

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](#_Toc61634546)

[2 Estrutura do projeto e desenvolvimento 3](#_Toc61634547)

[2.1 Associação entre as estruturas 7](#_Toc61634548)

[2.2 Output do projeto 8](#_Toc61634549)

[3 Conclusão 10](#_Toc61634550)

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 tranformação 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

# 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 diveros 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 fácilmente 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/lucasbmendonca/Rally).

# Estrutura do projeto e desenvolvimento

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

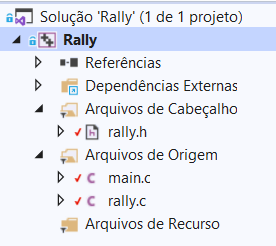


Figura 1: Arquivos do projeto

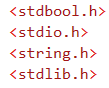


Figura 2: Bibliotecas C utilizadas

***rally.h***

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

Estruturas

1. **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;
2. **typedef** **struct** etapa {
3. **char** inicio[3];
4. **char** fim[3];
5. **float** distancia;
6. **int** tempo;
7. **struct** etapa\* next;
8. } Etapa;
9. **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;
10. **typedef** **struct** concorrente {
11. **int** id;
12. **char** nome[50];
13. **char** carro[10];
14. Etapa\* etapa;
15. Etapa\* etapa\_ult;
16. **int** qtdEtapas;
17. } Concorrente;
18. **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;
19. **typedef** **struct** corrida {
20. Concorrente concorrente;
21. **int** num\_etapas;
22. **int** tempoTotal;
23. **float** distanciaTotal;
24. **float** velocidadeMedia;
25. **struct** corrida\* next;
26. } InfoCorrida;
27. **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;

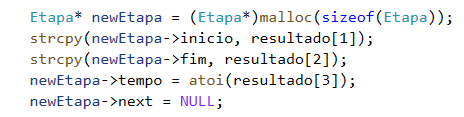
1. **typedef** **struct** prova {
2. InfoCorrida\* corrida; //inicio da lista
3. **int** quantidadeConcorrentes;
4. } Prova;
5. **MediaEtapa**  
   Guarda informações de forma agregada sobre uma determinada etapa;
6. //contém informações sobre as etapas e suas medias
7. **typedef** **struct** mediaEtapa {
8. **char** ini[3]; //inicio da etapa
9. **char** fim[3]; //fim da etapa
10. **int** tempoTotal; //tempo total percorrido por todos os concorrentes na etapa
11. **int** cont; //quantas vezes a etapa foi percorrida
12. **float** media; //media da etapa
13. **int** tempoMinimo; //Menor tempo feito na etapa
14. } MediaEtapa;

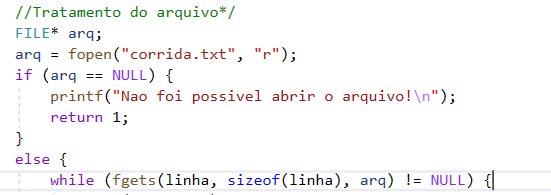
**Além disso, possui as seguintes principais funções definidas:**

1. /\*\*
2. Processa ficheiro de informações sobre uma corrida
3. \*/
4. **extern** **int** processaCorrida();
6. /\*\*
7. Processa as etapas que existem em uma corrida, assim como suas distancias totais
8. \*/
9. **extern** **int** processaEtapa();
11. /\*\*
12. Processa os concorrentes com provas validas ou não
13. \*/
14. **extern** **int** processaConcorrentes();
16. /\*\*
17. Retorna o resultado do carregamento efetuado
18. \*/
19. **extern** InfoCorrida\* getResult();
21. /\*\*
22. Busca quantidade de concorrentes.
23. \*/
24. **int** getQtdConcorrentes();
26. /\*\*
27. Quantidade de concorrentes com provas validas.
28. \*/
29. **int** getConcProvaVal();
31. /\*\*
32. Apresentacao do concorrente mais rapido / mais lento a efetuar uma prova valida.
33. \*/
34. **extern** **void** displayListTempoProva();
36. /\*\*
37. Calculo das medias dos tempos por etapa e ordenado por ocorrencia.
38. \*/
39. **extern** **void** calcMediaEtapa();
41. /\*\*
42. Apresentação do concorrente mais rápido / mais lento a efetuar uma prova válida.
43. \*/
44. **extern** **void** displayRapidoLento();
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 concorrentes com
49. provas válidas ou não;
50. \*/
51. **extern** **int** displayMinTempProva();
53. /\*
54. Listagem das velocidades médias de toda a prova, ordenada por ordem decrescente.
55. Considere apenas as velocidades de concorrentes que efetuaram uma prova válida.
56. \*/
57. **extern** **void** calcVelocidadeMedia();
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 para 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.

