PROJETO FINAL

Trabalho realizado por:

Duarte Domingues (99925)

Tiago Costa (100094)

Disciplina: Programação

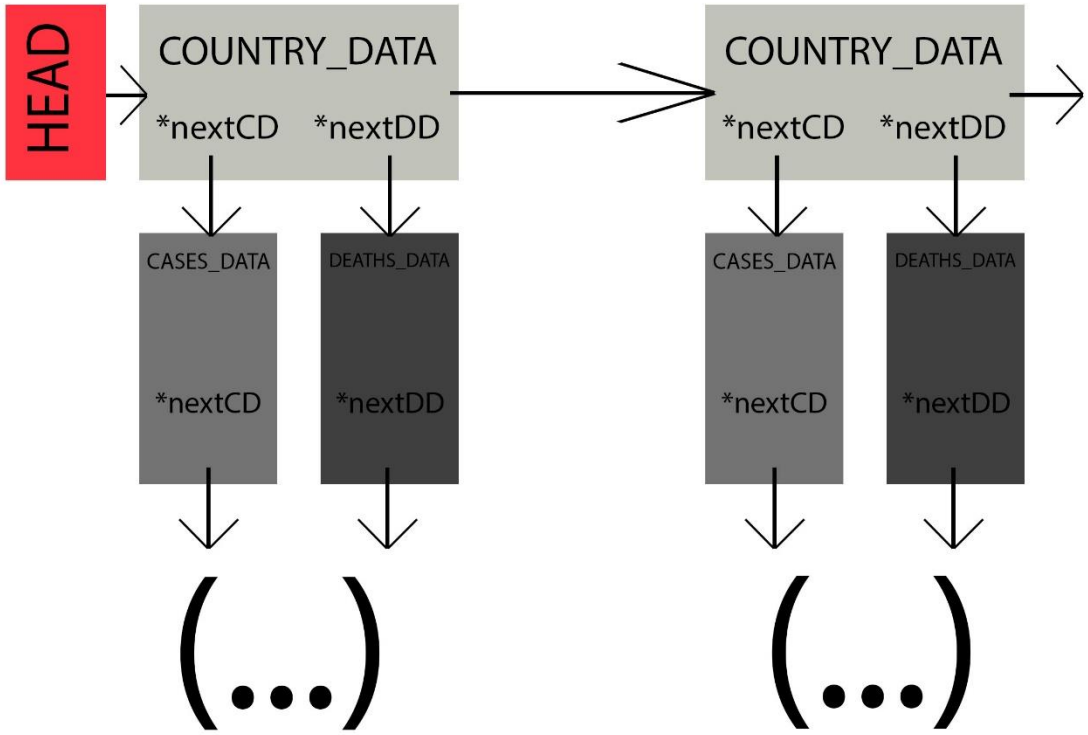
Docente Responsável: Nuno Horta

1. Introdução

Neste relatório descreveremos a estrutura do nosso programa, a sua estrutura de dados e alguns conceitos-chave para a sua compreensão. Aqui abordaremos as suas funções principais, a forma como tornámos eficiente a nossa estrutura de dados e de que modo pensamos ter atingido os objetivos de economia de memória propostos.

2. Estrutura de Dados

O nosso programa possui 3 estruturas diferentes. Uma principal que possui duas secundárias.



Como podemos observar pelo diagrama acima, o nosso programa possui uma lista principal que guarda os dados fixos relativos aos países. Depois, para cada nó da lista principal, i.e. para cada país, os dados relativos a mortes desse mesmo país serão guardados numa lista secundária para informação sobre casos e os dados relativos a casos desse país serão guardados numa segunda lista secundária.

Porquê?

Desde já queríamos esclarecer esta nossa escolha. Reparámos que grande parte das restrições/seleções/ordenações etc. se debruçavam sobre OU mortos OU infetados. Vimos aqui uma oportunidade perfeita para começar a trabalhar para a eficiência da estrutura de dados, não se chegando sequer a escrever dados de um ou de outro se alguma especificação o impedir, economizando quase metade da memória/estrutura de dados apenas com esta pequena "feature".

3. Estrutura do Código

3.1 As 3 funções fundamentais

Feita a descrição e explicação da nossa estrutura de dados passaremos agora à explicação do da estrutura do programa em si. O programa possui 3 funções principais **country_data* read_file(), country_data* data_structure_placement() e void write_file()**. Os nomes são intuitivos, não obstante, a função read_file() lê qualquer ficheiro no formato especificado pelo utilizador desde que este seja válido (.dat ou .csv). Esta função encarrega-se ainda de chamar a função data_structure_placement() de modo a criar a estrutura e ir inserindo os respetivos dados à medida que os vai lendo. Por fim, quando findar o processo de construção da estrutura de dados (passando primeiro pela seleção/restrição e depois pela ordenação - abordaremos mais à frente a nossa abordagem a estes detalhes) é escrita toda a estrutura num ficheiro do formato .csv ou .dat, com recurso à função write_file().

3.2 Funções int selection(), int restriction() e country_data* order()

Esta secção dedica-se à explicação da implementação destas 3 restrições. Começar por dizer que a restrição de leitura foi simplesmente cumprida por meio de uma condição if à entrada da função read, apenas permitindo a construção das estruturas de dados que cumprissem a condição de leitura especificada pelo utilizador. Quanto ao selection() e restriction(), estas duas funções funcionam como flags. Estas funções funcionam como entrave à criação da estrutura de dados, sendo apenas esta criada se o valor destas funções não se opuser, ou seja, se o valor delas for 0. Importante realçar que a não especificação de uma destas opções levará a respetiva função a assumir o valor 0 de modo que esta não interfira com o funcionamento da outra. Assim introduzimos um dos nossos princípios de economia de memória e de eficiência estrutural em termos de dados: **apenas são criadas as estruturas estritamente necessárias para satisfazer a especificação de análise do utilizador – qualquer estrutura que pudesse vir a ser eliminada nem chega a ser criada.** Isto permite manter a lista tão reduzida quanto possível e minimizar um conjunto de erros associados à alocação de grandes dimensões de memória. A void order() edita a lista depois de já não ser adicionada nenhuma estrutura à estrutura de dados, reordenando segundo a especificação do utilizador.

3.3 Funções Acessórias

Todas as voids/funções não citadas são voids/funções em tudo semelhante às lecionadas tendo como função a criação e inserção de estruturas na(s) lista(s).

4. Escrita/Leitura de ficheiro binário (.dat)

Para escrevermos o nosso ficheiro em binário começamos por escrever o número de estruturas da lista principal. Depois escrevemos, **para cada nó da lista principal,** o número de estruturas cases_data e o número de estruturas do tipo deaths_data. Estes valores permitem a criação do número de estruturas necessárias. A partir daqui a única regra que seguimos é que sempre que escrevemos uma char *, antes colocamos a sua dimensão para que a leitura seja feita sem problemas, do número correto de bytes.

A leitura do ficheiro criado é precisamente o mecanismo inverso, alocando o número de estruturas de dados necessárias e depois a leitura das variáveis é feita tendo em atenção o preciso facto que referimos e precisámos em relação a char *.

5. Conclusão

Consideramos que o objetivo principal foi atingido. Desenvolvemos uma estrutura de dados robusta e económica utilizando diversos conceitos abordados na vertente teórica da UC de Programação. Esta estrutura está capaz de organizar um número considerável de elementos segundo um vasto leque de opções, tendo ainda uma opção de fazer uma conversão/transcrição de binário para .csv e vice-versa.