

Estrutura de Dados

1º semestre de 2025

Arquivos

Tema #02

Professor Marcelo Eustáquio



Definições

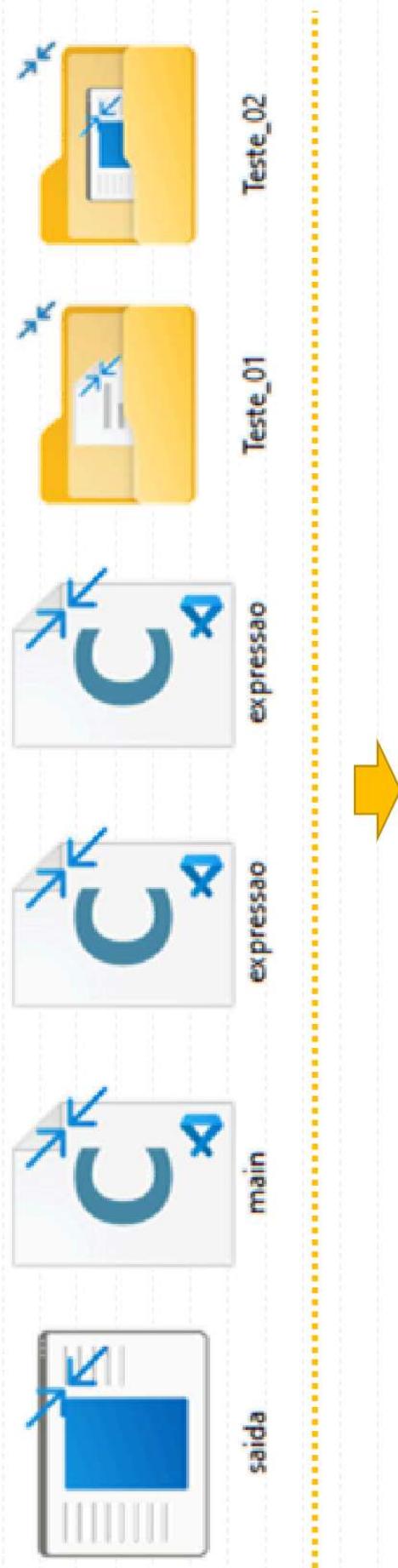
Um **arquivo** é uma coleção de bytes armazenados em um dispositivo de armazenamento. Em C, existem dois tipos principais de arquivos: os arquivos **binários** (que podem não ser legíveis) e os arquivos **texto** (que são sempre legíveis).

1	Binário	Armazena dados como são organizados em memória (arquivos executáveis).
2	Texto	Dados são armazenados como sequência de caracteres (arquivos fonte).
3	Pasta	É um conjunto de 0 ou mais arquivos ou pastas.

Além desses conceitos, é importante destacar o conceito de pasta (ou diretório), definido como uma coleção de 0 ou mais caracteres.

Definições

Um **arquivo** é uma coleção de bytes armazenados em um dispositivo de armazenamento. Em C, existem dois tipos principais de arquivos: os arquivos **binários** (que podem não ser legíveis) e os arquivos **texto** (que são sempre legíveis).



Na figura, vê-se um conjunto de 1 arquivo binário (saída), 3 arquivos texto (main, expressão e expressão), além de 2 pastas (Teste_01 e Teste_02).

Em C, existem dois tipos principais de arquivos: os arquivos binários (que **podem não ser legíveis**) e os arquivos texto (que **são sempre legíveis**). Considerando essa classificação, quanto são os arquivos textos indicados abaixo?

Nome	Data de modificação	Tipo	Tamanho	Data da criação
📄 cidades	26/11/2024 09:53	Arquivo Fonte C	4 KB	25/11/2024 19:01
📄 cidades	10/06/2024 10:13	Arquivo Fonte C Header	1 KB	05/11/2024 08:55
📄 main	08/11/2024 10:49	Arquivo Fonte C	6 KB	03/10/2024 13:02
📄 Resultado	25/11/2024 19:02	Documento de Texto	2 KB	25/11/2024 19:02
📄 saída	25/11/2024 19:02	Aplicativo	49 KB	25/11/2024 19:02
📄 Teste01	22/11/2024 10:26	Documento de Texto	1 KB	05/11/2024 08:55
📄 Teste02	14/06/2024 14:56	Documento de Texto	1 KB	05/11/2024 08:55
📄 Teste03	10/06/2024 11:54	Documento de Texto	1 KB	05/11/2024 08:55
📄 Teste04	07/06/2024 08:36	Documento de Texto	1 KB	05/11/2024 08:55

Manipulando um arquivo

São três as funções iniciais para manipular arquivos (criando uma variável do tipo arquivo, abrindo o arquivo apontado pela variável do tipo arquivo e fechando o arquivo apontado pela variável do tipo arquivo).

- | | | |
|---|--|-----------------------------|
| 1 | <code>FILE *fopen(char *NomeArquivo, char *Modo);</code> | Abrir um arquivo. |
| 2 | <code>int fclose(FILE * fp);</code> | Fechar um arquivo. |
| 3 | <code>int fprintf(FILE *stream, const char *format, ...);</code> | Escrever no arquivo. |
| 4 | <code>int fscanf(FILE * stream, const char *format, ...);</code> | Ler do arquivo. |

Existem outras funções, mas essas serão tratadas oportunamente.

