TÉCNICO LISBOA

Projeto Final - Pandemia

Relatório de Programação

G002
Projeto realizado por:
- Rita Mendes, ist1100072 MEEC
- Tomás Martins, ist1100102 MEEC

Ano letivo 20/21 - 2º Semestre

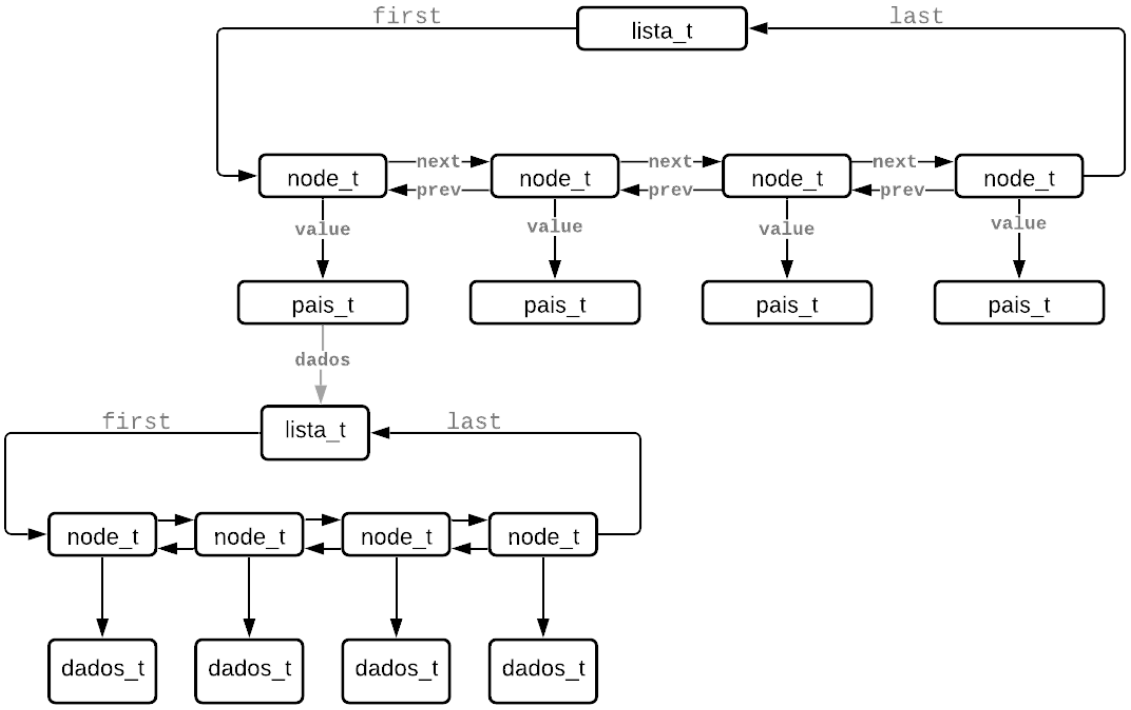
Este ano, o projeto final consistiu na análise e organização de diversos dados relacionados com a evolução da pandemia a nível mundial.

Como pedido no enunciado, separámos todas as structs e headers de funções para outro ficheiro, *headers.h* e também separámos o programa em vários ficheiros *.c,* um para tudo o que envolve as restrições, ordenações e seleções, outro para o que envolve a criação e leitura de ficheiros, outro que envolve listas e, por fim, o *main.* Assim, garantimos uma melhor estruturação do código.

Tal como no projeto intermédio, optámos pelo uso de várias structs para guardar vários tipos de variáveis diferentes. A struct *pais_t* contém os dados fixos de cada país, a *dados_t* contém os dados variáveis, a *yearWeek_t* contém a semana e o ano, facilitando a conversão e interpretação de datas, a lista_t consiste numa doubly linked list que usámos para guardar listas de países e de dados, a *node_t* que representa um elemento da *lista_t* (tanto dados, como países), e, por fim, a struct *settings_t* está associada a vários *enums* que tratam da parte dos argumentos na linha de comando. Dentro desta última *struct*, utilizámos *enums* para poder limitar o valor de uma variável a um conjunto restrito de opções, de modo a facilitar a organização das diferentes variáveis que foram utilizadas ao longo do programa.

