



Professor
Maromo

LINGUAGEM DE PROGRAMAÇÃO

Material 005



GitHub
maromo71

C



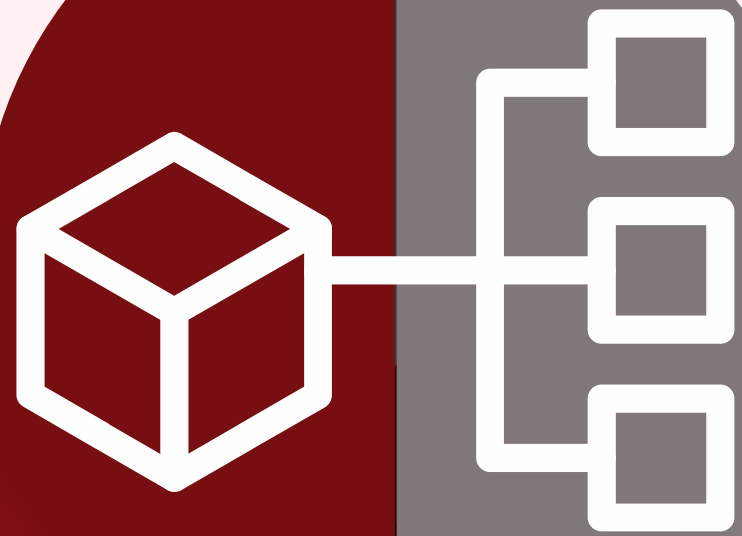
Agenda



Matrizes

- **Unidimensionais**
- **N-dimensionais**
- **Strings**

Material: LP_005



Matrizes

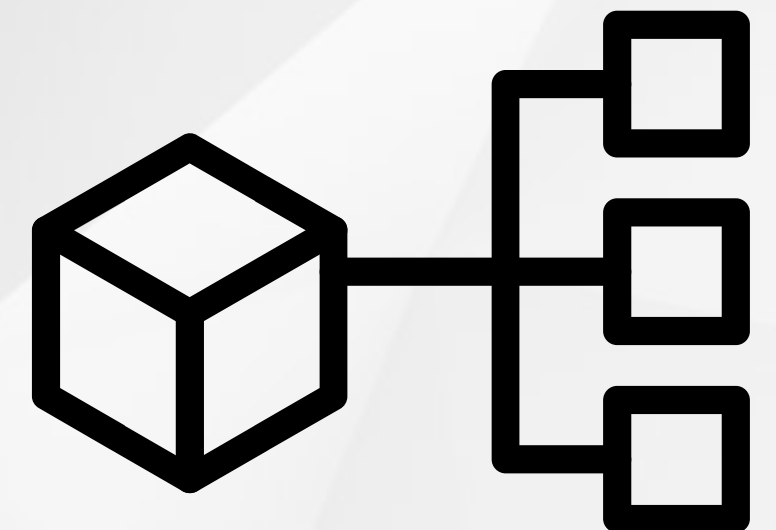


Matrizes são estruturas de dados utilizadas para armazenar conjuntos de elementos com características similares. Cada matriz é **identificada por um nome único**, e seus elementos são acessados por **meio de índices**. As matrizes podem ser classificadas como unidimensionais ou multidimensionais, dependendo do número de índices necessários para acessar seus elementos.

Nota sobre Matrizes

- É importante notar que matrizes de qualquer dimensão são caracterizadas por terem todos os elementos pertencentes ao mesmo tipo de dado.
- Para se declarar uma matriz/vetor podemos utilizar a seguinte forma geral:

tipo_da_variável nome_da_variável [tamanho];



Declaração de Vetores



Vetor de inteiros: Declara um vetor com espaço para 5 inteiros.

```
int numeros[5];
```

Vetor de floats: Declara um vetor com espaço para 10 números de ponto flutuante.

```
float notas[10];
```