Manipulando um arquivo

Em C, os modos de abertura de arquivo são usados para especificar como um arquivo será aberto e manipulado. Os modos de abertura são representados por strings passadas como argumentos para a função **fopen()**. Os modos de abertura mais comuns são:

"r"
(Read)

Abre o arquivo para leitura. Se o arquivo não existir, a operação de abertura falhará.

"w"
(Write)

Abre o arquivo para escrita. Se o arquivo já existir, seu conteúdo será apagado. Se o arquivo não existir, um novo arquivo será criado.

"a"
(Append)

Abre o arquivo para escrita, mas acrescenta dados ao final do arquivo, em vez de sobrescrevê-lo. Se o arquivo não existir, um novo arquivo será criado.

Manipulando um arquivo

Abrindo o arquivo “arquivo.txt” para escrita.

```
#include <stdio.h>
#include <stdlib.h>

int main() {

    FILE *arquivo;
    arquivo = fopen("arquivo.txt", "w");

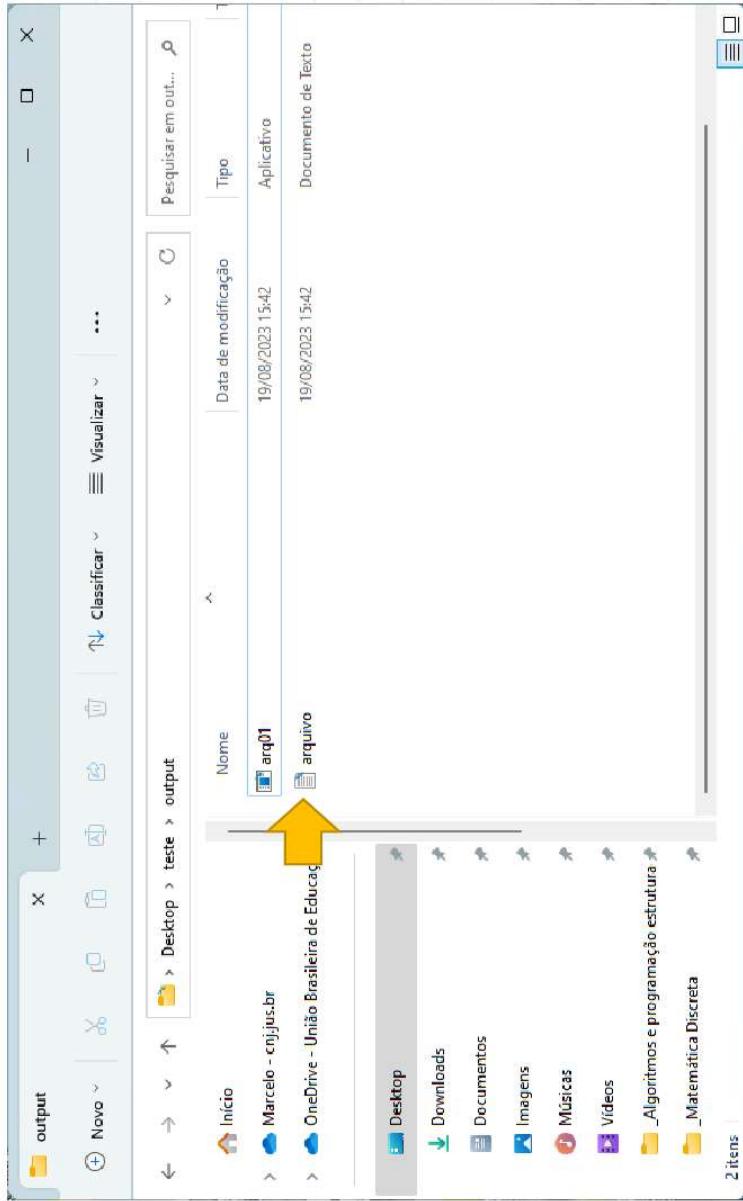
    if (arquivo == NULL) {
        printf("Erro ao abrir o arquivo. \n");
        exit(1);
    }

    fclose(arquivo);
    return 0;
}
```



Manipulando um arquivo

O arquivo será criado e, caso já exista, todo o seu conteúdo será sobrescrito.



Manipulando um arquivo

Ao ser executada, **fopen()** retornará um ponteiro para o arquivo aberto.

```
arquivo = fopen("arquivo.txt", "wb");

if (arquivo == NULL) {
    printf("Erro ao abrir o arquivo.\n");
    exit(1);
}
```

Observações:

- Erros podem ocorrer (por exemplo: abrir para leitura um arquivo que não existe).
- Em falhas, o retorno de fopen() é indicado por **NULL**.

exit() é uma função da biblioteca **stdlib.h** da linguagem C que retorna o controle ao SO, passando um código de retorno e terminando a execução do programa.

