

UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO

ALGORITMOS E ESTRUTURA DE DADOS 1

Relatório 1 – Sistema de Gerenciamento para Clínica Médica

Anderson Carlos da Silva Moraes

Marília Fonseca Andrade

Pau dos Ferros - RN

01/06/2025

Arquitetura e Organização

Para entendimento do projeto, a arquitetura e organização do projeto se encontram divididas em: Fluxograma, Estrutura de arquivo e lógica e funcionamento dos módulos, que são apresentadas em sequência.

Fluxograma

O projeto é construído seguindo uma lógica de máquina de estado em que cada módulo é representado por um estado próprio e, a partir da entrada fornecida pelo usuário, o estado pode ser mudado. Na Figura 2, uma ilustração das mudanças de estados:

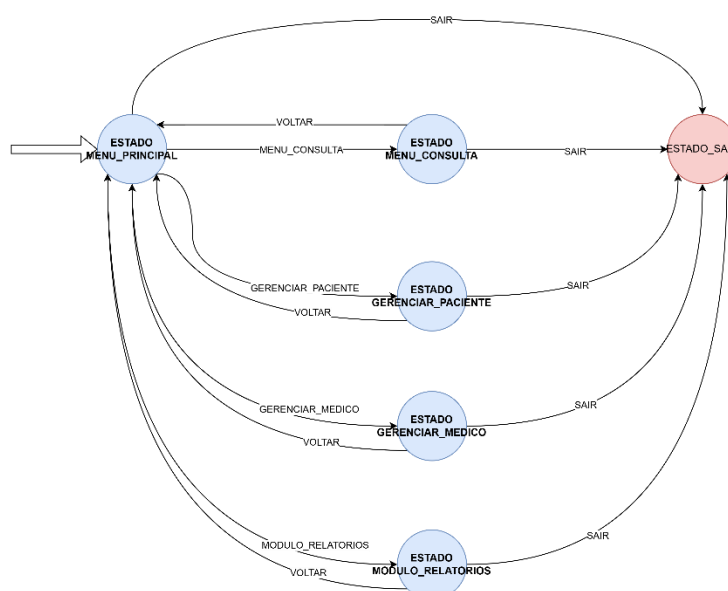


Figura 1. Lógica de transição de estados do algoritmo. Autoria própria.

Além da máquina de estados, o projeto foi organizado e elaborado em um fluxograma geral, representando as principais funcionalidades e blocos de códigos desenvolvidos, sendo acessível no link fornecido abaixo.

- Link de acesso:

<https://drive.google.com/file/d/1DoM9ObVXIDKSTWdcG0hmHc36eUR4g9HZ/view?usp=sharing>

Todos os fluxogramas são compostos por componentes fundamentais descritos abaixo e apresentados na Figura 1.

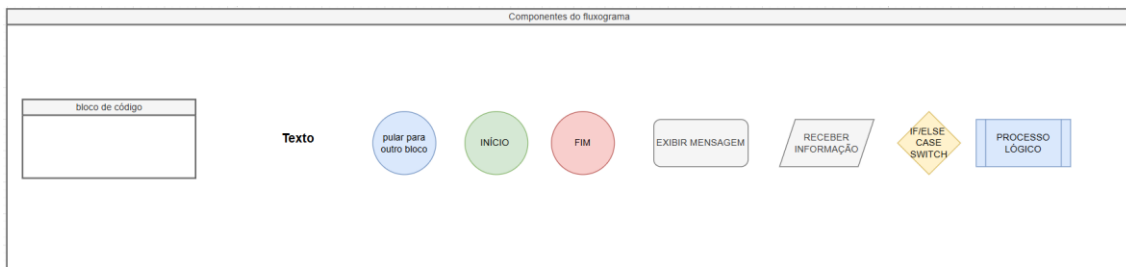


Figura 2: Componentes do fluxograma. Autoria própria.

- Bloco de código: quadro que organiza funções e módulos relevantes ao projeto;
- Texto: informações necessárias ao projeto;
- Pular para outro bloco: permite acessar outro trecho de código;
- Início: início do fluxo de código;
- Fim: fim do fluxo de código;
- Exibir mensagem: bloco de resposta ao usuário;
- Receber informação: bloco de captação de resposta do usuário;
- If/else case switch: bloco para tomada de decisão;
- Processo lógico: funcionalidade específica, podendo ser também a chamada de um outro bloco de código.

Segue abaixo as representações dos fluxogramas desenhados para os principais módulos do projeto.