  
Figura 3: Exemplo de alocação dinâmica e atribuição de valores

  
Figura 4: Exemplo de tratamento de ficheiro

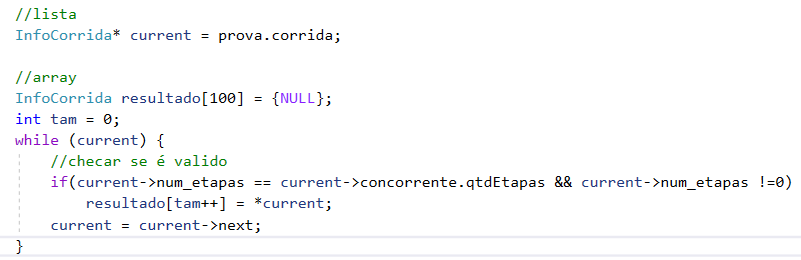


Figura 5: Exemplo de tranformação de uma lista dinâmica para um vetor não dinâmico

***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.

## 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:

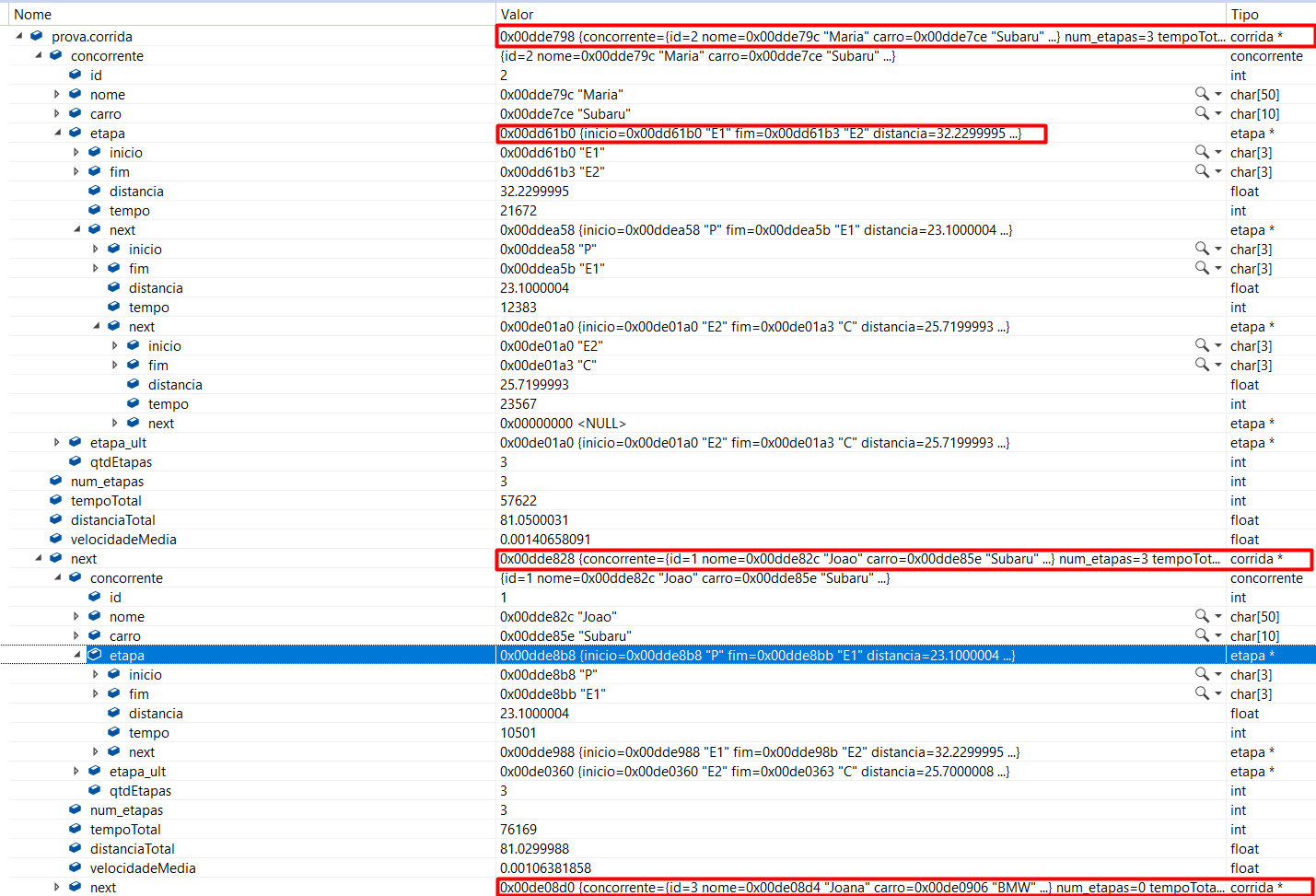


