UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO
Curso: Bacharelado em Engenharia da Computação

Relatório do sistema de gerenciamento da clínica

Disciplina: Algoritmos e Estrutura de Dados I
Professor: George Felipe Fernandes Vieira
Alunos: João Pedro Fernandes de Aquino;
Frederico Gregório Emídio dos Santos

Pau dos Ferros - RN
02/06/2025

Sumário

8. Arquivo "Auxiliar.c" – Funções Utilitárias

8.1. Gerenciamento de Arquivos

8.1.1. verificarArquivo

8.2. Controle de Entradas

8.2.1. entradaLimitada

8.3. Geração de IDs

8.3.1. gerarNovoId

8.4. Conversão de Enums

8.4.1. especialidadeParaTexto

8.4.2. statusConsultaParaTexto

8.5. Carregamento de Dados

8.5.1. carregarMedicos

8.5.2. carregarPacientes

8.6. Operações de Busca

8.6.1. buscaPacienteCPF

8.6.2. buscaPacienteId

8.6.3. buscaMedicoCRM

8.6.4. buscaMedicoId

8.7. Exibição de Registros

8.7.1. exibirMedico

8.7.2. exibirPaciente

8.7.3. exibirConsulta

8.8. Validação de Dados

8.8.1. validarCPF

8.8.2. CPFJaCadastrado

8.8.3. receberCPF

8.9. Entrada de Usuário

8.9.1. receberEspecialidade

8.9.2. validarNome

8.9.3. receberNome

8.9.4. validarContato

8.9.5. receberDataHora

1. Introdução

O sistema implementado permite o gerenciamento do sistema interno de uma clínica a partir de cadastros, atualizações, e remoções de dados de pacientes, médicos e consultas. Além disso, o sistema possibilita a geração de relatórios que possam ser do interesse da clínica a partir de parâmetros pré-estabelecidos. O sistema usa uma arquitetura modular dividida em:

- *tipos.h*: Que contém toda as estruturas de dados, e enums utilizados.
- *Main.c*: Menu inicial
- *medicos.c*: Gerenciamento de médicos
- *pacientes.c*: Gerenciamento de pacientes
- *consultas.c*: Agendamento e gestão de consultas
- *relatorios.c*: Geração de relatórios analíticos
- *auxiliar.c*: Funções utilitárias ao código, incluindo: exibição de estruturas; verificação e criação de arquivos; recebimento e validação de entradas das estruturas; conversão de tipos enums para texto, e geração de id.

Arquivos (*Medicos.txt*, *Pacientes.txt*, *Consultas.txt*) no diretório *Arquivos.* Diretório *Arquivos/Relatorios* para armazenar relatórios.

2. Arquivo "tipos.h" – Estruturas de dados utilizadas

2.1. Médicos

A estrutura *Medico* armazena informações sobre os médicos que atendem no sistema de saúde. Cada médico possui um identificador único (*id*), um nome (*nome*), um número de CRM (*crm*) e informações de contato (*contato*). Além disso, a especialidade do médico é definida por uma *enum* (*especialidade*), que pode ser uma das seguintes: *CLINICO*, *PEDIATRA*, *CARDIOLOGISTA*, *DERMATOLOGISTA*, *PSIQUIATRA* ou *ORTOPEDISTA*. Essa categorização permite que o sistema identifique rapidamente a área de atuação de cada médico.

2.2.Pacientes

A estrutura *Paciente* é responsável por armazenar informações essenciais sobre os pacientes. Cada paciente possui um identificador único (*id*), um nome (*nome*), um CPF (*cpf*) e informações de contato (*contato*).

2.3. Consultas

A estrutura *Consulta* é utilizada para gerenciar as consultas médicas. Cada consulta possui um identificador único (*id*), o identificador do paciente (*pacienteId*) e o identificador do médico (*medicoId*). A data e hora da consulta são armazenadas em uma estrutura *DataHora*, que inclui o dia (*dia*), mês (*mes*), ano (*ano*), hora (*hora*) e minuto (*minuto*). O status da consulta é definido por uma *enum* (*status*), que pode ser *AGENDADA*, *REALIZADA* ou *CANCELADA*. Essa estrutura permite que o sistema acompanhe o agendamento e a realização das consultas de forma organizada.

