Relatório

Mini-Projecto PPP

23 de maio de 2024

Vasco Alves (2022228207)
Miguel Silva (2023221244)

Conteúdo

1 Introdução

1.1 Objetivo

O objectivo deste trabalho é construir uma aplicação capaz de armazenar, alterar e gerir a informação dos doentes de um médico hipotético. Esta informação inclui: nome, data de nascimento, cartão de cidadão, telefone e email. Para além desta funcionalidade deve ser capaz de armazenar os seus dados clínicos. Estes são: tensão arterial mínima, tensão arterial máxima, peso e altura.

1.2 Funcionalidades

É necessário um menu através do qual o utilizador interage com todas as funcionalidades do programa. Estas incluem: introduzir e eliminar os dados de um novo doente, listar os doentes por ordem alfabética ou os com que têm tensões máximas acima de um determinado valor, apresentar toda a informação de um doente em especifico; e fazer um registo das tensões, do peso e da altura de um determinado doente num determinado dia.

2 Implementação do Programa

Para implementar todas as funcionalidades eficiente decidimos criar duas listas ligadas: uma para armazenar os registos e uma para os doentes. Isto permite gestão de memoria eficiente e ter várias ordenações da lista simultaneamente. As listas ligadas estão armazenadas em ficheiros de texto correspondentes (`doentes.txt` e `registos.txt`) e são atualizadas sempre que um nó é adicionado ou removido.

2.1 Estrutura de Ficheiros

`main.c`: é o ficheiro principal onde está declarado o comportamento do programa. Todas as funções e tipos de dados auxiliares estão declaradas em ficheiros externos.

`tipos.h`: todas as estruturas de dados estão declaradas neste ficheiro. Adicionalmente, estão definidas algumas funções para verificar e validar os tipos de dados quando são inseridos pelo utilizador. Os doentes e os registos têm as suas estruturas de dados correspondentes declaradas neste ficheiro.

`tipos.c`: este ficheiro contém a definição das funções que validam os dados inseridos pelo utilizador.

`doente.h`: declara todas as funções que manipulam a lista ligada correspondente à informação dos doentes incluindo as funções responsáveis por atualizar e guardar a informação dos doentes em ficheiro.

`registo.h`: faz o equivalente a `doente.h` para a lista ligada dos registos.

Por fim, `registo.c` e `doente.c` contém as definições das funções dos respectivos *header files*.

2.2 Estruturas de Dados

> tipos.h

```
typedef struct unsigned int size_tt;
```

`size_tt`: é um *alias* de `unsigned int`. É utilizado em outros *header files* quando queremos armazenar um numero inteiro positivo, por exemplo, no id e número de telefone do doente.

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

`struct Data`: estrutura de dados que guarda o ano, mês e dia correspondentes a uma data. É validada através da função `ler_data`.

```
typedef struct Doente {
  char nome[TAM_EMAIL];
  size_tt id;
  char email[TAM_EMAIL];
  char cc[TAM_CC];
  Data data;
  size_tt telefone;
} Doente;
```

`struct Doente`: contém todos os dados respectivos ao doente. As principais funções do programa dependem desta estrutura de dados.

```
typedef struct noDoente {
    struct Doente doente;
    struct noDoente* prox;
    struct noDoente* prox_alfabetica;
} noDoente ;
```

`struct noDoente`: nó da lista ligada respectiva. Têm dois `pointers`, um para ordem crescente por id e outro para ordem alfabética.

```
typedef struct Registo {
  size_tt id;
  Data data;
  double tensao_minima, tensao_maxima;
  double peso, altura;
} Registo;
```

`struct Registo`: contém todos os dados respectivos a um registo de um doente. Cada registo contém o id do doente a quem pertence. Pode haver id repetido, o que não é o caso para doentes e permite um doente ter vários registos mas não o contrario.

```
typedef struct noRegisto{
  struct Registo registo;
  struct noRegisto* prox;
} noRegisto;
```

`struct noRegisto`: nó da lista ligada respectiva.

2.3 Funções

> tipos.h

```
void limpar_buffer(FILE *stream);
void ler_data(Data *data);
int validar_email(char *input);
int validar_nome(char *input);
int validar_cc(char *input);
int validar_telefone(int input);
```

`limpar_buffer(stdin)` garante que não existe caracteres que "sobram" (como o `newline`) depois do utilizador inserir os dados, evitando errors de inserção. As restantes funções são responsáveis por validar os dados associados com `struct Doente` quando são pedidos ao utilizador.

