**ONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS**

**INSTITUTO DE CIÊNCIAS EXATAS E INFORMÁTICA**

**UNIDADE EDUCACIONAL PRAÇA DA LIBERDADE**

**Bacharelado em Engenharia de Software**

**Antônio Moncorvo Mascarenhas**

**Daniel Aquino Silva**

**Guilherme Oliveira Antônio**

**Mateus Pace Scafutto**

**CLÍNICA VIVER BEM**

****

1. **APRESENTAÇÃO**

O referente trabalho foi feito com o intuito de desenvolver uma aplicação que busca agilizar os processos de uma clinica médica. Nesse documento fica explicito quais foram os processos e suas respectivas etapas utilizados para a execução de tal atividade.

1. **BACKLOG DO PRODUTO:**

**Geral**

Criar função de cadastro de pacientes.

Criar função de cadastro de médicos.

Criar função de cadastro de consultas.

Criar função de cancelamento de consultas.

Criar função de relatório do dia.

Criar função de relatório do paciente.

Criar função de relatório do médico.

Criar função EXTRA.

**1º sprint**

**Requisitos de detalhas de cadastro**

Paciente

Código terá 4 dígitos (1000 a 2000). EX.: 1098.

Será gerado de forma aleatória.

Chamadas dos pacientes no programa serão através do nome.

Cada paciente terá um código.

Pacientes não poderão ter códigos iguais.

Médico

Código terá 2 dígitos (10 a 99). EX.: 36.

Será gerado de forma aleatória.

Chamadas dos médicos no programa serão através do nome.

Cada médico terá um código.

Médicos não poderão ter códigos iguais.

Será registrado: código, nome, telefone, especialidade.

Consulta

Código terá 5 dígitos(10000 a 20000). EX.:19876.

Será gerado de forma aleatória.

Códigos de consulta não podem ser iguais.

Cada consulta terá seu código.

Consultas não terão códigos iguais.

Para cadastrar uma consulta médico e paciente devem ter códigos registrados.

Cada consulta deve ser marcada em intervalos de 30 minutos.

Cada dia só pode ser realizado duas consultas por médico.

**Funções**

Função main

Deve exibir um menu

Deve Ficar em loop ate que se deseje sair

Informações devem ser gravadas em um arquivo de texto binário.

Função menu

1-Cadastro do paciente

2-Cadastro do médico

3-Marcar da consulta\n

4-Relatório do dia \\ Mostra todas as consultas do dia

5-Relatório do paciente \\ Mostra todas consultas de um determinado paciente

6-Relatório do médico \\ Mostra todas consultas de um determinado médico

7-Função extra

8-Cancelamento da consulta

9-Sair

**2º sprint**

**Estrutura para cadastro**

Paciente

Nome da estrutura: pacientes

Tipo inteiro: codidoP, telefoneP, dataN.

Tipo caractere: nomeP, ruaP, bairroP, cidadeP, estadoP.

Médico

Nome da estrutura: medico

Tipo inteiro: codigoM, telefoneM

Tipo caractere: nomeM, especialidade.

Consulta

Nome da estrutura: consulta

Tipo inteiro: codigoC, codigoM, codigoP, status, dataC, horaC.

Tipo caractere: nomeP, nomeM.

Obs.: variáveis com P significam paciente.

Obs.: variáveis com N significam nascimento.

Obs.:variáveis com M significam médico.

**Funções principais**

Definido a função extra. Essa função ira lista s os médicos cadastrados.

Nome de cada função

cadastrarPaciente

cadastrarMedico

cadastraConsulta

cancelaConsulta

relatorioDia

relatorioPaciente

relatorioMedico

listadeMedicos

Parâmetros

Deve ser escrito de forma que todos do grupo entendam.

Pode ser escrita de forma abreviada.

Feitos no momento de codificação.

Apos definido o nome das funções e definido como seria feita cada função, houve divisão de funções para ser deita por cada um do grupo.

Antônio Moncorvo Mascarenhas: relatorioDia; cadastrar\_medico

Daniel Aquino Silva: relatorioMedico; listadeMedicos

Guilherme Oliveira Antônio: cadastrar\_paciente; relatorioPaciente

Mateus Pace Scafutto: cadastraConsulta; cancelaConsulta

**3º sprint**

Juntar todas as funções em um código

Fazer testes no código

Consertar erros e defeitos

Organizar os códigos para criar funções e tornar o código mais limpo

Documentação dos parâmetros e das funções principais

Funções secundárias criadas

*limpatela()*

*pausa()*

*pausa()*

*limpaBuffer()*

*menu()*

*criaArquivos()*

*localizaPaciente()*

*localizaMedico()*

1. **LISTA DE ASSINATURAS DAS FUNÇÕES E PARÂMETROS**

**void limpaTela()**

Este procedimento foi criado para tornar o código mais simples, fácil de ser lido e de ser escrito.