2.4 DataHora

A estrutura *DataHora* é responsável por armazenar a data e hora de uma consulta. Ela inclui o dia (*dia*), mês (*mes*), ano (*ano*), hora (*hora*) e minuto (*minuto*). Esses dados são utilizados dentro do *struct consultas* (1.1.4).

3. Arquivo "Main.c" – O menu inicial

Inclui bibliotecas padrão do C, como <stdio.h> e <stdlib.h>, além de cabeçalhos locais: "tipos.h", "Medico.h", "Paciente.h", "Consulta.h" e "Relatorios.h", que definem estruturas de dados e funções específicas para cada módulo do sistema.

A função única do arquivo mplementa um menu interativo para o sistema de gerenciamento clínico. Utiliza um loop *do-while* para manter o programa em execução até que o usuário selecione a opção de sair (0). A seleção é feita com *scanf()* e tratada por um *switch-case*, que direciona para os módulos: *menuMedico()*, *menuPaciente()*, *menuConsultas()* e *menuRelatorios()*. Cada função é responsável pela navegação e operações do respectivo módulo.

4. Arquivo "medico.c" - Gerenciamento de médicos

Os *includes* permitem as operações do módulo: <*stdio.h*> e <*stdlib.h*> habilitam entradas, saídas e gestão de memória. A biblioteca <*string.h*> fornece manipulação segura de strings, enquanto os cabeçalhos locais *"tipos.h"*, *"Medico.h"* e *"Auxiliar.h"* definem estruturas de dados médicas, declaram funções específicas e compartilham funções utilitárias.

4.1 *salvarMedicos*

Abre o arquivo de médicos no modo especificado (*"w"* para sobrescrever ou *"a"* para adicionar). Quando no modo *"w"*, escreve o cabeçalho das colunas primeiro usando *fprintf()*. Para cada médico no array fornecido, verifica se precisa gerar um novo ID (quando *id = 0*) usando *gerarNovoId()*. Escreve os dados de cada médico no arquivo no formato CSV com *fprintf()*, incluindo *ID*, *nome*, *CRM*, *especialidade* e *contato*. Finalmente fecha o arquivo com *fclose()*.

4.2.*cadastrarMedico*

Para o cadastro, essa função aloca memória para uma nova estrutura *Medico* usando *malloc(), i*nicializa o *ID* como 0 e coleta os dados do médico através das funções auxiliares: *receberNome()*, *receberCRM()*, *receberEspecialidade()* e *receberContato()*, que validam cada entrada. Exibe os dados coletados com *exibirMedico()* e pede confirmação via *scanf()*. Se confirmado, chama *salvarMedicos()* no modo *"a"* (append). Libera a memória alocada com *free()* ao final.

4.3. *listarMedicos*

Carrega todos os médicos do arquivo para memória usando *carregarMedicos()*, que retorna um array dinâmico. Se não houver médicos cadastrados, exibe mensagem com *printf()*. Caso contrário, percorre o array e exibe cada médico com *exibirMedico()*. Libera a memória do array com *free()*.

4.4. *atualizarMedico*

Essa função arrega todos os médicos em um array dinâmico via *carregarMedicos(), s*olicita o *CRM* do médico a ser atualizado com *receberCRM()* e busca no array usando *strcmp()*. Se encontrado, permite selecionar campos para atualizar (usando *scanf()* para entrada), reutilizando as funções de validação do cadastro. Após alterações, salva todos os médicos com *salvarMedicos()* no modo *"w"*. Libera a memória com *free()*.