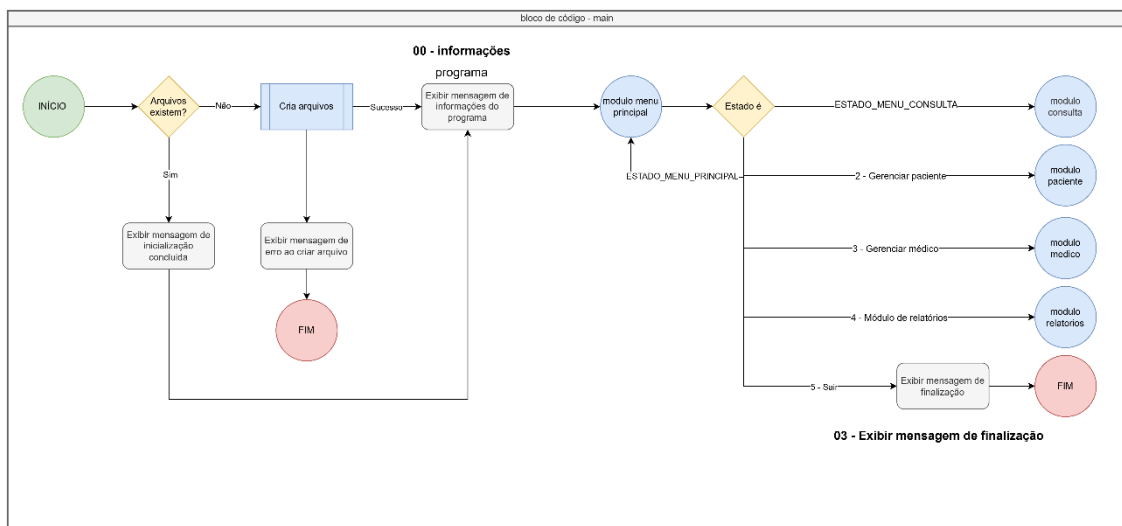


Figura 3. Fluxograma do bloco main. Autoria própria

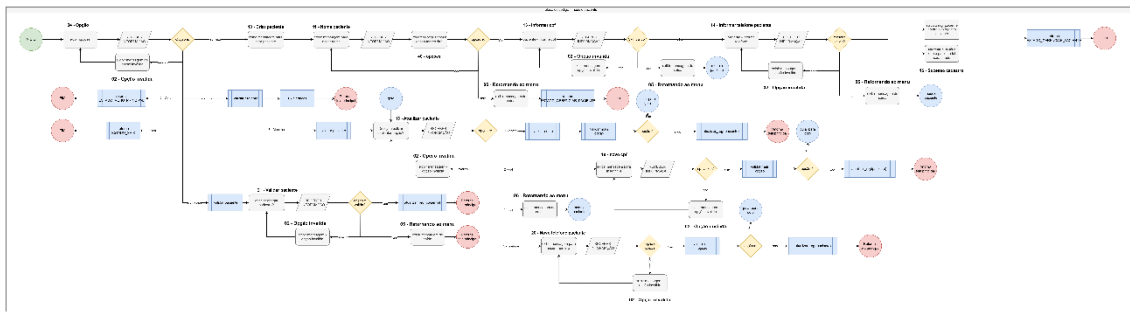


Figura 7. Fluxograma do módulo gerenciar pacientes. Autoria própria.

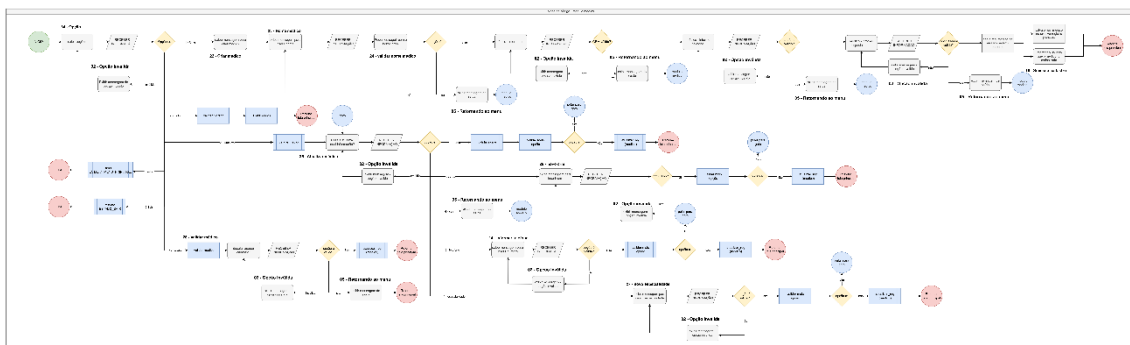


Figura 8: Fluxograma do módulo gerenciar médicos. Autoria própria.

Cada módulo faz uso de recursos compartilhados quanto de trechos específicos presentes apenas em seu módulo. É importante destacar que o fluxograma não representa de forma totalmente fiel e detalhada a implementação em código, visto as limitações na representação gráfica, alguns passos podem estar ausentes, simplificados ou, com etapas adicionais. O objetivo principal dos fluxogramas é ilustrar a lógica geral e as ideias centrais do funcionamento do código, e não servir como uma representação exata da implementação.

Estrutura de Arquivos

A organização dos arquivos pode ser vista pelo link do projeto fornecido abaixo e ilustrada nas Figuras 9 e 10.

Link para o github do projeto: [marifa16/Trabalho1 estrutura de dados 1](https://github.com/marifa16/Trabalho1 Estrutura de dados 1)

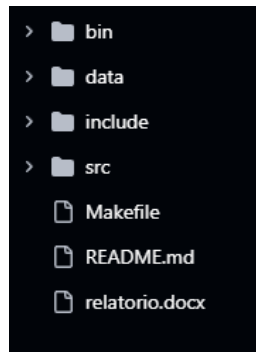


Figura 9: Organização de arquivos do diretório principal do projeto. Autoria própria.

Descrevendo os elementos do diretório principal:

- /bin: destinada ao arquivo executável gerado pelo projeto, representado pelo executável main;
- /data: diretório para armazenar os arquivos csv persistentes do projeto, com os arquivos: registro_consultas.csv, registro_medicos.csv e registro_pacientes.csv;
- /include: armazena os arquivos de cabeçalho (.h) usados para declarar os protótipos das funções, structs, enum e diretivas para melhor usabilidade e legibilidade do código, sendo ilustrados na Figura 8;
- /src: contém os arquivos fonte (.c) com os códigos e funcionalidades do projeto, ilustrados na Figura 8;
- Makefile: arquivo responsável pelo processo de compilação de todos os arquivos do projeto, gerando o executável principal e inicializando a execução do arquivo através da chamada da função main.c;
- README.md: arquivo de documentação do projeto com breve descrição do projeto e como se usar;
- relatorio.pdf / relatório.docx: documento de relatório do projeto.

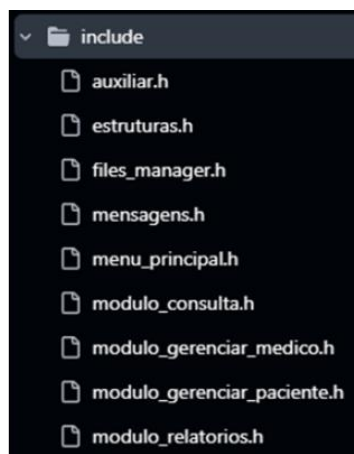
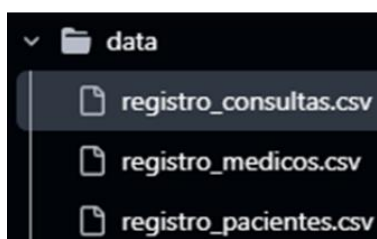
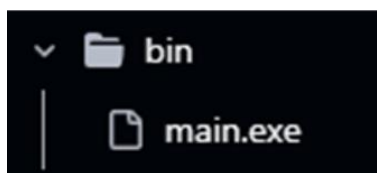


Figura 10: Arquivo e organização dos demais diretórios do projeto. Autoria própria.

Conceitos e estruturas

Essa seção é dedicada a descrever qual a lógica e necessidade de cada arquivo do projeto por diretório e arquivo, as funcionalidades são apresentadas na seção seguinte. Devido à complexidade e extensão dos arquivos, a descrição busca apresentar o propósito geral de cada módulo, destacando suas responsabilidades no funcionamento do sistema.