Vetor de caracteres (string): Declara um vetor de caracteres com espaço para 50 caracteres (incluindo o caractere nulo \0 que indica o fim da string).

```
char nome[50];
```

Inicialização na declaração: Declara e inicializa um vetor de inteiros.

```
int diasMes[12] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
```

Tamanho automático: Declara um vetor de inteiros e deixa o compilador determinar seu tamanho com base na inicialização.

```
int algunsNumeros[] = {2, 4, 6, 8, 10};
```

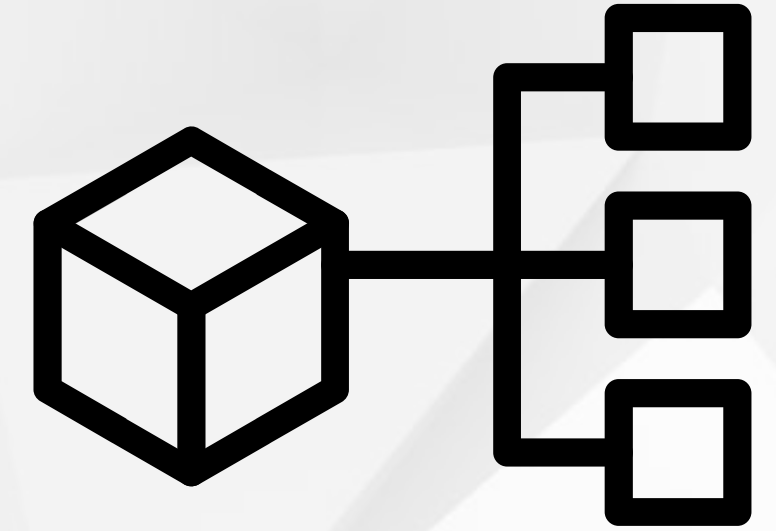
Vetor de strings: Declara um vetor de strings, onde cada string pode ter até 20 caracteres.

```
char nomes[5][20];
```

Declaração de Matrizes

Matriz 2x2 de inteiros: Declara uma matriz 2x2.

```
int matriz[2][2];
```



Matriz 3x4 de floats: Declara uma matriz com 3 linhas e 4 colunas para números de ponto flutuante.

```
float valores[3][4];
```

Inicialização na declaração: Declara e inicializa uma matriz 2x3 de inteiros.

```
int tabela[2][3] = {{1, 2, 3}, {4, 5, 6}};
```

Tamanho automático: Declara uma matriz e deixa o compilador determinar seu tamanho com base na inicialização.

```
int matrizAuto[][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
```

Matriz 3D: Declara uma matriz tridimensional. Por exemplo, uma matriz 2x3x4 pode ser visualizada como 2 matrizes de 3x4.

```
int cubo[2][3][4];
```

Inicialização de matriz 3D: Declara e inicializa uma matriz tridimensional 2x2x2.

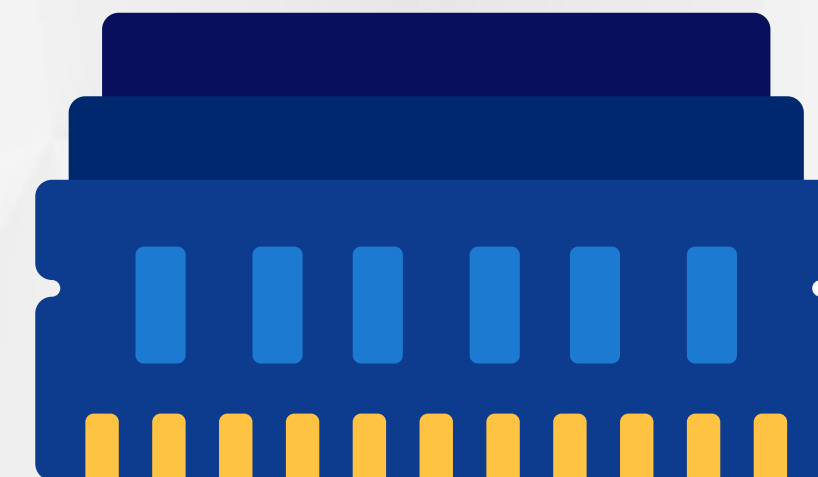
```
int matriz3D[2][2][2] = {{{1, 2}, {3, 4}}, {{5, 6}, {7, 8}}};
```

