



INSTITUTO POLITÉCNICO
DO CÁVADO E DO AVE
ESCOLA SUPERIOR DE TECNOLOGIA

RELATÓRIO DE TRABALHO PRÁTICO I

Linguagem de Programação I

LUCAS BRGA MENDONÇA

ALUNO Nº 17870

Trabalho realizado sob a orientação de:
Luís Ferreira

Linguagens de Programação I

Licenciatura em Engenharia de Sistemas Informáticos

Barcelos, Janeiro de 2021

Índice

1	INTRODUÇÃO	1
2	ESTRUTURA DO PROJETO E DESENVOLVIMENTO	3
2.1	Associação entre as estruturas	7
2.2	Output do projeto	8
3	CONCLUSÃO	10

Lista de Figuras

Figura 1: Arquivos do projeto	3
Figura 2: Bibliotecas C utilizadas	3
Figura 3: Exemplo de alocação dinâmica e atribuição de valores	6
Figura 4: Exemplo de tratamento de ficheiro	7
Figura 5: Exemplo de transformação de uma lista dinâmica para um vetor não dinâmico	7
Figura 6: Estruturas de dados e suas ligações depois do carregamento do ficheiro	8
Figura 7: Output final do programa	8
Figura 8: Ficheiro resultados.txt	9

1 Introdução

O presente trabalho tem como objetivo principal sedimentar os conhecimentos introduzidos nas aulas da unidade curricular de Programação I, especificamente na linguagem C. O sistema desenvolvido é capaz de auxiliar no processamento de dados dos concorrentes de uma prova de Rally e devolver o resultado de diversos cálculos sobre o seu conteúdo.

Para a entrega do presente projeto, foram definidas as principais estruturas, assim como o relacionamento entre estas mesmas estruturas.

O código está implementado de forma que as estruturas e ficheiros do projeto sejam facilmente identificadas e de forma que utilize alguns dos conceitos fundamentais vistos até hoje nas aulas de Linguagem de Programação I, como a criação de bibliotecas. O presente relatório, tal como o código completo do projeto está disponível no GitHub, através do link [17870_LP1.git](https://github.com/17870-LP1).

2 Estrutura do projeto e desenvolvimento

Para uma melhor disposição e estruturação dos dados, o projeto foi dividido da seguinte maneira:

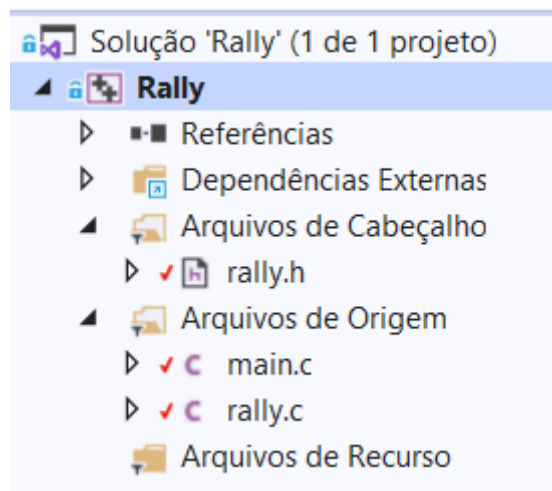


Figura 1: Arquivos do projeto

```
<stdbool.h>
<stdio.h>
<string.h>
<stdlib.h>
```

Figura 2: Bibliotecas C utilizadas

rally.h

Possui a definição das estruturas a se utilizar no projeto, assim como a definição de métodos e constantes globais.

Estruturas

a) **Etapa**

Etapa realizada por um concorrente em uma determinada prova, com nome de início da etapa, fim, distância percorrida, tempo em que foi percorrida pelo concorrente e um apontador para a próxima etapa efetuada;

```
1. typedef struct etapa {
2.     char inicio[3];
3.     char fim[3];
4.     float distancia;
5.     int tempo;
6.     struct etapa* next;
7. } Etapa;
```

b) **Concorrente**

Guarda os dados do concorrente, como seu nome, nome do carro, primeira etapa realizada, ultima etapa realizada e quantidade de etapas realizadas numa determinada prova;

```
1. typedef struct concorrente {
2.     int id;
3.     char nome[50];
4.     char carro[10];
5.     Etapa* etapa;
6.     Etapa* etapa_ult;
7.     int qtdEtapas;
8. } Concorrente;
```