Include

A maioria dos arquivos no diretório Include consiste em arquivos headers para que o compilador consiga ter as instruções de compilação corretamente, como mencionado anteriormente.

Alguns arquivos de atenção são:

Estruturas.h

O arquivo consta estruturas essenciais para todo o projeto funcionar, como:

- Constantes e variáveis globais.
- ENUMs do projeto: máquina de estado, do status_consulta, Especialidade médica,
- Structs usadas no projeto: reg_consulta, reg_medico, reg_paciente e Horario,

Auxiliar.h

Já o arquivo Auxiliar.h possui as diretivas com o caminho para acessar os arquivos persistentes.

Src

O diretório src concentra todo o código-fonte e a lógica do projeto, apresentando uma variada gama de trechos de código, em que cada um apresenta finalidades distintas, apresentadas a seguir.

Auxiliar.c

- Agrupa funções de suporte, especialmente para validação de dados;
- Garante que as informações estejam no formato correto antes de serem processadas;
- Usado para melhorar a legibilidade dos demais módulos e reutilizar funcionalidades;
- Exemplos de validação: datas, CPF, campos vazios, entre outros.

Files_manager.c

- Possui as funcionalidades de: criar, ler, escrever e deletar os dados de um arquivo, além da inicialização e exibição dos arquivos.

Main.c

- Arquivo principal do projeto, responsável por gerenciar o ciclo de vida da aplicação.
- Principais funções:
 - Inicialização dos recursos.
 - Controle do fluxo do programa por meio de uma máquina de estados, que evita chamadas excessivas na pilha.
 - Direcionamento das ações para os módulos correspondentes, de acordo com as escolhas do usuário.

Mensagens.c

- Centraliza todas as mensagens exibidas ao usuário;
- Garante padronização da interface textual, facilitando eventuais alterações, manutenções ou processos de internacionalização.

Menu_principal.c

- Controla o fluxo e as interações do menu principal do sistema.
- Permite o acesso aos módulos específicos, conforme a seleção do usuário.
- Atua diretamente no estado ESTADO_MENU_PRINCIPAL.

Modulo_consulta.c

- Gerencia as funcionalidades relacionadas às consultas médicas.

- Permite:
 - Cadastrar novas consultas;
 - Cancelar uma consulta com base no usuário, sendo médico ou paciente.

Modulo_gerenciar_medico.c

- Responsável pela gestão dos dados dos médicos;
- Permite operações como: cadastrar, visualizar, editar uma informação e buscar um registro médico.

Modulo_gerenciar_paciente.c

- Responsável por gerenciar as informações dos pacientes.
- Oferece funcionalidades para cadastrar, visualizar, editar uma informação e buscar um registro de paciente.

Modulo_relatorios.c

- Responsável pela **geração de relatórios** com os dados que o sistema possui e na escolha do usuário.
- Permite visualizar: consultas por usuário, consultas por médico, consulta por especialidade, consulta do dia atual.

Funcionalidades

Essa secção consiste apenas em identificar onde cada funcionalidade do relatório é implementada no projeto, otimizando o tempo de correção.

Cadastro médico

Cada médico deve possuir: Nome, CRM, Especialidade e Telefone de contato, permitindo inserir, remover e atualizar os dados do médico. No código é possível visualizar os conceitos na Struct `reg_medico` e dentro da estrutura da função `tratar_modulo_medico()`, ilustrado na Figura 11.

```
// Definição da struct para médicos
typedef struct
{
    int id_medico;
    char nome[100];
    char crm[16];
    Especialidade especialidade_medico;
    char especialidade[100];
    char telefone[12];
    Horario horarios[9];
} reg_medico;
```

```
data > registro_medicos.csv > data
1 id_medico,nome,crm,especialidade,telefone
2 1,Arthur Bernardes,123456,Cardiologista,83991234567
3 2,Beatriz Campos,234567,Pediatra,83992345678
4 3,Caio Dantas,345678,Clinico Geral,83993456789
5 4,Diana Esteves,456789,Dermatologista,83994567890
6 5,Eric Ferreira,567890,Psiquiatra,83995678901
7 6,Flavia Guedes,678901,Cardiologista,83996789012
8 7,Gabriel Holanda,789012,Pediatra,83997890123
9 8,Isabela Jales,890123,Clinico Geral,83998901234
10 9,Jonas Lacerda,901234,Dermatologista,83980123456
11 10,Karina Macedo,112233,Psiquiatra,83981234567
```

```
19 do
20 {
21     msg_38_mostrar_modulo_medico();
22     escolha_modulo = ler_opcao_menu(1, 6);
23
24     switch (escolha_modulo)
25     {
26 > case 1: // Criar medico...
164
165 > case 2: // Exibir medico...
219
220 > case 3: // Atualizar medico...
470
471 > case 4: // Deletar medico...
514
515
516 case 5: // Voltar ao menu principal
517     estado_atual = ESTADO_MENU_PRINCIPAL;
518     break;
519
520 case 6: // Sair do programa
521     estado_atual = ESTADO_SAIR;
522     break;
523
524     default:
525         break;
526 } while (estado_atual == ESTADO_GERENCIAR_MEDICO);
```

Figura 11: Funcionalidades do cadastro médico. Autoria própria.

Cadastro paciente

Cada paciente deve possuir: Nome, CPF e Telefone de contato, permitindo inserir, remover e atualizar os dados do paciente. No código é possível visualizar os conceitos na Struct `reg_paciente` e dentro da estrutura da função `tratar_modulo_paciente()`, ilustrado na Figura 12.

```
// Definição da struct para pacientes
typedef struct
{
    int id_paciente;
    char nome[120];
    char cpf[12]; // Alterado para string (11 dígitos + '\0')
    char telefone[12]; // Alterado para string (11 dígitos + '\0')
} reg_paciente;
```

```
data > registro_pacientes.csv > data
1 id_paciente,nome,cpf,telefone
2 1,Ana Beatriz Costa,11122233344,83999887766
3 2,Bruno Alves Dias,22233344455,83988776655
4 3,Carlos Eduardo Lima,33344455566,83977665544
5 4,Daniela Farias Gomes,44455566677,83966554433
6 5,Eduardo Martins Neto,55566677788,83955443322
7 6,Fernanda Oliveira Pinto,66677788899,83944332211
8 7,Gustavo Pereira Ramos,77788899900,83933221100
9 8,Helena Queiroz Silva,88899900011,83922110099
10 9,Igor Rocha Teixeira,99900011122,83911009988
11 10,Julia Souza Viana,00011122233,83900998877
```