Declaração de Matrizes

Dado o vetor:

```
int v[5] = {10,12,14,16,18};
```

Veja uma representação na memória



Índice	Valor (v[i])	Endereço de Memória
0	10	0x1000 + 0
1	12	0x1000 + 4
2	14	0x1000 + 8
3	16	0x1000 + 12
4	18	0x1000 + 16



- A primeira coluna **mostra o índice do vetor**.
- A segunda coluna **mostra o valor armazenado no índice correspondente** do vetor.
- A terceira coluna **representa um endereço de memória** simplificado para cada elemento do vetor.

Nesta simulação, os endereços de memória incrementam de 4 em 4, o que é comum para variáveis do tipo int em muitos sistemas. No entanto, em um sistema real, os endereços seriam mais complexos e não tão ordenados.

Sobre carga inicial

Quando declarados, vetores, assim como variáveis individuais, contêm valores indeterminados, frequentemente referidos como "**lixo**", em suas posições. Para evitar isso, podemos inicializar todos os elementos de um vetor automaticamente utilizando a seguinte sintaxe:

```
tipo var[n] = { valor1, valor2, ..., valorn };
```


Exemplos de carga inicial

Inicialização Manual com Valores Específicos

int meuVetor[5] = {1, 2, 3, 4, 5}; // Inicializa o vetor com valores específicos.

Inicialização Automática com Zeros

int meuVetor[5] = {0}; // Todos os elementos são inicializados com zero.

Inicialização Parcial

int meuVetor[5] = {1, 2}; // Os dois primeiros elementos são 1 e 2, os demais são inicializados com zero.

Tamanho Automático com Inicialização

int meuVetor[] = {1, 2, 3, 4, 5}; // O compilador determina o tamanho com base no número de elementos na inicialização

Nota sobre cargas

Um vetor declarado com N elementos, e se forem colocados apenas k valores ($k < N$) na carga inicial do vetor. Os elementos não carregados ficarão com o valor ZERO.

Suponhamos a seguinte declaração:

```
int v[10] = {10,20,30};
```

No exemplo anterior, os três primeiros elementos do vetor (índices 0, 1 e 2) ficam iniciados com os valores 10, 20 e 30, respectivamente, e todos os outros ficam iniciados com o valor 0.

Assim, as seguintes instruções são equivalentes

```
int v[10] = {10,20,30};
```

```
int v[10] = {10,20,30,0,0,0,0,0,0,0};
```