4.5. *removerMedico*

[Essa função arrega médicos em array dinâmico via *carregarMedicos(), s*olicita *CRM* com *receberCRM()* e busca no array. Se encontrado, pede confirmação via *scanf()*. Se confirmado, aloca novo array com *malloc*(tamanho original - 1) e copia todos os registros exceto o removido. Salva o novo array com *salvarMedicos()* no modo *"w"* e libera memória de ambos arrays com *free()*.

4.6. *menuMedico*

Essa função exibe um menu com *printf()* contendo opções numéricas. Usa *do-while* com *scanf()* para manter o menu ativo até seleção de saída (opção 0). Para opções válidas, chama *cadastrarMedico()*, *atualizarMedico()*, *removerMedico()* ou *listarMedicos()*.

5. Arquivo "paciente.c" – Gerenciamento de pacientes

Os *includes* permitem as operações do módulo: <*stdio.h*> e <*stdlib.h*> habilitam entradas, saídas e gestão de memória. A biblioteca <*string.h*> fornece manipulação segura de strings, enquanto os cabeçalhos locais *"tipos.h"*, *"Paciente.h"* e *"Auxiliar.h"* definem estruturas de dados dos pacientes, declaram funções específicas e compartilham funções utilitárias.

5.1. *salvarPacientes*

Abre o arquivo de pacientes no modo especificado (*"w"* para sobrescrever ou *"a"* para adicionar). Quando no modo *"w"*, escreve o cabeçalho das colunas primeiro usando *fprintf()*. Para cada paciente no array fornecido, verifica se precisa gerar um novo ID (quando *id = 0*) usando *gerarNovoId()*. Escreve os dados de cada paciente no arquivo no formato CSV com *fprintf()*, incluindo ID, nome, CPF, data de nascimento e contato. Finalmente, fecha o arquivo com *fclose()*.

5.2 *cadastrarPaciente*

Para o cadastro, essa função a loca memória para uma nova estrutura *Paciente* usando *malloc(), i*nicializa o ID como 0 e coleta os dados do paciente através das funções auxiliares: *receberNome()*, *receberCPF()*, *receberDataNascimento()* e *receberContato()*, que validam cada entrada. Exibe os dados coletados com *exibirPaciente()* e pede confirmação via *scanf()*. Se confirmado, chama *salvarPacientes()* no modo *"a"* (append). Libera a memória alocada com *free()* ao final.

5.3 *listarPacientes*

Carrega todos os pacientes do arquivo para memória usando *carregarPacientes()*, que retorna um array dinâmico. Se não houver pacientes cadastrados, exibe mensagem com *printf()*. Caso contrário, percorre o array e exibe cada paciente com *exibirPaciente()*. Libera a memória do array com *free()*.

5.4 *atualizarPaciente*

Essa função carrega todos os pacientes em um array dinâmico via *carregarPacientes(),* solicita o CPF do paciente a ser atualizado com *receberCPF()* e busca no array usando *strcmp()*. Se encontrado, permite selecionar campos para atualizar (usando *scanf()* para entrada), reutilizando as funções de validação do cadastro. Após alterações, salva todos os pacientes com *salvarPacientes()* no modo *"w"*. Libera a memória com *free()*.

5.5 *removerPaciente*

Essa função carrega pacientes em array dinâmico via *carregarPacientes()*. Solicita CPF com *receberCPF()* e busca no array. Se encontrado, pede confirmação via *scanf()*. Se confirmado, aloca novo array com *malloc(tamanho original - 1)* e copia todos os registros exceto o removido. Salva o novo array com *salvarPacientes()* no modo *"w"* e libera memória de ambos arrays com *free()*.

5.6 *menuPaciente*

Essa função exibe menu com *printf()* contendo opções numéricas. Usa *do-while* com *scanf()* para manter o menu ativo até seleção de saída (opção 0). Para opções válidas, chama *cadastrarPaciente()*, *atualizarPaciente()*, *removerPaciente()* ou *listarPacientes()*.