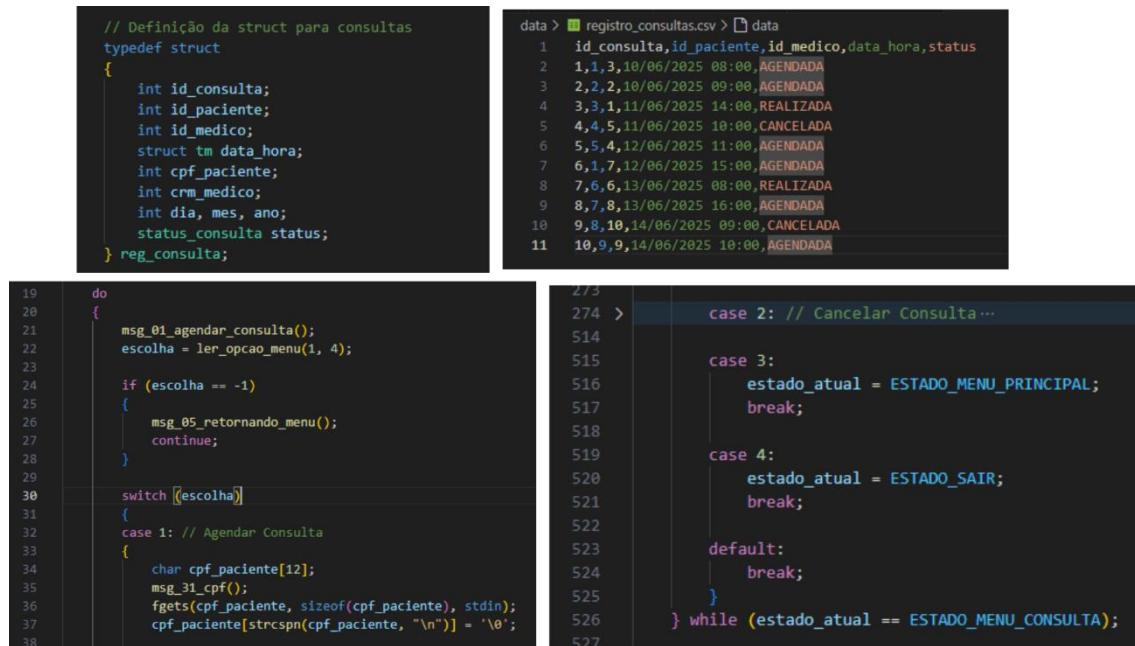
```
do // loop que irá manter o usuário no módulo enquanto o estado permanecer
{
    msg_37_mostrar_modulo_paciente(); // Exibe o menu de gerenciamento de pacientes
    escolha_modulo = ler_opcao_menu(1, 6); // valida se a entrada é válida

    switch (escolha_modulo) // pula para a funcionalidade selecionada
    {
    > case 1: // Criar Paciente...
    > case 2: // Exibir Paciente...
    > case 3: // Atualizar Paciente...
    > case 4: // Deletar Paciente...
    > case 5: // volta ao menu principal...
    > case 6: // sair do código
    {
        estado_atual = ESTADO_SAIR;
        break;
    }
    default: // opção inválida
    {
        msg_02_opcao_invalida();
        break;
    }
    }
```

Figura 12. Funcionalidades do cadastro paciente. Autoria própria.

Gerenciar consultas

Cada consulta deve possuir: Identificador único, Paciente, Médico, Data e hora e Status, também deve permitir agendar, cancelar e registrar consultas. No código é possível visualizar os conceitos na Struct `reg_consulta` e dentro da estrutura da função `tratar_modulo_consulta()`, ilustrado na Figura 13.



The figure consists of four screenshots arranged in a 2x2 grid, illustrating the functionalities of the consultation module.

- Top Left:** A C code snippet defining the `reg_consulta` struct. It includes fields for `id_consulta`, `id_paciente`, `id_medico`, `data_hora` (a `tm` struct), `cpf_paciente`, `crm_medico`, `dia`, `mes`, `ano`, and `status_consulta`.
- Top Right:** A terminal window showing a CSV file named `registro_consultas.csv`. The header row is `id_consulta,id_paciente,id_medico,data_hora,status`. The data rows show various consultations with their respective IDs, patient/doctor IDs, dates/times, and statuses (e.g., `AGENDADA`, `REALIZADA`, `CANCELADA`).
- Bottom Left:** A C code snippet showing a `do-while` loop for a menu. It calls `msg_01_agendar_consulta()` and `ler_opcao_menu(1, 4)`. It includes a `switch` statement for handling menu choices, with a specific case for scheduling a consultation that involves getting patient information.
- Bottom Right:** A C code snippet showing a `switch` statement for handling menu choices. It includes cases for canceling a consultation and returning to the main menu, and a `while` loop that continues to run as long as the current state is `ESTADO_MENU_CONSULTA`.

Figura 13. Funcionalidades do módulo consulta. Autoria própria.

Relatórios

O módulo relatórios precisar realizar as funcionalidades de: listar consultas de um determinado paciente ou médico, gerar relatório com contagem de consultas por especialidade e mostrar as consultas do dia atual. Sendo implementadas na função `tratar_modulo_relatorios()`, ilustrado na Figura 14.

```

9 Estado tratar_modulo_relatorios()
10 {
11     Estado estado_atual = ESTADO_MODULO_RELATORIOS;
12     int escolha_modulo;
13     do
14     {
15         msg_39_mostrar_modulo_relatorios();
16         escolha_modulo = ler_opcao_menu(1, 7);
17
18         switch (escolha_modulo)
19         {
20 >         case 1: // Consultas de pacientes ...
43
44 >         case 2: // Consultas de médicos ...
68
69 >         case 3: // Consulta apenas pela especialidade ...
101
102 >         case 4: // Consultas do dia atual ...
122 >         case 5: // Nova opção: Quantidade de consultas por especialidade ...
128 >         case 6: // Voltar ao menu principal ...
133 >         case 7: // Encerrar o programa ...
138
139         default:
140             msg_02_opcao_invalida();
141             break;
142     } while (estado_atual == ESTADO_MODULO_RELATORIOS);
143 }

```

Figura 14. Funcionalidades do módulo consulta. Autoria própria.