Exemplo de Vetor

Leitura e Impressão de vetor

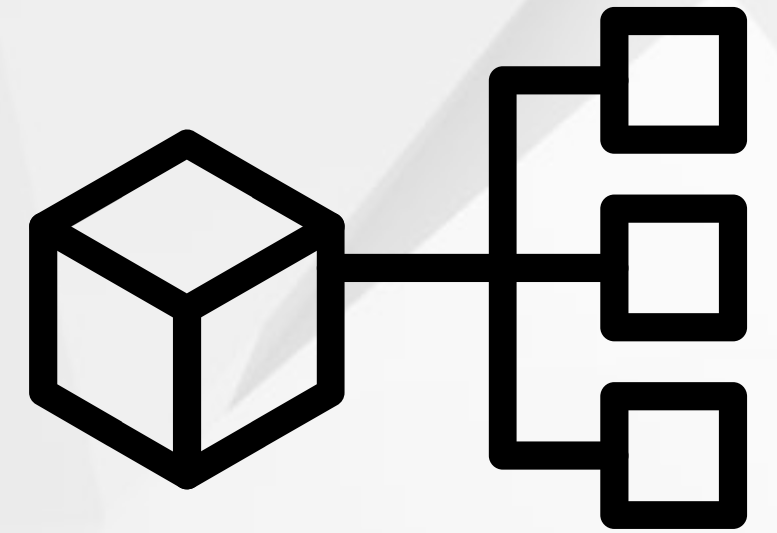
```
#include <stdio.h>

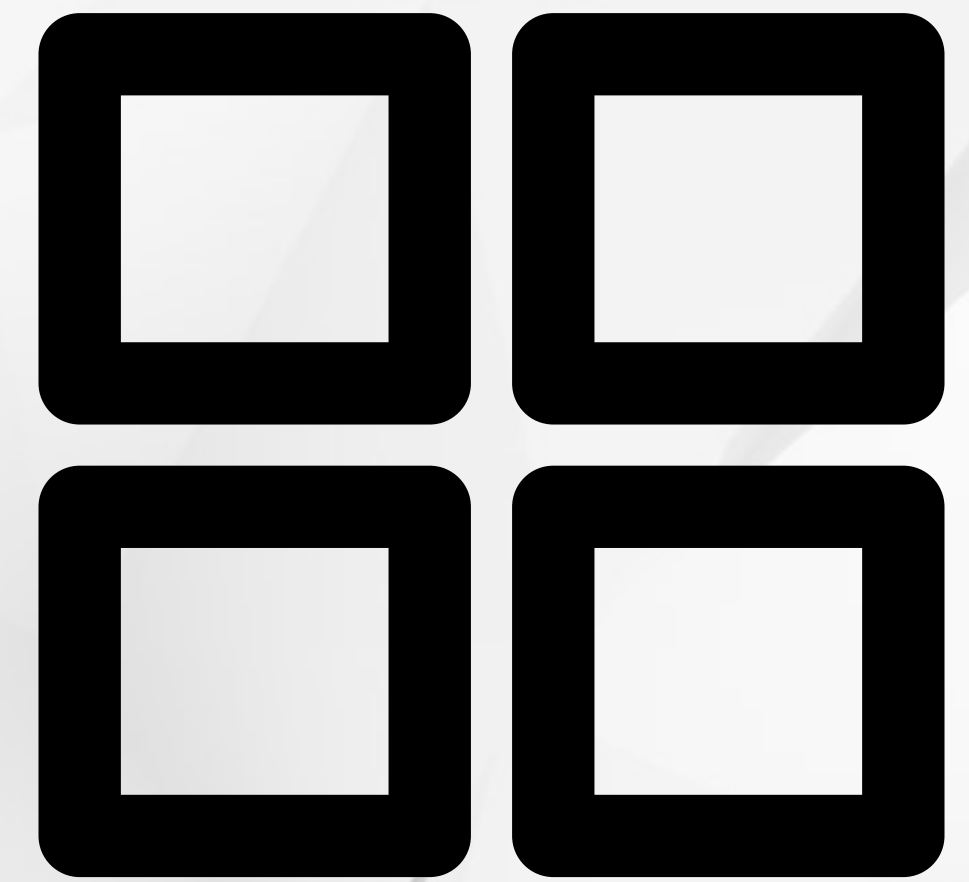
int main() {
    int numeros[5]; // Declaração do vetor
    int i; // Variável de controle para o loop

    // Leitura dos valores
    printf("Digite 5 numeros:\n");
    for (i = 0; i < 5; i++) {
        printf("Numero %d: ", i + 1);
        scanf("%d", &numeros[i]);
    }

    // Impressão dos valores
    printf("\nOs numeros digitados foram:\n");
    for (i = 0; i < 5; i++) {
        printf("%d ", numeros[i]);
    }

    return 0;
}
```





Matriz

2D

Matriz 2D

Uma matriz 2D (ou bidimensional) pode ser visualizada como uma tabela composta por linhas e colunas. Em termos de programação, particularmente em linguagens como C, uma matriz 2D é essencialmente um "**vetor de vetores**".

