 |  void auxLeString(int tipo, char texto[], char pergunta[30], char erro[50], int inferior, int superior);
0157 |  int auxLeInteiro (char pergunta[40], int menor, int maior); // Le e valida um número inteiro
0158 |  void auxLeData(int numero_etapa, etapa_t *ptr_etapa); // Le uma data válida para inserir uma etapa.
0159 |  void auxPausa(); // Muda de linha e pausa o sistema.
0160 |  void auxIniciaValores(ficha_t *ptr_piloto, etapa_t *ptr_etapa, conta_t *ptr_contador);
0161 |  char auxSair(conta_t *ptr_contador, ficha_t *ptr_piloto, etapa_t *ptr_etapa);
0162 |  tmp_t auxConverteMseg (int tempo_mseg);
0163 |  void auxVencedores ( int vencedores[], etapa_t *ptr_etapa);
0164 |  void auxOrdenaMatriz (int vector[][2], int max, char tipo);
0165 |  ficha_t *auxLer(conta_t *ptr_contador, ficha_t *ptr_piloto, etapa_t *ptr_etapa);
0166 |
0173 |
0174 |
0175 |
```

Programa Principal

```
0176 |  /*  ###################################################################
0177 |                          PROGRAMA PRINCIPAL
0178 |      ################################################################### */
0179 |
0180 |  main (void) {
0181 |
0182 |      char opcao_menu;
0183 |
0184 |      ficha_t *ptr_piloto;
0185 |      etapa_t ptr_etapa[MAX_ETAPAS];
0186 |      conta_t ptr_contador;
0187 |
0188 |      ptr_piloto = NULL;
0189 |
0190 |      ptr_piloto = auxLer(&ptr_contador, ptr_piloto, ptr_etapa); // le dados de um ficheiro ou inicializa os valores a zero
0191 |
0192 |      do {
0193 |          ptr_contador.realizadas = etaNumRealizadas(ptr_etapa); // conta as etapas realizadas antes de desenhar o menu
0194 |          opcao_menu = menuPrincipal(&ptr_contador);
0195 |          switch (opcao_menu) {
0196 |          case 'P':
0197 |              do { // espera pela tecla V, mantem o menu nos pilotos indeterminadamente
0198 |                  opcao_menu = pilMenu(&ptr_contador);
0199 |                  ptr_piloto = pilFuncoes(ptr_piloto, ptr_etapa, opcao_menu, &ptr_contador);
0200 |              } while (opcao_menu != 'V');
0201 |              break;
0202 |
0203 |          case 'E':
0204 |              do { // espera pela tecla V, mantem o menu nas etapas indeterminadamente
0205 |                  opcao_menu = etaMenu(&ptr_contador);
0206 |                  etaFuncoes(ptr_etapa, ptr_piloto, opcao_menu, &ptr_contador);
```

Bruno Tereso, 2091066@student.estg.ipleiria.pt
Marco Ferreira, 2092016@student.estg.ipleiria.pt

```
0207 |              } while (opcao_menu != 'V');
0208 |              break;
0209 |
0210 |          case 'R':
0211 |              do { // espera pela tecla V, mantem o menu nos resultados indeterminadamente
0212 |                  opcao_menu = resMenu(&ptr_contador, ptr_etapa);
0213 |                  resFuncoes(ptr_piloto, ptr_etapa, opcao_menu, &ptr_contador);
0214 |              } while (opcao_menu != 'V');
0215 |              break;
0216 |
0217 |          case 'T':
0218 |              do {
0219 |                  opcao_menu = estMenu(&ptr_contador);
0220 |                  estFuncoes(ptr_piloto, ptr_etapa, opcao_menu, &ptr_contador);
0221 |              } while (opcao_menu != 'V');
0222 |              break;
0223 |
0224 |          case 'G':
0225 |              fichGravar(&ptr_contador, ptr_etapa, ptr_piloto);
0226 |              break;
0227 |
0228 |          case 'S':
0229 |              opcao_menu = auxSair(&ptr_contador, ptr_piloto, ptr_etapa);
0230 |              break;
0231 |
0240 |          }
0241 |      } while (opcao_menu != SAIR);
0242 |      free(ptr_piloto);
0243 |  }
0244 |
0245 |
```

Funções Principais

```
0246 |  /*  ################################################################
0247 |                          FUNÇÕES PRINCIPAIS
0248 |      ################################################################ */
0249 |
0250 |  char menuPrincipal(conta_t *ptr_contador) {
0251 |      /*  Mostra o menu principal no ecrã.
0257 |
0258 |      char op;
0259 |
0260 |      do {
0261 |          menuCabecalho(0, ptr_contador); // 0 = menu principal | 1 = menu pilotos | 2 = menu etapas
0262 |          printf("\n\n\tP - Pilotos");
0263 |          printf("\n\tE - Etapas");
0264 |          printf("\n\tR - Resultados");
```

Bruno Tereso, 2091066@student.estg.ipleiria.pt
Marco Ferreira, 2092016@student.estg.ipleiria.pt

```
0265 |          printf("\n\tT - Estatisticas");
0266 |          if ( ptr_contador->grava == 1 )
0267 |              printf("\n\tG - Gravar Dados ( Necessario )");
0268 |          else
0269 |              printf("\n\tG - Gravar Dados");
0270 |          printf("\n\tS - Sair\n\t");
0271 |          //menu debug (ao remover verificar o while)
0272 |          printf("\n\n\n\tZ - DEBUG");
0273 |
0274 |          fflush(stdin);
0275 |          op = getch();
0276 |          op = toupper(op);
0277 |      } while (op != 'Z' && op != 'S' && op != 'P' && op != 'E' && op != 'R' && op != 'T' && op != 'G');
0278 |
0279 |      return op;
0280 |  }
0281 |
0282 |
0283 |  char pilMenu(conta_t *ptr_contador) {
0284 |      /*  Mostra o menu de Pilotos no ecrã.
0290 |
0291 |      char op;
0292 |
0293 |      do {
0294 |          menuCabecalho(1, ptr_contador); // 0 = menu principal | 1 = menu pilotos | 2 = menu etapas
0295 |          printf("\n\n\tI - Inserir Piloto");
0296 |          printf("\n\tE - Eliminar Piloto");
0297 |          printf("\n\tC - Consultar Dados Piloto");
0298 |          printf("\n\tL - Listar todos os Pilotos");
0299 |          printf("\n\n\tV - Voltar ao Menu Principal");
0300 |          fflush(stdin);
0301 |          op = getch();
0302 |          op = toupper(op);
0303 |      } while (op != 'V' && op != 'I' && op != 'E' && op != 'C' && op != 'L');
0304 |
0305 |      return op;
0306 |  }
0307 |
0308 |
0309 |  ficha_t *pilFuncoes(ficha_t *ptr_piloto, etapa_t *ptr_etapa, char op, conta_t *ptr_contador) {
0310 |      /*  Recebe a opção escolhida e direcciona para a função a usar.
0317 |
0318 |      switch (op) {
0319 |      case 'I':  //inserir piloto
0320 |          if (ptr_contador->pilotos < MAX_PILOTOS - 1)
0321 |              ptr_piloto = pilInserir(ptr_piloto, ptr_contador);
0322 |          else {
```

Bruno Tereso, 2091066@student.estg.ipleiria.pt
Marco Ferreira, 2092016@student.estg.ipleiria.pt

```
0323 |              printf("\n\tLamento, atingiu o numero maximo de pilotos...");
0324 |              auxPausa();
0325 |          }
0326 |          break;
0327 |
0328 |      case 'E':  //elimitar piloto
0329 |          if (ptr_contador->pilotos > 0)
0330 |              pilEliminar(ptr_piloto, ptr_etapa, ptr_contador);
0331 |          else {
0332 |              printf("\n\n\tImpossivel eliminar pilotos, de momento nao tem nenhum inscrito...");
0333 |              auxPausa();
0334 |          }
0335 |          break;
0336 |
0337 |      case 'C':  //consultar dados de piloto
0338 |          if (ptr_contador->pilotos > 0)
0339 |              pilConsultar(ptr_piloto, ptr_etapa, ptr_contador);
0340 |          else {
0341 |              printf("\n\n\tImpossivel consultar pilotos, de momento nao tem nenhum inscrito...");
0342 |              auxPausa();
0343 |          }
0344 |          break;
0345 |
0346 |      case 'L':  //lista de pilotos
0347 |          if (ptr_contador->pilotos > 0) {
0348 |              menuCabecalho(1, ptr_contador); // 0 = menu principal | 1 = menu pilotos | 2 = menu etapas
0349 |              pilListagem(0, ptr_piloto, ptr_contador);
0350 |              auxPausa();
0351 |          } else {
0352 |              printf("\n\n\tLista vazia, de momento nao tem nenhum inscrito...");
0353 |              auxPausa();
0354 |          }
0355 |          break;
0356 |      }
0357 |      return ptr_piloto;
0358 |  }
0359 |
0360 |
0361 |  ficha_t *pilInserir(ficha_t *ptr_piloto, conta_t *ptr_contador) {
0362 |      /*  Insere um novo registo nas fichas de pilotos
0367 |
0368 |      char apelido[15], nome[15];
0369 |      int i, aviao_validado = 1, numero_aviao, idade, indice;
0370 |      ficha_t *aux;
0371 |
0372 |      aux = NULL;
0373 |
```

Bruno Tereso, 2091066@student.estg.ipleiria.pt
Marco Ferreira, 2092016@student.estg.ipleiria.pt

```
0374 |      menuCabecalho(1, ptr_contador); // 0 = menu principal | 1 = menu pilotos | 2 = menu etapas
0375 |      printf("\n\n\tInserir Piloto: \n\n");
0376 |
0377 |      auxLeString(1, apelido, "\tApelido:", "Erro, O Apelido do piloto deve ter entre 2 e 15 caracteres\n", 2, 15);
0378 |      auxLeString(1, nome, "\tNome:", "Erro, O Nome do piloto deve ter entre 2 e 15 caracteres\n", 2, 15);
0379 |
0380 |      strcat(nome, " "); // coloca um espaço a seguir ao nome
0381 |      strcat(nome, apelido);   // completa a string do nome com o apelido, max 31 caracteres
0382 |
0383 |      do { // verifica q se trata de um valor unico
0384 |          numero_aviao = auxLeInteiro("\tNumero do Aviao: ", 1, ERRO_NAOEXISTE);
0385 |          for (i = 0; i < ptr_contador->pilotos; i++)
0386 |              if (numero_aviao == ptr_piloto[i].aviao) {
0387 |                  printf("\tErro, ja existe um piloto com esse numero de aviao.\n");
0388 |                  aviao_validado = 0;
0389 |                  i = ptr_contador->pilotos;
0390 |              } else
0391 |                  aviao_validado = 1;
0392 |      } while (aviao_validado == 0);
0393 |
0394 |      idade = auxLeInteiro ("\tIdade do Piloto: ", 18, 80); //le um valor inteiro entre o 'menor' e o 'maior' inclusive
0395 |      printf("\n\tPiloto inserido com sucesso...");
0396 |      aux = (ficha_t*)realloc(ptr_piloto, (ptr_contador->pilotos+1)*sizeof(ficha_t));
0397 |      if ( aux == NULL) {
0398 |          printf("\n\tErro de Memoria.");
0399 |          auxPausa();
0400 |          return ptr_piloto;
0401 |      } else {
0402 |          ptr_piloto = aux; // leu sem erros, aponta para o piloto
0403 |          strcpy(ptr_piloto[ptr_contador->pilotos].nome, nome);
0404 |          ptr_piloto[ptr_contador->pilotos].idade = idade;
0405 |          ptr_piloto[ptr_contador->pilotos].aviao = numero_aviao;
0406 |          ptr_piloto[ptr_contador->pilotos].pontos = 0;
0407 |
0408 |          ptr_contador->pilotos++; // incrementa os pilotos depois de inserir com sucesso.
0409 |          ptr_contador->grava = 1; // houve alterações, pergunta se quer gravar antes de sair
0410 |          auxPausa();
0411 |          return ptr_piloto;
0412 |      }
0413 |  }
0414 |
0415 |
0416 |  void pilConsultar(ficha_t *ptr_piloto, etapa_t *ptr_etapa, conta_t *ptr_contador) {
0417 |      /*  Pergunta ao utilizador o número ou nome de um piloto e mostra os seus dados no ecrã.
0418 |          Incluido Nome, Nº do Avião, Idade, Pontuação, Etapas onde está inscritos e o pontos destas.
0424 |
0425 |      char nomenumero[31];
```

```
0426 |      int duplicados = 0, i, j, indice;
0427 |
0428 |      menuCabecalho(1, ptr_contador); // 0 = menu principal | 1 = menu pilotos | 2 = menu etapas
0429 |      printf("\n\n\tConsultar dados de Piloto \n");
0430 |      pilListagem(1, ptr_piloto, ptr_contador); // lista os pilotos para facilitar; valor 1 para lista reduzida
0431 |
0432 |      do {
0433 |          auxLeString(2, nomenumero, "\n\n\tInsira o nome do piloto ou o numero do aviao:", "Erro…\n", 1, 31);
0434 |          indice = pilProcurar(nomenumero, ptr_piloto, ptr_contador);
0435 |          if (indice == ERRO_DUPLICADO) { // ==ERRO_NOME removido, para não apresentar duas mensagens de erro. se o nome
0436 |              // não pode existir n faz sentido avisar que a etapa n foi encontrada.
0437 |              printf("\n\tExiste mais do que um piloto com esse nome\n\tPor favor use o numero de aviao para a pesquisa.");
0438 |          }
0439 |          if (indice == ERRO_NAOEXISTE) // ==ERRO_NOME removido, para não apresentar duas mensagens de erro. se o nome não
0440 |              //  pode existir n faz sentido avisar que o piloto n foi encontrado.
0441 |              printf("\n\tPiloto não encontrado...");
0442 |      } while (indice == ERRO_NOME || indice == ERRO_NAOEXISTE || indice == ERRO_DUPLICADO);
0443 |
0444 |      system("cls");
0445 |      puts(PROGRAMA);
0446 |      printf("\n\n\n\tEncontrado o piloto:\n\n\t%s, de %d anos, a correr no aviao com o numero: %d",
0447 |             ptr_piloto[indice].nome, ptr_piloto[indice].idade, ptr_piloto[indice].aviao);
0448 |      printf("\n\tActualmente com %d pontos no campeonato.", ptr_piloto[indice].pontos);
0449 |      printf("\n\tAtribuidos nas seguintes etapas:\n\n");
0450 |
0451 |      pilMostraInscricoes(ptr_piloto, ptr_etapa, indice, 1);
0452 |      auxPausa();
0453 |  }
0454 |
0455 |
0456 |  void pilListagem(int tipo_lista, ficha_t *ptr_piloto, conta_t *ptr_contador) {
0457 |      /*  Mostra a lista de pilotos, em modo normal ou reduzido.
0463 |
0464 |      int i;
0465 |
0466 |      if (tipo_lista == 0) {
0467 |          printf("\n\n\t\t\t\t\t         Nome | Idade | Aviao\n");
0468 |          for (i = 0; i < (ptr_contador->pilotos); i++) { // lista os pilotos ate ao ultimo inscrito
0469 |              printf("\n\t\t      %31.31s |   %2.d |  %3.d  ", ptr_piloto[i].nome, ptr_piloto[i].idade, ptr_piloto[i].aviao);
0470 |              if (i != 0 && i % 10 == 0) // de 10 em 10 nomes, faz uma pausa para o utilizador ver os nomes
0471 |                  auxPausa();
0472 |          }
0473 |      } else {
0474 |          printf("\n\n\t    Nome | Aviao\n");
0475 |          for (i = 0; i < (ptr_contador->pilotos); i++) { // lista os pilotos ate ao ultimo inscrito
0476 |              printf("\n\t  %18.18s | %3.d  ", ptr_piloto[i].nome, ptr_piloto[i].aviao);
0477 |              if ( ptr_contador->pilotos > 1 && i+1 < ptr_contador->pilotos)
```

Bruno Tereso, 2091066@student.estg.ipleiria.pt
Marco Ferreira, 2092016@student.estg.ipleiria.pt