Especificações Técnicas

Ponteiros

Manipulação de variáveis e strings por meio de ponteiros

- Vetores dinâmicos como `reg_medico *medicos` (`src/modulo_gerenciar_medico.c`) e `reg_paciente *pacientes` (`src/main.c`) são declarados como ponteiros, permitindo que a memória seja alocada dinamicamente.
- Funções frequentemente recebem ponteiros para strings (arrays de char), como `const char *nome_arquivo` na função `get_id` em `auxiliar.c`.
- O tipo `FILE *` é um ponteiro usado para identificar e manipular fluxos de arquivos, como visto em `files_manager.c` nas funções `fopen` e `fclose`.

Criação de funções que recebam e retornem ponteiros

- Muitas funções recebem ponteiros como parâmetros para operar sobre dados ou evitar cópias desnecessárias. Por exemplo, `add_row` em `files_manager.c` recebe

char *valores[], que é um array de ponteiros para strings. Exemplificado na Figura 15.

```
// filepath: src/files_manager.c
// ...existing code...
int add_row(const char *nome_arquivo, int select_col, char *valores[])
{
    int novo_id = get_maior_id(nome_arquivo) + 1; // Busca o maior id e incrementa
    // ...existing code...
```

Figura 15. Exemplo de funções que recebem ponteiros. Autoria própria.

- Funções como buscar_paciente_por_cpf em auxiliar.c retornam um ponteiro para uma estrutura reg_paciente, permitindo acesso direto ao registro encontrado. Exemplo na Figura 16.

```
// filepath: src/auxiliar.c
// ...existing code...
reg_paciente *buscar_paciente_por_cpf(const char *cpf)
{
    for (int i = 0; i < total_pacientes; i++)
    {
        if (strcmp(pacientes[i].cpf, cpf) == 0)
            return &pacientes[i]; // Retorna ponteiro para o registro encontrado
    }
    return NULL; // Não encontrou
}
// ...existing code...
```

Figura 16: recorte de buscar_paciente_por_cpf. Autoria própria.

- A liberação de memória alocada dinamicamente é feita usando free, como em main.c ao final da execução do programa, apresentado na Figura 17.

```
// filepath: src/main.c
// ...existing code...
// Libera memória alocada dinamicamente
free(pacientes);
pacientes = NULL;

// Se usar médicos em memória:
extern reg_medico *medicos;
free(medicos);
// ...existing code...
```

Figura 17: uso do free para liberar memória. Autoria própria.

Manipulação de Strings com Ponteiros

Operações básicas com strings

No arquivo `modulo_gerenciar_paciente.c`, dentro do case 2: (Exibir Paciente), é utilizado operação básicas com strings através de ponteiros, ilustrados na Figura 18, em que:

- `strcpy(nome_copia, p->nome);` realiza a **cópia** da string `p->nome` para `nome_copia`. As funções da biblioteca `string.h` como `strcpy` recebem ponteiros para caracteres (`char *`) como argumentos.
- `strcat(mensagem, p->nome);` realiza a **concatenação** da string `p->nome` ao final da string `mensagem`. Similarmente, `strcat` opera com ponteiros.
- `strcmp(p->nome, "Anderson Carlos")` realiza a **comparação** entre a string `p->nome` e a string literal `"Anderson Carlos"`. `strcmp` também utiliza ponteiros.

```
// ...existing code...
// Busca o paciente em memória usando ponteiro
reg_paciente *p = buscar_paciente_por_cpf(cpf_paciente);
if (p == NULL)
{
    msg_paciente_nao_encontrado(cpf_paciente);
}
else
{
    printf("=====\\n");
    printf("Nome      : %s\\n", p->nome);
    printf("CPF       : %s\\n", p->cpf);
    printf("Telefone: %s\\n", p->telefone);
    printf("=====\\n");

    // Manipulação de string com ponteiros
    // Cópia
    char nome_copia[120];
    strcpy(nome_copia, p->nome);

    // Concatenação
    char mensagem[256] = "Paciente: ";
    strcat(mensagem, p->nome);
    strcat(mensagem, " | Telefone: ");
    strcat(mensagem, p->telefone);

    // Comparação
    if (strcmp(p->nome, "Anderson Carlos") == 0)
    {
        printf("Bem-vindo, Anderson!\\n");
    }

    printf("Mensagem concatenada: %s\\n", mensagem);
    printf("Nome copiado: %s\\n", nome_copia);
}
limpar_buffer();
break;
// ...existing code...
```

Figura 18: Operações básicas com strings usando ponteiros. Autoria própria.

Alocação Dinâmica de Memória

Emprego de `realloc` e `free`

- A função `realloc` é usada para aumentar o tamanho dos vetores dinâmicos `pacientes` e `medicos` conforme novos registros são adicionados. Isso é visto em `modulo_gerenciar_paciente.c` ao adicionar um novo paciente e em `carregar_medicos_do_arquivo` em `auxiliar.c` ao carregar médicos do arquivo, ilustrados na Figura 19.

```
// filepath: src/modulo_gerenciar_paciente.c
// ...existing code...
reg_paciente *novo = realloc(pacientes, (total_pacientes + 1) * sizeof(reg_paciente));
if (novo == NULL)
{
    msg_erro_memoria_paciente();
    break;
}
pacientes = novo;
// ...existing code...

// filepath: src/auxiliar.c
// ...existing code...
medicos = realloc(medicos, (total_medicos + 1) * sizeof(reg_medico));
medicos[total_medicos++] = temp;
// ...existing code...
```

Figura 19: uso do `realloc` no projeto. Autoria própria.

- A função `free` é utilizada para liberar a memória alocada para esses vetores quando não são mais necessários, prevenindo vazamentos de memória. Isso ocorre no final da execução do programa em `main.c` e também antes de recarregar os dados dos arquivos, como em `carregar_medicos_do_arquivo` em `auxiliar.c`, ilustrados na Figura 20.

```
// filepath: src/auxiliar.c
// ...existing code...
if (medicos)
{
    free(medicos);
    medicos = NULL;
    total_medicos = 0;
}
// ...existing code...
```

Figura 20: uso do `free` para liberar memória dos vetores. Autoria própria.