Sintaxe básica

```
tipo nome_da_matriz[num_linhas][num_colunas];
```

Por exemplo, para declarar uma matriz 2D de inteiros com 3 linhas e 4 colunas:

```
int minhaMatriz[3][4];
```

Loops

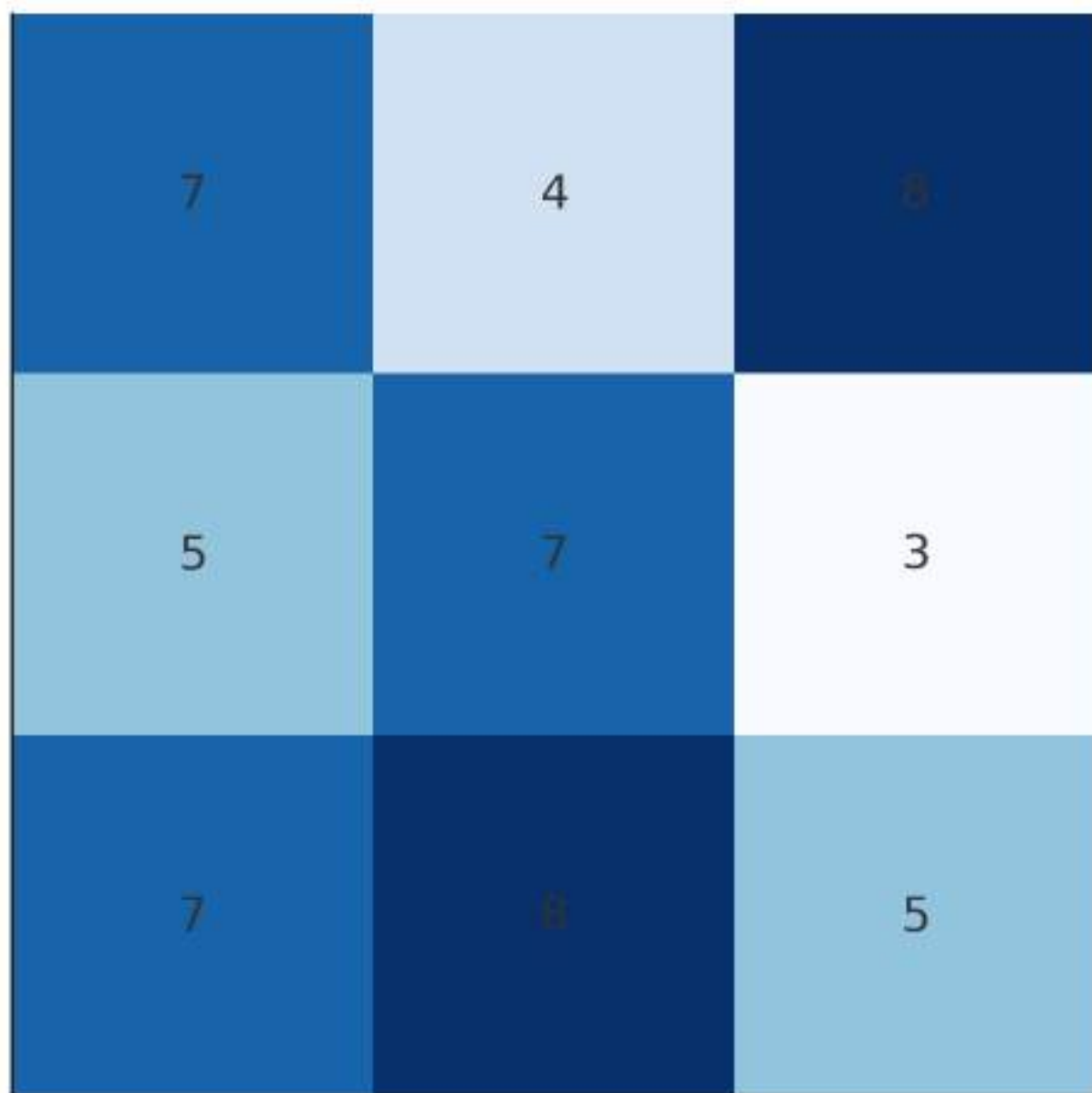
A maneira mais comum de trabalhar com matrizes 2D é usando loops aninhados – um loop para as linhas e outro para as colunas.