**void pausa()**

Este procedimento foi criado com o intuito de auxiliar o programador e cliente a ver os dados em sua tela.

**void limpaBuffer()**

Este procedimento foi criado para tornar o código mais limpo e acessível.

**void menu()**

Este procedimento foi criado para mostrar ao usuário as funções que ele poderá acessar, e facilitar a leitura do código.

**void criaArquivos()**

Este procedimento cria os arquivos necessários para as funções do programa.

Arquivos criados:

Fexistencia: este arquivo analisa a existência dos dados ao cadastrar um paciente, médico ou consulta.

Fpaciente: este arquivo cadastra os dados do paciente em seu interior.

Fmedico: este arquivo cadastra os dados do médico em seu interior.

Fconsulta: este arquivo cadastra os dados da consulta em seu interior.

**void cadastrarPaciente (FILE \*Fpaciente)**

Este procedimento cadastra os dados do paciente no arquivo **cadastro-paciente.dat**.

Arquivos utilizados:

\*Fpaciente: variável que é ponteiro para um arquivo que cadastra os dados do paciente.

Variáveis criadas:

int tam: criada para apontar qual o tamanho do vetor e também qual sua posição no registro.

Registros criados:

pacientes p[1000]: registro criado para armazenar os dados do paciente. Cada posição no vetor armazena um paciente.

**void cadastrarMedico (FILE \*Fmedico)**

Este procedimento cadastra os dados do médico no arquivo **cadastro-medico.dat**.

Arquivos utilizados:

\*Fmedico: variável que é ponteiro para um arquivo que cadastra os dados do médico.

Variáveis criadas:

int tam: criada para apontar qual o tamanho do vetor e também qual sua posição no registro.

Registros criados:

medico m[90]: registro criado para armazenar os dados do médico. Cada posição no vetor armazena um médico.

**int localizaPaciente (char nomeP[30])**

Esta função localiza o paciente no arquivo **cadastro-paciente.dat** para averiguar se o mesmo está cadastrado.

Variáveis utilizadas:

char nomeP[30]: variável do tipo string que armazena o nome do paciente.

int achou: variável do tipo int que afirma se o paciente desejado foi localizado na estrutura.

int posicao: variável do tipo int que encontra a posição do paciente desejado na estrutura.

Registros utilizados:

paciente p[1000]: registro criado para armazenar os dados do paciente. Cada posição no vetor armazena um paciente.

Arquivos utilizados:

**\***Fpaciente: variável que é ponteiro para um arquivo que cadastra os dados do paciente.

**int localizaMedico (char nomeM[30])**

Esta função localiza o médico no arquivo **cadastro-medico.dat** para averiguar se o mesmo está cadastrado.

Variáveis utilizadas:

int achou: variável do tipo int que afirma se o médico desejado foi localizado na estrutura.

int posicao: variável do tipo int que encontra a posição do paciente desejado na estrutura.

char nomeM[30]: variável do tipo string que armazena o nome do médico.

Registros utilizados:

medico m: registro criado para armazenar os dados do médico. Cada posição no vetor armazena um médico.

Arquivos utilizados:

**\***Fmedico: variável que é ponteiro para um arquivo que cadastra os dados do médico.

**void cadastraConsulta (FILE \*Fconsulta)**

Este procedimento cadastra os dados da consulta no arquivo **cadastro-consulta.dat.**

Arquivos utilizados:

\*Fconsulta: variável que é ponteiro para um arquivo que cadastra os dados da consulta.

\*Fpaciente: variável que é ponteiro para um arquivo que cadastra os dados do paciente.

\*Fmedico: variável que é ponteiro para um arquivo que cadastra os dados do médico.

Variáveis criadas:

char nomeP[30]: variável do tipo string que armazena o nome do paciente.

char nomeM[30]: variável do tipo string que armazena o nome do médico.

int posicaoP: variável do tipo int utilizada para confirmar a existência do paciente.

int posicaoM: variável do tipo int utilizada para confirmar a existência do médico.

int tam: criada para apontar qual o tamanho do vetor e também qual sua posição no registro.

int aux: variável do tipo int que auxilia na verificação da data e analisar se está no padrão proposto pelo trabalho.

Registros criados:

consulta c[10000]: registro criado para armazenar os dados da consulta. Cada posição no vetor armazena uma consulta.

pacientes p[1000]: registro criado para armazenar os dados do paciente. Cada posição no vetor armazena um paciente.

medico m[90]: registro criado para armazenar os dados do médico. Cada posição no vetor armazena um médico.

