

Estrutura de Dados

1º semestre de 2025

Pesquisa sequencial

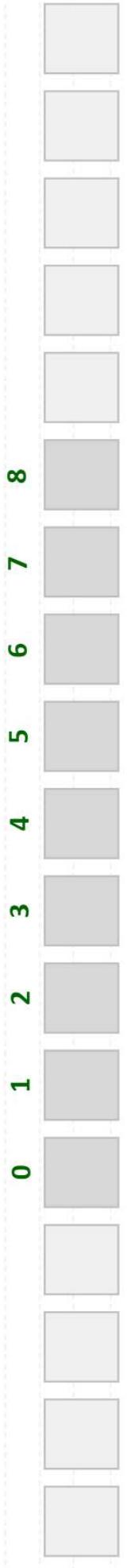
Tema #03

Professor Marcelo Eustáquio



Pesquisa sequencial

A **pesquisa sequencial** (ou busca linear) é um método simples de busca em estruturas de dados lineares, como listas ou vetores, onde cada elemento é verificado **sequencialmente**, um após o outro, até que o elemento desejado seja encontrado ou até que todos os elementos tenham sido examinados.

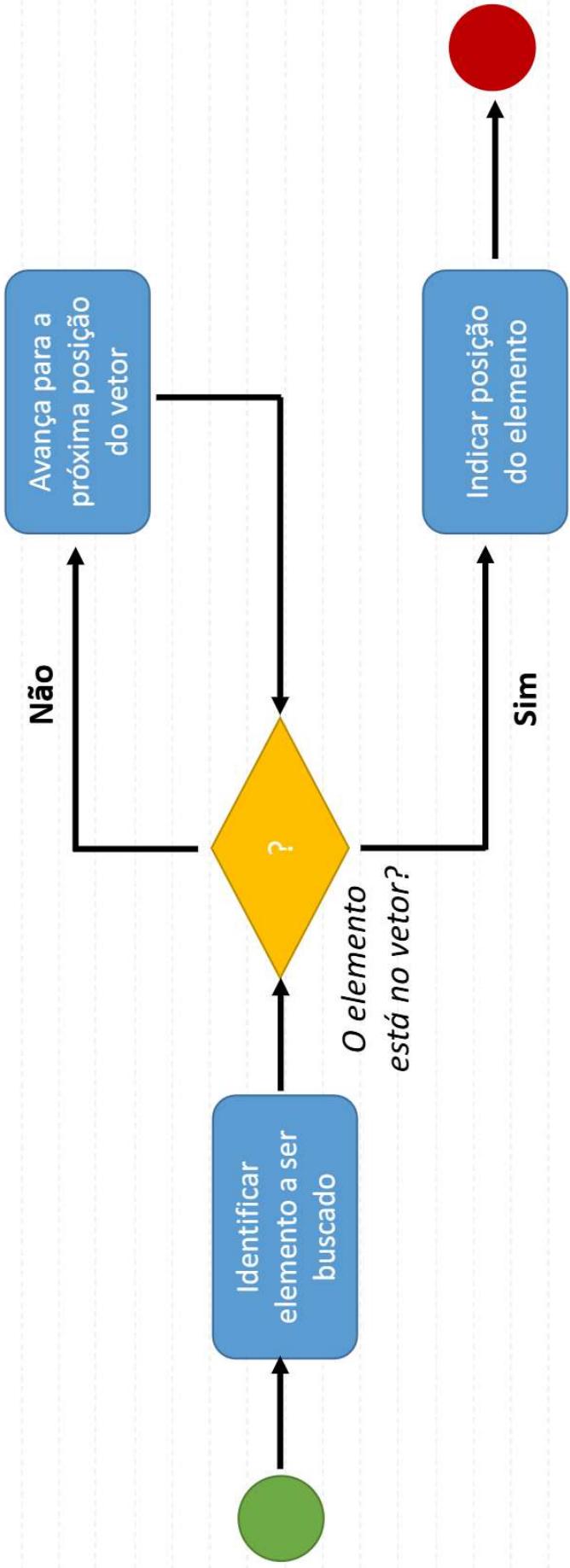


*Todos os elementos são
verificados!!!*

É um dos algoritmos mais básicos e intuitivos para busca, mas pode não ser o mais eficiente para conjuntos de dados grandes.