c) **InfoCorrida**

Guarda informações relativas ao concorrente uma corrida, ou seja, possui uma estrutura de dados de Concorrente, o número de etapas que a corrida possui, tempo total de corrida do concorrente, distancia total que o concorrente percorreu, assim como sua velocidade media ao longo da corrida. Ainda, há um apontador para outra informação de corrida de outro concorrente;

```
1. typedef struct corrida {
2.     Concorrente concorrente;
3.     int num_etapas;
4.     int tempoTotal;
5.     float distanciaTotal;
6.     float velocidadeMedia;
7.     struct corrida* next;
8. } InfoCorrida;
```

d) **Prova**

Guarda um apontador para a informação de um concorrente em uma corrida (início da lista), assim como a quantidade total de concorrentes que participaram daquela prova;


```

1. typedef struct prova {
2.     InfoCorrida* corrida; //inicio da lista
3.     int quantidadeConcorrentes;
4. } Prova;

```

e) MediaEtapa

Guarda informações de forma agregada sobre uma determinada etapa;

```

1. //contém informações sobre as etapas e suas medias
2. typedef struct mediaEtapa {
3.     char ini[3]; //inicio da etapa
4.     char fim[3]; //fim da etapa
5.     int tempoTotal; //tempo total percorrido por todos os concorrentes na et
    apa
6.     int cont; //quantas vezes a etapa foi percorrida
7.     float media; //media da etapa
8.     int tempoMinimo; //Menor tempo feito na etapa
9. } MediaEtapa;

```

Além disso, possuí as seguintes principais funções definidas:

```

1. /**
2.     Processa ficheiro de informações sobre uma corrida
3. */
4. extern int processaCorrida();
5.
6. /**
7.     Processa as etapas que existem em uma corrida, assim como suas distancias t
    otais
8. */
9. extern int processaEtapa();
10.
11. /**
12.     Processa os concorrentes com provas validas ou não
13. */
14. extern int processaConcorrentes();
15.
16. /**
17.     Retorna o resultado do carregamento efetuado
18. */
19. extern InfoCorrida* getResult();
20.
21. /**
22.     Busca quantidade de concorrentes.
23. */
24. int getQtdConcorrentes();
25.
26. /**
27.     Quantidade de concorrentes com provas validas.
28. */
29. int getConcProvaVal();
30.
31. /**
32.     Apresentacao do concorrente mais rapido / mais lento a efetuar uma prova va
    lida.
33. */
34. extern void displayListTempoProva();
35.
36. /**
37.     Calculo das medias dos tempos por etapa e ordenado por ocorrencia.
38. */
39. extern void calcMediaEtapa();

```

```

40.
41. /**
42.     Apresentação do concorrente mais rápido / mais lento a efetuar uma prova vá
        lida.
43. */
44. extern void displayRapidoLento();
45.
46. /*
47.     Cálculo do menor tempo em que é possível efetuar a prova na totalidade, ou
        seja, soma dos
48.     tempos mínimos por etapa independentemente de terem sido efetuados por conc
        orrentes com
49.     provas válidas ou não;
50. */
51. extern int displayMinTempProva();
52.
53. /*
54.     Listagem das velocidades médias de toda a prova, ordenada por ordem decresc
        ente.
55.     Considere apenas as velocidades de concorrentes que efetuaram uma prova vál
        ida.
56. */
57. extern void calcVelocidadeMedia();
58.
59. /*
60.     Geração da tabela classificativa da prova, onde constem os seguintes campos
        de informação:
61.     posição na prova, número do concorrente, tempo total de prova, diferença pa
        ra o concorrente
62.     anterior, diferença para o líder. Os concorrentes desclassificados deverão
        constar no final da
63.     tabela, ordenados por ordem crescente do seu número.
64. */
65. extern void displayTabela();

```

rally.c

Possui a implementação das funções definidas em rally.h com a utilização de alguns conceitos vistos nas aulas, como por exemplo, vetores, estruturas (ligadas ou não), alocação de memória dinâmica e não dinâmica, tratamento de ficheiros (leitura e escrita), assim como o desenvolvimento de toda a relação efetuada entre as estruturas descritas acima.

```

Etapa* newEtapa = (Etapa*)malloc(sizeof(Etapa));
strcpy(newEtapa->inicio, resultado[1]);
strcpy(newEtapa->fim, resultado[2]);
newEtapa->tempo = atoi(resultado[3]);
newEtapa->next = NULL;