**void relatorioDia (FILE \*Fconsulta)**

Este procedimento mostra todas as consultas marcadas para determinada data.

Variáveis utilizadas:

int dia: variável do tipo int que armazena o dia da data desejada.

int mes: variável do tipo int que armazena o mês da data desejada.

int ano: variável do tipo int que armazena o ano da data desejada.

Registros utilizados:

consulta c[10000]: registro criado para armazenar os dados da consulta. Cada posição no vetor armazena uma consulta.

**void relatorioPaciente (FILE \*Fconsulta)**

Este procedimento mostra todas as consultas de determinado paciente.

Variáveis utilizadas:

char nomeP[30]; variável do tipo string que armazena o nome do paciente.

Registro utilizados:

consulta c[10000]: registro criado para armazenar os dados da consulta. Cada posição no vetor armazena uma consulta.

Arquivos utilizados:

\*Fconsulta: variável que é ponteiro para um arquivo que cadastra os dados da consulta.

**void relatorioMedico (FILE \*Fconsulta)**

Este procedimento mostra todas as consultas de determinado médico.

Variáveis utilizadas:

char nomeM[30]: variável do tipo string que armazena o nome do médico.

Registros utilizados:

consulta c[10000]: registro criado para armazenar os dados da consulta. Cada posição no vetor armazena uma consulta.

Arquivos utilizados:

\*Fconsulta: variável que é ponteiro para um arquivo que cadastra os dados da consulta.

**Void cancelaConsulta (FILE \*Fconsulta)**

Este procedimento exclui uma consulta de acordo com seu código.

Variáveis utilizadas:

int codigo: variável do tipo int que armazena o código da consulta.

int aux: variável do tipo int que recebe a resposta ( 0 ou 1 ) do usuário e que define se determinada consulta será cancelada.

Registros utilizados:

consulta c[10000]: registro criado para armazenar os dados da consulta. Cada posição no vetor armazena uma consulta.

Arquivos utilizados:

\*Fconsulta: variável que é ponteiro para um arquivo que cadastra os dados da consulta.

**void listadeMedicos(FILE \*Fmedico)**

Este procedimento lista todos os médicos já cadastrados.

Registros utilizadas:

medico m[90]: registro criado para armazenar os dados do médico. Cada posição no vetor armazena um médico.

Arquivos utilizados:

\*Fmedico: variável que é ponteiro para um arquivo que cadastra os dados do médico.

1. **TESTES E SEUS RELÁTORIOS DE EXECUÇÃO**
2. **Menu**
3. Entrdas: apenas dígitos, de 1 a 9
4. Classes validas: Apenas 1 digito entre 1 e 9. Ex: 1, 2,3,4...9.
5. Resultado esperado: Inicia a opção escolhida pelo usuário
6. Classes inválidas: Mais de 1 digito (dentre 1 a 9), digito 0, letras, caracteres. Ex: 11, 45, 0, a, abc, 1a2b, !!, 1!, 1!a...
7. Resultado esperado: Mensagem de código invalido
8. **Cadastro de pacientes**
   1. **Nome do paciente, Bairro, Cidade, Estado**
9. Entradas: Apenas letras (números em algarismos romanos)
10. Classes validas: Apenas letras. Ex:Antonio, Antonio Moncorvo, Antonio IV, Santa Lucia, Sta Lucia, Belo Horizonte, BH, Minas Gerais, MG...
11. Resultado esperado: Usuário segue para a próxima etapa
12. Classes inválidas: Digitos e Caracteres. Ex: Antonio!, Antonio1234, Min@s Ger@is...
13. Resultado esperado: Mensagem de Erro
    1. **Telefone**
14. Entradas: Apenas números
15. Classes validas: Apenas números. Ex: 84208398, 33443535...
16. Resultado esperado: Usuário segue para a próxima etapa
17. Classes inválidas: Letras e Caracteres. Ex: aaaaaa, AA!!23, !@aa123...
18. Resultado esperado: Mensagem de Erro
    1. **Dia, Mês e Ano**
    2. Entradas: Apenas números
    3. Classes validas: Apenas números. Ex: 04, 12, 1999, 21, 05, 1935...
    4. Resultado esperado: Usuário segue para a próxima etapa
    5. Classes inválidas: Letras e Caracteres. Ex: aaabbb , ab!23, !12aa\*...
    6. Resultado esperado: Mensagem de Erro
    7. **Rua**
       * + 1. Entradas: Letras e números
           2. Classes validas: Letras e números. Ex: Avenida do Contorno 3513, Av Prudente de Moraes 223
           3. Resultado esperado: Usuário segue para a próxima etapa
           4. Classes inválidas: Caracteres. Ex: Av. Prudente de Moraes!
           5. Resultado esperado: Mensagem de Erro
19. **Cadastro do Médico**
    1. Nome e Especialidade
20. Entradas: Apenas Letras
21. Classes validas: Apenas Letras Ex: Antonio Mascarenhas , Antonio, Clinica Geral ...
22. Resultado esperado: Usuário segue para a próxima etapa
23. Classes inválidas: Números e Caracteres. Ex: Antonio 123, Antonio!, Clinic@ Ger@l...
24. Resultado esperado: Mensagem de Erro
    1. **Telefone**
25. Entradas: Apenas números
26. Classes validas: Apenas números. Ex: 84208398, 33443535...
27. Resultado esperado: Usuário volta ao menu
28. Classes inválidas: Letras e Caracteres. Ex: aaaaaa, AA!!23, !@aa123...
29. Resultado esperado: Mensagem de Erro
30. **Marcar Consulta**