Pesquisa sequencial

A **pesquisa sequencial** (ou busca linear) é um método simples de busca em estruturas de dados lineares, como listas ou vetores, onde cada elemento é verificado **sequencialmente**, um após o outro, até que o elemento desejado seja encontrado ou até que todos os elementos tenham sido examinados.



Pesquisa sequencial

A **pesquisa sequencial** (ou busca linear) é um método simples de busca em estruturas de dados lineares, como listas ou vetores, onde cada elemento é verificado **sequencialmente**, um após o outro, até que o elemento desejado seja encontrado ou até que todos os elementos tenham sido examinados.

```
for (int i = 0; i < N; i++)
    if (V[i] == Chave) return i; // i é a posição do elemento procurado!
```

Observação:

Qual o melhor caso do algoritmo, ou seja, qual a situação que o resultado da pesquisa sequencial é identificado de forma mais rápida? E qual o pior caso, no qual a resposta é identificada de forma mais demorada?

Pesquisa sequencial



Para responder esta questão, considere os elementos do seguinte vetor:

```
int v[] = {14, 29, 37, 11, 43, 25, 19, 32, 16, 22,  
          40, 13, 28, 35, 10, 45, 20, 38, 17, 24,  
          30, 41, 15, 27, 33, 18, 26, 39, 12, 21}
```

Agora responda:

- 1** Quantas comparações são necessárias para identificar a posição do elemento 16?
- 2** E qual o número de comparações para determinar a posição do elemento 21?
- 3** E quantas comparações são necessárias para descobrir que um elemento não está no vetor?

Pesquisa sequencial

Em relação à complexidade de tempo, são características da busca sequencial:

1	Melhor caso O(1) : o elemento procurado está na primeira posição.
2	Pior caso O(n) : o elemento procurado está na última posição.
3	Caso médio Em média, o algoritmo realiza $n/2$ comparações para realizar uma pesquisa.
4	Espaço Não requer memória adicional para realizar a pesquisa.

Aplicabilidade:

Funciona para listas ordenadas ou não, pois não depende de nenhuma organização específica.

Pesquisa sequencial

As vantagens da pesquisa sequencial estão relacionadas com a simplicidade e com a versatilidade do algoritmo ao passo de que as desvantagens residem na ineficiência em relação a alternativas mais rápidas como a pesquisa binária.



De fácil implementação, funciona em qualquer tipo de lista (ordenada, desordenada, dinâmica) e não tem requisitos de ordenação.