Para este projeto, decidiu-se usar *doubly linked lists* depois de terem sido encontradas algumas limitações ao usar *single linked lists*. A maior destas limitações é o facto da lista apenas “andar” num sentido, pelo que se quiséssemos retirar um elemento do meio da lista, precisaríamos de ter um pointer para o elemento que o antecedia. Numa lista singular, essa tarefa seria mais complicada. Em *doubly linked lists*, temos um *pointer* para o próximo elemento (*next*) e um *pointer* para o elemento anterior (*prev*), facilitando assim a implementação de várias funções ao longo do código.

No esquema abaixo, podemos visualizar melhor a relação entre cada lista. As structs *settings_t* e *yearWeek_t* não apresentam uma ligação direta e, por isso, não estão representadas:



Como a nossa estrutura *lista_t* pode guardar qualquer tipo de estrutura (dado que estamos a utilizar *void pointers*), precisámos de passar como argumento, funções que fizessem *free* da estrutura correspondente eliminando os elementos dessa lista independentemente de serem dados ou países (nomeadamente, nas funções **apagar_elemento_lista** e **liberta_lista**).

Foi utilizada uma função **compare_fn** com argumentos (*void**), para comparar nas diferentes seleções existentes, dois *nodes*, um que indica o *node* atual, e um outro de comparação, esta retorna um inteiro que depois vai ser utilizado para decidir o que fazer aos *nodes* comparados.

De forma a preservar a legibilidade do código e obedecer a uma boa prática da programação, não se usaram variáveis globais. Em vez disso, optou-se por passar as variáveis necessárias como argumentos de funções ou dentro de estruturas passadas por *pointers* (nomeadamente as estruturas *settings_t* e *lista_t*). Pelo mesmo motivo, tentamos dividir ao máximo as funcionalidades do programa em inúmeras funções.

Para o ficheiro binário, usámos o seguinte formato: Consoante o número de países, guardamos a informação de cada país (tendo em conta que todas as strings são precedidas de um inteiro que indica o seu tamanho, incluindo o “\0”). A mesma lógica é usada para os dados de cada país, primeiro vemos o seu tamanho e só depois é que o guardamos.

Para a leitura dos argumentos da linha de comando, foi utilizada a biblioteca “getopt.h”, pois esta facilita bastante a leitura destes argumentos. A cada elemento lido (por exemplo -L), era guardado para uma string o seu valor, que depois era enviado, em algumas situações, para uma função que metia na *struct settings_t* o seu valor. Para a leitura de elementos com espaços que a função *getopt* por si não verifica, foi utilizado a variável *optind*, que é o *index* do próximo elemento a ser processado no *argv*, inicializado a 1, assim ao ler o *optarg* que indica a palavra seguinte à indicação do “-”, o *optind* vê o elemento seguinte, por exemplo na indicação de *-S inf “data”*, o *argv[optind]* possui a palavra “data”. Com a verificação de erros que o utilizador poderia colocar na linha de comando, a função “argumentos” ficou bastante extensa, então resolvemos passar a leitura do -L, -S, -D e -P para funções à parte de modo a diminuir o tamanho da anterior.

Em relação aos erros possíveis na leitura dos argumentos da linha de comandos, tentamos verificar o máximo de erros possíveis, se havia introdução de -i e -o, dois elementos obrigatórios, se as opções de ordenação, seleção e restrição possuíam valores corretos para a sua utilização, entre outros.

Foram também verificados erros na leitura do ficheiro, como números onde apenas deveriam estar presentes letras, ou vice-versa, se o número de colunas está correto para todas as linhas, valores negativos ou semanas impossíveis. Ao ocorrer qualquer tipo de erro, tentamos ao máximo dar *free* de tudo o que tinha sido criado até ao momento do erro, de modo a não haver qualquer *leak* de memória.