Vetores Dinâmicos

Implementação e Gerenciamento

- Os ponteiros `reg_medico *medicos` (declarado em `src/modulo_gerenciar_medico.c`) e `reg_paciente *pacientes` (declarado em `src/main.c` e usado como `extern` em outros locais) servem como vetores dinâmicos. Eles são inicializados como `NULL`.
- Inserção: Quando um novo registro é adicionado (por exemplo, um novo médico em `tratar_modulo_medico` ou ao carregar do arquivo em `carregar_medicos_do_arquivo`), `realloc` é chamado para aumentar o tamanho do vetor. O novo elemento é então copiado para a nova posição, e o contador (`total_medicos` ou `total_pacientes`) é incrementado, sendo apresentado na Figura 21.

```
// filepath: src/auxiliar.c
// ...existing code...
// Libera vetor antigo, se houver
if (medicos)
{
    free(medicos);
    medicos = NULL;
    total_medicos = 0;
}

while (fgets(linha, sizeof(linha), arquivo))
{
    reg_medico temp;
    // ...existing code...
    medicos = realloc(medicos, (total_medicos + 1) * sizeof(reg_medico));
    medicos[total_medicos++] = temp;
    // ...existing code...
```

Figura 21: uso de vetores dinâmicos. Autoria própria.

- Acesso: Os elementos dos vetores dinâmicos são acessados usando a sintaxe de array, como `medicos[i].nome`, após a alocação e preenchimento.
- Liberação: A memória alocada para esses vetores é liberada com `free` ao final do programa em `main.c` para evitar vazamentos de memória.

Tipos Estruturados (Registros e Enumerações)

Definição de structs

- O arquivo estruturas.h define as principais estruturas:
 - reg_paciente (include/estruturas.h): Armazena ID, nome, CPF e telefone do paciente, ilustrados na Figura 22.
 - reg_medico (include/estruturas.h): Armazena ID, nome, CRM, especialidade (como enum e string) e telefone do médico.
 - reg_consulta (include/estruturas.h): Armazena IDs da consulta, paciente, médico, data/hora e status da consulta.

```
// filepath: include/estruturas.h
// ...existing code...
typedef struct
{
    int id_paciente;
    char nome[120];
    char cpf[12];      // Alterado para string (11 dígitos + '\0')
    char telefone[12]; // Alterado para string (11 dígitos + '\0')
} reg_paciente;
// ...existing code...
```

Figura 22: Struct de paciente. Autoria própria.

Utilização de enums

- Enumerações são definidas em estruturas.h para representar categorias e estados de forma clara, sendo exemplificados na Figura 23:
 - Estado (include/estruturas.h): Define os diferentes estados de navegação do menu do sistema (ex: ESTADO_MENU_PRINCIPAL, ESTADO_GERENCIAR_PACIENTE).
 - status_consulta (include/estruturas.h): Define o status de uma consulta (ex: AGENDADA, REALIZADA, CANCELADA).
 - Especialidade (include/estruturas.h): Define as especialidades médicas (ex: CLINICO_GERAL, PEDIATRA).

```
// filepath: include/estruturas.h
// ...existing code...
typedef enum
{
    CLINICO_GERAL,
    PEDIATRA,
    CARDIOLOGISTA,
    DERMATOLOGISTA,
    PSIQUIATRA
} Especialidade;
// ...existing code...
```

Figura 23: Enum para especialidades médicas. Autoria própria.

Manipulação de Arquivos

Leitura e Escrita

- Abertura e Fechamento: `fopen` é usado para abrir arquivos em diferentes modos ("r" para leitura, "w" para escrita, "a" para adição), e `fclose` é usado para fechá-los. Exemplos podem ser encontrados em funções como `criar_arquivo_csv`, `add_row`, e `exibir_arquivo` em `files_manager.c`.
- Leitura: `fgets` é comumente usado para ler uma linha inteira de um arquivo. Em seguida, `sscanf` é frequentemente aplicado para extrair e converter os campos da linha lida, como na função `exibir_arquivo` para processar os dados das consultas, sendo visto o uso na Figura 24.

```
// filepath: src/files_manager.c
// ...existing code...
while (fgets(linha, sizeof(linha), arquivo))
{
    // ...existing code...
    if (sscanf(linha, "%15[^,],%15[^,],%15[^,],%31[^,],%15[^\n]",
                id_consulta_csv, id_paciente_csv, id_medico_csv, data_hora_csv, status_csv) != 5)
    {
        // ...existing code...
    }
}
```

Figura 24: Uso de `sscanf` para extrair e converter campos. Autoria própria.

- Escrita: fprintf é usado para escrever dados formatados em arquivos, como ao adicionar um novo registro em add_row ou ao escrever em arquivos temporários durante atualizações, como visto na Figura 25.

```
// filepath: src/files_manager.c
// ...existing code...
fprintf(arquivo, "%d", novo_id);          // Escreve o novo id
for (int i = 0; i < select_col - 1; i++) // -1 pois ja escrevemos o id
{                                         // Escreve os demais campos
    fprintf(arquivo, "%s", valores[i]);
}
fprintf(arquivo, "\n"); // Nova linha
// ...existing code...
```

Figura 25: Uso da função fprintf. Autoria própria.

Atualização e Deleção

- Para atualizar (att_row) ou deletar (del_row) linhas em arquivos CSV, uma abordagem comum é criar um arquivo temporário. O conteúdo do arquivo original é lido, as modificações são aplicadas (ou a linha a ser deletada é omitida) e o resultado é escrito no arquivo temporário.
- Após o processamento, o arquivo original é removido usando remove, e o arquivo temporário é renomeado para o nome do arquivo original usando rename, ilustrado na Figura 26.

```
// filepath: src/files_manager.c
// ...existing code...
// Substitui o arquivo original pelo temporario
if (remove(nome_arquivo) != 0 || rename("temp.csv", nome_arquivo) != 0)
{
    msg_erro_abrir_arquivo_nome(nome_arquivo);
    return 0;
}
// ...existing code...
```

Figura 26: Uso da função remove. Autoria própria.