```

Figura 3: Exemplo de alocação dinâmica e atribuição de valores

```
//Tratamento do arquivo*/
FILE* arq;
arq = fopen("corrida.txt", "r");
if (arq == NULL) {
    printf("Nao foi possivel abrir o arquivo!\n");
    return 1;
}
else {
    while (fgets(linha, sizeof(linha), arq) != NULL) {
```

Figura 4: Exemplo de tratamento de ficheiro

```
//lista
InfoCorrida* current = prova.corrida;

//array
InfoCorrida resultado[100] = {NULL};
int tam = 0;
while (current) {
    //checar se é valido
    if(current->num_etapas == current->concorrente.qtdEtapas && current->num_etapas !=0)
        resultado[tam++] = *current;
    current = current->next;
}
```

Figura 5: Exemplo de transformação de uma lista dinâmica para um vetor não dinâmico

main.c

Possui a chamada das funções criadas pela ordem em que encontram-se no enunciado do trabalho. Aqui, de forma a facilitar o desenvolvimento, não há nenhuma interação com o utilizador já que o que se pede está bem definido.

2.1 Associação entre as estruturas

A ligação entre as estruturas foi feita de forma a facilitar todo o processo de conexão entre as mais variadas informações obtidas pelo programa. Logo abaixo, é apresentada uma imagem retirada em debug de como os dados ficam estruturados ao longo do programa:

Nome	Valor	Tipo
prova.corrida	0x00dde798 (concorrente=[id=2 nome=0x00dde79c "Maria" carro=0x00dde7ce "Subaru" ...] num_etapas=3 tempoTot... corrida *	
concorrente	(id=2 nome=0x00dde79c "Maria" carro=0x00dde7ce "Subaru" ...)	concorrente
id	2	int
nome	0x00dde79c "Maria"	char[50]
carro	0x00dde7ce "Subaru"	char[10]
etapa	0x00dd61b0 (inicio=0x00dd61b0 "E1" fim=0x00dd61b3 "E2" distancia=32.2299995 ...)	etapa *
inicio	0x00dd61b0 "E1"	char[3]
fim	0x00dd61b3 "E2"	char[3]
distancia	32.2299995	float
tempo	21672	int
next	0x00ddea58 (inicio=0x00ddea58 "P" fim=0x00ddea5b "E1" distancia=23.1000004 ...)	etapa *
inicio	0x00ddea58 "P"	char[3]
fim	0x00ddea5b "E1"	char[3]
distancia	23.1000004	float
tempo	12383	int
next	0x00de01a0 (inicio=0x00de01a0 "E2" fim=0x00de01a3 "C" distancia=25.7199993 ...)	etapa *
inicio	0x00de01a0 "E2"	char[3]
fim	0x00de01a3 "C"	char[3]
distancia	25.7199993	float
tempo	23567	int
next	0x00000000 <NULL>	etapa *
etapa_ult	0x00de01a0 (inicio=0x00de01a0 "E2" fim=0x00de01a3 "C" distancia=25.7199993 ...)	etapa *
qtdEtapas	3	int
num_etapas	3	int
tempoTotal	57622	int
distanciaTotal	81.0500031	float
velocidadeMedia	0.00140658091	float
next	0x00dde828 (concorrente=[id=1 nome=0x00dde82c "Joao" carro=0x00dde85e "Subaru" ...] num_etapas=3 tempoTot... corrida *	
concorrente	(id=1 nome=0x00dde82c "Joao" carro=0x00dde85e "Subaru" ...)	concorrente
id	1	int
nome	0x00dde82c "Joao"	char[50]
carro	0x00dde85e "Subaru"	char[10]
etapa	0x00dde8b8 (inicio=0x00dde8b8 "P" fim=0x00dde8bb "E1" distancia=23.1000004 ...)	etapa *
inicio	0x00dde8b8 "P"	char[3]
fim	0x00dde8bb "E1"	char[3]
distancia	23.1000004	float
tempo	10501	int
next	0x00dde988 (inicio=0x00dde988 "E1" fim=0x00dde98b "E2" distancia=32.2299995 ...)	etapa *
etapa_ult	0x00de0360 (inicio=0x00de0360 "E2" fim=0x00de0363 "C" distancia=25.7000008 ...)	etapa *
qtdEtapas	3	int
num_etapas	3	int
tempoTotal	76169	int
distanciaTotal	81.0299988	float
velocidadeMedia	0.00106381858	float
next	0x00de08d0 (concorrente=[id=3 nome=0x00de08d4 "Joana" carro=0x00de0906 "BMW" ...] num_etapas=0 tempoTot... corrida *	