```
0478 |                  printf("\t  %18.18s | %3.d    ", ptr_piloto[i+1].nome, ptr_piloto[i+1].aviao);
0479 |                  i++;
0480 |              }
0481 |              if (i != 0 && i % 10 == 0
0482 |                  auxPausa();
0483 |          }
0484 |      }
0485 |  }
0486 |
0487 |
0488 |  ficha_t *pilEliminar(ficha_t *ptr_piloto, etapa_t *ptr_etapa, conta_t *ptr_contador) {
0489 |      /*  Lista os pilotos, recebe o nome ou numero de avião do utilizador e apaga o piloto.
0495 |
0496 |      char nomenumero[31], confirma, nome[31];
0497 |      int confirmaapaga=0, verifica_inscrito, i, j, numero_aviao, saiciclo = 1, indice;
0498 |      ficha_t *aux;
0499 |
0500 |      aux = NULL;
0501 |
0502 |      menuCabecalho(1, ptr_contador); // 0 = menu principal | 1 = menu pilotos | 2 = menu etapas
0503 |      printf("\n\n\tEliminar Piloto \n");
0504 |      pilListagem(1, ptr_piloto, ptr_contador); // lista os pilotos para facilitar; valor 1 para lista reduzida
0505 |      do {
0506 |          printf("\n\n\tInsira o nome do piloto ou o numero do aviao: "); // detecta automaticamente se e' aviao ou nome
0507 |          fflush(stdin);
0508 |          //gets(nomenumero); // usar scanf
0509 |          scanf("%31[^\n]", nomenumero);
0510 |          indice = pilProcurar(nomenumero, ptr_piloto, ptr_contador);
0511 |          if (indice != ERRO_NOME && indice != ERRO_NAOEXISTE && indice != ERRO_DUPLICADO)
0512 |              saiciclo = 0;
0513 |          else
0514 |              if (indice == ERRO_NAOEXISTE)
0515 |                  printf("\tO nome\\aviao nao existe na base de dados\n");
0516 |              else
0517 |                  if (indice == ERRO_DUPLICADO) {
0518 |                      printf("\n\tExiste mais do que um piloto com esse nome\n\tPor favor use o numero de aviao para a pesquisa.");
0519 |                  }
0520 |      } while (saiciclo);
0521 |      // verifica se o piloto está inscrito em alguma etapa ; max_etapas+1 para salvaguardar que e' uma etapa que nao existe
0522 |      verifica_inscrito = etaVerificaInscricao(ptr_piloto, ptr_etapa, ptr_piloto[indice].aviao, MAX_ETAPAS + 1, ptr_contador);
0523 |      if (verifica_inscrito == -2) {
0524 |          printf ("\n\tImpossivel apagar o piloto.");
0525 |          printf ("\n\t%s encontra-se inscrito numa etapa ja realizada...", ptr_piloto[indice].nome);
0526 |          auxPausa();
0527 |      } else {
0528 |          do { // pedir confirmação para apagar
0529 |              if (verifica_inscrito == -1) {
```

```
0530 |                 printf ("\n\tATENCAO: O piloto esta inscrito nas seguintes etapas:\n");
0531 |                 pilMostraInscricoes(ptr_piloto, ptr_etapa, indice, 2);
0532 |                 confirmaapaga = 1;
0533 |             }
0534 |             printf("\n\tTem a certeza que pretente eliminar o piloto");
0535 |             printf("\n\t%s, com o aviao numero %d, [S]im [N]ao", ptr_piloto[indice].nome, ptr_piloto[indice].aviao);
0536 |             confirma = getch();
0537 |             confirma = toupper(confirma);
0538 |         } while (confirma != 'S' && confirma != 'N');
0539 |
0540 |         if (confirma == 'S') {
0541 |             if (confirmaapaga)
0542 |                 pilLimpaInscricoes(ptr_piloto, ptr_etapa, indice);
0543 |             ptr_piloto = pilApaga(ptr_piloto, ptr_contador, indice);
0544 |             printf("\n\tPiloto apagado com sucesso...");
0545 |             ptr_contador->grava = 1; // houve alterações, pergunta se quer gravar antes de sair
0546 |         } else
0547 |             printf("\n\tCancelado.");
0548 |         auxPausa();
0549 |     }
0550 |     return ptr_piloto;
0551 | }
0552 |
0553 |
0554 | char etaMenu(conta_t *ptr_contador) {
0555 |     /*  Mostra o menu de Etapas no ecrã.
0561 |
0562 |     char op;
0563 |
0564 |     do {
0565 |         menuCabecalho(2, ptr_contador); // 0 = menu principal | 1 = menu pilotos | 2 = menu etapas
0566 |         printf("\n\n\tI - Inserir Etapa");
0567 |         printf("\n\tE - Editar Etapa");
0568 |         printf("\n\tN - Inscrever Piloto em Etapa");
0569 |         printf("\n\tD - Dados da Etapa");
0570 |         printf("\n\tL - Listar Inscricoes de Piloto");
0571 |         printf("\n\n\tV - Voltar ao Menu Principal");
0572 |         fflush(stdin);
0573 |         op = getch();
0574 |         op = toupper(op);
0575 |     } while (op != 'V' && op != 'I' && op != 'E' && op != 'N' && op != 'D' && op != 'L');
0576 |
0577 |     return op;
0578 | }
0579 |
0580 |
0581 | void etaFuncoes(etapa_t *ptr_etapa, ficha_t *ptr_piloto, char op, conta_t *ptr_contador) {
```

Bruno Tereso, 2091066@student.estg.ipleiria.pt
Marco Ferreira, 2092016@student.estg.ipleiria.pt

```
0582 |      /*  Recebe a opção escolhida e direcciona para a função a usar.
0589 |
0590 |      int vagas=1;
0591 |
0592 |      switch (op) {
0593 |      case 'I':  //inserir etapa
0594 |          if (ptr_contador->etapas < MAX_ETAPAS - 1)
0595 |              etaInserir(ptr_etapa, ptr_contador);
0596 |          else {
0597 |              printf("\n\tLamento, atingiu o numero maximo de etapas...");
0598 |              auxPausa();
0599 |          }
0600 |          break;
0601 |
0602 |      case 'E':  //editar etapa
0603 |          if (ptr_contador->etapas > 0)
0604 |              etaEditar(ptr_etapa, ptr_contador);
0605 |          else {
0606 |              printf("\n\tNao existe qualquer etapa para editar...\n\n\t");
0607 |              system("PAUSE");
0608 |          }
0609 |          break;
0610 |
0611 |      case 'N':  //Inscrever piloto em etapa // etaInscricoesVagas
0612 |          vagas = etaInscricoesVagas(ptr_etapa, ptr_contador);
0613 |          if (ptr_contador->etapas > 0 && ptr_contador->pilotos > 0 && vagas !=0)
0614 |              etaInscreverPiloto(ptr_etapa, ptr_piloto, ptr_contador);
0615 |          else
0616 |              if (vagas == 0 && ptr_contador->etapas > 0) {
0617 |                  printf("\n\n\tImpossivel inscrever piloto.\n\tTodas as etapas estao preenchidas...\n\n\t");
0618 |                  system("PAUSE");
0619 |              } else {
0620 |                  printf("\n\tImpossivel inscrever piloto em etapas...");
0621 |                  printf("\n\tCertifique-se que tem pelo menos 1 piloto e 1 etapa.\n\n\t");
0622 |                  system("PAUSE");
0623 |              }
0624 |          break;
0625 |
0626 |      case 'D':  //dados da etapa
0627 |          if (ptr_contador->etapas > 0)
0628 |              etaDados(ptr_etapa, ptr_piloto, ptr_contador);
0629 |          else {
0630 |              printf("\n\tNao existem etapas disponiveis...\n\n\t");
0631 |              system("PAUSE");
0632 |          }
0633 |          break;
0634 |
```

Bruno Tereso, 2091066@student.estg.ipleiria.pt
Marco Ferreira, 2092016@student.estg.ipleiria.pt

```
0635 |      case 'L':  //lista inscrições de piloto
0636 |          if (ptr_contador->etapas > 0 && ptr_contador->pilotos > 0)
0637 |              pilInscricoes(ptr_etapa, ptr_piloto, ptr_contador);
0638 |          else {
0639 |              printf("\n\tImpossivel listar inscricoes...");
0640 |              printf("\n\tCertifique-se que tem pelos menos 1 piloto e 1 etapa.\n\n\t");
0641 |              system("PAUSE");
0642 |          }
0643 |          break;
0644 |
0645 |      }
0646 |  }
0647 |
0648 |
0649 |  void etaInserir(etapa_t *ptr_etapa, conta_t *ptr_contador) {
0650 |      /*  Insere um novo registo na lista de etapas
0655 |
0656 |      char nome_etapa[31], data[10];
0657 |      int i, validado = 0, ordem_etapa, participantes, indice;
0658 |
0659 |      menuCabecalho(2, ptr_contador); // 0 = menu principal | 1 = menu pilotos | 2 = menu etapas
0660 |      printf("\n\n\tInserir Etapa | Listagem das etapas actuais");
0661 |      etaListar(1, ptr_etapa, ptr_contador);
0662 |      printf("\n\n");
0663 |
0664 |      auxLeString(2, nome_etapa, "\n\tDesignacao da nova etapa: ", "\n\tInsira um nome de etapa entre 3 e 30 caracteres.\n", 3, 30);
0665 |
0666 |      do { // verifica se ja existe uma etapa na ordem inserida
0667 |          ordem_etapa = auxLeInteiro("\tOrdem da etapa no campeonato: ", 1, MAX_ETAPAS);
0668 |          if (ptr_etapa[ordem_etapa-1].data.ano != 0) {
0669 |              printf("\tErro, ja existe uma etapa nessa posicao.\n");
0670 |              validado = 0;
0671 |              i = MAX_ETAPAS;
0672 |          } else
0673 |              validado = 1;
0674 |      } while (validado == 0);
0675 |
0676 |      ordem_etapa--; // decrementa 1 para passar a usar o indice correcto do vector ; depois do ciclo
0677 |      auxLeData(ordem_etapa, ptr_etapa); // aqui preciso da ordem da etapa, nao do indice
0678 |      participantes = auxLeInteiro ("\tNumero de participantes: ", 0, MAX_PILOTOS_ETAPA);
0679 |      ptr_etapa[ordem_etapa].participantes = participantes;
0680 |      strcpy(ptr_etapa[ordem_etapa].nome, nome_etapa);
0681 |      ptr_contador->etapas++; // incrementa uma etapa depois de inserir com sucesso.
0682 |      printf("\n\tEtapa inserida com sucesso...");
0683 |      for ( i = 0; i < participantes; i++) { // limpa os vectores usados nos resultados e estatisticas para a etapa inserida
0684 |          ptr_etapa[ordem_etapa].inscritos[i] = 0;
0685 |          ptr_etapa[ordem_etapa].pontos[i] = 0;
```

Bruno Tereso, 2091066@student.estg.ipleiria.pt
Marco Ferreira, 2092016@student.estg.ipleiria.pt

```
0686 |          ptr_etapa[ordem_etapa].tempos[i] = 0;
0687 |      }
0688 |      ptr_etapa[ordem_etapa].realizada = 0;
0689 |      ptr_etapa[ordem_etapa].totalinscritos = 0;
0690 |      etaActCorrente(ptr_etapa, ptr_contador); // verifica qual e proxima etapa corrente
0691 |      ptr_contador->grava = 1; // houve alterações, pergunta se quer gravar antes de sair
0692 |      auxPausa();
0693 |  }
0694 |
0695 |
0696 |  void etaActCorrente(etapa_t *ptr_etapa, conta_t *ptr_contador) {
0697 |      /*  Actualiza a etapa corrente e certifica-se que é uma etapa q existe
0702 |
0703 |      int i;
0704 |
0705 |      for ( i = 0 ; i < MAX_ETAPAS; i++) // vai procura a etapa mais proxima por realizar
0706 |          if ( ptr_etapa[i].realizada == 0 && ptr_etapa[i].data.ano != 0) {// se não foi realizada
0707 |              ptr_contador->etapa_corrente = i;
0708 |              break;
0709 |          }
0710 |  }
0711 |
0712 |
0713 |  void etaEditar(etapa_t *ptr_etapa, conta_t *ptr_contador) {
0714 |      /*  Lista as etapas e altera uma à escolha do utilizador
0719 |
0720 |      int numero_etapa, participantes;
0721 |      char confirma, nome_etapa[31];
0722 |
0723 |      menuCabecalho(2, ptr_contador); // 0 = menu principal | 1 = menu pilotos | 2 = menu etapas
0724 |      printf("\n\n\tEditar Etapa | Listagem por ordem de realizacao");
0725 |
0726 |      etaListar(1, ptr_etapa, ptr_contador);
0727 |      printf("\n\n");
0728 |
0729 |      do {
0730 |          numero_etapa = auxLeInteiro("\n\tInsira a ordem da etapa a editar: ", 1, MAX_ETAPAS);
0731 |          numero_etapa--; // coloca a variavel no indice correcto
0732 |          if (ptr_etapa[numero_etapa].data.ano == 0)
0733 |              printf("\tEtapa nao existe.\n");
0734 |      } while (ptr_etapa[numero_etapa].data.ano == 0);
0735 |
0736 |      if (ptr_etapa[numero_etapa].realizada != 1) {
0737 |          auxLeString(2, nome_etapa, "\n\tDesignacao da etapa: ", "\n\tInsira um nome de etapa entre 3 e 30 caracteres.\n", 3, 30);
0738 |          strcpy(ptr_etapa[numero_etapa].nome, nome_etapa);
0739 |
0740 |          auxLeData(numero_etapa, ptr_etapa);
```

Bruno Tereso, 2091066@student.estg.ipleiria.pt
Marco Ferreira, 2092016@student.estg.ipleiria.pt