for aninhado

```
for (int i = 0; i < num_linhas; i++) {  
    for (int j = 0; j < num_colunas; j++) {  
        // Faça algo com minhaMatriz[i][j]  
    }  
}
```

Matriz Quadrada 3x3 com valores aleatórios

Representação Gráfica da Matriz 3x3



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

int main() {
    int matriz[3][3];
    int i, j;

    // Inicializa a semente do gerador de números aleatórios
    srand(time(NULL));

    // Preenche a matriz com valores aleatórios entre 1 e 9
    for (i = 0; i < 3; i++) {
        for (j = 0; j < 3; j++) {
            matriz[i][j] = (rand() % 9) + 1;
        }
    }


    // Imprime a matriz
    printf("Matriz 3x3 com valores aleatórios:\n");
    for (i = 0; i < 3; i++) {
        for (j = 0; j < 3; j++) {
            printf("%d ", matriz[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

...the ground or stays
iverse is vast, and you
s also beautiful. You a
omething bigger than yc
t of something that ma
most of your time. Tak
e a blog post. Make a

usando Strings





Embora a linguagem C apresente certas restrições quanto à manipulação direta de vetores e strings — como a incapacidade de atribuir diretamente uma string a uma variável ou de concatená-la com outra — ela compensa com uma rica biblioteca de funções dedicadas. Estas funções facilitam a realização de quase todas as operações necessárias com strings. Exploraremos essas funcionalidades neste material.

Caracteres / Strings em C

Aspecto	Caractere (char)	String
Definição	Um único símbolo alfabético, numérico ou especial	Sequência de caracteres terminada por um caractere nulo (' \ 0')
Tipo de Dado	char	Array de char
Tamanho	1 byte (típico)	Número de caracteres + 1 (para o ' \ \ 0')
Representação	Aspas simples (')	Aspas duplas (")
Exemplo de Declaração	char letra = 'B';	char nome[6] = "Hello"; (5 caracteres + 1 para o ' \ \ 0')

String – Problema do Tamanho

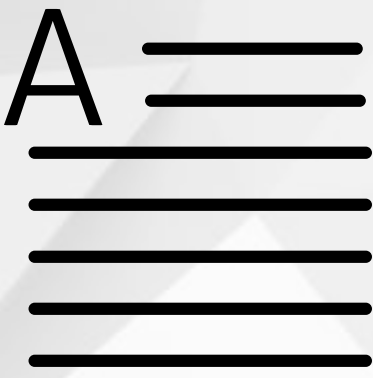
Suponhamos que você declare um vetor **v** com **100 posições** para armazenar um nome, mas só insira o nome **"ZÉ"**. Isso levanta uma questão válida: **como saber quais caracteres estão efetivamente sendo usados e quantos deles há?**

A resposta está na convenção padrão de C para strings: **o terminador de string, representado pelo caractere nulo ('**\0**').**

Quando você armazena a string "ZÉ" em um vetor, na realidade, o que é armazenado se parece com isso: **'Z', 'É', '\0'**. Esse caractere nulo serve como um sinalizador para funções de manipulação de strings (como `strlen()`, `strcpy()`, etc.) para identificar onde a string termina.

'Z' **'É'** **'\0'**

Delimitador '\0**'**



Carga inicial de String

A declaração e a carga inicial segue a sintaxe normal de vetores.