Figura 6: Estruturas de dados e suas ligações depois do carregamento do ficheiro

## Output do projeto

O output do projeto pode ser visto a seguir:

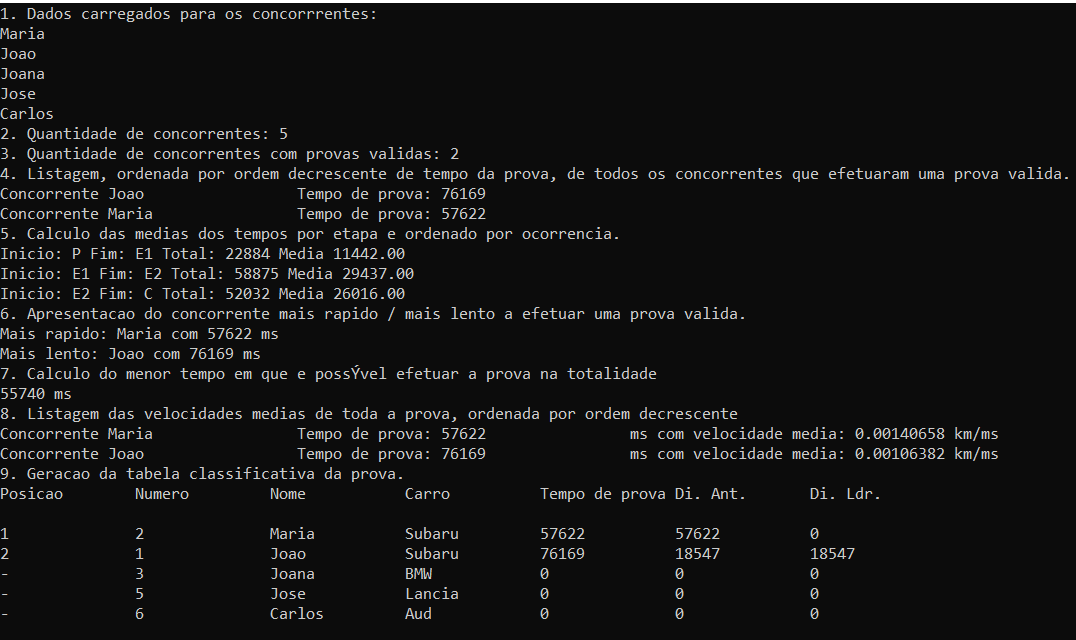


Figura 7: Output final do programa

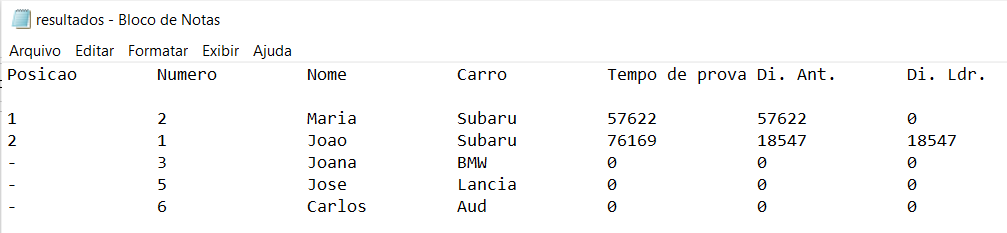


Figura 8: Ficheiro resultados.txt

# 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 faciltar 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.