Manipulando um arquivo

A função **fprintf()** é usada para escrever dados formatados em um arquivo em C, assim como a função printf() é usada para escrever dados formatados na saída padrão (normalmente o console). A sintaxe básica da função fprintf() é a seguinte:

```
int fprintf (FILE *stream, const char *format, ...);
```

Observações:

- **stream**: O ponteiro para o arquivo no qual os dados serão gravados.
- **format**: string de formato que especifica como os dados devem ser formatados e escritos no arquivo.
- **...:** argumentos variáveis relativos aos valores que serão escritos no arquivo.



```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main() {

    FILE *arquivo;
    int numero = 125;
    char palavra[] = "Esta é uma frase que será gravada no arquivo!";

    arquivo = fopen("saída.txt", "w");

    if (arquivo == NULL) {
        printf("Erro ao abrir o arquivo.\n");
        exit(1);
    }

    fprintf(arquivo, "Número: %d\nPalavra: %s\n", numero, palavra);

    return 0;
}
```

Manipulando um arquivo

A função **fscanf()** é usada para ler dados formatados de um arquivo em C, assim como a função `scanf()` é usada para ler dados formatados da entrada padrão (normalmente o teclado). A sintaxe básica da função `fscanf()` é a seguinte:

```
int fscanf (FILE *stream, const char *format, ...);
```

Observações:

- **stream**: O ponteiro para o arquivo no qual os dados serão gravados.
- **format**: string de formato que especifica como os dados devem ser formatados e escritos no arquivo.
- **...:** argumentos variáveis relativos aos valores que serão lidos do arquivo.



```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main() {
    FILE *arquivo;
    int numero;
    char palavra[50];

    arquivo = fopen("arquivo.txt", "r");
    if (arquivo == NULL) {
        printf("Erro ao abrir o arquivo.\n");
        exit(1);
    }

    // Lendo um número inteiro e uma palavra do arquivo

    fscanf(arquivo, "%d %s", &numero, palavra);
    printf("Número: %d\nPalavra: %s\n", numero, palavra);
    fclose(arquivo);
    return 0;
}
```



O arquivo do tipo CSV

Escreva uma função, usando a linguagem C padrão, que leia o arquivo cujo nome foi passado como parâmetro, exiba seu conteúdo na tela e retorne o número de linhas nele existente (exceto o cabeçalho). Um exemplo de arquivo é mostrado a seguir:

dados.csv

ID	Nome	Idade
1	Ana	23
2	João	35
3	Maria	28



```
#include <stdio.h>
#include <stdlib.h>

int main() {
    FILE *arquivo;
    int id, idade;
    char nome[50];

    arquivo = fopen("dados.csv", "r"); // Abre o arquivo CSV para leitura
    if (arquivo == NULL) {
        printf("Erro ao abrir o arquivo.\n");
        return 1;
    }

    fscanf(arquivo, "%*[^\n],%*[^\n],%*s"); // Lê e descarta a primeira linha
    printf("ID\tNome\n");
    while (fscanf(arquivo, "%d,%[^\n],%d", &id, nome, &idade) == 3) {
        printf("%d\t%s\t%d\n", id, nome, idade);
    }

    fclose(arquivo);
    return 0;
}
```

Manipulando um arquivo

A função **fgets()** em C é usada para ler uma linha de texto de um arquivo ou da entrada padrão (normalmente o teclado) e armazená-la em uma string. Ela é útil para ler linhas completas de texto, incluindo espaços em branco, e armazená-las em variáveis de string. A sintaxe é detalhada a seguir:

```
char *fgets(char *str, int num, FILE *stream);
```

Observações:

- **stream**: ponteiro para o arquivo no qual os dados serão gravados.
- **str**: ponteiro para a string na qual os dados lidos serão armazenados.
- **num**: número máximo de caracteres a serem lidos, incluindo os caracteres '\n' e '\0'.



```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main() {
    FILE *arquivo;
    char linha[100];

    arquivo = fopen("dados.csv", "r");
    if (arquivo == NULL) {
        printf("Erro ao abrir o arquivo. \n");
        exit(1);
    }

    // Lê e exibe cada linha do arquivo

    while (fgets(linha, sizeof(linha), arquivo) != NULL) {
        printf("%s", linha);
    }

    fclose(arquivo);
    return 0;
}
```

Manipulando um arquivo

A função **fputs()** é usada para escrever uma string em um arquivo. Ela é útil para gravar uma sequência de caracteres em um arquivo, sem a formatação adicional que as funções de formatação. A função fputs() não adiciona automaticamente caracteres de nova linha e tem a seguinte sintaxe:

```
int fputs(const char *str, FILE *stream);
```

Observações:

- **stream**: ponteiro para o arquivo no qual os dados serão gravados.
- **str**: ponteiro para a string que será gravada no arquivo texto



```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main() {

    FILE *arquivo;
    arquivo = fopen("saída.txt", "w");
    if (arquivo == NULL) {
        printf("Erro ao abrir o arquivo.\n");
        return 1;
    }

    // Escrevendo uma string no arquivo
```

```
const char *mensagem = "Esta é uma mensagem de exemplo.";
fputs(mensagem, arquivo);

fclose(arquivo);
return 0;
}
```

Estrutura de Dados

1º semestre de 2025

Arquivos

Tema #01

Professor Marcelo Eustáquio



Q01

Descreva o funcionamento do seguinte código-fonte:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main() {