```
0741 |
0742 |          do {
0743 |              printf("\n\tDeseja alterar as inscricoes actuais na etapa?\n");
0744 |              printf("\n\tM - Manter inscricoes\n");
0745 |              printf("\tA - Apagar todas as inscricoes (pede novo valor de inscritos)\n");
0746 |              if (ptr_etapa[numero_etapa].participantes < MAX_ETAPAS) // esconde o C no caso de n dar para acrescentar
0747 |                  printf("\tC - Acrescentar incricoes (mantem as actuais)\n\t");
0748 |              confirma = getch();
0749 |              confirma = toupper(confirma);
0750 |          } while (confirma != 'M' && confirma != 'A' && confirma != 'C');
0751 |
0752 |          if (confirma == 'A') {
0753 |              etaLimpaInscricoes(numero_etapa, ptr_etapa);
0754 |              printf("\n\tInscricoes apagadas...\n");
0755 |              participantes = auxLeInteiro ("\tNovo Numero de participantes: ", 0, MAX_PILOTOS_ETAPA);
0756 |              ptr_etapa[numero_etapa].participantes = participantes;
0757 |          } else
0758 |              if (confirma == 'C') {
0759 |                  printf("\n\tAumentar as inscricoes para:");
0760 |                  participantes = auxLeInteiro (" ", (ptr_etapa[numero_etapa].participantes)+1, MAX_PILOTOS_ETAPA);
0761 |                  ptr_etapa[numero_etapa].participantes = participantes;
0762 |                  printf("\n\tNumero de inscricoes aumentado com sucesso...");
0763 |              }
0764 |          printf("\n\tEtapa editada com sucesso...");
0765 |      } else
0766 |          printf("\n\tImpossivel editar a etapa porque ja foi realizada...");
0767 |      etaActCorrente(ptr_etapa, ptr_contador); // verifica qual e proxima etapa corrente
0768 |      ptr_contador->grava = 1; // houve alterações, pergunta se quer gravar antes de sair
0769 |      auxPausa();
0770 |  }
0771 |
0772 |
0773 |  void etaInscreverPiloto(etapa_t *ptr_etapa, ficha_t *ptr_piloto, conta_t *ptr_contador) {
0774 |      /*  Mostra as etapas, inscreve um piloto na etapa escolhida.
0780 |
0781 |      int verifica_inscrito, indice, num_etapa, numero_aviao;
0782 |      char nomenumero[40], nome[40];
0783 |
0784 |      menuCabecalho(2, ptr_contador); // 0 = menu principal | 1 = menu pilotos | 2 = menu etapas
0785 |      printf("\n\n\tInscrever Piloto em Etapa\n");
0786 |      printf("\n\tEtapas disponiveis:\n"); // lista etapas matriz etaListar(1, ptr_etapa, ptr_contador);
0787 |      etaListar(1, ptr_etapa, ptr_contador);
0788 |      do {
0789 |          num_etapa = auxLeInteiro("\n\n\tPretende inscrever piloto na etapa de ordem: ", 0, MAX_ETAPAS);
0790 |          num_etapa--; // coloca o valor no indice correcto para trabalhar com o vector
0791 |          if (ptr_etapa[num_etapa].data.ano == 0)
0792 |              printf("\tErro, a etapa que escolheu nao existe...\n");
```

Bruno Tereso, 2091066@student.estg.ipleiria.pt
Marco Ferreira, 2092016@student.estg.ipleiria.pt

```
0793 |          else
0794 |              if (ptr_etapa[num_etapa].realizada == 1)
0795 |                  printf("\tErro, a etapa que escolheu ja se realizou...\n");
0796 |              else
0797 |                  if (ptr_etapa[num_etapa].totalinscritos >= ptr_etapa[num_etapa].participantes)
0798 |                      printf("\tErro, a etapa que escolheu atingiu o limite maximo de inscritos...\n");
0799 |       } while (ptr_etapa[num_etapa].realizada == 1 || ptr_etapa[num_etapa].data.ano == 0 || (ptr_etapa[num_etapa].totalinscritos >=
0799 |                ptr_etapa[num_etapa].participantes));
0800 |      pilListagem(1, ptr_piloto, ptr_contador); // lista os pilotos para facilitar; valor 1 para lista reduzida
0801 |      do {
0802 |          auxLeString(2, nomenumero, "\n\n\tInsira o nome do piloto ou o numero do aviao: ", "Erro…\n", 1, 31);
0803 |          indice = pilProcurar(nomenumero, ptr_piloto, ptr_contador);
0804 |          if (indice == ERRO_NAOEXISTE) // ==ERRO_NOME removido, para não apresentar duas mensagens de erro.
0805 |              //   se o nome não pode existir n faz sentido avisar que o piloto n foi encontrado.
0806 |              printf("\n\tPiloto nao encontrado...");
0807 |          if (indice == ERRO_DUPLICADO) {
0808 |              printf("\n\tExiste mais do que um piloto com esse nome\n\tPor favor use o numero de aviao.");
0809 |          }
0810 |      } while (indice == ERRO_NOME || indice == ERRO_NAOEXISTE || indice == ERRO_DUPLICADO);
0811 |      // verifica se o piloto já se encontra inscrito na etapa escolhida
0812 |      verifica_inscrito = etaVerificaInscricao(ptr_piloto, ptr_etapa, ptr_piloto[indice].aviao, num_etapa, ptr_contador);
0813 |      if (verifica_inscrito == 1) {
0814 |          printf ("\n\tErro, o piloto ja esta inscrito nesta etapa...");
0815 |          printf ("\n\tPor favor tente de novo.");
0816 |          auxPausa();
0817 |      } else {
0818 |          printf("\n\n\tSucesso, inscreveu o piloto na etapa: %s", ptr_etapa[num_etapa].nome);
0819 |          ptr_etapa[num_etapa].inscritos[ptr_etapa[num_etapa].totalinscritos] = ptr_piloto[indice].aviao;
0820 |          ptr_etapa[num_etapa].pontos[ptr_etapa[num_etapa].totalinscritos] = 0; // inicia os pontos a zero (causava erros)
0821 |          ptr_etapa[num_etapa].tempos[ptr_etapa[num_etapa].totalinscritos] = 0; // inicia o tempo a zero
0822 |          ptr_etapa[num_etapa].totalinscritos++;
0823 |          printf("\n\tA etapa conta actualmente com %d piloto(s)...\n", ptr_etapa[num_etapa].totalinscritos);
0824 |          auxPausa();
0825 |      }
0826 |      ptr_contador->grava = 1; // houve alterações, pergunta se quer gravar antes de sair
0827 |  }
0828 |
0829 |
0830 |  void etaDados(etapa_t *ptr_etapa, ficha_t *ptr_piloto, conta_t *ptr_contador) {
0831 |      /*  Lista etapas e mostra os dados de uma delas à escolha do utilizador.
0832 |          Mostra Designação, Ordem da etapa no campeonato, Data, Nº participantes,
0833 |          Nº de inscritos, Pilotos inscritos e, caso já se tenha realizado, Resultados.
0839 |
0840 |      int indice_aviao, inscritos, j = 5, i, indice;
0841 |      char nomenumero[31];
0842 |
0843 |      menuCabecalho(2, ptr_contador); // 0 = menu principal | 1 = menu pilotos | 2 = menu etapas
```

Bruno Tereso, 2091066@student.estg.ipleiria.pt
Marco Ferreira, 2092016@student.estg.ipleiria.pt

```
0844 |      printf("\n\n\tListagem de Etapas\n");
0845 |      etaListar(1, ptr_etapa, ptr_contador);
0846 |
0847 |      do {
0848 |          auxLeString(2, nomenumero, "\n\n\tInsira o nome da etapa ou a sua ordem:", "Erro, nome ou numero invalido\n", 1, 30);
0849 |          indice = etaProcurar(nomenumero, ptr_etapa, ptr_contador);
0850 |          if (indice == ERRO_DUPLICADO) {
0851 |              printf("\n\tExiste mais do que uma etapa esse nome\n\tPor favor use a ordem de etapa a pesquisa.");
0852 |          }
0853 |          if (indice == ERRO_NAOEXISTE) // ==ERRO_NOME removido, para não apresentar duas mensagens de erro.
0854 |              //  se o nome não pode existir n faz sentido avisar que a etapa n foi encontrada.
0855 |              printf("\n\tEtapa nao encontrada...");
0856 |      } while (indice == ERRO_NOME || indice == ERRO_NAOEXISTE || indice == ERRO_DUPLICADO);
0857 |
0858 |      system("cls");
0859 |      puts(PROGRAMA);
0860 |      printf("\n\tDesignacao da etapa: %s", ptr_etapa[indice].nome);
0861 |      printf("\n\tOrdem da etapa no campeonato: %d", ptr_etapa[indice].ordem);
0862 |      printf("\t   Data [ %.2d.%.2d.%.4d ]", ptr_etapa[indice].data.dia, ptr_etapa[indice].data.mes, ptr_etapa[indice].data.ano);
0863 |      printf("\n\tTem um maximo de %d inscricoes", ptr_etapa[indice].participantes);
0864 |      printf(" actualmente com %d", ptr_etapa[indice].totalinscritos);
0865 |      if (ptr_etapa[indice].totalinscritos > 0) { // Se não tem inscritos não faz sentido mostrar listas
0866 |          printf("\n\n\tLista de pilotos e pontuacoes para a etapa escolhida:");
0867 |          resEtapa (indice, ptr_etapa, ptr_piloto, ptr_contador); // lista etapa
0868 |      }
0869 |      auxPausa();
0870 |  }
0871 |
0872 |
0873 |  void pilInscricoes(etapa_t *ptr_etapa, ficha_t *ptr_piloto, conta_t *ptr_contador) {
0874 |      /*  Lista as etapas em que um determinado piloto se inscreveu.
0880 |
0881 |      int cont=0, indice, i, j; //temp
0882 |      char nomenumero[31];
0883 |
0884 |      system("cls");
0885 |      puts(PROGRAMA);
0886 |      printf("\n\n\tInscricoes Pilotos\n");
0887 |      pilListagem(1, ptr_piloto, ptr_contador); // lista os pilotos para facilitar; valor 1 para lista reduzida
0888 |      printf("\n");
0889 |      do {
0890 |          auxLeString(2, nomenumero, "\n\tInsira o nome do piloto ou o numero do aviao: ", "Erro, nome ou numero invalido\n", 1, 31);
0891 |          indice = pilProcurar(nomenumero, ptr_piloto, ptr_contador);
0892 |          if (indice == ERRO_NAOEXISTE) // ==ERRO_NOME removido, para não apresentar duas mensagens de erro.
0893 |              printf("\n\tPiloto não encontrado...");
0894 |          if (indice == ERRO_DUPLICADO) {
0895 |              printf("\n\tExiste mais do que um piloto com esse nome\n\tPor favor use o numero de aviao para a pesquisa.");
```

Bruno Tereso, 2091066@student.estg.ipleiria.pt
Marco Ferreira, 2092016@student.estg.ipleiria.pt

```
0896 |          }
0897 |      } while (indice == ERRO_NOME || indice == ERRO_NAOEXISTE || indice == ERRO_DUPLICADO);
0898 |
0899 |      system("cls");
0900 |      puts(PROGRAMA);
0901 |      printf("\n\n\tInscricoes Piloto | %s\n\n", ptr_piloto[indice].nome);
0902 |      printf("\t\t\t\t\t\t  Etapa |     Data\n\n");
0903 |
0904 |      for (i = 0; i <= MAX_ETAPAS - 1; i++) // de i ate max etapas (-1 para ter indice 9) senao usa o /0 e pumm
0905 |          for (j = 0;j <= ptr_etapa[i].participantes;j++) { // de j ate ao numero actual de participantes na etapa i
0906 |              if (ptr_etapa[i].inscritos[j] == ptr_piloto[indice].aviao) {// se o vector tiver um inscrito
0907 |                  printf("\t\t\t%31.31s |  %.2d.%.2d.%.4d\n", ptr_etapa[i].nome,
0908 |                         ptr_etapa[i].data.dia, ptr_etapa[i].data.mes, ptr_etapa[i].data.ano);
0909 |                  cont++;
0910 |              }
0911 |          }
0912 |      if ( cont == 0 )
0913 |          printf("\n\tO piloto escolhido nao esta inscrito em nenhuma etapa...");
0914 |      auxPausa();
0915 |  }
0916 |
0917 |
0918 |
0919 |
0920 |
```

Funções Diversas

```
0921 |  /*  ################################################################
0922 |                          FUNÇÕES DIVERSAS
0923 |      ################################################################ */
0924 |
0925 |  void menuCabecalho(int tipo, conta_t *ptr_contador) {
0926 |      /*  Mostra o cabeçalho do Programa.
0937 |
0938 |      int etapas_realizadas;
0939 |
0940 |      system("cls");
0941 |      puts(PROGRAMA);
0942 |      switch (tipo) {
0943 |      case 0:
0944 |          printf("\n\n\tPilotos Incritos: %.2d/%.2d", ptr_contador->pilotos, MAX_PILOTOS);
0945 |          printf("\t\tEtapas Realizadas: %.2d", ptr_contador->realizadas); // 2ª parte do trabalho
0946 |          printf("\n\t\t\t\t\t\t     Inseridas: %.2d/%.2d", ptr_contador->etapas, MAX_ETAPAS);
0947 |          break;
0948 |      case 1:
0949 |          printf("\n\n\tPilotos Incritos: %.2d/%.2d", ptr_contador->pilotos, MAX_PILOTOS);
0950 |          break;
```

```
0951 |      case 2:
0952 |          printf("\n\n\tEtapas Realizadas/Inseridas: %.2d/%.2d\t\tMaximo: %2d",
0953 |                 ptr_contador->realizadas, ptr_contador->etapas, MAX_ETAPAS);
0954 |          break;
0960 |      case 4:
0961 |          printf("\n\n\tEstatisticas");
0962 |          break;
0963 |      }
0964 |  }
0965 |
0966 |
0967 |  void resCabecalho(etapa_t *ptr_etapa, conta_t *ptr_contador) {
0968 |      /*  Cabeçalho para o menu Resultados. Independente porque precisa das etapas
0973 |
0974 |      system("cls");
0975 |      puts(PROGRAMA);
0976 |      if (ptr_contador->etapa_corrente == 0 || ptr_etapa[ptr_contador->etapa_corrente].realizada == 1)
0977 |          //  verifica se está sem etapas ou se a ultima etapa já foi lida
0978 |          printf("\n\tNenhuma Etapa actualmente em curso");
0979 |      else {
0980 |          printf("\n\n\tEtapa Corrente: %.2d", ptr_contador->etapa_corrente+1);
0981 |          printf(" - %s", ptr_etapa[ptr_contador->etapa_corrente].nome);
0982 |      }
0983 |  }
0984 |
0985 |
0986 |  int pilProcurar(char nomenumero[], ficha_t *ptr_piloto, conta_t *ptr_contador) {
0987 |      /*  Procura o indice de um piloto nas fichas de piloto.
0998 |
0999 |      int duplicados, i, numero_aviao;
1000 |
1001 |      if (strlen(nomenumero) <= 3 && atoi(nomenumero) != 0) { //entao e' um numero de aviao
1002 |          numero_aviao = atoi(nomenumero);
1003 |          for (i = 0; i < ptr_contador->pilotos; i++) { // procura a posição com o aviao inserido
1004 |              if (numero_aviao == ptr_piloto[i].aviao)
1005 |                  return i;
1006 |          }
1007 |      } else
1008 |          if (strlen(nomenumero) <= 3 && atoi(nomenumero) == 0) {
1009 |              printf("\tInseriu um nome muito curto, ou um numero de aviao invalido\n");
1010 |              return ERRO_NOME;
1011 |          } else {
1012 |              for (i = 0; i < ptr_contador->pilotos; i++) // procura um nome igual
1013 |                  if (strcasecmp(nomenumero, ptr_piloto[i].nome) == 0) {
1014 |                      duplicados = pilProcuraDuplicado(ptr_piloto[i].nome, ptr_piloto, ptr_contador);
1015 |                      if (duplicados == 1)
1016 |                          return ERRO_DUPLICADO;
```

Bruno Tereso, 2091066@student.estg.ipleiria.pt
Marco Ferreira, 2092016@student.estg.ipleiria.pt

