Projeto de Princípios de Programação Procedimental – Manual do Programador

Alexandre Rodrigues
2022249408

Bernardo Luz
2022230815

Ficheiros:

No âmbito da conclusão do projeto, criamos e utilizamos 3 ficheiros. Um header file, “projeto.h”, para incluir bibliotecas, criar e conter as estruturas de dados necessárias, definir constantes e para declarar funções. E dois ficheiros “.c”, o “projeto.c” e o “projeto_func.c”.

O nosso ficheiro principal, “projeto.c”, é onde está localizada interface e a função “main”, o qual irá buscar as funções ao “projeto_func.c”, este contem todas as nossas funções necessárias ao bom funcionamento do projeto.

projeto.h :

Começamos por definir o header file, depois incluímos diversas bibliotecas necessitadas, em seguida criamos uma constante “Tam” que posteriormente será utilizada na estrutura de dados entre outros. Criamos 5 estruturas de dados, 2 para guardar as variáveis relacionadas com os doentes, a “DadosDoente” para o registo e a “Doente” para os dados do doente, em ambas introduzimos uma data, por isso, também criamos uma struct “Data” facilitando o armazenamento das mesmas. Também criamos duas estruturas para a realização de nós para listas ligadas, “noLista” para a estrutura “Doente” e “nodados” para “DadosDoente”.

A nível de funções, encontram-se os cabeçalhos de todas as funções usadas no projeto.

porjeto_func.c :

- **Funções base de listas ligadas:**
  - **cria / criaDados** -> inicializa a lista ligada com um header node;
  - **vazia / vaziaDados** -> percorre a lista ligada, não contando o header node, verificando se está vazia;
  - **destrói / destroiDados** -> percorre a lista ligada eliminando cada no, até estar vazia;
  - **insere** -> Verifica se a lista está vazia ou se o novo doente deve ser inserido no início, se não se verifica encontra o local adequado na lista para inserir o novo doente por ordem alfabética;

- **insereDados / insereTens** -> Cria um novo nó que vai conter a informação a ser inserida, depois de verificar que existe lista, e insere na posição a seguir ao último nó;

- **impime / imprimeDados** -> Á medida que percorre a lista ligada vai imprimindo a informação contida em cada nó;

- **imprimeEspecifico** -> Imprime os dados contidos no nó fornecido pelo utilizador;

- **elimina** -> procura o nó com informação fornecido dada pelo utilizador e elimina-o;

- **procura_(*)** -> vai atribuir ao ponteiro do tipo "pLista" ou "pDados" "atual", o nó que contenha a característica desejada;

- **pesquisa_Id** -> através da função "procura_Id" do tipo "pLista" retorna o ponteiro "atual" do tipo da função;

- **pesquisa_CC / pesquisa_email / pesquisa_telefone** -> através da função procura do tipo equivalente à da função especifica, retorna 1 caso encontre e 0 caso contrário.

*(Id; Id_Dados; Nome; CC; email; telefone; Tens)

- **Funções da nossa autoria:**
  - **verifica_id** -> percorre a lista ligada comparando os id's existentes retornando o maior;

  - **escreve_ficheiro** -> Abrimos o ficheiro em modo *write*, e à medida que percorre a lista ligada vai escrevendo a informação contida em cada nó no ficheiro;

  - **escreve_ficheiro_Dados** -> Abrimos o ficheiro em modo *append*, e à medida que percorre a lista ligada vai escrevendo a informação contida em cada nó no ficheiro, utilizando um "count" para saber se será a primeira linha ou não, caso não seja introduz um "\n" antes de começar a escrever;

  - **retira_doentes_txt / retira_registos_txt** -> Ambas as funções têm o mesmo raciocínio, contudo, destinam-se a ficheiros e listas ligadas diferentes. Criamos um método para que enquanto lê o ficheiro, insira as linhas do mesmo na lista ligada, garantindo que o mesmo não esteja corrompido nem deficiente através das devidas

verificações, ou seja, a cada 6 linhas insere um novo nó, no fim fazemos mais uma condição para colocar as últimas 6 linhas do ficheiro;

- **analisa_nome_comp** -> a função recebe um nome na totalidade através de um ponteiro e copia-o para uma variável, em seguida realiza um ciclo “while” para dividir por espaços e verificando se é um nome adequado ou real através de uma verificação local e outra através da função “analisa_nome”;

- **analisa_nome** -> recebe um parte do nome total vinda da função “analisa_nome_comp”e verifica se o mesmo só contem letras do alfabeto;