    FILE *f1, *f2;

    f1 = fopen("entrada.txt", "r");
    f2 = fopen("saída.bin", "w");

    fclose(f1);
    fclose(f2);

    return 0;
}
```

O código apresentado realiza apenas a abertura e o fechamento de dois arquivos, sem efetuar nenhuma operação de leitura ou escrita entre eles. De forma resumida, o funcionamento dele é apresentado a seguir:

- 1) Inclui as bibliotecas padrão: stdio.h, stdlib.h e string.h (manipulação de strings, embora não seja usada).
- 2) Declara dois ponteiros para arquivos: FILE *f1, *f2;.
- 3) Abre o arquivo "entrada.txt" no modo leitura ("r").
- 4) Se o arquivo não existir, a função fopen retornará NULL.
- 5) Cria (ou substitui, se já existir) o arquivo "saída.bin" no modo escrita ("w").
- 6) Esse arquivo será binário apenas por convenção do nome, já que não se especificou "wb".
- 7) Fecha os dois arquivos com fclose(f1) e fclose(f2).
- 8) Retorna 0, indicando término normal do programa.

Resumindo:

O programa abre um arquivo de entrada em texto e cria um arquivo de saída vazio, fechando ambos em seguida, sem manipular os dados.

Q02

Descreva o funcionamento do seguinte código-fonte:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main() {

    FILE *f1, *f2;
    int x;
    f1 = fopen("entrada.txt", "r");
    f2 = fopen("saída.bin", "w");

    fscanf(f1, "%d", &x);
    fprintf(f2, "%d", x);

    fclose(f1);
    fclose(f2);
    return 0;
}
```

Esse código em C realiza uma leitura simples de um valor inteiro de um arquivo texto e grava esse valor em outro arquivo. De forma resumida, o funcionamento dele é apresentado a seguir:

- 1) Inclui as bibliotecas padrão necessárias.
- 2) Declara dois ponteiros para arquivos (FILE *f1, *f2;) e uma variável inteira x.
- 3) Abre o arquivo "entrada.txt" em modo leitura ("r").
- 4) Cria (ou sobrescreve) o arquivo "saída.bin" em modo escrita ("w").
- 5) Usa fscanf(f1, "%d", &x); para ler um número inteiro do arquivo de entrada e armazena-lo em x.
- 6) Usa fprintf(f2, "%d", x); para escrever esse número inteiro no arquivo de saída.
- 7) Fecha os dois arquivos (fclose(f1); fclose(f2);).
- 8) Retorna 0, indicando término normal do programa.

Resumindo:

O programa copia um número inteiro contido em entrada.txt e o grava em saída.bin.

Q03

Descreva o funcionamento do seguinte código-fonte:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main() {

    FILE *arquivo;
    int X;
    arquivo = fopen("entrada.txt", "r");
    if (arquivo == NULL) exit(1);

    while (fscanf(arquivo, "%d", &X) != EOF) {
        printf("%d\n", X);
    }

    fclose(arquivo);
    return 0;
}
```

Esse código em C faz a leitura de todos os números inteiros presentes em um arquivo texto e os imprime na tela. De forma resumida, o funcionamento dele é apresentado a seguir:

- 1) Inclui as bibliotecas padrão (stdio.h, stdlib.h, string.h).
- 2) Declara um ponteiro para arquivo (FILE *arquivo) e uma variável inteira X.
- 3) Abre o arquivo "entrada.txt" em modo leitura ("r").
- 4) Caso o arquivo não exista ou não possa ser aberto, o programa encerra com exit(1).
- 5) Usa um laço while com fscanf(arquivo, "%d", &X) != EOF para ler sucessivamente todos os inteiros do arquivo até o fim.
- 6) Para cada número lido, imprime na tela (printf("%d\n", X);).
- 7) Ao final, fecha o arquivo com fclose(arquivo);
- 8) Retorna 0, indicando execução bem-sucedida.

Resumindo:

O programa lê todos os números inteiros de entrada.txt e os exibe no console, um por linha.

Q04

Escreva um programa que solicite do usuário os nomes de 2 arquivos texto para, a seguir, criar um terceiro arquivo, também texto, com o conteúdo dos dois arquivos fornecidos (o conteúdo do primeiro arquivo seguido do conteúdo do segundo arquivo).

```
#include <stdio.h>
#include <stdlib.h>

int main() {
    char arquivo1[100], arquivo2[100], arquivo3[100];
    FILE *f1, *f2, *f3;
    int ch;

    printf("Digite o nome do primeiro arquivo de texto: ");
    scanf("%s", arquivo1);
    printf("Digite o nome do segundo arquivo de texto: ");
    scanf("%s", arquivo2);
    printf("Digite o nome do arquivo de saída: ");
    scanf("%s", arquivo3);

    f1 = fopen(arquivo1, "r");
    f2 = fopen(arquivo2, "r");
    f3 = fopen(arquivo3, "w");
```

```
if (f1 == NULL || f2 == NULL || f3 == NULL) exit(1);

while ((ch = fgetc(f1)) != EOF) fputc(ch, f3); // Copia o conteúdo do 1º arquivo para o 3º
while ((ch = fgetc(f2)) != EOF) fputc(ch, f3); // Copia o conteúdo do 2º arquivo para o 3º

fclose(f1);
fclose(f2);
fclose(f3);

printf("Conteúdo de %s e %s foi copiado para %s.\n", arquivo1, arquivo2, arquivo3);

return 0;
}
```

Q05

Escreva um programa que leia um arquivo texto contendo uma lista de compras em que cada linha possui o nome do produto, a quantidade e o valor unitário de cada item para, em seguida, informar o valor total da compra. Caso o arquivo texto tenha:

Arroz 2 5.50
Feijão 3 3.00
Carne 1 15.75
Leite 4 2.50

A saída (na tela) deverá ser:

- 1) Arroz: $2 * 5.50 = 11.00$
- 2) Feijão: $3 * 3.00 = 9.00$
- 3) Carne: $1 * 15.75 = 15.75$
- 4) Leite: $4 * 2.50 = 10.00$

O valor total da compra é R\$ 45.75