```
1017 |                     return i;
1018 |                 }
1019 |             }
1020 |         return ERRO_NAOEXISTE;
1021 |     }
1022 |
1023 |
1024 |     void auxLeString(int tipo, char texto[], char pergunta[30], char erro[50], int inferior, int superior) {
1025 |         /*  Le e valida um vector de caracteres.
1035 |
1036 |         int i, numero=0;
1037 |
1038 |         do {
1039 |             printf("%s ", pergunta);  // mostra a pergunta
1040 |             fflush(stdin);
1041 |             gets(texto);
1042 |             numero = 0;
1043 |
1044 |             if (tipo == 1) { // se for apenas texto
1045 |                 for (i = 0; i < strlen(texto); i++) // do inicio ate ao final do vector
1046 |                     if (!isalpha(texto[i])) // verifica se o caracter é alfanumérico
1047 |                         numero++;
1048 |                 if (numero > 0)
1049 |                     printf("\tErro, introduza apenas texto...\n");
1050 |             }
1051 |
1052 |             if ( ( strlen(texto) < inferior || strlen(texto) > superior ) && numero == 0)
1053 |                 printf("\t%s", erro);
1054 |         } while (numero > 0 || strlen(texto) < inferior || strlen(texto) > superior);
1055 |     }
1056 |
1057 |
1058 |     int auxLeInteiro (char pergunta[40], int menor, int maior) {
1059 |         /*  Le e valida um número inteiro
1067 |
1068 |         int comp = 0, cont = 0, aux;
1069 |         char astring[10]; // string usada para ler valor inteiro
1070 |
1071 |         do {
1072 |             cont = comp = 0;
1073 |             printf("%s", pergunta);
1074 |             fflush(stdin);
1076 |             scanf("%10[^\n]", astring);
1077 |             cont = atoi(astring); // converte a string para um inteiro, sem os caracteres misturados
1078 |             aux = cont; // cria um auxiliar para n alterar o valor do cont
1079 |             while (aux > 0) { // comp conta o numero de digitos do numero, dividindo por 10
1080 |                 comp++;
```

Bruno Tereso, 2091066@student.estg.ipleiria.pt
Marco Ferreira, 2092016@student.estg.ipleiria.pt

```
1081 |              aux /= 10;
1082 |          } // vai comparar o comprimento da string inicial com caracteres com o comprimento do numero convertido
1083 |          if (strlen(astring) != comp || cont == 0 || cont < menor || cont > maior)
1084 |              printf("\tValor incorrecto. Entre %d e %d", menor, maior);
1085 |      } while (strlen(astring) != comp || cont == 0 || cont < menor || cont > maior);
1086 |
1087 |      return(cont); // devolve inteiro validado
1088 |  }
1089 |
1090 |
1091 |  void etaListar(int tipo, etapa_t *ptr_etapa, conta_t *ptr_contador) {
1092 |      /*  Mostra uma lista de etapas para auxílio em funções
1099 |
1100 |      int inscritos, i;
1101 |
1102 |      printf("\n\n\t\t           Designacao |  Ordem |     Data       | Pilotos\n");
1103 |
1104 |      if (tipo == 1) {
1105 |          for (i = 0; i <= MAX_ETAPAS - 1; i++) {  // tem que usar max etapas pq senao salta ordens de etapa n usadas
1106 |              if (ptr_etapa[i].data.ano != 0) { // se a etapa tem dados, mostra no ecra
1107 |                  printf("\n\t\t %20.20s |     %.2d |  %.2d.%.2d.%.4d ", ptr_etapa[i].nome, ptr_etapa[i].ordem,
1108 |                         ptr_etapa[i].data.dia, ptr_etapa[i].data.mes, ptr_etapa[i].data.ano);
1109 |                  if ( ptr_etapa[i].realizada == 1)
1110 |                      printf(" R"); // mostra R ou outro simbolo para etapas realizadas
1111 |                  else
1112 |                      printf("  ");
1113 |                  printf(" |  %.2d/%.2d    ", ptr_etapa[i].totalinscritos, ptr_etapa[i].participantes);
1114 |              }
1115 |              if (i != 0 && i % 10 == 0)
1116 |                  auxPausa();
1117 |          }
1118 |      } else
1119 |          if (tipo == 2) {
1120 |              for (i = 0; i <= MAX_ETAPAS - 1; i++) {  // tem que usar max etapas pq senao salta ordens de etapa n usadas
1121 |                  if (ptr_etapa[i].data.ano != 0 && ptr_etapa[i].realizada == 1) {
1122 |                      //inscritos = auxCompVector(ptr_etapa[i].inscritos, 20);
1123 |                      printf("\n\t\t %20.20s |     %.2d |  %.2d.%.2d.%.4d  R", ptr_etapa[i].nome, ptr_etapa[i].ordem,
1124 |                             ptr_etapa[i].data.dia, ptr_etapa[i].data.mes, ptr_etapa[i].data.ano);
1125 |                      printf(" |  %.2d/%.2d    ", ptr_etapa[i].totalinscritos, ptr_etapa[i].participantes);
1126 |                  }
1127 |                  if (i != 0 && i % 10 == 0)
1128 |                      auxPausa();
1129 |              }
1130 |          }
1131 |  }
1132 |
1133 |
```

Bruno Tereso, 2091066@student.estg.ipleiria.pt
Marco Ferreira, 2092016@student.estg.ipleiria.pt

```
1134 |  int etaProcurar(char nomenumero[40], etapa_t *ptr_etapa, conta_t *ptr_contador) {
1135 |      /*  Analisa o nome ou ordem de uma etapa e devolve o indice desta.
1146 |
1147 |      int duplicados, i, ordem_etapa;
1148 |
1149 |      if (strlen(nomenumero) <= 2 && atoi(nomenumero) != 0) { //entao e' ordem de etapa
1150 |          ordem_etapa = atoi(nomenumero);
1151 |          if (ptr_etapa[ordem_etapa-1].data.ano != 0 && ordem_etapa<=MAX_ETAPAS)
1152 |              return ordem_etapa -1; // devolve o indice, nao a ordem
1153 |          else
1154 |              return ERRO_NAOEXISTE;
1155 |      } else
1156 |          if (strlen(nomenumero) <= 2 && atoi(nomenumero) == 0) {
1157 |              printf("\tInseriu um nome muito curto, ou uma ordem invalida\n");
1158 |              return ERRO_NOME;
1159 |          } else {
1160 |              for (i = 0; i <= MAX_ETAPAS; i++) // procura um nome igual
1161 |                  if (strcasecmp(nomenumero, ptr_etapa[i].nome) == 0) { // compara as strings case insensitive
1162 |                      duplicados = etaProcuraDuplicada(ptr_etapa[i].nome, ptr_etapa);
1163 |                      if (duplicados == 1)
1164 |                          return ERRO_DUPLICADO;
1165 |                      return i; // devolve o indice com o nome inserido
1166 |                  }
1167 |          }
1168 |      return ERRO_NAOEXISTE;
1169 |  }
1170 |
1171 |
1172 |  void auxLeData(int numero_etapa, etapa_t *ptr_etapa) {
1173 |      /*  Le uma data válida para inserir uma etapa.
1179 |
1180 |      int i, passaciclo = 1, dataactual_dias, datainserida_dias, etapaseguinte_dias,
1181 |                          etapaanterior_dias, etapa_anterior, etapa_seguinte;
1182 |      data_t data_inserida;
1183 |
1184 |      // declarações para data do sistema
1185 |      time_t  dataactual;
1186 |      struct tm *infodata;
1187 |      char string_data[32] = {0};
1188 |
1189 |      time(&dataactual);
1190 |      infodata = localtime(&dataactual);
1191 |
1192 |      // INWORK considera todos os meses com 30 dias
1193 |      infodata->tm_mon++; //corrige bug mes
1194 |      dataactual_dias = ( ((infodata->tm_mon) - 1) * 30 + (infodata->tm_mday) + ((infodata->tm_year + 1900) - 1) * 365 );
1195 |
```

Bruno Tereso, 2091066@student.estg.ipleiria.pt
Marco Ferreira, 2092016@student.estg.ipleiria.pt

```
1196 |      do {
1197 |          data_inserida.ano = auxLeInteiro("\n\tData da etapa, ano: ", 2010, 2100);
1198 |          data_inserida.mes = auxLeInteiro("\tData da etapa, mes: ", 1, 12);
1199 |          switch (data_inserida.mes) {
1200 |          case 2:
1201 |              data_inserida.dia = auxLeInteiro("\tData da etapa, dia: ", 1, 28);
1202 |              break;
1203 |
1204 |          case 1:
1205 |          case 3:
1206 |          case 5:
1207 |          case 7:
1208 |          case 8:
1209 |          case 10:
1210 |          case 12:
1211 |              data_inserida.dia = auxLeInteiro("\tData da etapa, dia: ", 1, 31);
1212 |              break;
1213 |          default:
1214 |              data_inserida.dia = auxLeInteiro("\tData da etapa, dia: ", 1, 30);
1215 |              break;
1216 |
1217 |          }
1218 |
1219 |          datainserida_dias = ( ((data_inserida.mes) - 1) * 30 + (data_inserida.dia) + ((data_inserida.ano) - 1) * 365 );
1220 |
1221 |          // vai verificar se a data esta' entre a data da etapa anterior e da etapa seguinte
1222 |          // obtem a contagem de dias para a etapa seguinte
1223 |          if (numero_etapa + 1 == MAX_ETAPAS) { // se inseriu a ultima etapa
1224 |              etapaseguinte_dias = DIAS_MAX;  // se nao tem nenhum etapa seguinte, guarda um valor mais alto imaginario
1225 |              etapa_seguinte = ETAPA_EXTREMO;
1226 |          } else
1227 |              for (i = numero_etapa + 1; i < MAX_ETAPAS; i++) {
1228 |                  if (ptr_etapa[i].data.ano != 0) {
1229 |                       etapaseguinte_dias =  ( ((ptr_etapa[i].data.mes) - 1) * 30 + (ptr_etapa[i].data.dia) + ((ptr_etapa[i].data.ano)
1229 |                       - 1) * 365 );
1230 |                      etapa_seguinte = i; // guarda o indice da etapa seguinte para mostrar no caso de erro
1231 |                      i = MAX_ETAPAS; //se encontrou uma etapa seguinte, sai do ciclo
1232 |                  } else {
1233 |                      etapaseguinte_dias = DIAS_MAX;
1234 |                      etapa_seguinte = ETAPA_EXTREMO; // uma etapa que está em primeiro ou ultimo lugar
1235 |                  }
1236 |              }
1237 |          if (numero_etapa == 0) { // se esta a referir-se à primeira etapa
1238 |              etapaanterior_dias = DIAS_MIN; // se nao tem nenhum etapa antes, guarda o valor mais baixo e aceita todas
1239 |              etapa_anterior = ETAPA_EXTREMO;
1240 |          } else
1241 |              for (i = numero_etapa - 1; i >= 0; i--) {
```

Bruno Tereso, 2091066@student.estg.ipleiria.pt
Marco Ferreira, 2092016@student.estg.ipleiria.pt

```
1242 |                 if (ptr_etapa[i].data.ano != 0) {
1243 |                     etapaanterior_dias =  ( ((ptr_etapa[i].data.mes) - 1) * 30 + (ptr_etapa[i].data.dia) + ((ptr_etapa[i].data.ano)
1243 |                     - 1) * 365 );
1244 |                     etapa_anterior = i; // guarda o indice da etapa seguinte para mostrar no caso de erro
1245 |                     i = 0; //se encontrou uma etapa anterior, i=0 para sair o ciclo for
1246 |                 } else {
1247 |                     etapaanterior_dias = DIAS_MIN;
1248 |                     etapa_anterior = ETAPA_EXTREMO; // uma etapa que está em primeiro ou ultimo lugar
1249 |                 }
1250 |             }
1251 |
1252 |         if (dataactual_dias > datainserida_dias) {
1253 |             printf("\n\tErro, inseriu uma data passada...");
1254 |             passaciclo = 0;
1255 |         } else
1256 |             if ( (datainserida_dias > dataactual_dias) &&
1257 |                     (datainserida_dias > etapaanterior_dias) && (datainserida_dias < etapaseguinte_dias) )
1258 |                 passaciclo = 1;
1259 |             else {
1260 |                 printf("\n\tData Invalida, para inserir a etapa na ordem %d,", numero_etapa + 1);
1261 |                 if (etapa_anterior != ETAPA_EXTREMO && etapa_seguinte != ETAPA_EXTREMO) {
1262 |                     printf("\n\ttem que escolher uma data entre a etapa de %s e %s.",
1263 |                             ptr_etapa[etapa_anterior].nome, ptr_etapa[etapa_seguinte].nome);
1264 |                     printf("\n\tOu seja entre %d.%d.%d e", ptr_etapa[etapa_anterior].data.dia,
1265 |                             ptr_etapa[etapa_anterior].data.mes, ptr_etapa[etapa_anterior].data.ano);
1266 |                     printf(" %d.%d.%d\n", ptr_etapa[etapa_seguinte].data.dia,
1267 |                             ptr_etapa[etapa_seguinte].data.mes, ptr_etapa[etapa_seguinte].data.ano);
1268 |                 } else
1269 |                     if (etapa_anterior != ETAPA_EXTREMO) {
1270 |                         printf("\n\ttem que escolher uma etapa posterior a %s.", ptr_etapa[etapa_anterior].nome);
1271 |                         printf("\n\tOu seja com data depois de %d.%d.%d e", ptr_etapa[etapa_anterior].data.dia,
1272 |                                 ptr_etapa[etapa_anterior].data.mes, ptr_etapa[etapa_anterior].data.ano);
1273 |                     } else
1274 |                         if (etapa_seguinte != ETAPA_EXTREMO) {
1275 |                             printf("\n\ttem que escolher uma etapa anterior a %s.", ptr_etapa[etapa_seguinte].nome);
1276 |                             printf("\n\tOu seja com data antes de %d.%d.%d e", ptr_etapa[etapa_seguinte].data.dia,
1277 |                                     ptr_etapa[etapa_seguinte].data.mes, ptr_etapa[etapa_seguinte].data.ano);
1278 |                         }
1279 |                 passaciclo = 0;
1280 |             }
1281 |     } while (passaciclo != 1);
1282 |
1283 |     //guarda os valores da data depois de validados
1284 |     ptr_etapa[numero_etapa].data.ano = data_inserida.ano;
1285 |     ptr_etapa[numero_etapa].data.mes = data_inserida.mes;
1286 |     ptr_etapa[numero_etapa].data.dia = data_inserida.dia;
1287 | }
```

Bruno Tereso, 2091066@student.estg.ipleiria.pt
Marco Ferreira, 2092016@student.estg.ipleiria.pt

