FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE D COIMBRA

Princípios de Programação Procedimental

Engenharia Informática • 2023-2024

Manual do Programador • Projeto

Trabalho realizado por:

Daniel Pereira – 2021237092

Henrique Oliveira – 2022211169

Índice

1. Introdução

No âmbito da cadeira de Princípios de Programação Procedimental, foi desenvolvida uma aplicação em C que permite fazer a gestão dos vários doentes de um médico. O médico pode:

- adicionar e remover doentes,
- listar os doentes por ordem alfabética (id e nome),
- listar os doentes por ordem decrescente do valor da sua tensão máxima,
- apresentar toda a informação de um determinado doente,
- fazer registos, ou seja, adicionar o peso, a altura e as tensões num determinado dia a um doente.

A cada interação, o programa apresenta um menu com todas as opções até que ele escolha a opção Sair.

O programa também implementa uma proteção de dados, garantindo que, sempre que necessário, os dados são atualizados nos ficheiros de texto.

2. Estrutura do Programa

A estrutura do projeto é composta pelos seguintes ficheiros:

“projeto.c” : responsável por inicializar o programa e apresentar o menu principal e onde estão implementadas as funções necessários para o funcionamento do projeto;

“projeto.h” : onde estão as estruturas de dados e a declaração das funções utilizadas no “projeto.c”;

“doentes.txt” : ficheiro de texto onde estão armazenados todos os dados correspondestes aos doentes;

“utilizadores.txt” : ficheiro de texto onde estão armazenados todos os dados correspondentes aos registos.

3. Estruturas de dados

Estrutura Doente

```
struct Doente{
    int id;
    char nome[tamanho_maximo];
    char data_nascimento[11];
    char cc[15];
    char telefone[tamanho_maximo];
    char mail[tamanho_maximo];
};
```

A estrutura Doente armazena informações pessoais dos pacientes.

Estrutura Registo

```
struct Registo{
    int id;
    char data_registo[11];
    int tensao_max;
    int tensao_min;
    int peso;
    int altura;
};
```

A estrutura Registo armazena dados médicos dos pacientes.

Listas ligadas

Para administrar as estruturas Doente e Registo, foram usadas listas ligadas. Cada nó dessas listas armazena a estrutura correspondente e um ponteiro para o próximo nó.

- Nó da lista de doentes

```
typedef struct noListaDoentes{
    struct Doente pessoaLista;
    struct noListaDoentes *prox;
}noListaDoentes;
```

- Nó da lista de registos

```
typedef struct noListaRegistos{
    struct Registo registoLista;
    struct noListaRegistos *prox;
}noListaRegistos;
```

Os ponteiros para as listas de registos e de doentes são definidos da seguinte maneira:

```
typedef noListaDoentes *pListaDoentes;
typedef noListaRegistos *pListaRegistos;
```

4. Funções utilizadas

- **void limparBuffer() :** Esta função serve para limpar o buffer de entrada, garantindo assim que não haja problemas nas leituras.

- **int validarData(const char *data):** Esta função serve para validar se a data está no formato correto (dd/mm/aaaa) e verificar se os valores estão dentro dos intervalos aceitáveis.

- **int validarCC(const char *cc):** Serve para validar se o número do cartão de cidadão português está no formato correto (aaaaaaaa-a-lla, onde a=algarismo e l=letra).

- **int validarTelefone(const char *telefone):** É útil para verificar a inserção correta do número de telefone. Só pode conter algarismos, espaços e o indicativo "+". O tamanho máximo que pode ter é 20.

- **int validarEMail(const char *email):** Esta função valida emails. Os emails têm de conter um "@" algures no meio da string e um "." depois do "@" e antes do fim da string. Não podem conter espaços.

- **int valoresAjustados(int valor , int min, int max):** Verifica se um valor está dentro de um intervalo. É útil para a inserção dos valores numéricos nos registos.

- **void adicionar_doente_lista(pListaDoentes *lista, struct Doente novoDoente:** Permite adicionar um doente à lista de doentes por ordem alfabética do seu nome. Vai criar um nó e adicioná-lo à posição correta depois de comparar os nomes correspondentes aos vários nós com a função *strcmp*.

- **void ler_doentes(pListaDoentes *lista):** Função que permite ler o ficheiro "doentes.txt" e armazenar os doentes usando a função anterior.