```
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    char in_name[128], out_name[128];
    FILE *in, *out;
    char produto[64];
    int qtd, item = 1;
    double preco, total_item, total_compra = 0.0;

    printf("Arquivo de entrada (ex.: compras.txt): ");
    if (scanf("%127s", in_name) != 1) return 1;

    printf("Arquivo HTML de saida (ex.: relatorio.html): ");
    if (scanf("%127s", out_name) != 1) return 1;

    in = fopen(in_name, "r");
    if (!in) { perror("Erro ao abrir arquivo de entrada"); return 1; }
    out = fopen(out_name, "w");
    if (!out) { perror("Erro ao criar arquivo HTML"); fclose(in); return 1; }
```

```
/* Cabeçalho HTML + CSS simples */
fprintf(out,
"<!DOCTYPE html>\n<html lang=\"pt-BR\">\n<head>\n"
"<meta charset=\"UTF-8\">\n<title>Lista de Compras</title>\n"
"<style>\n"
"body{font-family:Arial,Helvetica,sans-serif;margin:24px;}"
"table{border-collapse:collapse;width:720px;max-width:100%}"
"th,td{border:1px solid #ccc;padding:8px;text-align:left}"
"th{background:#f2f2f2}"
"tfoot td{font-weight:bold}"
"tbody tr:nth-child(even){background:#fafafa}"
"</style>\n</head>\n<body>\n"
"<h2>Relatório de Compras</h2>\n"
"<table>\n"
"<thead><tr>"
"<th>#</th><th>Produto</th><th>Quantidade</th>" 
"<th>Preço unitário (R$)</th><th>Total do item (R$)</th>" 
"</tr></thead>\n<tbody>\n"
```

```
/* Lê linhas no formato: Produto Quantidade Preco */

while (fscanf(in, "%63s %d %lf", &produto, &qtd, &preco) == 3) {
    total_item = qtd * preco;
    total_compra += total_item;
    fprintf(out,
            "<tr><td>%d</td><td>%s</td><td>%d</td>" 
            "<td>% .2f </td><td>% .2f </td></tr>\n",
            item++, produto, qtd, preco, total_item);
}

fprintf(out,
        "</tbody>\n<tfoot>\n<tr><td colspan=\"4\">Valor total da compra</td>" 
        "<td>R$ % .2f </td></tr>\n"
        "</tfoot>\n</table>\n</body>\n</html>\n", total_compra);

fclose(in);
fclose(out);
printf("Relatorio gerado em: %s\n", out_name);
return 0;
}
```

Q06

Faça um que abra um arquivo HTML e elimine todas as “tags” do texto, ou seja, o programa deve gerar um novo arquivo em disco que elimine todas as tags HTML, que são caracterizadas por comandos entre “<>”. Veja abaixo um exemplo de um texto em HTML e do texto que deverá ser gerado pelo programa após eliminar as tags Html

```
<HTML>
<HEAD>
<TITLE>Minha Pagina Web</TITLE>
</HEAD>
<BODY>
Meu Texto<BR>
Minha Imagem<IMG SRC="img.jpg">
<A HREF="pag1.html">Meu Link</A>
</BODY>
</HTML>
```

```
Minha Pagina Web
Meu Texto
Minha Imagem
Meu Link
```

```
#include <stdio.h>
#include <stdlib.h>

int main() {
    FILE *entrada, *saída;
    char in_name[100], out_name[100];
    int ch;
    int dentro_tag = 0; // flag para indicar se está dentro de <...>

    printf("Digite o nome do arquivo HTML de entrada: ");
    scanf("%s", in_name);

    printf("Digite o nome do arquivo de saída (texto limpo): ");
    scanf("%s", out_name);

    entrada = fopen(in_name, "r");
    if (entrada == NULL) {
        perror("Erro ao abrir arquivo de entrada");
        return 1;
    }
```

```
saida = fopen(out_name, "w");

if (saida == NULL) {
    perror("Erro ao criar arquivo de saída");
    fclose(entrada);
    return 1;
}

while ((ch = fgetc(entrada)) != EOF) {
    if (ch == '<') dentro_tag = 1; // entrou em uma tag
    else if (ch == '>') dentro_tag = 0; // saiu da tag
    else if (!dentro_tag) fputc(ch, saida); // grava só o que não estiver em tags
}

fclose(entrada);
fclose(saida);

printf("Arquivo %s gerado sem tags HTML.\n", out_name);

return 0;
}
```

Q07

Escreva uma função, usando a linguagem C padrão, que leia o arquivo cujo nome é passado como parâmetro e calcule a retorne a soma de todos os elementos da coluna Quantidade. Um exemplo de arquivo é mostrado a seguir:

vendas.csv