```
1288 |
1289 |
1290 |  int pilIndiceAviao(int numero_aviao, ficha_t *ptr_piloto, conta_t *ptr_contador) {
1291 |      /*  Procura o avião indicado na lista de pilotos e devolve o indice correspondente.
1298 |
1299 |      int i;
1300 |
1301 |      for (i = 0; i < ptr_contador->pilotos; i++)
1302 |          if (ptr_piloto[i].aviao == numero_aviao)
1303 |              return i;
1304 |  }
1305 |
1306 |
1307 |  void auxPausa() {
1308 |      // Muda de linha e pausa o sistema.
1309 |
1310 |      printf("\n\n\t");
1311 |      system("PAUSE");
1312 |  }
1313 |
1314 |
1315 |  int etaVerificaInscricao(ficha_t *ptr_piloto, etapa_t *ptr_etapa, int aviao_piloto, int indice_etapa, conta_t *ptr_contador) {
1316 |      /*  Verifica se o avião está inscrito numa etapa ou qualquer outra etapa.
1329 |
1330 |      int i, j, aux=1;
1331 |
1332 |      if (indice_etapa < MAX_ETAPAS)
1333 |          for (i = 0; i < ptr_contador->pilotos; i++) // ve se esta' inscrito na etapa dada
1334 |              if (ptr_etapa[indice_etapa].inscritos[i] == aviao_piloto)
1335 |                  return 1;
1336 |
1337 |      for (i = 0; i <= MAX_ETAPAS - 1; i++) // ve se esta' inscrito em qualquer etapa
1338 |          for (j = 0;j <= MAX_PILOTOS_ETAPA - 1;j++)
1339 |              if (ptr_etapa[i].inscritos[j] == aviao_piloto && ptr_etapa[i].realizada == 1)
1340 |                  return -2;
1341 |              else
1342 |                  if (ptr_etapa[i].inscritos[j] == aviao_piloto && ptr_etapa[i].realizada == 0)
1343 |                      return -1;
1344 |                  else
1345 |                      aux = 0;
1346 |      return aux;
1347 |  }
1348 |
1349 |
1350 |  int pilProcuraDuplicado(char nomenumero[], ficha_t *ptr_piloto, conta_t *ptr_contador) {
1351 |      /*  Verifica se existe mais do que um piloto com o mesmo nome.
1360 |
```

Bruno Tereso, 2091066@student.estg.ipleiria.pt
Marco Ferreira, 2092016@student.estg.ipleiria.pt

```
1361 |      int contador = 0, i;
1362 |
1363 |      for (i = 0; i < ptr_contador->pilotos; i++)
1364 |          if (strcasecmp(nomenumero, ptr_piloto[i].nome) == 0)
1365 |              contador++;
1366 |      if (contador > 1) {
1367 |          return 1;
1368 |      }
1369 |      return 0;
1370 |  }
1371 |
1372 |
1373 |  int etaProcuraDuplicada(char nomenumero[], etapa_t *ptr_etapa) {
1374 |      /*  Verifica se existe mais do que uma etapa com o mesmo nome.
1382 |
1383 |      int contador = 0, i;
1384 |
1385 |      for (i = 0; i <= MAX_ETAPAS - 1; i++)
1386 |          if (strcasecmp(nomenumero, ptr_etapa[i].nome) == 0)
1387 |              contador++;
1388 |      if (contador > 1) {
1389 |          return 1;
1390 |      }
1391 |      return 0;
1392 |  }
1393 |
1394 |
1395 |  int etaInscricoesVagas(etapa_t *ptr_etapa, conta_t *ptr_contador) {
1396 |      /*  Conta o número de vagas do campeonato, em todas as etapas.
1403 |
1404 |      int i, j, contador = 0;
1405 |
1406 |      for (i = 0; i <= MAX_ETAPAS - 1; i++)
1407 |          for (j = 0; j <= (ptr_etapa[i].participantes) - 1; j++)
1408 |              if (ptr_etapa[i].inscritos[j] == 0)
1409 |                  contador++;
1410 |      return contador;
1411 |  }
1412 |
1413 |
1414 |  void etaLimpaInscricoes(int indice_etapa, etapa_t *ptr_etapa) {
1415 |      /*  Apaga todas as inscrições de uma etapa.
1420 |
1421 |      int i;
1422 |
1423 |      for (i = 0; i < ptr_etapa[indice_etapa].participantes; i++)
1424 |          ptr_etapa[indice_etapa].inscritos[i] = 0;
```

Bruno Tereso, 2091066@student.estg.ipleiria.pt
Marco Ferreira, 2092016@student.estg.ipleiria.pt

```
1425 |  }
1426 |
1427 |
1428 |  int etaNumRealizadas(etapa_t *ptr_etapa) {
1429 |      /*  Conta o número de etapa realizadas para apresentar no menu principal.
1435 |
1436 |      int i, cont=0;
1437 |
1438 |      for ( i=0 ; i< MAX_ETAPAS ; i++ )
1439 |          if ( ptr_etapa[i].realizada == 1 )
1440 |              cont++;
1441 |
1442 |      return cont;
1443 |  }
1444 |
1445 |
1446 |  int etaProcuraPiloto (int etapa, int aviao, etapa_t *ptr_etapa) {
1447 |      /*  Procura um piloto dentro da etapa especificada.
1456 |
1457 |      int i;
1458 |
1459 |      for ( i = 0 ; i < MAX_PILOTOS_ETAPA ; i++)
1460 |          if ( ptr_etapa[etapa].inscritos[i] == aviao )
1461 |              return i;
1462 |
1463 |      return ERRO_NAOEXISTE;
1464 |  }
1465 |
1466 |
1467 |  void pilMostraInscricoes(ficha_t *ptr_piloto, etapa_t *ptr_etapa, int indice, int tipo) {
1468 |      /*  Mostra uma lista com as inscrições de um piloto.
1476 |
1477 |      int i, j;
1478 |
1479 |      if (tipo == 1) {
1480 |          printf("\n\t\t\t\t\t   Etapa | Posicao | Pontuacao\n\n");
1481 |
1482 |
1483 |          for (i = 0; i < MAX_ETAPAS; i++) // de i ate max etapas
1484 |              for (j = 0;j < ptr_etapa[i].participantes;j++)  // de j ate ao numero actual de participantes na etapa i
1485 |                  if (ptr_etapa[i].inscritos[j] == ptr_piloto[indice].aviao) {
1486 |                      if ( ptr_etapa[i].realizada == 1 ) // se a etapa já tem pontos e posição ( se esta realizada )
1487 |                          printf("\t\t%31.31s |    %2.d  |    %3.d\n", ptr_etapa[i].nome, j+1 ,ptr_etapa[i].pontos[j]);
1488 |                      else // se n está realizada, mostra não disponivel
1489 |                          printf("\t\t%31.31s |   N/D  |     N/D \n", ptr_etapa[i].nome);
1490 |                  }
1491 |      } else
```

Bruno Tereso, 2091066@student.estg.ipleiria.pt
Marco Ferreira, 2092016@student.estg.ipleiria.pt

```
1492 |          if (tipo == 2) {
1493 |              printf("\n\t\t\t\t\t  Etapa |   Data\n\n");
1494 |              for (i = 0; i < MAX_ETAPAS; i++) // de i ate max etapas
1495 |                  for (j = 0;j < ptr_etapa[i].participantes;j++)  // de j ate ao numero actual de participantes na etapa i
1496 |                      if (ptr_etapa[i].inscritos[j] == ptr_piloto[indice].aviao)
1497 |                          printf("\t\t%31.31s  | %.2d.%.2d.%d\n", ptr_etapa[i].nome,
1498 |                                 ptr_etapa[i].data.dia, ptr_etapa[i].data.mes, ptr_etapa[i].data.ano);
1499 |          }
1500 |
1501 |  }
1502 |
1503 |
1504 |  void pilLimpaInscricoes(ficha_t *ptr_piloto, etapa_t *ptr_etapa, int indice) {
1505 |      /*  Limpa as inscrições de um piloto em todas as etapas.
1511 |
1512 |      int inscritos, i, j, k;
1513 |
1514 |      for (i = 0; i < MAX_ETAPAS; i++)
1515 |          for (j = 0; j < ptr_etapa[i].participantes; j++)  // de j ate ao numero actual de participantes na etapa i
1516 |              if (ptr_etapa[i].inscritos[j] == ptr_piloto[indice].aviao) {
1517 |                  //inscritos = auxCompVector(ptr_etapa[i].inscritos, 20);
1518 |                  ptr_etapa[i].totalinscritos--; // decrementa um inscrito, porque encontrou o aviao no vector
1519 |                  for (k = j; k < ptr_etapa[i].totalinscritos; k++) // da posição actual até ao ultimo inscrito
1520 |                      ptr_etapa[i].inscritos[k] = ptr_etapa[i].inscritos[k+1];
1521 |              }
1522 |  }
1523 |
1524 |
1525 |  ficha_t *pilApaga(ficha_t *ptr_piloto, conta_t *ptr_contador, int indice) {
1526 |      /*  Apaga o registo de um piloto.
1532 |
1533 |      int i;
1534 |
1535 |      for (i = indice; i < ptr_contador->pilotos; i++) {
1536 |          strcpy(ptr_piloto[i].nome, ptr_piloto[i+1].nome);
1537 |          ptr_piloto[i].idade = ptr_piloto[i+1].idade;
1538 |          ptr_piloto[i].aviao = ptr_piloto[i+1].aviao;
1539 |      }
1540 |
1541 |      ptr_contador->pilotos--; // decrementa os pilotos depois de eliminar com sucesso.
1542 |      ptr_piloto = (ficha_t*)realloc(ptr_piloto, (ptr_contador->pilotos)*sizeof(ficha_t));
1543 |
1544 |      return ptr_piloto;
1545 |  }
1546 |
1547 |
1548 |  char resMenu(conta_t *ptr_contador, etapa_t *ptr_etapa) {
```

```
1549 |      /*  Mostra o menu de Resultados no ecrã.
1555 |
1556 |      char op;
1557 |
1558 |      do {
1559 |          resCabecalho(ptr_etapa, ptr_contador);
1560 |          printf("\n\n\tR - Resultados da Etapa Corrente");
1561 |          printf("\n\tA - Alterar Resultados de uma Etapa");
1562 |          printf("\n\tO - Consultar Resultados de uma Etapa");
1563 |          printf("\n\tC - Classificacao Final");
1564 |          printf("\n\n\tV - Voltar ao Menu Principal");
1565 |          fflush(stdin);
1566 |          op = getch();
1567 |          op = toupper(op);
1568 |      } while (op != 'V' && op != 'R' && op != 'A' && op != 'O' && op != 'C');
1569 |
1570 |      return op;
1571 |  }
1572 |
1573 |
1574 |  void resFuncoes(ficha_t *ptr_piloto, etapa_t *ptr_etapa, char op, conta_t *ptr_contador) {
1575 |      /*  Recebe a opção escolhida e direcciona para a função a usar.
1583 |      int vagas, aux;
1584 |
1585 |      aux = etaNumRealizadas(ptr_etapa);
1586 |
1587 |      switch (op) {
1588 |      case 'R':  // le os valores da etapa corrente
1589 |          resLeCorrente(ptr_etapa, ptr_piloto, ptr_contador);
1590 |          break;
1591 |
1592 |      case 'A':  //elimitar piloto
1593 |          if ( aux > 0 )
1594 |              resAlterarResultados(ptr_etapa, ptr_piloto, ptr_contador);
1595 |          else {
1596 |              printf("\n\n\tNao existem etapa realizadas para editar...");
1597 |              auxPausa();
1598 |          }
1599 |          break;
1600 |
1601 |      case 'O':  //elimitar piloto
1602 |          if ( aux > 0 )
1603 |              resConsultar (ptr_etapa, ptr_piloto, ptr_contador);
1604 |          else {
1605 |              printf("\n\n\tAinda nao existem etapa realizadas...");
1606 |              auxPausa();
1607 |          }
```

Bruno Tereso, 2091066@student.estg.ipleiria.pt
Marco Ferreira, 2092016@student.estg.ipleiria.pt

```
1608 |          break;
1609 |
1610 |      case 'C':  //lista de pilotos
1611 |          if ( aux > 0 ) {
1612 |              estMostraClassificacao (1, ptr_piloto, ptr_contador); // listagem simples sem opções
1613 |              auxPausa();
1614 |          } else {
1615 |              printf("\n\n\tAinda nao existem etapa realizadas...");
1616 |              auxPausa();
1617 |          }
1618 |          break;
1619 |      }
1620 |  }
1621 |
1622 |
1623 |  void resLeCorrente(etapa_t *ptr_etapa, ficha_t *ptr_piloto, conta_t *ptr_contador) {
1624 |      /*  Verifica a Etapa corrente e recebe os tempos dados pelo utilizador.
1630 |
1631 |      int i, totalpilotos, etapa, piloto, resultado;
1632 |
1633 |      resCabecalho(ptr_etapa, ptr_contador);
1634 |      etapa = ptr_contador->etapa_corrente; // guarda o indice da etapa em questao
1635 |      if ( ptr_etapa[etapa].totalinscritos == 0)
1636 |          printf("\n\n\tA etapa corrente nao tem inscritos, adicione pilotos primeiro...");
1637 |      else
1638 |          if ( ptr_etapa[etapa].realizada == 1)
1639 |              printf("\n\n\tJa leu resultados para todas as etapas...");
1640 |          else {
1641 |          printf("\n\n\tResultados da Etapa: %s \n\tTotal Inscritos: %d\n", ptr_etapa[etapa].nome, ptr_etapa[etapa].totalinscritos);
1642 |              resLeTempos(etapa, ptr_etapa, ptr_piloto, ptr_contador);
1643 |              printf("\n\n\tValores recolhidos com sucesso...");
1644 |              for ( i=etapa+1 ; i < MAX_ETAPAS; i++)
1645 |                  if (ptr_etapa[i].realizada != 1 && ptr_etapa[i].data.ano!=0) { // procura a proxima etapa inserida
1646 |                      ptr_contador->etapa_corrente = i; // guarda o indice da proxima etapa valida
1647 |                      break;
1648 |                  }
1649 |              ptr_etapa[etapa].realizada = 1;
1650 |              ptr_contador->grava = 1; // houve alterações, pergunta se quer gravar antes de sair
1651 |          }
1652 |      auxPausa();
1653 |  }
1654 |
1655 |
1656 |  void resLeTempos(int etapa, etapa_t *ptr_etapa, ficha_t *ptr_piloto, conta_t *ptr_contador) {
1657 |      /*  Lista o nome do piloto e Recebe o valor do tempo no formato mm.ss,mseg.
1663 |
1664 |      int piloto, totalpilotos, i, resultado;
```

Bruno Tereso, 2091066@student.estg.ipleiria.pt
Marco Ferreira, 2092016@student.estg.ipleiria.pt