- **void adicionar_registo_lista(pListaRegistos *lista, struct Registo novoRegisto):** Serve para adicionar um novo registo à lista de registos por ordem decrescente de tensão máxima. Vai criar um nó e adicioná-lo à posição correta depois de comparar as tensões maximas correspondentes aos vários nós.

- **void ler_registos(pListaRegistos *lista):** Permite ler o ficheiro “registos.txt” e armazenar os registos usando a função anterior.

- **void listar_doentes(const pListaDoentes *lista):** Esta função permite listar os vários doentes por ordem alfabética, imprime o id e o nome de cada um.

- **void listar_doentes_tensao_maxima(const pListaRegistos *lista, const pListaDoentes *listaDoentes, int valor):** Esta função permite listar decrescentemente todos os doentes com tensões máximas superiores a um certo valor. São impressos o nome e a tensão máxima de cada um (para isso é procurado na lista de doentes o nome do doente com o id correspondente ao registo).

- **void apresentar_informacao_doente(const pListaRegistos *lista_registos, const pListaDoentes *lista_doentes, int id_doente):** Esta função serve para apresentar toda a informação de um doente incluindo todos os seus registos.

- **void adicionar_doente_arquivo(struct Doente novoDoente):** Esta função serve para inserir no ficheiro “doente.txt” um novo doente.

- **int gerar_novo_id(const pListaDoentes lista):** Esta função vai percorrer todos os doentes e verificar qual é o id mais alto atribuído e retorna esse valor + 1. Esse valor será o id do novo doente.

- **void introduzir_novo_doente(pListaDoentes *lista):** Esta função é usada para pedir os dados do novo doente ao utilizador e colocá-los no ficheiro de texto e na lista de doentes.

- **void remover_doente_arquivos(int id):** Permite remover um doente e os respetivos registos dos arquivos que os armazenam, utilizando o seu ID.

- **void remover_doente_lista(pListaDoentes *lista, int id):** Possibilita a remoção de um doente da lista de doentes.

- **void remover_registos_lista(int id, pListaRegistos * lista_registos):** Esta função remove todos os registos associados a um doente específico da lista de registos.

- **void remover_doente(int id, pListaDoente * lista_doentes, pListaRegistos *lista_registos):** Esta função elimina todas as informações do doente que existem (informação e registos), confirmando primeiro se o id inserido existe.

- **void inserir_registo(const pListaDoentes *lista_doentes, pListaRegistos *lista_registos, int id):** Esta função permite adicionar um novo registo para um doente em específico. Primeiro verifica se o doente está na lista através do seu id. Caso exista, solicita e valida os dados do registo. Após isso irá adicioná-los a lista de registos e ao arquivo de registos.

- **void libertar_lista_registos(pListaRegistos *lista):** Esta função serve para libertar a memória alocada para a lista de registos. Ela é utilizada quando o utilizador encerra o programa.

- **void libertar_lista_doentes(pListaDoentes *lista):** Esta função serve para libertar a memória alocada para a lista de doentes. Ela é utilizada quando o utilizador encerra o programa.

5. Interface

A interface apresentada ao utilizador tem o seguinte aspeto :

```
Bem-vindo ao portal de saude.

Opções:

1-Introduzir um novo doente
2-Eliminar um doente
3-Listar doentes por ordem alfabetica
4-Listar doentes com tensoes maximas acima de um valor
5-Apresentar toda a informação de um doente
6-Inserir tensoes, peso e altura de um doente num determinado dia
7-Sair
```

Após iniciar o programa é apresentado o menu para que o utilizador saiba as opções que tem. Para selecionar uma das opções basta digitar o seu algarismo correspondente. O programa utiliza um *switch* para verificar se o algarismo inserido corresponde a algum dos seus blocos. Se inserir outra coisa que não um algarismo entre e 1 e 7 o programa avisa que foi dada uma entrada inválida e apresenta novamente o menu.

Todas as entradas dos utilizadores são verificadas para que o programa funcione corretamente.

6. Informações

Todas as estruturas deste projeto foram desenvolvidas e testadas no sistema operativo *Windows 11* usando o *Visual Studio Code*.

Através do *Valgrind,* constatámos que não ocorreu nenhum vazamento de memória durante a execução do programa.