6. Arquivo “relatorios.c” - Geração de relatórios analíticos

O cabeçalho do código contém as inclusões essenciais para o funcionamento do módulo de relatórios do sistema. São incluídas as bibliotecas padrão stdio.h (entrada e saída), stdlib.h (alocação dinâmica), string.h (manipulação de strings) e time.h (operações com data e hora). Além disso, são incluídos os arquivos personalizados tipos.h, Relatorios.h, Auxiliar.h e

Consulta.h, que definem estruturas, funções auxiliares e funcionalidades relacionadas ao gerenciamento de consultas médicas.

6.1. *relatorioConsultaPacientes*

Essa função gera um relatório com todas as consultas realizadas por um paciente específico. Primeiramente, solicita o CPF do paciente com a função *receberCPF*. Em seguida, carrega todas as consultas do arquivo utilizando *carregarConsultas*. Percorre o vetor de consultas e, para cada uma cujo campo cpfPaciente coincida com o CPF informado, exibe os dados utilizando *exibirConsulta*. Ao final, libera a memória alocada com *free*.

6.2. *relatorioConsultaMedicos*

Essa função exibe todas as consultas associadas a um determinado médico. O CRM é solicitado com *receberCRM*, e as consultas são carregadas com *carregarConsultas*. O vetor de consultas é percorrido e, para cada consulta cujo crmMedico corresponda ao CRM informado, os dados são exibidos com *exibirConsulta*. A função libera a memória ao final da execução.

6.3. *relatorioConsultaPorEspecialidae*

Essa função contabiliza a quantidade de consultas realizadas por especialidade médica. Após carregar todas as consultas com *carregarConsultas*, a função inicializa um vetor de contadores correspondente às especialidades definidas no enum Especialidade. Em seguida, percorre o vetor de consultas, incrementando o contador da especialidade de cada consulta. Os resultados são gravados no arquivo “Relatorio_Especialidades.txt”, com o nome da especialidade e o total de consultas. A memória utilizada é liberada ao final com *free*.

6.4. *relatorioConsultaDiaAtual*

Essa função exibe todas as consultas agendadas para a data atual. Utiliza a biblioteca time.h para obter a data do sistema, formatando-a no padrão dd/mm/yyyy. Carrega todas as consultas com *carregarConsultas* e compara a data de cada consulta com a data atual. As consultas correspondentes são exibidas usando *exibirConsulta*. A memória alocada é liberada após o término da função.

6.5. *menuRelatorios*

Essa função exibe o menu principal para geração de relatórios. O menu apresenta opções para listar consultas por paciente, por médico, por especialidade e por data atual. A escolha do usuário é capturada com *scanf* e, conforme a opção selecionada, a função correspondente é executada. O menu permanece ativo até que o usuário escolha retornar.

7. Arquivo “consulta.c” - Agendamento e gestão de consultas

O cabeçalho do código contém as *inclusões* essenciais para o funcionamento do sistema de consultas médicas. São incluídas as bibliotecas padrão de entrada e saída (*stdio.h*), manipulação de strings (*string.h*) e alocação dinâmica de memória (*stdlib.h*). Além disso, há a inclusão dos arquivos de cabeçalho personalizados — *tipos.h*, *Relatorios.h*, *Consulta.h* e *Auxiliar.h* — que definem as estruturas, funções auxiliares e relatórios necessários para o correto funcionamento das operações relacionadas às consultas médicas.

7.1. *carregarConsultas*

Essa função é responsável por carregar as consultas armazenadas no arquivo "Consultas.txt". Para isso, ela abre o arquivo utilizando a *função fopen* no modo leitura. Caso o arquivo não seja encontrado, ela define o total de consultas como zero e o ponteiro de consultas como *NULL*, encerrando a execução. Em seguida, a função pula o cabeçalho da primeira linha com a *função fgets*. Para cada linha de dados do arquivo, ela lê os campos usando a *função fscanf* e armazena temporariamente em uma estrutura do tipo Consulta. Para cada nova consulta lida, a função realoca dinamicamente o vetor de consultas com

realloc, adiciona a consulta ao vetor e incrementa o contador total. No final, o arquivo é fechado com *fclose*.