**4.1 Nome do Paciente e Nome do Médico**

1. Entradas: Apenas Letras, Nome já Registrado
2. Classes validas: Apenas Letras, Nome já Registrado Ex: Antonio Mascarenhas, Antonio Moncorvo
3. Resultado esperado: Usuário segue para a próxima etapa
4. Classes inválidas: Números, Caracteres e Nomes não Registrados. Ex: Antonio 123, Antonio!, Antonioo Moncorvoo, Antonioo Mascarenhass
5. Resultado esperado: Mensagem de Erro, Mensagem de Medico ou Paciente não registrado
   1. **Dia, Mês e Ano da Consulta**
6. Entradas: Apenas números
7. Classes validas: Apenas números. Ex: 04, 12, 1999, 21, 05, 1935...
8. Resultado esperado: Usuário segue para a próxima etapa
9. Classes inválidas: Letras e Caracteres. Ex: aaabbb , ab!23, !12aa\*...
10. Resultado esperado: Mensagem de Erro
    1. **Hora da Consulta**
    2. Entradas: Apenas números entre 13 horas e 18 horas
    3. Classes validas: Apenas números entre 13 horas e 18 horas 1. Ex: 13, 14... 18
    4. Resultado esperado: Usuário segue para a próxima etapa
    5. Classes inválidas: Letras e Caracteres. Ex: aaabbb , ab!23, !12aa\*...
    6. Resultado esperado: Mensagem de Hora Inválida
    7. **Minutos da Consulta**
11. Entradas: Apenas 30 (para meia hora) ou 00 (para hora inteira)
12. Classes validas: Apenas 30 (para meia hora) ou 00 (para hora inteira)1. Ex: 30 ou 00
13. Resultado esperado: Usuário volta ao menu
14. Classes inválidas: Qualquer coisa diferente de 30 ou 00 Ex: aaabbb , ab!23, !12aa, 25, 33, 01, 60...
15. Resultado esperado: Mensagem de Minuto Inválido

OBS: Ao final da Marcação de Consulta será fornecido um código de consulta

1. **Relatório do Dia**
   1. **Dia, Mês e Ano da Consulta**
2. Entradas: Apenas números
3. Classes validas: Apenas números de dias que foram marcadas consultas Ex: 04, 12, 1999, 21, 05, 1935...
4. Resultado esperado: O console exibe as consultas do dia em questão
5. Classes inválidas: Letras , Caracteres e números de dias que não tem consultas. Ex: aaabbb , ab!23, !12aa\*...
6. Resultado esperado: Mensagem de Erro, Mensagem de que não existem consultas registradas nesse dia
7. **Relatório do Paciente ou do Médico**

**6.1 Nome do médico ou Paciente**

1. Entradas: Apenas Letras, Nome já Registrado
2. Classes validas: Apenas Letras, Nome já Registrado Ex: Antonio Mascarenhas, Antonio Moncorvo
3. Resultado esperado: Console Exibe todas as consultas do médico ou paciente em questão.
4. Classes inválidas: Números, Caracteres e Nomes não Registrados. Ex: Antonio 123, Antonio!, Antonioo Moncorvoo, Antonioo Mascarenhass
5. Resultado esperado: Mensagem de Erro, Mensagem de Medico ou Paciente não registrado
6. **Lista de Médicos**
   1. Entradas: apenas dígitos
   2. Classes validas: Apenas o digito referente à lista de médicos. Ex: 7
   3. Resultado esperado: Inicia a opção escolhida pelo usuário e mostra uma lista com todos os médicos e suas respectivas especialidades.
   4. Classes inválidas: Mais de 1 digito, digito diferente do referente à lista de medicos, letras, caracteres. Ex: 11, 45, 0, a, abc, 1a2b, !!, 1!, 1!a, 2, 6, 9,1...
   5. Resultado esperado: Mensagem de código invalido, Abre outra opção no menu
7. **Cancelamento de consulta**
   1. Entradas: apenas dígitos, Numero de consulta que foi fornecido quando a consulta foi marcada
   2. Classes validas: Códigos de consulta validos. Ex: 7341
   3. Resultado esperado: Mensagem de Consulta cancelada, retorna ao menu
   4. Classes inválidas: código invalido, letras, caracteres. Ex: 11, 45, 0, a, abc, 1a2b, !!, 1!, 1!a, 2, 6, 9,1...
   5. Resultado esperado: Mensagem de erro, Mensagem de Código invalido
8. **CÓDIGO EM C**