Figura 6: Estruturas de dados e suas ligações depois do carregamento do ficheiro

2.2 Output do projeto

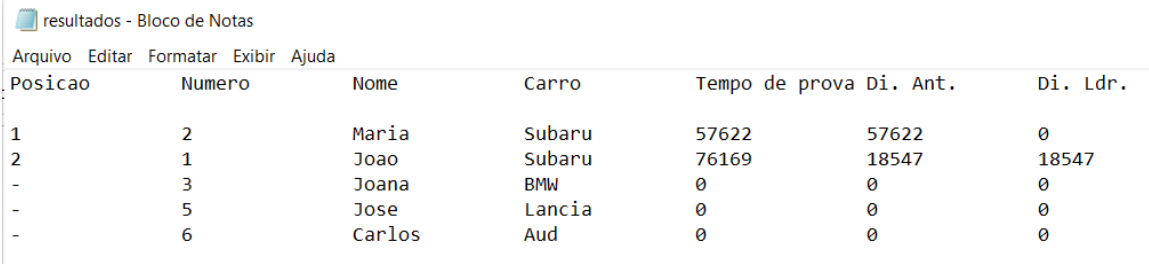
O output do projeto pode ser visto a seguir:

```

1. Dados carregados para os concorrentes:
Maria
Joao
Joana
Jose
Carlos
2. Quantidade de concorrentes: 5
3. Quantidade de concorrentes com provas validas: 2
4. Listagem, ordenada por ordem decrescente de tempo da prova, de todos os concorrentes que efetuaram uma prova valida.
Concorrente Joao          Tempo de prova: 76169
Concorrente Maria         Tempo de prova: 57622
5. Calculo das medias dos tempos por etapa e ordenado por ocorrencia.
Inicio: P Fim: E1 Total: 22884 Media 11442.00
Inicio: E1 Fim: E2 Total: 58875 Media 29437.00
Inicio: E2 Fim: C Total: 52032 Media 26016.00
6. Apresentacao do concorrente mais rapido / mais lento a efetuar uma prova valida.
Mais rapido: Maria com 57622 ms
Mais lento: Joao com 76169 ms
7. Calculo do menor tempo em que e possivel efetuar a prova na totalidade
55740 ms
8. Listagem das velocidades medias de toda a prova, ordenada por ordem decrescente
Concorrente Maria          Tempo de prova: 57622          ms com velocidade media: 0.00140658 km/ms
Concorrente Joao           Tempo de prova: 76169          ms com velocidade media: 0.00106382 km/ms
9. Geracao da tabela classificativa da prova.
Posicao      Numero      Nome      Carro      Tempo de prova Di. Ant.      Di. Ldr.
1            2          Maria     Subaru     57622       57622       0
2            1          Joao      Subaru     76169       18547       18547
-            3          Joana     BMW         0           0           0
-            5          Jose      Lancia     0           0           0
-            6          Carlos    Aud         0           0           0

```

Figura 7: Output final do programa



resultados - Bloco de Notas

Arquivo Editar Formatar Exibir Ajuda

Posicao	Numero	Nome	Carro	Tempo de prova	Di. Ant.	Di. Ldr.
1	2	Maria	Subaru	57622	57622	0
2	1	Joao	Subaru	76169	18547	18547
-	3	Joana	BMW	0	0	0
-	5	Jose	Lancia	0	0	0
-	6	Carlos	Aud	0	0	0

Figura 8: Ficheiro resultados.txt

3 Conclusão

Conclui-se que as structs definem tipos de dados que agrupam variáveis sob um mesmo tipo de dados. No presente trabalho, as estruturas foram desenvolvidas de modo a criar interações entre si e facilitar o processamento, busca e o resultado dos dados inseridos.

Funcionalidades adicionais poderão ser criadas para melhorias futuras de forma a facilitar mais ainda a organização do código e todo o processamento dos dados. Tanto o código como os arquivos gerados pelo DoxyGen encontram-se no link do git mencionado na introdução.