```
Produto,Quantidade,Preço
Camisa,10,30.50
Calça,5,80.00
Tênis,8,120.00
```

Observação:

No final do arquivo podem existir uma ou mais linhas vazias, sem dados a serem considerados.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int soma_quantidades(const char *nome_arquivo) {
    FILE *fp;
    char linha[256];
    int soma = 0, qtd;
    float preco;
    char produto[100];

    fp = fopen(nome_arquivo, "r");
    if (fp == NULL) {
        perror("Erro ao abrir o arquivo");
        return -1;
    }

    // Lê a primeira linha (cabecalho) e descarta
    fgets(linha, sizeof(linha), fp);
```

```
// Lê as demais linhas até o final

while (fgets(linha, sizeof(linha), fp)) {

    // Ignora linhas vazias

    if (strlen(linha) <= 1) continue;

    // Exemplo de linha: Camisa,10,30.50

    if (sscanf(linha, "%[^,],%d,%f", produto, &qtd, &preco) == 3) soma += qtd;

}

fclose(fp);
return soma;
}
```

```
int main() {  
  
    char *arquivo = "vendas.csv";  
    int total = soma_quantidades(arquivo);  
  
    if (total >= 0) {  
        printf("A soma das quantidades é: %d\n", total);  
    }  
    return 0;  
}
```

Q08

Escreva uma função que leia o arquivo cujo nome fora passado como parâmetro e retorne um vetor com os nomes dos alunos que obtiveram nota média maior ou igual a 7. A título de exemplo, para o arquivo alunos.csv a seguir, a função deverá retornar {"Carlos", "Fernanda"}.

alunos.csv

```
Nome,Nota1,Nota2
Carlos,6.5,8.0
Fernanda,7.0,7.5
Lucas,5.0,6.0
```

Observação:

No final do arquivo podem existir uma ou mais linhas vazias, sem dados a serem considerados.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_ALUNOS 100
#define MAX_NOME 50

typedef struct {
    char nome[MAX_NOME];
    float n1, n2;
    float media;
} Aluno;

/* Lê o CSV, preenche o vetor de Aluno e devolve a quantidade lida */
size_t ler_alunos_csv(const char *arquivo, Aluno alunos[], size_t max_alunos) {
    FILE *fp = fopen(arquivo, "r");
    if (!fp) {
        perror("Erro ao abrir o arquivo");
        return 0;
    }

    /* Lógica para ler os dados do arquivo CSV e preencher o vetor de alunos */
    // Implementação detalhada da leitura dos dados CSV é omitida por brevidade.
}
```

Q09

Implemente uma função `atualizar(char *Arquivo, char *Produto, int Quantidade, float Preco)` que atualiza o preço e a quantidade de certo Produto no Arquivo cujo nome for passado como parâmetro. O formato (layout) do arquivo é mostrado a seguir:

produtos.csv

```
Produto,Quantidade,Preço  
Mouse,15,25.00  
Teclado,10,50.00  
Monitor,5,500.00
```

Observação:

A título de exemplo, na execução de `atualizar("funcionários.csv", "Mouse", 35, 32.50)`, a primeira linha de dados passará a ser `Mouse,35,32.50`.

Q10

Implemente uma função que leia o arquivo de clientes (passado como parâmetro) e remova um cliente específico baseado no ID fornecido pelo usuário, além de salvar o arquivo atualizado. Para esta atividade, o arquivo apresenta a seguinte estrutura:

clientes.csv

```
ID, Nome, Email  
1, Alice, alice@email.com  
2, Breno, breno@email.com  
3, Camila, camila@email.com
```

Observação:

Considere que ID seja do tipo inteiro.

Q11

Implemente uma função, em C padrão, que divide o arquivo de transações (passado como parâmetro) em dois arquivos com base no tipo de transação (vendas.csv e compras.csv). O layout do arquivo de transações é apresentado a seguir:

transacoes.csv

ID	Tipo	Valor
1	Compra	150.00
2	Venda	200.00
3	Compra	300.00
4	Venda	400.00

Observação:

No âmbito de banco de dados, essa operação de dividir um arquivo em partes, como a indicada acima, é chamada de particionamento.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void particionarArquivo(const char *arquivoEntrada) {

    FILE *entrada, *compras, *vendas;
    char linha[200];

    entrada = fopen(arquivoEntrada, "r");
    if (entrada == NULL) exit(1);

    compras = fopen("compras.csv", "w");
    vendas = fopen("vendas.csv", "w");

    if (compras == NULL || vendas == NULL) {
        perror("Erro ao criar arquivos de saída");
        fclose(entrada);
        exit(1);
    }

    while (fscanf(entrada, "%s", linha) != EOF) {
        if (strcmp(linha, "Compras") == 0) {
            fprintf(compras, "%s", linha);
        } else if (strcmp(linha, "Vendas") == 0) {
            fprintf(vendas, "%s", linha);
        }
    }

    fclose(entrada);
    fclose(compras);
    fclose(vendas);
}
```

```
// Lê cabeçalho e grava nos dois arquivos

if (fgets(linha, sizeof(linha), entrada)) {
    fprintf(compras, "%s", linha);
    fprintf(vendas, "%s", linha);
}

// Lê linha por linha
while (fgets(linha, sizeof(linha), entrada)) {
    if (strstr(linha, "Compra") != NULL) fprintf(compras, "%s", linha);
    else if (strstr(linha, "Venda") != NULL) fprintf(vendas, "%s", linha);
}

// Fecha arquivos
fclose(entrada);
fclose(compras);
fclose(vendas);
printf("Arquivos 'compras.csv' e 'vendas.csv' gerados com sucesso!\n");
}
```

```
int main() {  
    particionarArquivo("transacoes.csv");  
    return 0;  
}
```



Um atributo multivalorado é um atributo que pode ter mais de um valor associado a ele para uma única entidade ou registro. Em outras palavras, em vez de ter um único valor (como seria comum em atributos "simples"), um atributo multivalorado pode conter um conjunto de valores.