#include <stdio.h>

#include <stdlib.h>

#include <locale.h>

#include <string.h>

#include <math.h>

/\*-----------------------ESTRUTURA PACIENTES----------------------\*/

// struct do paciente

struct Tpacientes

{

int codigoP, telefoneP;

int dataN[3];//Dia, Mes, Ano

char nomeP[30], ruaP[30], bairroP[30], cidadeP[30], estadoP[30] ;

};

typedef struct Tpacientes pacientes;

/\*-----------------------ESTRUTURA MEDICO------------------------\*/

// struct do m�dico

struct Tmedico

{

int codigoM, telefoneM;

char nomeM[30], especialidade[50];

};

typedef struct Tmedico medico;

/\*--------------------------ESTRUTURA CONSULTA--------------------\*/

// struct da consulta

struct Tconsulta

{

int codigoC, codigoM, codigoP, status;

int dataC[3]; //Dia, Mes, Ano

int horaC[2];//Hora, Minutos

char nomeP[30], nomeM[30];

};

typedef struct Tconsulta consulta;

/\*------------------------DECLARANDO FUNCOES--------------------------\*/

void limpaTela();

void pausa();

void limpaBuffer();

void menu();

void criaArquivos();

void cadastrarPaciente(FILE\*Fpaciente);

void cadastrarMedico(FILE\*Fmedico);

int localizaPaciente(char nomeP[30]);

int localizaMedico(char nomeM[30]);

void cadastraConsulta(FILE\*Fconsulta);

void relatorioDia(FILE\*Fconsulta);

void relatorioPaciente(FILE\*Fconsulta);

void relatorioMedico(FILE\*Fconsulta);

void listadeMedicos(FILE\*Fmedico);

void cancelaConsulta(FILE\*Fconsulta);

/\*--------------------------MAIN----------------------------\*/

int main()

{

setlocale(LC\_ALL, "portuguese");

int op;

FILE \*Fexistencia, \*Fpaciente, \*Fmedico, \*Fconsulta;

//condi��o para criar arquivos

Fexistencia = fopen("C\_existencia.dat","r+b");

if(Fexistencia == NULL)

{

criaArquivos();

}

fclose(Fexistencia);

//entrando no menu em loop

do

{

limpaTela();

menu();

fflush(stdin);

scanf("%i", &op);

switch(op)

{

case 1:

printf("\t\tCADASTRAR PACIENTE\n");

cadastrarPaciente(Fpaciente);

limpaTela();

break;

case 2:

printf("\t\tCADASTRAR MEDICO\n");

cadastrarMedico(Fmedico);

limpaTela();

break;

case 3:

printf("\t\tMARCAR CONSULTA\n");

cadastraConsulta(Fconsulta);

limpaTela();

break;

case 4:

printf("\t\tRELATORIO DO DIA\n");

relatorioDia(Fconsulta);

limpaTela();

break;

case 5:

printf("\t\tRELATORIO DO PACIENTE \n");

relatorioPaciente(Fconsulta);

limpaTela();

break;

case 6:

printf("\t\tRELATORIO DO MEDICO\n");

relatorioMedico(Fconsulta);

limpaTela();

break;

case 7:

printf("\t\tLISTA MEDICO\n");

listadeMedicos(Fmedico);

limpaTela();

break;

case 8:

printf("\t\tCANCELANDO CONSULTA\n");

cancelaConsulta(Fconsulta);

limpaTela();

break;

case 9:

op = 9;

break;

default:

printf("CODIGO INVALIDO");

pausa();

break;

}

}

while(op!=9);

}

/\*---------------------------------FUNCOES---------------------------------\*/

//PAUSA

void pausa()

{

system("pause");

}

//LIMPA TELA

void limpaTela()

{

system("cls");

}

//LIMPA BUFFER

void limpaBuffer()

{

fflush(stdin);

}

//CRIA ARQUIVOS

void criaArquivos()

{

FILE \*Fexistencia, \*Fpaciente, \*Fmedico, \*Fconsulta;