Manual de Uso

Módulo Consultas

Módulo que realiza as funcionalidades de agendamento e cancelamento de consultas. Funcionando da seguinte forma:

1. **Agendar Consulta:** o usuário seleciona inicialmente a opção "Agendar" no menu de consultas. Em seguida, ele informa o CPF do paciente, e o sistema verifica a existência desse paciente no cadastro. Após validação do paciente, o sistema lista os médicos disponíveis, permitindo que o usuário escolha um profissional. Com o médico selecionado, o usuário informa o mês e o dia para a consulta, e o sistema valida esses dados. Na sequência, o sistema exibe os horários livres para o médico e a data, após serem escolhidos. Finalmente, a consulta é salva no sistema com o status "AGENDADA", e uma mensagem de sucesso é apresentada ao usuário;
2. **Cancelar Consulta:** O usuário primeiro escolhe a opção "Cancelar" no menu de consultas. Em seguida, ele se identifica como Paciente ou Médico. Após a identificação, o usuário informa seu CRM (para médico) ou CPF (para paciente); o sistema verifica essas informações e lista as consultas relacionadas a esse usuário. Com a lista de consultas exibida, o usuário insere o ID da consulta que deseja cancelar. O próximo passo é o usuário confirmar o cancelamento. Por fim, é confirmado que o sistema atualizou o status da consulta para "CANCELADA" e o usuário recebe uma mensagem de sucesso.
3. **Voltar:** O usuário primeiro escolhe a opção " Voltar", na qual o usuário volta ao menu principal;
4. **Sair:** O usuário primeiro escolhe a opção "Sair", em que o programa é encerrado.

Módulo Paciente

Módulo que é responsável por todas as operações relacionadas ao cadastro de pacientes. Funcionando da seguinte forma:

1. **Criar Paciente:** O usuário seleciona a opção "Criar" no menu do paciente para iniciar o cadastro de um novo paciente. O usuário então informa o nome completo, que foi lido e verificado, o sistema então solicita a confirmação para prosseguir

com o nome fornecido. O próximo passo é inserir o CPF e o telefone separadamente, que também passam por um processo de revisão e validação após uma solicitação de confirmação. Após todas as confirmações, os dados do paciente são inseridos no banco de dados de pacientes e em uma estrutura de memória, resultando em uma mensagem de sucesso no cadastro.

2. **Exibir Paciente:** Quando o usuário seleciona a opção "Exibir", o sistema solicita o CPF do paciente cujos dados deseja visualizar. Ele é validado após a leitura do CPF, caso seja inválido, uma mensagem de erro é exibida. Em contrapartida, o sistema busca o paciente na memória. Se o paciente for localizado, suas informações (nome, CPF e telefone) são formatadas e exibidas na tela. Caso o paciente não seja encontrado no cadastro, é enviada uma mensagem informando que ele não foi encontrado;
3. **Atualizar Paciente:** Quando o usuário seleciona "Atualizar", o sistema pede o CPF do paciente, cujos dados serão alterados. O CPF é lido e validado; uma entrada inválida interrompe o processo com uma mensagem de erro. Se o CPF for válido, a função de atualização é ativada, procurando o paciente no registro. Se o paciente não for localizado, aparece uma mensagem que o usuário não foi encontrado. Caso contrário, o usuário pode escolher em um menu quais informações (nome, CPF ou número de telefone) deseja alterar. Os novos dados são solicitados, verificados e, em seguida, o sistema pergunta se o usuário deseja receber mais informações. Como resultado das alterações, a linha correspondente ao arquivo do paciente é atualizada e o sistema notifica o usuário se a operação ocorreu sem problemas ou se ocorreu algum erro.
4. **Deletar Paciente:** O usuário seleciona a opção "Excluir" para remover um paciente, e o sistema então solicita CPF do paciente. Após a leitura, o CPF, caso seja inválido, uma mensagem de erro é exibida, caso seja válido, ele verifica se está no registro. Caso o paciente não seja localizado, a mensagem "paciente não encontrado" é exibida. Uma vez localizado, o registro do paciente é excluído do arquivo, e o sistema exibe uma mensagem indicando se a operação de exclusão foi bem-sucedida ou não.
5. **Voltar:** O usuário primeiro escolhe a opção " Voltar", na qual o usuário volta ao menu principal;
6. **Sair:** O usuário primeiro escolhe a opção "Sair", em que o programa é encerrado.

Módulo Médico

Módulo que é responsável por todas as operações relacionadas ao cadastro dos médicos.

Funcionando da seguinte forma:

1. **Criar Médico:** Quando o usuário seleciona a opção "Criar" no menu de médicos para cadastrar um novo médico, e o sistema envia uma mensagem de Boas-Vindas. O usuário insere o nome completo e verificado, o sistema solicita confirmação para prosseguir. O procedimento é repetido para o CRM, telefone e especialidade, que também são lidos, verificados e validados. Após todas as confirmações, os dados do médico são salvos no arquivo médico e em uma estrutura de memória, e uma mensagem de sucesso é exibida.
2. **Exibir Médico:** O sistema solicita o CRM do médico quando o usuário seleciona "Exibir". O sistema busca o ID correspondente no registro do médico após a leitura do CRM. Se o médico for encontrado, seus dados são achados, irá exibi-los na tela. Nesse caso, é exibida uma mensagem informando que o médico não foi encontrado.
3. **Atualizar Médico:** Quando o usuário seleciona a opção "Atualizar", o sistema pede o CRM do médico. O sistema procura o médico no banco de dados, caso não o encontre, uma mensagem de erro é exibida. Caso seja encontrado, os dados médicos são registrados e o usuário pode escolher em um menu quais informações (nome, CRM, telefone ou especialidade) deseja alterar. Após cada alteração, o sistema pergunta se o usuário deseja receber mais informações. Como resultado das alterações, a linha correspondente ao arquivo médico é atualizada e o sistema registra o sucesso da operação.
4. **Deletar Médico:** Quando o usuário seleciona "Excluir" para remover um médico, e o sistema solicita o CRM. O sistema busca em registro. Caso não seja encontrado, a mensagem "médico não encontrado" é exibida. Uma vez localizado, o registro do médico é excluído do arquivo, e o sistema envia uma mensagem confirmando o sucesso da operação de exclusão ou indicando uma falha.
5. **Voltar:** O usuário primeiro escolhe a opção " Voltar", no qual o usuário volta ao menu principal;
6. **Sair:** O usuário primeiro escolhe a opção "Sair", em que o programe é encerrado.

Módulo Relatórios

Este módulo permite gerar diversos relatórios sobre as atividades da clínica.

1. [1] Consultas de Pacientes:

- O sistema solicita o CPF do paciente.
- Após a inserção, o CPF é validado. Se o paciente não for encontrado, uma mensagem informativa é exibida.
- Caso encontrado, o sistema lista todas as consultas agendadas, realizadas ou canceladas para o paciente informado, exibindo detalhes como ID da consulta, nome do médico, data/hora e status.