> doente.h

```
pDoente doente_criar();
void doente_carregar(pDoente raiz);
void doente_guardar(pDoente raiz);
void doente_destroi(pDoente raiz);
int doente_vazia(pDoente raiz);
size_tt doente_obter_id(pDoente raiz);
void doente_insere(pDoente raiz, Doente doente);
void doente_retira(pDoente raiz, size_tt id);
void doente_procura(pDoente raiz, size_tt id, pDoente *anterior, pDoente *atual);
void doente_info(pDoente raiz, size_tt id);
void doente_listar_ordem_alfabetica(pDoente raiz);
Doente doente_id_para_doente(pDoente raiz, size_tt id);
```

`doente_criar`: cria o primeiro nó da lista ligada, a "raiz", e retorna o seu endereço.
`doente_carregar`: lê os dados guardados em `doentes.txt` e insere-os na lista ligada.
`doente_guardar`: guarda todos doentes da lista ligada em `doentes.txt`. `doente_destroi`: destroi a lista ligada e liberta toda a memoria que lhe foi alocada.
`doente_vazia`: devolve 1 se o endereço para o próximo nó da fila for nulo e 0 se for o contrário.

`doente_obter_id`: soma 1 ao maior id da lista ligada evitando que haja ids reutilizados. Parte da funcionalidade do programa afirma que quando as informações de um doente são apagadas o seu id não volta a estar disponível para doentes futuros. O maior id da lista ligada está sempre armazenado no nó raiz.

`doente_insere`: insere um novo nó na lista ligada na posição certa de acordo com a ordenação por ids e de acordo com a ordem alfabética.

`doente_retira`: retira um doente da lista ligada e liberta a sua memoria.
`doente_procura`: procura o endereço do nó anterior por ordem crescente de ids.
`doente_procura_alfabeticamente`: procura o endereço do nó anterior por ordem alfabética.
`doente_info`: imprime na consola a informação de o doente correspondente ao id dado como argumento.
`doente_listar_ordem_alfabetica`: imprime na consola todos os doentes por ordem alfabética.
`doente_id_para_doente`: devolve um `struct Doente` com a informação correspondente ao id pedido.

> registo.h

```
pRegisto registo_criar();
void registo_carregar(pRegisto raiz);
void registo_guardar(pRegisto raiz);
void registo_destroi(pRegisto raiz);
int registo_vazia(pRegisto raiz);
void registo_insere(pRegisto raiz, Registo registo);
void registo_retira(pRegisto raiz, size_tt id);

void registo_procura(pRegisto raiz, int chave, pRegisto *anterior, pRegisto *atual)
void registo_procura_id(pRegisto raiz, size_tt id, pRegisto *anterior, pRegisto *at
void registo_listar_tensoes_max(pDoente raiz_doente, pRegisto raiz, int n);
void registo_listar_doente(pRegisto raiz, size_tt id);
int registo_validar_data(Data d_registo, Data d_doente);
```

Ás primeiras 7 funções são equivalentes às anteriores mas adaptadas para o `struct Registo`.

`registo_procura`: Procura um registo de saúde com a tensão arterial especificada `tensao_maxima`. Define `anterior` para o endereço do nó anterior ao nó encontrado e `atual` para o nó encontrado.

`registo_procura_id`: Faz o equivalente à função anterior para o ID especificado.

`registo_listar_tensoes_max`: Lista todos os registos de saúde com uma tensão arterial (`tensao_maxima`) maior ou igual a `n`, exibindo o nome do paciente e a tensão arterial.

`registo_listar_doente`: Lista todos os registos de saúde de um paciente com o ID especificado, exibindo a data, peso, altura e tensão arterial.

`registo_validar_data`: Valida se a data do registo de saúde é igual ou posterior à data especificada do paciente.

3 Conclusão

Com este projeto desenvolvemos uma aplicação capaz de gerir eficazmente os dados dos doentes e os seus registos clínicos. Usando listas ligadas organizamos a informação de forma eficiente. A validação dos dados inseridos garante que não haja error desnecessário. Ao guardar os dados em ficheiros de texto garantiu que a informação não é perdida quando a aplicação é encerrada.

Foram aplicados vários conceitos como estruturas de dados, alocação dinâmica de memória e escrita e leitura de ficheiros em C. A divisão do código em múltiplos header files manteve o código organizado e mais fácil de manter.

No futuro, a aplicação pode ser expandida com novas funcionalidades, como por exemplo a integração de uma interface gráfica os dados sensíveis dos doentes.