Fexistencia = fopen("C\_existencia.dat","w+b");

if(Fexistencia == NULL)

{

printf("Erro\n");

}

fclose(Fexistencia);

Fpaciente = fopen("cadastro-paciente.dat","w+b");

if(Fpaciente == NULL)

{

printf("Erro na abertura\n");

exit(1);

}

fclose(Fpaciente);

Fmedico = fopen("cadastro-medico.dat", "w+b");

if(Fmedico == NULL)

{

printf("Erro na abertura\n");

exit(2);

}

fclose(Fmedico);

Fconsulta = fopen("cadastro-consulta.dat","w+b");

if(Fconsulta == NULL)

{

printf("Erro na abertura\n");

exit(3);

}

fclose(Fconsulta);

}

//MENU

void menu()

{

printf("\t\t----- Clinica Viva Bem -----\n\n");

printf("1- Cadastro do paciente\n");

printf("2- Cadastro do medico\n");

printf("3- Marcar da consulta\n");

printf("4- Relatorio do dia \\ Mostra todas as consultas do dia\n");

printf("5- Relatorio do paciente \\ Mostra todas conultas de um determinado paciente\n");

printf("6- Relatorio do medico \\ Mostra todas consultas de um determinado medico\n");

printf("7- Lista todos medicos\n");

printf("8- Cancelamento da consulta\n");

printf("\n\n9- Sair\n\n");

printf("\n\nSelecione a opcao: ");

}

//CADASTRO DE PACIENTE

void cadastrarPaciente(FILE\*Fpaciente)

{

int tam=-1,achei=0;

pacientes p[1000];

Fpaciente = fopen("cadastro-paciente.dat", "r+b");

if(Fpaciente == NULL)

{

printf("Erro");

exit(1);

}

//conta o tamanho do vetor

fseek(Fpaciente,0,SEEK\_SET);

if(!feof(Fpaciente))

{

tam=0;

}

else

{

while(!feof(Fpaciente))

{

fread(p,sizeof(p),1,Fpaciente);

tam++;

}

}

printf("Nome do paciente: ");

limpaBuffer();

gets(p[tam].nomeP);

printf("Digite o numero de telefone: ");

limpaBuffer();

scanf("%d",&p[tam].telefoneP);

printf("----------DATA DE NASCIMENTO-------------\n");

printf("Digite o DIA: ");

limpaBuffer();

scanf("%d",&p[tam].dataN[0]);

printf("Digite o MES: ");

limpaBuffer();

scanf("%d",&p[tam].dataN[1]);

printf("Digite o ANO: ");

limpaBuffer();

scanf("%d",&p[tam].dataN[2]);

printf("----------ENDERECO-----------\n");

printf("RUA: ");

limpaBuffer();

gets(p[tam].ruaP);

printf("BAIRRO: ");

limpaBuffer();

gets(p[tam].bairroP);

printf("CIDADE: ");

limpaBuffer();

gets(p[tam].cidadeP);

printf("ESTADO: ");

limpaBuffer();

gets(p[tam].estadoP);

p[tam].codigoP = (rand()%999)+1000;

fseek(Fpaciente,0,SEEK\_SET);

/\*while(!achei)

{

while(!feof(Fpaciente))

{

fread(&p,sizeof(p),1,Fpaciente);

if(p.codigoP == p[tam].codigoP)

{

p[tam].codigoP = (rand()%999)+1000;

achei = 0;

}

else

{

achei = 1;

}

}

}\*/

fseek(Fpaciente,tam,SEEK\_END);

fwrite(p,sizeof(p),1,Fpaciente);

limpaBuffer();

fclose(Fpaciente);

}

//CADASTRO DE MEDICO

void cadastrarMedico(FILE\*Fmedico)

{

int tam=-1;

medico m[90];

Fmedico = fopen("cadastro-medico.dat", "r+b");

if(Fmedico == NULL)

{

printf("Erro");

exit(1);

}

//conta o tamanho do vetor

fseek(Fmedico,0,SEEK\_SET);

if(!feof(Fmedico))

{

tam=0;

}

else

{

while(!feof(Fmedico))

{

fread(m,sizeof(m),1,Fmedico);

tam++;

}

}

printf("Nome: ");

limpaBuffer();

gets(&m[tam].nomeM);

printf("Especialidade: ");

limpaBuffer();

gets(m[tam].especialidade);

printf("Telefone: ");

limpaBuffer();

scanf("%d",&m[tam].telefoneM);

m[tam].codigoM = (rand()%89)+10;

fseek(Fmedico,tam,SEEK\_END);

fwrite(m,sizeof(m),1,Fmedico);

fseek(Fmedico,0,SEEK\_END);

fclose(Fmedico);