7.2. salvarConsultas

Essa função salva as consultas em um arquivo texto. Ela abre o arquivo "Consultas.txt" usando *fopen* no modo especificado por seu parâmetro, que pode ser escrita ("w") ou anexação ("a"). Se o modo for escrita, a função sobrescreve o arquivo e grava o cabeçalho com *fprintf*. Depois, ela percorre todas as consultas passadas e grava cada uma no arquivo, formatando os dados com *fprintf*. Finalmente, o arquivo é fechado com *fclose*.

7.3. alterarStatusConsulta

Essa função permite alterar o status de uma consulta já cadastrada. Primeiramente, ela solicita ao usuário o ID da consulta que deseja alterar utilizando a *função scanf*. Em seguida, carrega todas as consultas com a *função carregarConsultas*. Percorre a lista de consultas buscando pelo ID informado e, se encontrado, exibe os dados da consulta com a *função exibirConsulta*. O usuário é então perguntado se deseja alterar o status para o novo valor, que é exibido em texto através da *função statusConsultaParaTexto*. Se o usuário confirmar a alteração, a consulta tem seu status atualizado e as consultas são salvas no arquivo usando a *função salvarConsultas* no modo escrita. Caso o usuário opte por não salvar, a alteração é descartada. Por fim, a memória alocada para as consultas é liberada com *free*.

7.4. listarConsultas

Essa função exibe a lista completa das consultas cadastradas. Ela carrega as consultas do arquivo chamando a *função carregarConsultas*, percorre todas as consultas e exibe cada uma usando a *função exibirConsulta*. Ao final, libera a memória alocada.

7.5. agendarConsulta

Essa função é responsável por cadastrar uma nova consulta. Primeiro, ela verifica se o arquivo "Consultas.txt" existe e tem cabeçalho com a *função verificarArquivo*. Em seguida, recebe o CPF do paciente através da *função receberCPF* e procura o paciente no cadastro com *buscaPacienteCPF*. Caso o paciente não exista, a função exibe uma mensagem de erro e termina. O mesmo processo é feito para o CRM do médico, com *receberCRM* e *buscaMedicoCRM*. Se ambos existirem, o status da nova consulta é definido como *AGENDADA*. A função então coleta a data e hora da consulta com *receberDataHora*. Em seguida, exibe um resumo da consulta, mostrando os dados do paciente (obtidos por *buscaPacienteId*) e do médico (com *buscaMedicoId* e *exibirMedico*), além da data e hora. O usuário deve confirmar se deseja salvar a consulta; se confirmado, um novo ID é gerado com *gerarNovoId* e a consulta é salva no arquivo usando *salvarConsultas* em modo anexação ("a"). Caso contrário, o agendamento é cancelado.

7.6. *menuConsultas*

Essa função exibe o menu principal para o usuário gerenciar as consultas. Ela apresenta opções para agendar uma consulta, cancelar uma consulta, registrar uma consulta realizada, listar consultas por paciente, listar consultas por médico, e sair. A escolha do usuário é capturada com scanf e, dependendo da opção, chama as funções correspondentes: agendarConsulta, alterarStatusConsulta com status CANCELADA, alterarStatusConsulta com status REALIZADA, relatorioConsultaPacientes ou relatorioConsultaMedicos. O menu permanece ativo até que o usuário escolha sair.

8. Arquivo “Auxiliar.c" - As funções utilitárias do código

Este arquivo funciona como uma caixa de ferramentas que sustenta e da suporte a todo sistema. Nele estão concentradas as funções que garantem o funcionamento do programa.

8.1. Gerenciamento de Arquivos

8.1.1 *verificarArquivo*

Utiliza operações de arquivo (*fopen, fprintf, fclose*) para criar arquivos com cabeçalhos quando não existem. Implementa verificação de existência antes da criação para evitar sobrescritas acidentais.