Ineficiente para grandes conjuntos de dados (ou seja, o tempo de execução cresce linearmente com o tamanho da lista ou conjunto de elementos).

```

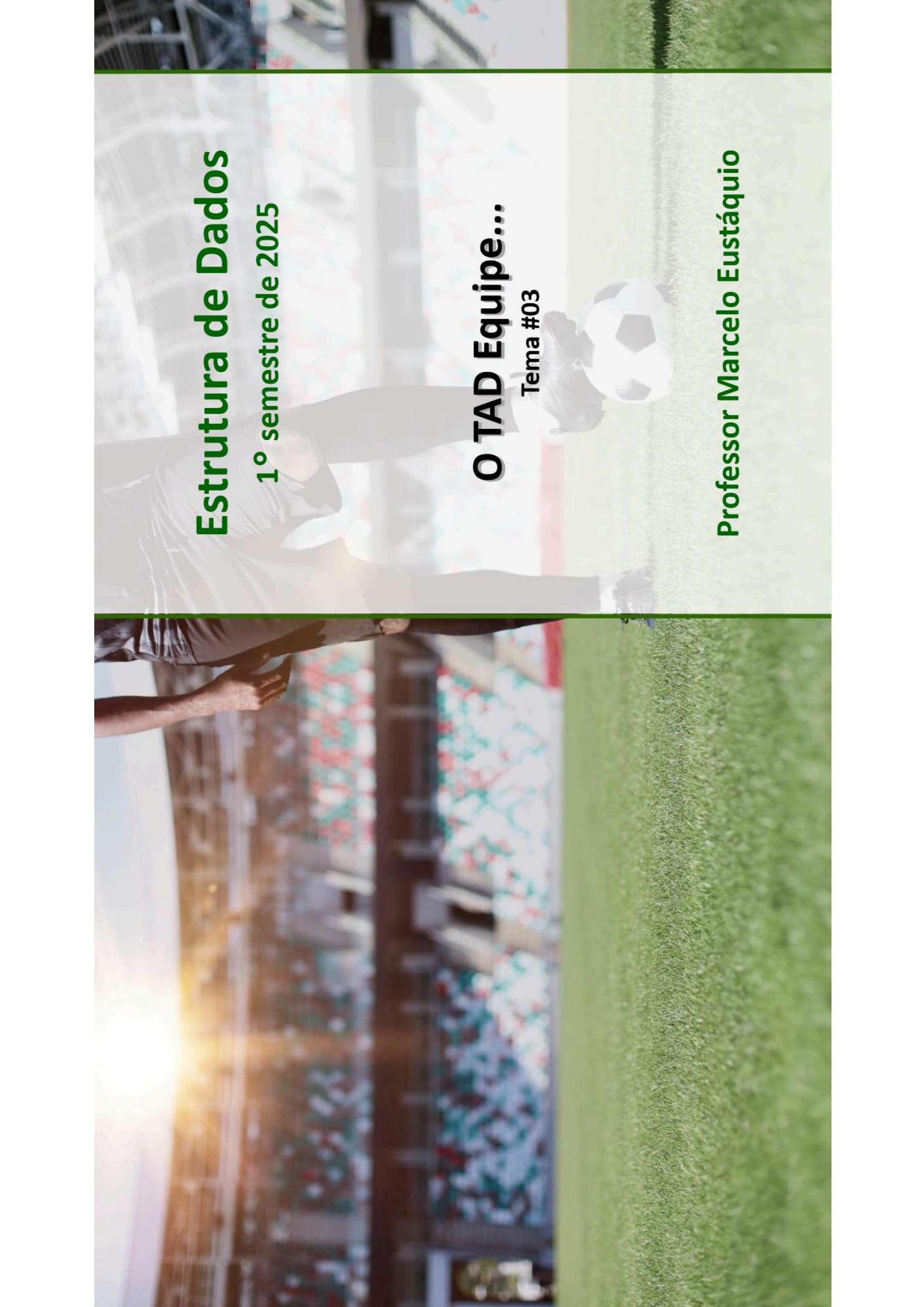
#include <stdio.h>
#include <stdlib.h>

int encontrar_posicao(int *vetor, int tamanho, int elemento) { // Usando int *vetor
    for (int i = 0; i < tamanho; i++)
        if (vetor[i] == elemento)
            return i;
    return -1;
}

int main() {
    int numeros[] = {14, 29, 37, 11, 43, 25, 19, 32, 16, 22,
                    40, 13, 28, 35, 10, 45, 20, 38, 17, 24,
                    30, 41, 15, 27, 33, 18, 26, 39, 12, 21};
    int tamanho = sizeof(numeros) / sizeof(numeros[0]);
    int elemento = 35;
    int posicao = encontrar_posicao(numeros, tamanho, elemento);

    if (posicao != -1) printf("Elemento %d encontrado na posição: %d\n", elemento, posicao);
    else printf("Elemento %d não encontrado no vetor.\n", elemento);
    return 0;
}

```

A composite background image featuring a soccer player in mid-air performing a bicycle kick over a soccer ball on a green field, and a person's arm reaching out towards a stack of books on a shelf.