Livro ID	Título	Gêneros
1	<i>Dom Quixote</i>	<i>Romance, Aventura</i>
2	<i>O Senhor dos Anéis</i>	<i>Fantasia, Aventura, Épico</i>

Pergunta:

Dos atributos acima, qual é multivalorado?

Atributos multivvalorados

Um atributo multivvalorado é um atributo que pode ter mais de um valor associado a ele para uma única entidade ou registro. Em outras palavras, em vez de ter um único valor (como seria comum em atributos "simples"), um atributo multivvalorado pode conter um conjunto de valores.

Exemplo:

Na tabela, “**Gêneros**” é multivvalorado, dado que um livro pode pertencer a mais de um gênero:

Livro ID	Título	Autor	Gêneros
1	<i>Dom Quixote</i>	<i>Miguel de Cervantes</i>	<i>Romance, Aventura</i>
2	<i>O Senhor dos Anéis</i>	<i>J.R.R. Tolkien</i>	<i>Fantasia, Aventura, Épico</i>

Atributos multivvalorados

Um atributo multivvalorado é um atributo que pode ter mais de um valor associado a ele para uma única entidade ou registro. Em outras palavras, em vez de ter um único valor (como seria comum em atributos "simples"), um atributo multivvalorado pode conter um conjunto de valores.

Livro ID	Título	Autor	Gêneros
1	<i>Dom Quixote</i>	Miguel de Cervantes	Romance, Aventura
2	<i>O Senhor dos Anéis</i>	J.R.R. Tolkien	Fantasia, Aventura, Épico



Como o atributo multivvalorado “Gêneros” poderia ser modelado em um arquivo/planilha CSV?

Atributos multivvalorados

Um atributo multivvalorado é um atributo que pode ter mais de um valor associado a ele para uma única entidade ou registro. Ou seja, pode conter um conjunto de valores.



Um caractere especial pode ser usado para delimitar que um atributo pode ser multivvalorado.
No caso, uma possibilidade é usar aspas.

[livros.csv](#)

```
LivroID,Titulo,Autor,Genero  
1,"Dom Quixote",Miguel de Cervantes,"Romance, Aventura"  
2,"O Senhor dos Anéis",J.R.R. Tolkien,"Fantasia, Aventura, Épico"
```

Estrutura de Dados

1º semestre de 2025

Pastas

Tópico adicional

Professor Marcelo Eustáquio



Atributos de um arquivo

A estrutura **struct stat** é definida em **<sys/stat.h>** e armazena características relacionadas a arquivos e diretórios no sistema de arquivos. Aqui está um detalhamento dos principais membros da estrutura e seus significados:

```
struct stat {  
    dev_t      st_dev;        // Identificador do dispositivo onde o arquivo está armazenado  
    ino_t      st_ino;        // Número do inode do arquivo  
    mode_t     st_mode;       // Tipo de arquivo e permissões (bits combinados)  
    nlink_t    st_nlink;      // Número de Links físicos (hard links)  
    uid_t      st_uid;        // ID do proprietário (usuário)  
    gid_t      st_gid;        // ID do grupo do proprietário  
    dev_t      st_rdev;       // Identificador de dispositivo, se for especial  
    off_t      st_size;       // Tamanho do arquivo em bytes  
    blksize_t  st_blksize;    // Tamanho ideal de bloco para operações de I/O  
    blkcnt_t   st_blocks;     // Número de blocos alocados para o arquivo  
    time_t     st_atime;      // Último acesso ao arquivo  
    time_t     st_mtime;      // Última modificação no conteúdo do arquivo  
    time_t     st_ctime;      // Última mudança no status do arquivo  
};
```

Atributos de um arquivo

A seguir, é apresentada a correspondência entre os membros da estrutura **struct stat** e os códigos de formato (%) apropriados para o printf (**1^a parte**):

1	st_dev	dev_t (inteiro)	%ld ou %llu	ID do dispositivo
2	st_ino	ino_t (inteiro)	%ld ou %llu	Número do inode
3	st_mode	mode_t (inteiro)	%o	Tipo e permissões de arquivo (octal)
4	st_nlink	nlink_t (inteiro)	%d	Número do links
5	st_uid	uid_t (inteiro)	%d	ID do proprietário
6	st_gid	gid_t (inteiro)	%d	ID do grupo

Atributos de um arquivo

A seguir, é apresentada a correspondência entre os membros da estrutura **struct stat** e os códigos de formato (%) apropriados para o printf (**2ª parte**):

1	st_rdev	dev_t (inteiro)	%ld ou %llu	ID do dispositivo especial
2	st_size	off_t (inteiro)	%ld	Tamanho do arquivo, em bytes
3	st_blksize	blksize_t (inteiro)	%ld	Tamanho ideal de bloco para I/O
4	st_blocks	blkcnt_t (inteiro)	%ld	Número de blocos alocados
5	st_atime	time_t (inteiro)	%ld	Último acesso
6	st_mtime	time_t (inteiro)	%ld	Última modificação