2. [2] Consultas de Médicos:

- O sistema solicita o CRM do médico.
- Após a inserção, o CRM é validado. Se o médico não for encontrado, uma mensagem informativa é exibida.
- Caso encontrado, o sistema lista todas as consultas associadas a esse médico, detalhando ID da consulta, nome do paciente, data/hora e status.

3. [3] Consultas por Especialidade:

- O sistema exibe uma lista numerada das especialidades médicas disponíveis (Clínico Geral, Pediatra, etc.).
- O usuário digita o número correspondente à especialidade desejada.
- O sistema então lista todas as consultas da especialidade selecionada, com seus respectivos detalhes.

4. [4] Consultas do Dia Atual:

- Ao selecionar esta opção, o sistema automaticamente identifica a data atual.
- Em seguida, exibe todas as consultas agendadas para o dia corrente, mostrando ID da consulta, paciente, médico, data/hora e status.

5. **[5] Quantidade de Consultas por Especialidade:**

- Esta opção gera um relatório sumarizado.
- O sistema calcula e exibe o número total de consultas realizadas para cada especialidade médica cadastrada.

6. **[6] Voltar ao Menu Principal:**

- Retorna o usuário ao menu principal do sistema.

7. **[7] Encerrar o Programa:**

- Finaliza a execução do sistema.

README.md

Sistema de Gerenciamento Médico

Este projeto é um sistema de gerenciamento para clínicas ou consultórios médicos, desenvolvido em linguagem C. Ele permite o cadastro e gerenciamento de pacientes, médicos e consultas, além da geração de relatórios básicos. Os dados são persistidos em arquivos CSV.

Funcionalidades Principais

O sistema é modularizado e oferece as seguintes funcionalidades:

1. ****Módulo de Consultas:****

- Agendar novas consultas, selecionando paciente, médico, data e horário.
- Cancelar consultas existentes (por médico ou paciente).

2. ****Módulo de Gerenciamento de Pacientes:****

- Criar novos registros de pacientes (nome, CPF, telefone).
- Exibir informações de um paciente específico (buscando por CPF).
- Atualizar dados cadastrais de pacientes.

- Deletar registros de pacientes.

3. ****Módulo de Gerenciamento de Médicos:****

- Criar novos registros de médicos (nome, CRM, especialidade, telefone).
- Exibir informações de um médico específico (buscando por CRM).
- Atualizar dados cadastrais de médicos.
- Deletar registros de médicos.

4. ****Módulo de Relatórios:****

- Listar todas as consultas de um paciente específico (buscando por CPF).
- Listar todas as consultas de um médico específico (buscando por CRM).
- Listar consultas por especialidade médica.
- Listar consultas agendadas para o dia atual.
- Exibir a quantidade total de consultas agrupadas por especialidade médica.

Estrutura do Projeto

O projeto está organizado da seguinte forma:

- - `src/`: Contém os arquivos de código-fonte (.c) com a lógica dos módulos e funcionalidades.
 - - `main.c`: Ponto de entrada do programa e loop principal de controle de estados.
 - - `menu_principal.c`: Lógica do menu principal.
 - - `modulo_consulta.c`: Lógica para agendamento e cancelamento de consultas.
 - - `modulo_gerenciar_paciente.c`: Lógica para CRUD de pacientes.
 - - `modulo_gerenciar_medico.c`: Lógica para CRUD de médicos.
 - - `modulo_relatorios.c`: Lógica para geração de relatórios.
 - - `files_manager.c`: Funções para manipulação dos arquivos CSV (leitura, escrita, atualização, deleção).
 - - `auxiliar.c`: Funções utilitárias diversas (validações, limpeza de buffer, conversões, etc.).

- - ``mensagens.c``: Funções para exibição de mensagens e menus padronizados.
- - ``include/``: Contém os arquivos de cabeçalho (`.h``) com as declarações de estruturas, funções e constantes.
 - - ``estruturas.h``: Definição das principais estruturas de dados (paciente, médico, consulta, enums de estado e especialidade).
 - - ``files_manager.h``: Cabeçalho para ``files_manager.c``.
 - - ``auxiliar.h``: Cabeçalho para ``auxiliar.c``.
 - - ``mensagens.h``: Cabeçalho para ``mensagens.c``.
 - - ``menu_principal.h``: Cabeçalho para ``menu_principal.c``.
 - - ``modulo_consulta.h``: Cabeçalho para ``modulo_consulta.c``.
 - - ``modulo_gerenciar_paciente.h``: Cabeçalho para ``modulo_gerenciar_paciente.c``.
 - - ``modulo_gerenciar_medico.h``: Cabeçalho para ``modulo_gerenciar_medico.c``.
 - - ``modulo_relatorios.h``: Cabeçalho para ``modulo_relatorios.c``.
- - ``data/``: Contém os arquivos CSV utilizados para persistência dos dados.
 - - ``registro_pacientes.csv``: Armazena os dados dos pacientes (id_paciente, nome, cpf, telefone).
 - - ``registro_medicos.csv``: Armazena os dados dos médicos (id_medico, nome, crm, especialidade, telefone).
 - - ``registro_consultas.csv``: Armazena os dados das consultas (id_consulta, id_paciente, id_medico, data_hora, status).
- - ``Makefile``: Arquivo para automatizar o processo de compilação (usando ``mingw32-make``).
- - ``bin/``: (Opcional, geralmente criado pela compilação) Contém o executável do programa.

Tecnologias Utilizadas

- Linguagem C
- Arquivos CSV para persistência de dados

Pré-requisitos

- Compilador C (como o GCC, parte do MinGW para Windows)

- Utilitário `make` (como `mingw32-make` para Windows)

Compilação e Execução

Para compilar e executar o projeto, navegue até a pasta raiz do projeto (`Trabalho1_estrutura_de_dados_1`) pelo terminal e utilize o seguinte comando:

```
```bash
```

```
mingw32-make run
```

```
```
```

Este comando irá compilar os arquivos-fonte e, se a compilação for bem-sucedida, executará o programa `bin/main.exe`.

Se desejar apenas compilar, você pode usar:

```
```bash
```

```
mingw32-make
```

```
```
```

E para limpar os arquivos compilados:

```
```bash
```

```
mingw32-make clean
```

```
```
```

Autores

- ANDERSON CARLOS DA SILVA MORAIS
- MARILIA FONSECA ANDRADE

Versão

- 1.0 (Conforme `mensagens.c`)