- **ver_data** -> a função verifica se as datas introduzidas são validas, começando pelo ano estar dentro de m limite, e fazemos uma verificação para os anos bissextos, por causa dos dias, chamando a função “ano_bi”, e limitando os dias;

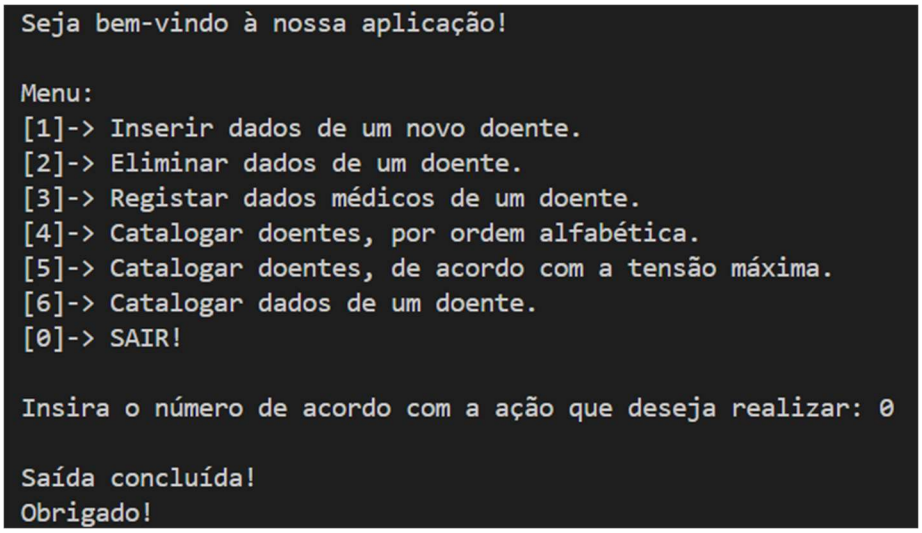
- **ano_bi** -> recebe o ano pela função “ver_data” e realiza contas para verificar se é bissexto;

- **insere_um_doente** -> esta função é chamada quando algum utilizador deseja introduzir um doente, esta faz a verificação do id automaticamente através da “verifica_id” e atribui, pede nome, data de nascimento, cartão de cidadão, telefone e email. Esta função utiliza outras previamente criadas para a verificação, como por exemplo a pesquisa telefone e também faz outras verificações locais. Isto tudo dentro de um ciclo que caso o utilizador se engane ou já exista o cartão de cidadão por exemplo o avisa e pede outra vez a sua introdução. No fim insere na lista ligada um novo nó com os dados.

- **Isnerir_registo** -> esta função recebe como argumento o id do doente, e vai pedindo ao utilizador as informações respetivas do doente, com verificação externa e interna usando funções como “ver_data”

porjeto.c :

Começamos o ficheiro principal por declarar todas as variáveis necessárias para o nosso programa, verificamos se os ficheiros estão bons para leitura e criamos a lista “doente” onde vamos inserir as reservas e pré reservas.



A aplicação dá ao utilizador 7 ações, 6 que pode realizar as vezes que desejar e uma para sair:

1. Inserir dados de um novo doente -> Realiza as funções "inserir_um_doente" e "imprimeEspecifico" que pede ao utilizador as variáveis todas e depois introduz, em seguida imprime os dados do utilizador que inseriu;

2. Eliminar dados de um doente -> Pede o id ao utilizador do doente que deseja eliminar depois chama a função elimina, no fim confirma que foi eliminado;
3. Registar dados médicos de um doente -> Pede o id ao utilizador do doente que deseja inserir os registos dos dados de um doente utilizando a "pesquisa_Id" e a "insere_Dados";

4. Catalogar doentes, por ordem alfabética -> Utiliza a função "imprime" para imprimir todos os nós da lista por ordem alfabética;

5. Catalogar doentes, de acordo com a tensão máxima -> cria uma lista ligada de tensões, pede ao utilizador uma tensão máxima e depois através da função "procura_Tens" introduz os nós que tem a tensão maior do que a introduzida pelo utilizador, e no fim imprime as mesmas;

6. Catalogar dados de um doente -> Pede um Id ao utilizador e depois se encontrar através da "procura_Id" imprime os dados do nó da lista ligada "Doentes" e da "DadosDoente", caso contrário avisa o utilizador que não existe doente;

0. Sair -> abre o ficheiro registos em modo “write” e verifica se esta apto para leitura, em seguida vai escrevendo todos os registos nos ficheiros através de um “while” com a “escreve_ficheiro_Dados”, em seguida escreve a lista ligada dos doentes no ficheiro através da função “escreve_ficheiro” e no fim apresenta a mensagem “Saída concluída! Obrigado!”;