}

//LOCALIZA PACIENTE

int localizaPaciente(char nomeP[30])

{

int achou=0, posicao=-1;

pacientes p;

FILE\*Fpaciente;

Fpaciente = fopen("cadastro-paciente.dat", "r+b");

if(Fpaciente == NULL)

{

printf("Erro");

exit(1);

}

fseek(Fpaciente,0,SEEK\_SET);

fread(&p,sizeof(p),1,Fpaciente);

while(!feof(Fpaciente) && !achou )

{

posicao++;

if(strcmp(nomeP,p.nomeP)==0)

{

achou = 1;

}

fread(&p,sizeof(p),1,Fpaciente);

}

//printf("%d\n",posicao);

//pausa();

if(achou)

{

return posicao;

}

else

{

return -1;

}

fclose(Fpaciente);

}

//LOCALIZA MEDICO

int localizaMedico(char nomeM[30])

{

int achou=0, posicao=-1;

medico m;

FILE\*Fmedico;

Fmedico = fopen("cadastro-medico.dat", "r+b");

if(Fmedico == NULL)

{

printf("Erro");

exit(1);

}

fseek(Fmedico,0,SEEK\_SET);

fread(&m,sizeof(m),1,Fmedico);

while(!feof(Fmedico) && !achou )

{

posicao++;

if(strcmp(nomeM, m.nomeM)==0)

{

achou = 1;

}

fread(&m,sizeof(m),1,Fmedico);

}

//printf("%d\n",posicao);

//pausa();

if(achou)

{

return posicao;

}

else

{

return -1;

}

fclose(Fmedico);

}

//CADASTRO DE CONSULTA

void cadastraConsulta(FILE\*Fconsulta)

{

char nomeP[30],nomeM[30];

int posicaoP, posicaoM, tam=-1,aux=0;

FILE \*Fpaciente, \*Fmedico;

consulta c[10000];

pacientes p;

medico m;

printf("Nome do Paciente: ");

limpaBuffer();

gets(nomeP);

printf("Nome do medico: ");

limpaBuffer();

gets(nomeM);

posicaoP = localizaPaciente(nomeP);

posicaoM = localizaMedico(nomeM);

if(posicaoP == -1 || posicaoM == -1 )

{

printf("Pessoa ou Medico nao exitem\n");

pausa();

}

else

{

Fconsulta = fopen("cadastro-consulta.dat","r+b");

if(Fconsulta == NULL)

{

printf("Erro");

exit(1);

}

//conta o tamnho da consulta

fseek(Fconsulta,0,SEEK\_SET);

if(!feof(Fconsulta))

{

tam=0;

}

else

{

while(!feof(Fconsulta))

{

fread(c,sizeof(c),1,Fconsulta);

tam++;

}

}

printf("DIA da consulta: ");

limpaBuffer();

scanf("%d",&c[tam].dataC[0]);

printf("MES da consulta: ");

limpaBuffer();

scanf("%d",&c[tam].dataC[1]);

printf("ANO da consulta: ");

limpaBuffer();

scanf("%d",&c[tam].dataC[2]);

printf("Hora marcada de 13h as 18h\n");

while(!aux)

{

printf("HORA da consulta de 13h as 18h: ");

limpaBuffer();

scanf("%d",&c[tam].horaC[0]);

if(c[tam].horaC[0]>= 13 && c[tam].horaC[0]<=18)

{

printf("Consulta feita acada 30 minutos\n");

printf("MINUTOS da consulta: ");

limpaBuffer();

scanf("%d",&c[tam].horaC[1]);

if(c[tam].horaC[1]==0 || c[tam].horaC[1]==30)

{

aux=1;

}else{

printf("Minutos nao valido");

}

}else{

printf("Horario nao valido");

}

}

c[tam].status = 1;

c[tam].codigoC =(rand() % 9999) + 10000;

//salvando codigo do paciente

Fpaciente = fopen("cadastro-paciente.dat","r+b");

if(Fpaciente == NULL)

{

printf("Erro na abertura\n");

exit(1);

}

fseek(Fpaciente,posicaoP,SEEK\_SET);

fread(&p,sizeof(p),1,Fpaciente);

strcpy(c[tam].nomeP,p.nomeP);

c[tam].codigoP = p.codigoP;

fclose(Fpaciente);

//salvando codigo no medico

Fmedico = fopen("cadastro-medico.dat","r+b");

if(Fmedico == NULL)

{

printf("Erro na abertura\n");

exit(1);