Estrutura de Dados

1º semestre de 2025

O TAD Equipe...

Tema #03

Professor Marcelo Eustáquio



Pos	Estado	Equipe	Pts	J	V	E	D	GP	GC	SG
1	Rio de Janeiro	Botafogo (C)	79	38	23	10	5	59	29	+30
2	São Paulo	Palmeiras	73	38	22	7	9	60	33	+27
3	Rio de Janeiro	Flamengo	70	38	20	10	8	61	42	+19
4	Ceará	Fortaleza	68	38	19	11	8	53	39	+14
5	Rio Grande do Sul	Internacional	65	38	18	11	9	53	36	+17
6	São Paulo	São Paulo	59	38	17	8	13	53	43	+10
7	São Paulo	Corinthians	56	38	15	11	12	54	45	+9
8	Bahia	Bahia	53	38	15	8	15	49	49	0
9	Minas Gerais	Cruzeiro	52	38	14	10	14	43	41	+2
10	Rio de Janeiro	Vasco da Gama	50	38	14	8	16	43	56	-13
11	Bahia	Vitória	47	38	13	8	17	45	52	-7
12	Minas Gerais	Atlético Mineiro	47	38	11	14	13	47	54	-7
13	Rio de Janeiro	Fluminense	46	38	12	10	16	33	39	-6
14	Rio Grande do Sul	Grêmio	45	38	12	9	17	44	50	-6
15	Rio Grande do Sul	Juventude	45	38	11	12	15	47	59	-12
16	São Paulo	Red Bull Bragantino	44	38	10	14	14	44	48	-4
17	Paraná	Athletico Paranaense	42	38	11	9	18	40	46	-6
18	Santa Catarina	Criciúma	38	38	9	11	18	42	61	-19
19	Goiás	Atlético Goianiense	30	38	7	9	22	29	58	-29
20	Mato Grosso	Cuiabá	30	38	6	12	20	29	49	-20

Pesquisa sequencial

Como implementar o tipo abstrato de dados Equipe, considerando os atributos presentes no arquivo **Tabela2024.csv**?

```
typedef struct {
    int Posicao;
    char Estado[32];
    char Time[32];
    int Pontos;
    int J; // Nº de jogos
    int V; // Nº de vitórias
    int E; // Nº de empates
    int D; // Nº de derrotas
    int GP; // Gols marcados
    int GC; // Gols sofridos
    int SD; // Saldo de gols
    float Aproveitamento; // % de pontos conquistados em relação ao total de pontos disputados
} Equipe;
```





Pesquisa sequencial

Implementando uma função para ler o conjunto de dados do arquivo CSV e armazenar o resultado em um vetor de “Equipes”.

Pesquisa sequencial



Implementando uma função para determinar o número de gols marcados por uma determinada equipe, cujo nome fora passado como parâmetro.

```
int getGolsMarcados(Equipe *Tabela, char *NomeEquipe) {  
    for (int i = 0; i < 20; i++)  
        if (strcmp(Tabela[i].Time, NomeEquipe) == 0)  
            return Tabela[i].GP;  
    return -1;  
}
```

Pesquisa sequencial



Considerando o tipo abstrato de dados Equipe, implemente funções para realizar as seguintes operações (atente-se para os tipos dos valores retornados em cada caso):

- 1** Determinar o número de pontos da equipe está em uma posição específica.
- 2** Determinar o aproveitamento de uma certa equipe.
- 3** Determinar a equipe que teve o maior saldo de gols.
- 4** Determinar a equipe que teve o menor número de gols sofridos.

Atente-se para a solução de cada um desses exercícios: elas serão importantes na resolução do primeiro trabalho prático deste semestre.