```
1665 |
1666 |      printf("\n\tInsira os valores do piloto: [ Exemplo 00:00.000 ]");
1667 |      for (i = 0 ; i < ptr_etapa[etapa].totalinscritos; i++) {
1668 |          piloto = pilIndiceAviao(ptr_etapa[etapa].inscritos[i], ptr_piloto, ptr_contador); // guarda o indice do piloto
1669 |          do {
1670 |              printf("\n\t%.3d | %s : ", ptr_piloto[piloto].aviao, ptr_piloto[piloto].nome);
1671 |              resultado = resLeSegundos();
1672 |              if ( resultado == 0)
1673 |                  printf("\tValor Invalido.");
1674 |          } while ( resultado == 0);
1675 |          ptr_etapa[etapa].tempos[i] = resultado;
1676 |      }
1677 |      fflush(stdin);
1678 |      resOrdenaTempos (etapa, ptr_etapa);
1679 |      fflush(stdin);
1680 |      resPontuacao (etapa, ptr_piloto, ptr_etapa, ptr_contador);
1681 |  }
1682 |
1683 |
1684 |  int resLeSegundos() {
1685 |      /*  Le um valor de tempo no formato mm.ss,mseg. e transforma em milesimos de segundo para guardar no vector etapas
1689 |
1690 |      int aux=0, valido=0;
1691 |      int min, seg, mseg, total;
1692 |
1693 |      fflush(stdin);
1694 |      aux = scanf("%2d:%2d.%d", &min, &seg, &mseg);
1695 |
1696 |      if (min < 0 || seg < 0 || mseg <0 )
1697 |          valido++; // encontrou erro com valores negativos
1698 |      if (min > 59 || seg > 60 || mseg > 999 )
1699 |          valido++; // encontrou valores impossiveis
1700 |      if ( aux != 3)
1701 |          valido++; // encontrou um erro de leitura com virgula e ponto
1702 |      if ( valido != 0)
1703 |          return 0; // devolve valor invalido
1704 |
1705 |      total = mseg + seg*1000 + min*60*1000;
1706 |      return total; // devolve total convertido em ms
1707 |  }
1708 |
1709 |
1710 |  void resAlterarResultados (etapa_t *ptr_etapa, ficha_t *ptr_piloto, conta_t *ptr_contador) {
1711 |      /*  Lista as etapas e permite alterar os resultados de uma delas à escolha do utilizador.
1717 |
1718 |      int i, etapa;
1719 |
```

Bruno Tereso, 2091066@student.estg.ipleiria.pt
Marco Ferreira, 2092016@student.estg.ipleiria.pt

```
1720 |      resCabecalho(ptr_etapa, ptr_contador); // 0 = menu principal | 1 = menu pilotos | 2 = menu etapas
1721 |      printf("\n\tEscolha a etapa para alterar resultados:");
1722 |      etaListar(2, ptr_etapa, ptr_contador);
1723 |      do {
1724 |          etapa = auxLeInteiro("\n\tInsira a ordem da etapa a editar: ", 1, ptr_contador->etapa_corrente+1);
1725 |          etapa--; // coloca a variavel no indice correcto
1726 |          if (ptr_etapa[etapa].data.ano == 0)
1727 |              printf("\tEtapa nao existe.\n");
1728 |      } while (ptr_etapa[etapa].data.ano == 0);
1729 |      fflush(stdin);
1730 |      resRemPontuacao (etapa, ptr_piloto, ptr_etapa, ptr_contador); // remove os pontos anteriores
1731 |      fflush(stdin);
1732 |      resLeTempos(etapa, ptr_etapa, ptr_piloto, ptr_contador); // le os novos tempos e distribui a pontuação
1733 |      printf("\n\tEtapa alterada com sucesso....");
1734 |      ptr_contador->grava = 1; // houve alterações, pergunta se quer gravar antes de sair
1735 |      auxPausa();
1736 |  }
1737 |
1738 |
1739 |  void resOrdenaTempos (int etapa, etapa_t *ptr_etapa) {
1740 |      /*  Ordena todos os tempos recebidos crescentemente e actualiza as posições no vector de inscritos.
1746 |
1747 |      int aux, aux2, aux3, ha_trocas,totalpilotos, i,j;
1748 |
1749 |      for ( i = 0; i < ptr_etapa[etapa].totalinscritos && ha_trocas; i++) {
1750 |          ha_trocas = 0;
1751 |
1752 |          for ( j = 0; j < ptr_etapa[etapa].totalinscritos - i - 1; j++)
1753 |              if ( ptr_etapa[etapa].tempos[j] > ptr_etapa[etapa].tempos[j+1]) { // Inverte para decrescemaxte
1754 |                  ha_trocas=1;
1755 |
1756 |                  aux=ptr_etapa[etapa].tempos[j]; //guarda o aviao
1757 |                  aux2=ptr_etapa[etapa].inscritos[j]; // guarda o tempo
1758 |                  aux3=ptr_etapa[etapa].pontos[j]; // guarda os pontos
1759 |
1760 |                  ptr_etapa[etapa].tempos[j]=ptr_etapa[etapa].tempos[j+1]; // troca o numero do aviao
1761 |                  ptr_etapa[etapa].inscritos[j]=ptr_etapa[etapa].inscritos[j+1]; // troca o tempo
1762 |                  ptr_etapa[etapa].pontos[j]=ptr_etapa[etapa].pontos[j+1]; // troca os pontos
1763 |
1764 |                  ptr_etapa[etapa].tempos[j+1]=aux;
1765 |                  ptr_etapa[etapa].inscritos[j+1]=aux2;
1766 |                  ptr_etapa[etapa].pontos[j+1]=aux3;
1767 |
1768 |              }
1769 |      }
1770 |  }
1771 |
```

Bruno Tereso, 2091066@student.estg.ipleiria.pt
Marco Ferreira, 2092016@student.estg.ipleiria.pt

```
1772 |
1773 |  void resPontuacao (int etapa, ficha_t *ptr_piloto, etapa_t *ptr_etapa, conta_t *ptr_contador) {
1774 |      /*  Preenche o vector das pontuações para a etapa indicada, le os tempos.
1781 |
1782 |      int indice, i, pontos=10;
1783 |
1784 |      // do primeiro ao sexto classificado
1785 |      for (i = 0; i < 6;i++) {
1786 |          indice = pilIndiceAviao(ptr_etapa[etapa].inscritos[i], ptr_piloto, ptr_contador);
1787 |          if ( ptr_etapa[etapa].inscritos[i] != 0 ) {// se existe um piloto para pontuar
1788 |              if ( i != 5 ) { // se for 1 dos 5 primeiros tempos
1789 |                  ptr_piloto[indice].pontos += pontos; // aumenta os pontos do piloto
1790 |                  ptr_etapa[etapa].pontos[i] = pontos; // pontua a etapa
1791 |                  pontos-=2; // o proximo piloto vai ter -2 pontos
1792 |              } else { // se for o 6º classificado, atribui 1 ponto
1793 |                  ptr_etapa[etapa].pontos[i] = 1;
1794 |                  ptr_piloto[indice].pontos += 1;
1795 |              }
1796 |          }
1797 |      }
1798 |  }
1799 |
1800 |
1801 |  void resRemPontuacao (int etapa, ficha_t *ptr_piloto, etapa_t *ptr_etapa, conta_t *ptr_contador) {
1802 |      /*  Limpa as pontuações atribuidas a pilotos e etapa,
1803 |          usada quando se pretende alterar os resultados de uma etapa.
1810 |      int indice, i, pontos=10;
1811 |
1812 |
1813 |      // do primeiro ao sexto classificado
1814 |      for (i = 0; i < 6;i++) {
1815 |          indice = pilIndiceAviao(ptr_etapa[etapa].inscritos[i], ptr_piloto, ptr_contador);
1816 |          if ( i != 5) {
1817 |              ptr_piloto[indice].pontos -= pontos;
1818 |              ptr_etapa[etapa].pontos[i] = 0;
1819 |              pontos-=2;
1820 |          } else { // se for o 6º classificado, atribui 1 ponto
1821 |              ptr_etapa[etapa].pontos[i] = 0;
1822 |              ptr_piloto[indice].pontos -= 1;
1823 |          }
1824 |      }
1825 |  }
1826 |
1827 |
1828 |  void resConsultar (etapa_t *ptr_etapa, ficha_t *ptr_piloto, conta_t *ptr_contador) {
1829 |      /*  Lista etapas, recebe a ordem da etapa a visualizar e chama a função que mostra os resultados.
1835 |
```

Bruno Tereso, 2091066@student.estg.ipleiria.pt
Marco Ferreira, 2092016@student.estg.ipleiria.pt

```
1836 |      int pontos, tempo, totalpilotos, etapa, i;
1837 |
1838 |      resCabecalho(ptr_etapa, ptr_contador); // 0 = menu principal | 1 = menu pilotos | 2 = menu etapas
1839 |      printf("\n\tEscolha a etapa para consultar resultados:");
1840 |      etaListar(2, ptr_etapa, ptr_contador);
1841 |      do { // INWORK mudar verificação para se esta realizada ou nao, pode ler de 1 a max_etapas pq depois vai dar erro na mesma
1842 |          etapa = auxLeInteiro("\n\tInsira a ordem da etapa: ", 1, MAX_ETAPAS);
1843 |          etapa--; // coloca a variavel no indice correcto
1844 |          if (ptr_etapa[etapa].data.ano == 0)
1845 |              printf("\tEtapa nao existe.\n");
1846 |          if (ptr_etapa[etapa].realizada == 0)
1847 |              printf("\tErro, Etapa por realizar.\n");
1848 |      } while (ptr_etapa[etapa].data.ano == 0 || ptr_etapa[etapa].realizada == 0);
1849 |
1850 |      resEtapa(etapa, ptr_etapa, ptr_piloto, ptr_contador);
1851 |      auxPausa();
1852 |  }
1853 |
1854 |
1855 |  void resEtapa (int etapa, etapa_t *ptr_etapa, ficha_t *ptr_piloto, conta_t *ptr_contador) {
1856 |      /*  Mostra os dados de um etapa realizada, Nº Aviao, Nome do Piloto, Tempo e Pontos.
1857 |          Ordenado pela classificação.
1864 |
1865 |      int piloto, i, totalpilotos;
1866 |      tmp_t tempo;
1867 |
1868 |      printf("\n\n\t  Aviao |\t\t\t     Nome |   Tempo    | Pontos\n");
1869 |      for (i=0; i< ptr_etapa[etapa].totalinscritos; i++) {
1870 |          piloto = pilIndiceAviao(ptr_etapa[etapa].inscritos[i], ptr_piloto, ptr_contador); // guarda o indice do piloto
1871 |          tempo = auxConverteMseg( ptr_etapa[etapa].tempos[i] ); // converte o tempo para min/seg/mseg para mostrar no ecra
1872 |          printf("\n\t    %.3d | %31.31s | ", ptr_etapa[etapa].inscritos[i], ptr_piloto[piloto].nome); //inwork
1873 |          printf("%.2d:%.2d.%.3d |", tempo.min, tempo.seg, tempo.mseg);
1874 |          printf(" %d", ptr_etapa[etapa].pontos[i]);
1875 |      }
1876 |  }
1877 |
1878 |
1879 |  tmp_t auxConverteMseg (int tempo_mseg) {
1880 |      /*  Converte o tempo de milesimos de segundos para o formato mm.ss,mseg
1881 |          de modo a imprimir no ecrã
1887 |
1888 |      tmp_t tempo;
1889 |      float min, seg, mseg;
1890 |
1891 |      min = tempo_mseg / (1000 * 60);
1892 |      seg = tempo_mseg - ( min * 60 * 1000 );
1893 |      mseg = seg - ( ((int)seg/1000) * 1000 );
```

Bruno Tereso, 2091066@student.estg.ipleiria.pt
Marco Ferreira, 2092016@student.estg.ipleiria.pt

```
1894 |
1895 |      tempo.min = (int)min;
1896 |      tempo.seg = (int)seg/1000;
1897 |      tempo.mseg = (int)mseg;
1898 |
1899 |      return tempo;
1900 |  }
1901 |
1902 |
1903 |  void resOrdenaClass (ficha_t *ptr_piloto, conta_t *ptr_contador) {
1904 |      /*  Ordena a Classificação Geral do campeonato, com os valores guardados em cada piloto
1908 |
1909 |      ficha_t aux;
1910 |      int i, j, ha_trocas,posmenor;
1911 |
1912 |      for ( i = 0; i < ptr_contador->pilotos && ha_trocas; i++) {
1913 |          ha_trocas = 0;
1914 |          for ( j = 0; j < ptr_contador->pilotos - i - 1; j++)
1915 |              if ( ptr_piloto[j].pontos < ptr_piloto[j+1].pontos) {
1916 |                  ha_trocas=1;
1917 |
1918 |                  strcpy(aux.nome, ptr_piloto[j].nome);
1919 |                  aux.aviao = ptr_piloto[j].aviao;
1920 |                  aux.idade = ptr_piloto[j].idade;
1921 |                  aux.pontos = ptr_piloto[j].pontos;
1922 |
1923 |                  strcpy(ptr_piloto[j].nome,ptr_piloto[j+1].nome);
1924 |                  ptr_piloto[j].aviao = ptr_piloto[j+1].aviao;
1925 |                  ptr_piloto[j].idade = ptr_piloto[j+1].idade;
1926 |                  ptr_piloto[j].pontos = ptr_piloto[j+1].pontos;
1927 |
1928 |                  strcpy(ptr_piloto[j+1].nome,aux.nome);
1929 |                  ptr_piloto[j+1].aviao = aux.aviao;
1930 |                  ptr_piloto[j+1].idade = aux.idade;
1931 |                  ptr_piloto[j+1].pontos = aux.pontos;
1932 |              }
1933 |      }
1934 |  }
1935 |
1936 |
1937 |  void resOrdenaClassAviao (ficha_t *ptr_piloto, conta_t *ptr_contador) {
1938 |      /*  Ordena a classificação geral pelo número do avião
1944 |
1945 |      ficha_t aux;
1946 |      int i, j, ha_trocas,posmenor;
1947 |
1948 |      for ( i = 0; i < ptr_contador->pilotos && ha_trocas; i++) {
```

Bruno Tereso, 2091066@student.estg.ipleiria.pt
Marco Ferreira, 2092016@student.estg.ipleiria.pt

```
1949 |          ha_trocas = 0;
1950 |          for ( j = 0; j < ptr_contador->pilotos - i - 1; j++)
1951 |              if ( ptr_piloto[j].aviao > ptr_piloto[j+1].aviao) {
1952 |                  ha_trocas=1;
1953 |
1954 |                  strcpy(aux.nome, ptr_piloto[j+1].nome);
1955 |                  aux.aviao = ptr_piloto[j+1].aviao;
1956 |                  aux.idade = ptr_piloto[j+1].idade;
1957 |                  aux.pontos = ptr_piloto[j+1].pontos;
1958 |
1959 |                  strcpy(ptr_piloto[j+1].nome,ptr_piloto[j].nome);
1960 |                  ptr_piloto[j+1].aviao = ptr_piloto[j].aviao;
1961 |                  ptr_piloto[j+1].idade = ptr_piloto[j].idade;
1962 |                  ptr_piloto[j+1].pontos = ptr_piloto[j].pontos;
1963 |
1964 |                  strcpy(ptr_piloto[j].nome,aux.nome);
1965 |                  ptr_piloto[j].aviao = aux.aviao;
1966 |                  ptr_piloto[j].idade = aux.idade;
1967 |                  ptr_piloto[j].pontos = aux.pontos;
1968 |              }
1969 |      }
1970 |  }
1971 |
1972 |
1973 |  void estMostraClassificacao (int simples, ficha_t *ptr_piloto, conta_t *ptr_contador) {
1974 |      /*  Lista a classificação geral nas estatísticas, ordenada por pontos ou avião.
1982 |
1983 |      int i, aux = 0, alternador = 0;
1984 |      char opcao;
1985 |
1986 |      do {
1987 |          fflush(stdin);
1988 |          aux = 0;
1989 |          if ( alternador % 2 == 0)// se for par mostra ordenado por pontuação, impar por aviao, o X alterna
1990 |              resOrdenaClass (ptr_piloto, ptr_contador);
1991 |          else
1992 |              resOrdenaClassAviao (ptr_piloto, ptr_contador);
1993 |
1994 |          system("CLS");
1995 |          puts(PROGRAMA);
1996 |          printf("\n\n\tClassificacao Geral: ( Pilotos sem pontuacao omitidos )\n\n");
1997 |          printf("\n\t\t\t\t\t\t Nome | Aviao |  Pontos\n");
1998 |
1999 |          for (i=0 ; i< ptr_contador->pilotos; i++) {
2000 |              if ( ptr_piloto[i].aviao != 0 && ptr_piloto[i].pontos != 0) {
2001 |                  printf("\n\t\t      %31.31s |  %3.d |  %3.d", ptr_piloto[i].nome, ptr_piloto[i].aviao, ptr_piloto[i].pontos);
2002 |                  aux++; // conta os pilotos que vai mostrar, para fazer pausa a cada 10
```

Bruno Tereso, 2091066@student.estg.ipleiria.pt
Marco Ferreira, 2092016@student.estg.ipleiria.pt