}

fseek(Fmedico,posicaoM,SEEK\_SET);

fread(&m,sizeof(m),1,Fmedico);

strcpy(c[tam].nomeM,m.nomeM);

c[tam].codigoM = m.codigoM;

fclose(Fmedico);

fseek(Fconsulta,tam,SEEK\_SET);

fwrite(&c,sizeof(c),1,Fconsulta);

fclose(Fconsulta);

}

pausa();

}

//RELATORIOS DO DIA

void relatorioDia(FILE\*Fconsulta)

{

int dia, mes, ano;

consulta c;

printf("DIA: ");

limpaBuffer();

scanf("%d",&dia);

printf("MES: ");

limpaBuffer();

scanf("%d",&mes);

printf("ANO: ");

limpaBuffer();

scanf("%d",&ano);

Fconsulta = fopen("cadastro-consulta.dat","r+b");

if(Fconsulta == NULL)

{

printf("Erro");

exit(1);

}

fseek(Fconsulta,0,SEEK\_SET);

printf("\t\tData: %d/%d/%d\n",dia,mes,ano);

while(!feof(Fconsulta))

{

fread(&c,sizeof(c),1,Fconsulta);

if(dia == c.dataC[0])

{

if(mes == c.dataC[1])

{

if(ano == c.dataC[2])

{

if(c.status != 0)

{

printf("Nome do paciente: %s\tCodigo: %d",c.nomeP,c.codigoP);

printf("\tNome do medico: %s\tCodigo: %d",c.nomeM,c.codigoM);

printf("\tCodigo da consulta: %d\n\n",c.codigoC);

}

}

}

}

}

pausa();

fclose(Fconsulta);

}

//RELATORIO PACIENTE

void relatorioPaciente(FILE\*Fconsulta)

{

char nomeP[30];

consulta c;

printf("Nome do paciente: ");

limpaBuffer();

gets(nomeP);

Fconsulta = fopen("cadastro-consulta.dat","r+b");

if(Fconsulta == NULL)

{

printf("Erro");

exit(1);

}

fseek(Fconsulta,0,SEEK\_SET);

printf("Consultas do %s\n",nomeP);

while(feof(Fconsulta))

{

fread(&c,sizeof(c),1,Fconsulta);

if(strcmp(nomeP,c.nomeP) == 0)

{

if(c.status != 0)

{

printf("Data: %d/%d/%d",c.dataC[0],c.dataC[2],c.dataC[2]);

printf("\tCodigo da consulta: %d",c.codigoC);

printf("\tNome do medico: %s",c.nomeM);

}

}

}

pausa();

fclose(Fconsulta);

}

//RELATORIO DO MEDICO

void relatorioMedico(FILE\*Fconsulta)

{

char nomeM[30];

consulta c;

printf("Nome do medico: ");

limpaBuffer();

gets(nomeM);

Fconsulta = fopen("cadastro-consulta.dat","r+b");

if(Fconsulta == NULL)

{

printf("Erro");

exit(1);

}

fseek(Fconsulta,0,SEEK\_SET);

while(!feof(Fconsulta))

{

fread(&c,sizeof(c),1,Fconsulta);

if(strcmp(nomeM,c.nomeM) == 0)

{

if(c.status != 0)

{

printf("Data %d/%d/%d\t",c.dataC[0],c.dataC[1],c.dataC[2]);

printf("Codigo da cosulta: %d",c.codigoC);

}

}

}

pausa();

fclose(Fconsulta);

}

//CANCELA CONSULTA

void cancelaConsulta(FILE\*Fconsulta)

{

int codigo, aux;

consulta c;

printf("Digite o codigo da consulta");

limpaBuffer();

scanf("%d",&codigo);

Fconsulta = fopen("cadastro-consulta.dat","r+b");

if(Fconsulta == NULL)

{

printf("Erro");

exit(1);

}

fseek(Fconsulta,0, SEEK\_SET);

while(feof(Fconsulta))

{

fread(&c,sizeof(c),1,Fconsulta);

if(codigo == c.codigoC)

{

printf("Para cancelar consulta aperte 0.");

limpaBuffer();

scanf("%d",&aux);

if(aux == 0)

{

c.status = 0;

printf("Consulta cancelada");

}

}

else

{

printf("Consulta nao econtrada");

}

}

pausa();

fclose(Fconsulta);

}

//FUNÇÃO EXTRA

void listadeMedicos(FILE\*Fmedico)

{

medico m;

Fmedico = fopen("cadastro-medico.dat", "r+b");

if(Fmedico == NULL)

{

printf("Erro");

exit(1);

}

fseek(Fmedico,0,SEEK\_SET);

while(!feof(Fmedico))

{

fread(&m,sizeof(m),1,Fmedico);

printf("\n Nome: %s\t",m.nomeM);

printf("Codigo: %d\n",m.codigoM);

}

pausa();

fclose(Fmedico);

}