Estrutura de Dados

1º semestre de 2025

Pesquisa sequencial

Questões

Professor Marcelo Eustáquio



Q01

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>

#define MAX_LINE 256

typedef struct { // Struct para armazenar cada registro do CSV
    char produto[50];
    int quantidade;
    float preco;
} Venda;

int soma_quantidades(char *nome_arquivo) {

    FILE *fp = fopen(nome_arquivo, "r");
    if (!fp) return -1;

    char linha[MAX_LINE];
    int soma = 0;
```

```
fgets(linha, MAX_LINE, fp); // Lê a primeira linha (cabeçalho) e descarta

while (fgets(linha, MAX_LINE, fp)) {
    if (strlen(linha) <= 1) continue;
    Venda v;
    if (sscanf(linha, "%49[^,],%d,%f", &v.produto, &v.quantidade, &v.preco) == 3)
        soma += v.quantidade;
}

fclose(fp);
return soma;
}

int main() {

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

Q02

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_LINE 256
#define MAX_ALUNOS 100

// Struct para armazenar dados de cada aluno

typedef struct {
    char nome[50];
    float nota1;
    float nota2;
} Aluno;

// Função que lê o arquivo e retorna os nomes dos alunos com média >= 7

int alunos_aprovados(const char *nome_arquivo, char aprovados[][50]);
```

```
int main() {
    char aprovados[MAX_ALUNOS][50];
    int n = alunos_aprovados("alunos.csv", aprovados);

    if (n >= 0) {
        printf("Alunos aprovados:\n");
        for (int i = 0; i < n; i++) printf("%s\n", aprovados[i]);
    }
    return 0;
}

int alunos_aprovados(const char *nome_arquivo, char aprovados[][50]) {
    FILE *fp = fopen(nome_arquivo, "r");
    if (!fp) {
        printf("Erro ao abrir arquivo");
        return -1;
    }

    char linha[MAX_TNF1:
```

```
int alunos_aprovados(const char *nome_arquivo, char aprovados[][50]) {
    FILE *fp = fopen(nome_arquivo, "r");
    if (!fp) return -1;
    char linha[MAX_LINE];
    int count = 0;
    fgets(linha, MAX_LINE, fp); // Ignora o cabeçalho
    while (fgets(linha, MAX_LINE, fp)) {
        if (strlen(linha) <= 1) continue; // ignora linhas vazias
        Aluno a;
        if (sscanf(linha, "%49[^,],%f,%f", a.nome, &a.notas1, &a.notas2) == 3) {
            float media = (a.notas1 + a.notas2) / 2.0;
            if (media >= 7.0) {
                strcpy(aprovados[count], a.nome);
                count++;
            }
        }
    }
    fclose(fp);
    return count; // retorna quantidade de alunos aprovados
}
```

Q03

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`.

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

#define MAX_LINE 256

// Função que atualiza quantidade e preço de um produto

void atualizar(const char *arquivo, const char *produto, int quantidade, float preco) {

    FILE *fp = fopen(arquivo, "r");
    if (!fp) return;

    FILE *temp = fopen("temp.csv", "w");
    if (!temp) {
        perror("Erro ao criar arquivo temporário");
        fclose(fp);
        return;
    }

    // Implementação da função para escrever no arquivo temporário
    // ...

    fclose(temp);
    fclose(fp);
}
```

```
char linha[MAX_LINE];
int encontrado = 0;

if (fgets(linha, MAX_LINE, fp)) fputs(linha, temp);

while (fgets(linha, MAX_LINE, fp)) {
    if (strlen(linha) <= 1) continue; // ignora linhas vazias

    char nome[50];
    int qtd;
    float prc;

    if (sscanf(linha, "%49[^,],%d,%f", nome, &qtd, &prc) == 3) {

        if (strcmp(nome, produto) == 0) {
            // Atualiza o produto
            fprintf(temp, "%s,%d,%f\n", nome, qtdade, preco);
            encontrado = 1;
        } else {
    }
```

```
// Copia sem alterações
fprintf(temp, "%s,%d,%f\n", nome, qtd, prc);
}

fclose(fp);
fclose(temp);

remove(arquivo); // Substitui o arquivo original pelo atualizado
rename("temp.csv", arquivo);
if (encontrado) printf("Produto '%s' atualizado com sucesso!\n", produto);
else printf("Produto '%s' não encontrado no arquivo.\n", produto);
}

// Exemplo de uso
int main() {
    atualizar("produtos.csv", "Mouse", 35, 32.50);
    return 0;
}
```