Acessando atributos

```
#include <sys/stat.h>
#include <stdio.h>
#include <time.h>

int main() {

    const char *filePath = ".../main.c"; // Caminho do arquivo
    struct stat fileStat;
    if (stat(filePath, &fileStat) == -1) return 1; // Recupera informações do arquivo

    printf("Tamanho do arquivo: %ld bytes\n", fileStat.st_size);
    printf("Última modificação: %s", ctime(&fileStat.st_mtime));
    printf("Número de links: %ld\n", fileStat.st_nlink);
    printf("ID do proprietário: %d\n", fileStat.st_uid);
    printf("Permissões: %o\n", fileStat.st_mode & 0777);

    return 0;
}
```



Exercício

Modifique o código-fonte apresentado anteriormente para mostrar outros atributos do arquivo cujo nome está armazenado na string `filePath`. Como exemplos, considere o número de blocos alocados para o arquivo (`st_blocks`), o último acesso ao arquivo (`st_atime`) e a data da última modificação sofrida pelo arquivo (`st_mtime`).

Listando arquivos

O código apresentado a seguir, escrito na linguagem C, lista todos os arquivos (não diretórios) em um diretório especificado no sistema Windows.

```
#include <stdio.h>
#include <windows.h>
#include <string.h>
```

Observação:

Em windows.h, encontram-se a estrutura **WIN32_FIND_DATA** e o tipo **HANDLE**, que são úteis na manipulação de diretórios. A estrutura WIN32_FIND_DATA é usada pela API do Windows para armazenar informações sobre arquivos ou diretórios encontrados por funções como FindFirstFile e FindNextFile. Ademais, o tipo HANDLE é um identificador genérico usado pelas APIs do Windows para representar referências a objetos no sistema operacional, como arquivos, diretórios, threads, eventos, entre outros.

Listando arquivos

```
void listFilesInDirectory(const char *directoryPath) {
    WIN32_FIND_DATA findFileData;
    HANDLE hFind;
    char searchPath[MAX_PATH];
    sprintf(searchPath, sizeof(searchPath), "%s\\*", directoryPath);
    hFind = FindFirstFile(searchPath, &findFileData);
    if (hFind == INVALID_HANDLE_VALUE) {
        printf("Erro ao abrir o diretório ou diretório vazio.\n");
        return;
    }

    do {
        if (strcmp(findFileData.cFileName, ".") == 0 || strcmp(findFileData.cFileName, "..") == 0)
            continue;
        if (!(findFileData.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY))
            printf("%s\\%s\n", directoryPath, findFileData.cFileName);
    } while (FindNextFile(hFind, &findFileData) != 0); // Continua buscando
    FindClose(hFind);
}
```

Listando arquivos

```
int main() {  
  
    const char *directoryPath = ".."; // Substitua pelo caminho do diretório desejado  
    listFilesInDirectory(directoryPath);  
    return 0;  
}
```

Atributos de um arquivo

Considere a seguinte função que recebe o caminho de um arquivo e utiliza a estrutura struct stat para identificar seu tipo:

```
#include <stdio.h>
#include <sys/stat.h>

void checkFileType(const char *filePath) {
    struct stat fileStat;
    if (stat(filePath, &fileStat) == -1) {
        perror("Erro ao acessar o arquivo");
        return;
    }

    // Verificar tipo de arquivo aqui !!!
}
```

Complete a função checkFileType para imprimir o tipo do arquivo (se é um arquivo regular, diretório, link simbólico, etc.), utilizando as macros apropriadas da struct stat. Dica: Utilize as macros S_ISREG(), S_ISDIR(), S_ISLNK() e outras.





Atributos de um arquivo

Implemente uma função que recebe o caminho de um diretório e calcula o tamanho total dos arquivos regulares presentes nele (não contando subdiretórios). A função deve usar a struct stat para verificar o tamanho de cada arquivo.

A função deve percorrer o diretório e somar o valor de st_size de cada arquivo regular.

Dica: Use opendir(), readdir() e stat().



Atributos de um arquivo

Utilizando a estrutura `struct stat`, escreva uma função que, dado um caminho de arquivo, exiba as permissões de leitura, escrita e execução para o proprietário, grupo e outros. A função deve imprimir as permissões em formato legível, como `rwxr-xr--`, onde `r` significa leitura, `w` significa escrita e `x` significa execução.

Dica: Use os bits de `st_mode` e as macros `S_IRUSR`, `S_IWUSR`, `S_IXUSR`, etc., para verificar as permissões.



Atributos de um arquivo

Implemente uma função que, dada a data de criação de um arquivo, calcula e imprime a "idade" do arquivo em dias. A função deve usar a estrutura `stat` e seus campos `st_ctime` (última modificação do status) e `st_mtime` (última modificação do conteúdo).

Dica: Utilize a função `time()` e manipule os timestamps para calcular a diferença de dias.



Atributos de um arquivo

Implemente uma função que percorre todos os arquivos de um diretório e encontra o arquivo de maior tamanho. Para cada arquivo, utilize a estrutura struct stat para acessar o campo st_size e determinar o maior arquivo. A função deve retornar o nome do arquivo e seu tamanho.

Dica: Utilize opendir(), readdir() e stat(). Lembre-se de ignorar subdiretórios.