```
2003 |              }
2004 |              if (aux != 0 && aux % 10 == 0)
2005 |                  auxPausa();
2006 |          }
2007 |          if ( simples != 1 ) { // se for listagem com opcoes para Estatisticas
2008 |              printf("\n\n\n\t[ X ] - Alterna a ordem Pontos/Aviao\t\t [ S ] - Sair\n\t");
2009 |              opcao = getch();
2010 |              opcao = toupper(opcao);
2011 |              if ( opcao == 'X' )
2012 |                  alternador++; // se carregou X muda a visualização
2013 |          } else // se nao, sai de imediato
2014 |              opcao = 'S';
2015 |      } while ( opcao != 'S' );
2016 |  }
2017 |
2018 |
2019 |  char estMenu(conta_t *ptr_contador) {
2020 |      /*  Mostra o menu de Pilotos no ecrã.
2026 |
2027 |      char op;
2028 |
2029 |      do {
2030 |          menuCabecalho(4, ptr_contador); // 0 = menu principal | 1 = menu pilotos | 2 = menu etapas
2031 |          printf("\n\n\tR - Resultados do Campeonato");
2032 |          printf("\n\tH - Historico por Piloto");
2033 |          printf("\n\tD - Dados Estatísticos");
2034 |          printf("\n\n\tV - Voltar ao Menu Principal");
2035 |          fflush(stdin);
2036 |          op = getch();
2037 |          op = toupper(op);
2038 |      } while (op != 'V' && op != 'R' && op != 'H' && op != 'D');
2039 |
2040 |      return op;
2041 |  }
2042 |
2043 |
2044 |
2045 |  void estFuncoes(ficha_t *ptr_piloto, etapa_t *ptr_etapa, char op, conta_t *ptr_contador) {
2046 |      /*  Recebe a opção escolhida e direcciona para a função a usar.
2053 |
2054 |      int aux;
2055 |
2056 |      aux = etaNumRealizadas(ptr_etapa);
2057 |
2058 |      switch (op) {
2059 |      case 'R':  // (R)esultados do Campeonato
2060 |          if ( aux > 0 )
```

Bruno Tereso, 2091066@student.estg.ipleiria.pt
Marco Ferreira, 2092016@student.estg.ipleiria.pt

```
2061 |              estMostraClassificacao (0, ptr_piloto, ptr_contador); // 0 = listagem com opções
2062 |          else {
2063 |              printf("\n\tAinda nao existem etapas realizadas...");
2064 |              auxPausa();
2065 |          }
2066 |          break;
2067 |
2068 |      case 'H':  // (H)istórico por Piloto
2069 |          if ( aux > 0 )
2070 |              estHistorico (ptr_etapa, ptr_piloto, ptr_contador);
2071 |          else {
2072 |              printf("\n\tAinda nao existem etapas realizadas...");
2073 |              auxPausa();
2074 |          }
2075 |          break;
2076 |
2077 |      case 'D':  // (D)ados Estatísticos
2078 |          if ( aux > 2 ) {
2079 |              estDados (ptr_contador, ptr_etapa, ptr_piloto);
2080 |              auxPausa();
2081 |          } else {
2082 |              printf("\n\tPara ver as estatisticas precisa de pelo menos 3 etapas realizadas...");
2083 |              auxPausa();
2084 |          }
2085 |          break;
2086 |      }
2087 |  }
2088 |
2089 |
2090 |  void estHistorico (etapa_t *ptr_etapa, ficha_t *ptr_piloto, conta_t *ptr_contador) {
2091 |      /*  Mostra uma tabela de Pilotos/Etapas com a classificação de cada.
2097 |
2098 |      int encontrou_piloto, pontos, tempo, piloto, totalpilotos, etapa, i,j;
2099 |
2100 |      fflush(stdin);
2101 |      resOrdenaClass (ptr_piloto, ptr_contador); // ordena a classificação geral
2102 |      estCabecalhoHistorico (ptr_etapa, ptr_contador); // mostra o cabeçalho
2103 |
2104 |      for ( i = 0 ; i < ptr_contador->pilotos ; i++ ) {
2105 |          printf("\n| %.3d-%15.15s | %.3d |", ptr_piloto[i].aviao, ptr_piloto[i].nome, ptr_piloto[i].pontos);
2106 |          for ( j = 0; j < MAX_ETAPAS; j++ ) {
2107 |              encontrou_piloto = etaProcuraPiloto (j, ptr_piloto[i].aviao, ptr_etapa);
2108 |              if ( encontrou_piloto != ERRO_NAOEXISTE )
2109 |                  printf(" %2d |", ptr_etapa[j].pontos[encontrou_piloto]);
2110 |              else
2111 |                  printf("  0 |");
2112 |          }
```

Bruno Tereso, 2091066@student.estg.ipleiria.pt
Marco Ferreira, 2092016@student.estg.ipleiria.pt

```
2113 |          printf("\n+-------------------------------------------------------------------------------+");
2114 |          if (i != 0 && i % 5 == 0 && i <ptr_contador->pilotos) {
2115 |              auxPausa(); // se não tiver i <ptr_contador->pilotos faz um pause a mais no fim da listagem quando o fim dá resto 0
2116 |              system("CLS");
2117 |              estCabecalhoHistorico (ptr_etapa, ptr_contador);
2118 |          }
2119 |      }
2120 |      auxPausa();
2121 |  }
2122 |
2123 |
2124 |  void estCabecalhoHistorico (etapa_t *ptr_etapa, conta_t *ptr_contador) {
2125 |      /*  Cabeçalho para a Listagem do Histórico.
2130 |
2131 |      int i;
2132 |      char titulo[5];
2133 |
2134 |      system("CLS");
2135 |      printf("\n\n                        Tabela Geral de Classificacoes\n");
2136 |      printf("+-------------------------------------------------------------------------------+\n");
2137 |      printf("|                    |    |   |   |   |   |   |   |   |   |   |    |\n");
2138 |      printf("| Aviao / Nome        |Total|");
2139 |
2140 |      for ( i=0 ; i<MAX_ETAPAS ; i++ ) {
2141 |          strncpy(titulo, ptr_etapa[i].nome, 4);
2142 |          titulo[3] = '.';
2143 |          titulo[4] = '\0';
2144 |          if ( ptr_etapa[i].data.ano != 0)
2145 |              printf("%s|", titulo);
2146 |          else
2147 |              printf(" -- |");
2148 |      }
2149 |      printf("\n|                     |(pts)|");
2150 |      for ( i=0 ; i<MAX_ETAPAS ; i++ ) // mostra as ordens
2151 |          printf("(%2.d)|", i+1);
2152 |      printf("\n|                     |    |   |   |   |   |   |   |   |   |   |    |");
2153 |      printf("\n|---------------------+-----+----+----+----+----+----+----+----+----+----+----|");
2154 |  }
2155 |
2156 |
2157 |  void estDados (conta_t *ptr_contador, etapa_t *ptr_etapa, ficha_t *ptr_piloto) {
2158 |      /*  Mostra os dados estatísticos do campeonato.
2159 |          número e percentagem de etapas já realizadas, etapa(s) com maior e menor número de participantes,
2160 |          número médio de participantes por etapas, piloto(s) com maior número de vitórias,
2161 |          número e percentagem de pilotos com vitórias em etapas, piloto(s) com o maior número de vitórias consecutivas.
2167 |
2168 |      int i;
```

Bruno Tereso, 2091066@student.estg.ipleiria.pt
Marco Ferreira, 2092016@student.estg.ipleiria.pt

```
2169 |      int vencedores[MAX_ETAPAS]; // Vector para auxilio a outras funções.
2170 |      int pilotos[ptr_contador->pilotos][2]; // Matriz para auxilio a outras funções, usada para ordenações.
2171 |      int etapas[MAX_ETAPAS][2]; // Matriz para auxilio a outras funções, usada para ordenações.
2172 |
2173 |      float pct_realizadas;
2174 |
2175 |      system("CLS");
2176 |      //puts(PROGRAMA); falta de espaço
2177 |      printf("\n\tESTG AIR RACE | Estatisticas Gerais:\n");
2178 |
2179 |
2180 |      pct_realizadas = ( (float)ptr_contador->realizadas * 100 ) / (float)ptr_contador->etapas;
2181 |      printf("\n\tEstao actualmente %.2f%% [ %d em %d ] de Etapas Realizadas.\n", pct_realizadas,
2182 |             ptr_contador->realizadas, ptr_contador->etapas ); // percentagem de etapas realizadas
2183 |
2184 |      auxVencedores ( vencedores, ptr_etapa); // preenche o vector vencedores com o vencedor de cada etapa
2185 |      printf("\n\tPilotos com mais Vitorias:");
2186 |      estNumVitorias ( pilotos, vencedores, ptr_piloto, ptr_contador, 'D'); // pilotos com mais vitorias
2187 |      fflush(stdin);
2188 |      estPctVictorias ( pilotos, ptr_piloto, ptr_contador);
2189 |      printf("\n\n\tEtapas com menos inscritos:");
2190 |      fflush(stdin); // limpa buffers porque mostrava resultados inválidos
2191 |      estEtapas ( etapas, ptr_etapa, ptr_contador, 'A');
2192 |      printf("\n\n\tEtapas com mais inscritos:");
2193 |      fflush(stdin);
2194 |      estEtapas ( etapas, ptr_etapa, ptr_contador, 'D');
2195 |      fflush(stdin);
2196 |      estMediaParticipantes ( ptr_etapa, ptr_contador ); //media de participantes
2197 |      fflush(stdin);
2198 |      estVitoriasConsecutivas (vencedores, ptr_etapa, ptr_piloto, ptr_contador);
2199 |  }
2200 |
2201 |
2202 |  void auxVencedores ( int vencedores[], etapa_t *ptr_etapa) {
2203 |      /*  Preenche um vector Vencedores com o numero dos aviões que venceram cada etapa.
2204 |          Vector para auxilio a outras funções.
2209 |
2210 |      int i;
2211 |
2212 |      for ( i = 0; i < MAX_ETAPAS; i++) { // preenche o vector vencedores com o vencedor de cada etapa
2213 |          fflush(stdin);
2214 |          resOrdenaTempos (i, ptr_etapa);
2215 |          if ( ptr_etapa[i].realizada == 1) // se a etapa não esta realizada preenche com 0
2216 |              vencedores[i] = ptr_etapa[i].inscritos[0];
2217 |          else
2218 |              vencedores[i] = 0;
2219 |      }
```

```
2220 |  }
2221 |
2222 |
2223 |  void estNumVitorias ( int pilotos[][2], int vencedores[], ficha_t *ptr_piloto, conta_t *ptr_contador, char tipo) {
2224 |      /*  Preenche a matriz pilotos com o numero do aviao [][0] e numero de vitorias [][1],
2225 |          ordena pelo numero de vitorias e mostra ao utilizador.
2233 |
2234 |      int aux = 0, indice, i,j;
2235 |
2236 |
2237 |      for (i = 0; i < ptr_contador->pilotos; i++) { // copia os numeros de avioes para um vector temporario
2238 |          pilotos[i][0] = ptr_piloto[i].aviao;
2239 |          pilotos[i][1] = 0;
2240 |      }
2241 |      for (j = 0; j < ptr_contador->pilotos; j++) // percorre a matriz pilotos
2242 |          for (i = 0; i < MAX_ETAPAS; i++) // percorre o vector q tem os vencedores de todas as etapas (tinha contador-1)
2243 |              if ( vencedores[i] == pilotos[j][0] ) // se um dos vencedores é o pilotos[j]
2244 |                  pilotos[j][1]++; // aumenta o numero de vitorias na matriz
2245 |
2246 |      if ( tipo == 'D' ) {
2247 |          fflush(stdin);
2248 |          auxOrdenaMatriz ( pilotos, ptr_contador->pilotos, tipo); // vai ordenar
2249 |          for (i=0 ; i < ptr_contador->pilotos && aux < NUMTOPS; i++) {
2250 |              indice = pilIndiceAviao( pilotos[i][0], ptr_piloto, ptr_contador); // procura o indice do aviao pilotos[i][0]
2251 |              if ( pilotos[i][1] != 0 ) { // esconde os pilotos q n têm vitorias
2252 |                  printf("\n\t\t%2.d |  %.31s", pilotos[i][1], ptr_piloto[indice].nome);
2253 |                  aux++;
2254 |              }
2255 |          }
2256 |      } else
2257 |          if ( tipo == 'A' ) {
2258 |              fflush(stdin);
2259 |              auxOrdenaMatriz ( pilotos, ptr_contador->pilotos, tipo); // vai ordenar
2260 |              for (i=0 ; i < ptr_contador->pilotos; i++)
2261 |                  printf("\n -vitorias: o piloto %d tem %d victorias", pilotos[i][0], pilotos[i][1]);
2262 |          }
2263 |  }
2264 |
2265 |
2266 |  void estPctVictorias ( int pilotos[][2], ficha_t *ptr_piloto, conta_t *ptr_contador) {
2267 |      /*  Recebe a matriz pilotos com o numero do aviao [][0] e numero de vitorias [][1],
2268 |          Conta o número de vitórias e calcula a média com o número total de pilotos
2274 |
2275 |      int i, numpilotos, contavitorias = 0;
2276 |      float media;
2277 |
2278 |      for ( i = 0; i < ptr_contador->pilotos - 1; i++)
```

Bruno Tereso, 2091066@student.estg.ipleiria.pt
Marco Ferreira, 2092016@student.estg.ipleiria.pt