Q04

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.

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

#define MAX_LINE 256

// Função que remove cliente pelo ID

void remover_cliente(const char *arquivo, int id_remover) {

    FILE *fp = fopen(arquivo, "r");
    if (!fp) return;

    FILE *temp = fopen("temp.csv", "w");
    if (!temp) {
        perror("Erro ao criar arquivo temporário");
        fclose(fp);
        return;
    }

    int id_lido;
    while (fscanf(fp, "%d", &id_lido) != EOF) {
        if (id_lido != id_remover)
            fprintf(temp, "%d\n", id_lido);
    }

    fclose(fp);
    fclose(temp);

    remove(arquivo);
    rename("temp.csv", arquivo);
}
```

```
char linha[MAX_LINE];
int encontrado = 0;

if (fgets(linha, MAX_LINE, fp)) fputs(linha, temp); // Copiar cabeçalho
while (fgets(linha, MAX_LINE, fp)) {

    if (strlen(linha) <= 1) continue; // ignora linhas vazias
    int id;
    char nome[50], email[100];

    if (sscanf(linha, "%d,%49[^,],%99[^\\n]", &id, nome, email) == 3) {
        if (id == id_remover) {
            encontrado = 1; // não grava esta linha
            continue;
        }
        fprintf(temp, "%d,%s,%s\n", id, nome, email);
    }
}
```

```
fclose(fp);
fclose(temp);

// Substitui o arquivo original pelo atualizado

remove(arquivo);
rename("temp.csv", arquivo);

if (encontrado) printf("Cliente com ID %d removido com sucesso!\n", id_remover);
else printf("Cliente com ID %d não encontrado no arquivo.\n", id_remover);
}

// Exemplo de uso

int main() {
    remover_cliente("clientes.csv", 2); // Remove o cliente com ID = 2
    return 0;
}
```

Q05

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>

#define MAX_LINE 256

void particionar_transacoes(const char *arquivo) {

    FILE *fp = fopen(arquivo, "r");
    if (!fp) return;

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

    if (!compras || !vendas) {
        perror("Erro ao criar arquivos de saída");
        fclose(fp);
        return;
    }

    // Implementação da lógica de particionamento
    // Aqui iria o código para ler linhas do arquivo
    // e escrever em compras.csv e vendas.csv
}
```

```
char linha[MAX_LINE];

// Lê cabeçalho e escreve nos dois arquivos

if (fgets(linha, MAX_LINE, fp)) {
    fputs(linha, compras);
    fputs(linha, vendas);
}

// Processa cada linha do arquivo

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

    if (strlen(linha) <= 1) continue; // ignora linhas vazias

    int id;
    char tipo[20];
    float valor;
```

```
if (sscanf(linha, "%d,%19[^,],%f", &id, tipo, &valor) == 3) {
    if (strcmp(tipo, "Compra") == 0) fprintf(compras, "%d,%s,%.2f\n", id, tipo, valor);
    else if (strcmp(tipo, "Venda") == 0) fprintf(vendas, "%d,%s,%.2f\n", id, tipo, valor);
}

fclose(fp);
fclose(compras);
fclose(vendas);

printf("Particionamento concluído: arquivos 'compras.csv' e 'vendas.csv' gerados.\n");

// Exemplo de uso

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