8.2. Controle de Entradas

8.2.1. entradaLimitada

Manipula entrada de caracteres via *getchar()* com controle preciso de *buffer*. Usa ponteiros para gravar caracteres diretamente na memória, garantindo terminação adequada com '\0' e prevenindo overflow.

8.3. Geração de IDs

8.3.1. *gerarNovoId*

Combina leitura de arquivo (f*gets*) com leitura de *strings* (*sscanf)* para identificar o maior ID existente. Implementa lógica de incremento sequencial com tratamento para arquivos vazios ou inválidos.

8.4. Conversão de *Enums*

8.4.1. *especialidadeParaTexto*

Mapeia valores enumerados para strings usando ponteiros de retorno. Não requer alocação dinâmica, operando como tabela de consulta estática.

8.4.2. *statusConsultaParaTexto*

Similar à anterior, converte códigos de status em descrições legíveis através de strings.

8.5. *Carregamento de Dados*

8.5.1. *carregarMedicos*

Gerencia memória dinâmica com realloc para vetores expansíveis. Lê arquivos com *fscanf* e converte dados brutos para estruturas.

8.5.2. *carregarPacientes*

Opera com ponteiros duplos para modificar vetores diretamente. Implementa leitura segura com delimitadores específicos para campos de tamanho variável.

8.6. Operações de Busca

8.6.1. *buscaPacienteCPF*

Realiza busca sequencial em arquivos com *fscanf*. Usa ponteiros para *strings* em comparações diretas com *strcmp*, retornando IDs sem carregar toda a estrutura.

8.6.2. *buscaPacienteId()*

Retorna *structs* completas copiando dados para memória temporária, e utiliza strcpy para transferência segura entre *buffer*s.

8.6.3. *buscaMedicoCRM()*

Combina manipulação de arquivos e *strings* para localizar CRMs. Otimiza desempenho com leitura sequencial direta do disco.

8.6.4. *buscaMedicoId()*

Converte dados de arquivo para estruturas Médico com casting direto de *enums*. Essa função copia *strings* para membros da struct via ponteiros.

8.7. Exibição de Registros

8.7.1. *exibirMedico()*

Formata saída combinando dados primitivos e strings convertidas. Acessa membros de *struct* diretamente via operador ponto.

8.7.2. *exibirPaciente()*

Exibe dados formatados sem processamento adicional a partir de acesso direto a membros de *struct*.

8.7.3. *exibirConsulta*

Exibe dados de Consultas com algumas formatações. Formata datas numéricas para exibição e converte status via ponteiro para função de conversão para texto.

8.8. Validação de Dados

8.8.1*. validarCPF*

Verifica tamanho e composição com *strlen* e *isdigit.* Opera diretamente em *buffer*s de caracteres via ponteiros.

8.8.2*. CPFJaCadastrado*

Percorre arquivos com *fscanf* comparando *strings*. Gerencia recursos de arquivo com abertura e fechamento explícitos.

8.8.3*. receberCPF*

Implementa loop de validação combinando entradaLimitada, validarCPF e CPFJaCadastrado. Usa ponteiros para modificação direta do *buffer*.

8.9. Entrada de Usuário

8.9.1. *receberEspecialidade*

Utiliza ponteiros para modificar *enums* diretamente. Implementa menu interativo com prevenção de entrada inválida.

8.9.2. *validarNome*

Analisa *strings* com combinação de *isspace* e *isalpha, e v*erifica composição caractere-por-caractere via indexação.

8.9.3 *receberNome*

Integra *entradaLimitada* com *validarNome* em loop e opera com passagem por ponteiro para modificação direta.

8.9.4. validarContato

Combina verificação de tamanho e composição numérica com *isdigit().*

8.9.5 *receberDataHora*

Usa *fgets* para entrada segura e *sscanf* para leitura. Implementa validação em múltiplas camadas (formato, data, hora) com estruturas passadas por ponteiro.