```
2279 |          if ( pilotos[i][1] != 0 ) // se 1 piloto tem uma ou mais vitorias
2280 |              contavitorias++;
2281 |
2282 |      numpilotos = ptr_contador->pilotos;
2283 |      media = (contavitorias*100) / numpilotos;
2284 |      printf("\n\n\t%.2f%% [ %d em %d ] dos pilotos tem pelo menos uma vitoria.", media, contavitorias, numpilotos );
2285 |  }
2286 |
2287 |
2288 |  void estEtapas ( int etapas[][2], etapa_t *ptr_etapa, conta_t *ptr_contador, char tipo) {
2289 |      /*  Preenche a matriz etapas com o numero do aviao [][0] e numero de inscritos [][1],
2290 |          ordena pelo numero de inscritos ascendente ou descendente e mostra ao utilizador.
2297 |
2298 |      int aux = 0, i,j;
2299 |
2300 |      for (i = 0; i < MAX_ETAPAS; i++) { // copia os numeros de avioes para um vector temporario
2301 |          etapas[i][0] = i;
2302 |          etapas[i][1] = ptr_etapa[i].totalinscritos;
2303 |      }
2304 |
2305 |      auxOrdenaMatriz ( etapas, MAX_ETAPAS, tipo); // vai ordenar
2306 |
2307 |      if ( tipo == 'D' )  // mais inscritos
2308 |          for (i=0 ; i < MAX_ETAPAS & aux < NUMTOPS; i++)
2309 |              if ( ptr_etapa[etapas[i][0]].data.ano != 0 ) {// esconde etapas vazias
2310 |                  aux++;
2311 |                  printf("\n\t\t%2d |  %.31s", etapas[i][1], ptr_etapa[etapas[i][0]].nome);
2312 |              } // ptr_etapa[etapas[i][0]].nome  -> etapas[i][0] é o indice da etapa em questão
2313 |
2314 |      if ( tipo == 'A' )  // menos inscritos
2315 |          for (i=0 ; i < MAX_ETAPAS && aux < NUMTOPS; i++)
2316 |              if ( ptr_etapa[etapas[i][0]].data.ano != 0 ) { // esconde etapas vazias
2317 |                  aux++;
2318 |                  printf("\n\t\t%2d |  %.31s", etapas[i][1], ptr_etapa[etapas[i][0]].nome);
2319 |              } // ptr_etapa[etapas[i][0]].nome  -> etapas[i][0] é o indice da etapa em questão
2320 |  }
2321 |
2322 |
2323 |  void estMediaParticipantes (etapa_t *ptr_etapa, conta_t *ptr_contador) {
2324 |      /*  Conta o número de etapas inseridas e os participantes de cada uma e
2325 |          mostra ao utilizador a média de pilotos por etapa.
2330 |
2331 |      int total = 0, aux = 0, i;
2332 |
2333 |      for (i=0 ; i < MAX_ETAPAS; i++)
2334 |          if ( ptr_etapa[i].data.ano !=0) {
2335 |              aux++; // encontrou uma etapa
```

Bruno Tereso, 2091066@student.estg.ipleiria.pt
Marco Ferreira, 2092016@student.estg.ipleiria.pt

```
2336 |              total += ptr_etapa[i].participantes;
2337 |          } // ptr_etapa[etapas[i][0]].nome  -> etapas[i][0] é o indice da etapa em questão
2338 |      printf("\n\n\tExiste um total maximo de %2d Participantes, distribuidos por %d Etapas", total, ptr_contador->etapas);
2339 |      printf("\n\tO que resulta numa Media de %.2f pilotos por etapa.", (float)total/aux);
2340 |  }
2341 |
2342 |
2343 |  void estVitoriasConsecutivas (int vencedores[], etapa_t *ptr_etapa, ficha_t *ptr_piloto, conta_t *ptr_contador) {
2344 |      /*  Analiza o vector vencedores e procura pilotos com 2 ou mais vitórias consecutivas.
2351 |
2352 |      int contatops = 0, aux = 0, i, indice;
2353 |      int vitorias[MAX_ETAPAS][2];
2354 |
2355 |      for ( i = 0; i < MAX_ETAPAS; i++)
2356 |          vitorias[i][1] = 0; // inicia o contador de vitorias a zero
2357 |      for ( i = 0; i < MAX_ETAPAS; i++) {
2358 |          if ( vencedores[i] != 0) {
2359 |              indice = pilIndiceAviao( vencedores[i], ptr_piloto, ptr_contador);
2360 |              if ( indice == vitorias[aux-1][0] && i!=0)
2361 |                  vitorias[aux-1][1]++;
2362 |              else {
2363 |                  vitorias[aux][0] = indice; // guarda o indice do piloto
2364 |                  vitorias[aux][1]++; // conta uma vitoria para o piloto
2365 |                  aux++; // auxiliar usado porque o i tem valores que não vamos usar na matriz ( quando salta etapas )
2366 |              }
2367 |
2368 |          }
2369 |      }
2370 |      auxOrdenaMatriz ( vitorias, aux, 'D'); // ordena a matriz criada, descencente
2371 |      printf("\n\n\tPilotos com maior numero de vitorias consecutivas:");
2372 |      for ( i = 0; i < aux && contatops++ < NUMTOPS; i++) // mostra top 3
2373 |          if ( vitorias[i][1] > 1 ) // esconde os que só têm uma vitória
2374 |              printf("\n\t\t%2d | %s", vitorias[i][1], ptr_piloto[vitorias[i][0]].nome);
2375 |  }
2376 |
2377 |
2378 |  void auxOrdenaMatriz (int vector[][2], int max, char tipo) { // ordena pelo segundo valor da matriz [][1]
2379 |      /*  Função auxiliar para ordenar matrizes com base no valor [][x].
2385 |
2386 |      int aux, aux2;
2387 |      int i, j, ha_trocas, posmenor;
2388 |
2389 |      fflush(stdin);
2390 |      if ( tipo == 'D')
2391 |          for ( i = 0; i < max && ha_trocas; i++) {
2392 |              ha_trocas = 0;
2393 |              for ( j = 0; j < max - i - 1; j++)
```

Bruno Tereso, 2091066@student.estg.ipleiria.pt
Marco Ferreira, 2092016@student.estg.ipleiria.pt

```
2394 |                  if ( vector[j][1] < vector[j+1][1]) { // Inverte para decrescemaxte
2395 |                      ha_trocas=1;
2396 |
2397 |                      aux = vector[j][0];
2398 |                      aux2 = vector[j][1];
2399 |
2400 |                      vector[j][0] = vector[j+1][0];
2401 |                      vector[j][1] = vector[j+1][1];
2402 |
2403 |                      vector[j+1][0] = aux;
2404 |                      vector[j+1][1] = aux2;
2405 |                  }
2406 |              }
2407 |          else
2408 |              if ( tipo == 'A')
2409 |                  for ( i = 0; i < max && ha_trocas; i++) {
2410 |                      ha_trocas = 0;
2411 |                      for ( j = 0; j < max - i - 1; j++)
2412 |                          if ( vector[j][1] > vector[j+1][1]) { // Inverte para decrescemaxte
2413 |                              ha_trocas=1;
2414 |
2415 |                              aux = vector[j][0];
2416 |                              aux2 = vector[j][1];
2417 |
2418 |                              vector[j][0] = vector[j+1][0];
2419 |                              vector[j][1] = vector[j+1][1];
2420 |
2421 |                              vector[j+1][0] = aux;
2422 |                              vector[j+1][1] = aux2;
2423 |                          }
2424 |                  }
2425 |
2426 |  }
2427 |
2428 |
2429 |  void fichGravar(conta_t *ptr_contador, etapa_t *ptr_etapa, ficha_t *ptr_piloto) {
2430 |      /*  Grava a informação num ficheiro binário
2436 |
2437 |      FILE *f;
2438 |      int num;
2439 |
2440 |      f = fopen( "dados.dat", "wb");
2441 |      if ( f == NULL )
2442 |          printf("Erro ao criar ficheiro...");
2443 |      else {
2444 |
2445 |          num = fwrite(ptr_contador, sizeof(conta_t), 1, f);
```

Bruno Tereso, 2091066@student.estg.ipleiria.pt
Marco Ferreira, 2092016@student.estg.ipleiria.pt

```
2446 |
2447 |          if ( num != 1  )  {
2448 |              printf("Erro de gravação...");
2449 |              auxPausa();
2450 |          } else {
2451 |              fwrite(ptr_piloto, sizeof(ficha_t), ptr_contador->pilotos , f);
2452 |              fwrite(ptr_etapa, sizeof(etapa_t), MAX_ETAPAS, f);
2453 |          }
2454 |      }
2455 |
2456 |      fclose(f);
2457 |      system("CLS");
2458 |      printf("\n\n\tInformacao guardada com sucesso...");
2459 |      ptr_contador->grava = 0;
2460 |      auxPausa();
2461 |  }
2462 |
2463 |
2464 |  ficha_t *fichLer(conta_t *ptr_contador, etapa_t *ptr_etapa) {
2465 |      /*  Le a informação de num ficheiro binário
2471 |
2472 |      FILE *f;
2473 |      int num;
2474 |      ficha_t *aux;
2475 |
2476 |      aux = NULL;
2477 |
2478 |      f = fopen( "dados.dat", "rb");
2479 |      if ( f == NULL )
2480 |          printf("Erro ao ler ficheiro...");
2481 |      else {
2482 |
2483 |          num = fread(ptr_contador, sizeof(conta_t), 1, f);
2484 |          if ( num != 1 )  {
2485 |              printf("Erro de leitura...");
2486 |              auxPausa();
2487 |          } else {
2488 |              aux = (ficha_t*)malloc((ptr_contador->pilotos)*sizeof(ficha_t));
2489 |              if ( aux == NULL )
2490 |                  printf("\n\tErro de Memoria...");
2491 |              else {
2492 |                  fread(aux, sizeof(ficha_t), ptr_contador->pilotos , f);
2493 |                  fread(ptr_etapa, sizeof(etapa_t), 10, f);
2494 |              }
2495 |          }
2496 |      }
2497 |      fclose(f);
```

Bruno Tereso, 2091066@student.estg.ipleiria.pt
Marco Ferreira, 2092016@student.estg.ipleiria.pt

```
2498 |      printf("\n\n\tInformacao recuperada com sucesso...");
2499 |      ptr_contador->grava = 0;
2500 |      auxPausa();
2501 |
2502 |      return aux;
2503 |  }
2504 |
2505 |
2506 |  char auxSair(conta_t *ptr_contador, ficha_t *ptr_piloto, etapa_t *ptr_etapa) {
2507 |      /*  Função auxiliar para perguntar ao utilizador se quer gravar antes de sair
2513 |
2514 |      char op;
2515 |
2516 |      if ( ptr_contador->grava == 0) // se não precisa de gravar, sai logo
2517 |          return SAIR;
2518 |
2519 |      do {
2520 |          system("CLS");
2521 |          puts(PROGRAMA);
2522 |          printf("\n\n\tNao gravou o seu trabalho, deseja:\n");
2523 |          printf("\n\tS - Sair sem gravar");
2524 |          printf("\n\tG - Gravar e Sair");
2525 |          printf("\n\tC - Cancelar e voltar ao menu principal");
2526 |          fflush(stdin);
2527 |          op = getch();
2528 |          op = toupper(op);
2529 |      } while (op != 'S' && op != 'G' && op != 'C');
2530 |
2531 |      if ( op == 'G') {
2532 |          fichGravar(ptr_contador, ptr_etapa, ptr_piloto);
2533 |          return SAIR;
2534 |      } else
2535 |          if ( op == 'C')
2536 |              return NAOSAIR;
2537 |          else
2538 |              return SAIR;
2539 |  }
2540 |
2541 |
2542 |  ficha_t *auxLer(conta_t *ptr_contador, ficha_t *ptr_piloto, etapa_t *ptr_etapa) {
2543 |      /*  Função auxiliar para perguntar ao utilizador se quer ler os dados à entrada
2549 |
2550 |      char op;
2551 |      FILE *f;
2552 |      int num;
2553 |
2554 |      f = fopen( "dados.dat", "rb");
```

Bruno Tereso, 2091066@student.estg.ipleiria.pt
Marco Ferreira, 2092016@student.estg.ipleiria.pt

```
2555 |
2556 |      if ( f != NULL ) {
2557 |          do {
2558 |              system ("CLS");
2559 |              puts(PROGRAMA);
2560 |
2561 |              printf("\n\n\tDeseja ler os dados armazenados?");
2562 |              printf("\n\t\tS - Sim");
2563 |              printf("\n\t\tN - Nao");
2564 |              fflush(stdin);
2565 |              op = getch();
2566 |              op = toupper(op);
2567 |          } while ( op != 'S' && op !='N');
2568 |
2569 |
2570 |          if ( op == 'S' )
2571 |              ptr_piloto = fichLer(ptr_contador, ptr_etapa);
2572 |          else
2573 |              auxIniciaValores(ptr_piloto, ptr_etapa, ptr_contador);
2574 |      } else
2575 |          auxIniciaValores(ptr_piloto, ptr_etapa, ptr_contador);
2576 |
2577 |      return ptr_piloto;
2578 |  }
2579 |
2580 |
2581 |  void auxIniciaValores(ficha_t *ptr_piloto, etapa_t *ptr_etapa, conta_t *ptr_contador) {
2582 |      /*  Função auxiliar para inicializar valores das estruturas a zero
2588 |
2589 |      int i, j;
2590 |
2591 |      ptr_contador->etapas = 0;
2592 |      ptr_contador->pilotos = 0;
2593 |      ptr_contador->etapa_corrente = 0;
2594 |      ptr_contador->grava = 0;
2595 |
2596 |      for (i = 0; i < MAX_ETAPAS; i++) {
2597 |          for (j = 0;j < MAX_PILOTOS_ETAPA;j++) {
2598 |              ptr_etapa[i].inscritos[j] = 0;
2599 |              ptr_etapa[i].tempos[j] = 0;
2600 |              ptr_etapa[i].pontos[j] = 0;
2601 |          }
2602 |          ptr_etapa[i].totalinscritos = 0;
2603 |          ptr_etapa[i].data.dia = 0;
2604 |          ptr_etapa[i].data.mes = 0;
2605 |          ptr_etapa[i].data.ano = 0;
2606 |          ptr_etapa[i].ordem = i + 1; // inicia logo as posições todas
```

Bruno Tereso, 2091066@student.estg.ipleiria.pt
Marco Ferreira, 2092016@student.estg.ipleiria.pt

```
2607 |          ptr_etapa[i].participantes = 0;
2608 |          strcpy(ptr_etapa[i].nome, "\0");
2609 |          ptr_etapa[i].realizada = 0;
2610 |      }
2611 |  }
```

Bruno Tereso, 2091066@student.estg.ipleiria.pt
Marco Ferreira, 2092016@student.estg.ipleiria.pt

Software Utilizado

Dev-C ++ | http://www.bloodshed.net/
Compilador onde o programa foi desenvolvido e testado.

Microsoft Visual Studio 2008 | http://www.microsoft.com/exPress/
Compilador secundário usado para impressões

UniversalIndentGUI | http://universalindent.sourceforge.net/
Software de indentação automática, usado para limpar espaços no código e facilitar a importação do código para o Microsoft Word.

Bibliografia

Aulas Teórico-Práticas (TP) | Materiais de Apoio
PDFs Programação I - EI - 2009/10 Prof. Vítor Távora

Cplusplus | http://www.cplusplus.com/
Biblioteca online com informação diversificada sobre linguagem C/C++.

Bruno Tereso, 2091066@student.estg.ipleiria.pt
Marco Ferreira, 2092016@student.estg.ipleiria.pt