Exemplos:

```
char nome[20] = "André";  
char nome[20] = {'A', 'n', 'd', 'r', 'é'};  
char nome[] = "André";    /* Equivalente a char nome[5+1] = "André"; */  
char *nome = "André";    /* idem */
```




printf e puts (put string).



scanf e gets (get string).

Leitura e escrita **Strings**

Função printf:

Uso: Utilizada para imprimir dados formatados na saída padrão (geralmente a tela).

Sintaxe:

```
printf(format_string, argumentos...);
```

Exemplos:

```
char nome[] = "João";  
printf("Olá, %s!\n", nome); // Olá, João!
```

Função puts:

Uso: Uma função simples para imprimir uma string na saída padrão, seguida por uma quebra de linha.

Sintaxe:

```
puts(string);
```

Exemplos:

```
char saudacao[] = "Olá, Mundo!";  
puts(saudacao); // Olá, Mundo!
```

Função scanf:

Uso: Utilizada para ler dados formatados da entrada padrão (geralmente do teclado).

Sintaxe:

```
scanf(format_string, &variavel1, &variavel2, ...);
```

Exemplos:

```
char nome[50];  
printf("Digite seu nome: ");  
scanf("%s", nome); // Lê uma palavra da entrada padrão e armazena em 'nome'.
```


Função gets:

Nota Importante: A função gets é **considerada insegura e foi oficialmente removida do padrão C em 2011 (C11)**. Ela pode causar desbordamento de buffer, pois não verifica o tamanho do buffer de entrada. É aconselhável usar fgets em vez de gets.

Uso: Era usada para ler uma linha de texto da entrada padrão até encontrar uma quebra de linha.

Sintaxe:

```
gets(string);
```

Exemplos:

```
char frase[100];  
gets(frase); // Lê uma linha da entrada padrão e armazena em 'frase'.
```

Recomendação: Em vez de usar gets, é preferível usar fgets da seguinte forma:

```
char frase[100];  
fgets(frase, sizeof(frase), stdin);
```

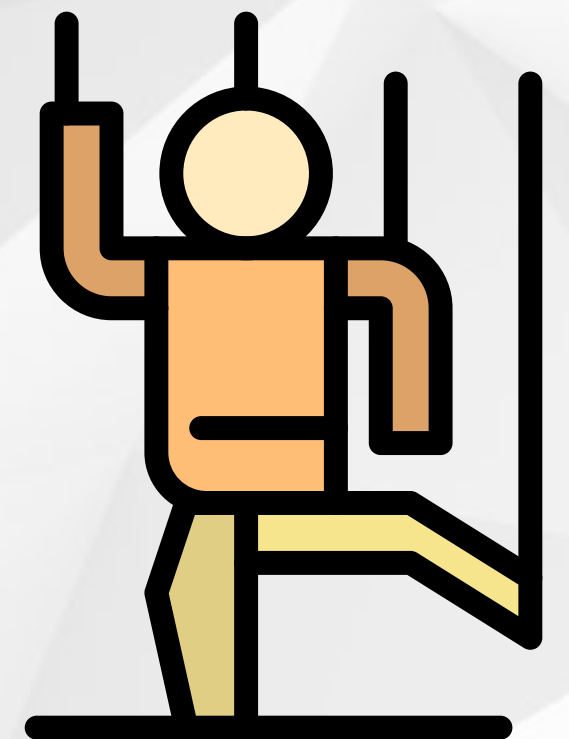
Ao usar fgets, você especifica o tamanho máximo de caracteres a serem lidos (incluindo o caractere nulo), o que ajuda a prevenir desbordamentos de buffer.



funções que
manipulam
Strings

Manipulação de Strings

A manipulação de strings é uma tarefa fundamental em programação. Em muitos programas, **grande parte da entrada e saída envolve o processamento de texto**. Desde a análise de arquivos de texto até a manipulação de entradas do usuário ou comunicação através de redes, a capacidade de trabalhar com strings de forma eficaz e segura é crucial.



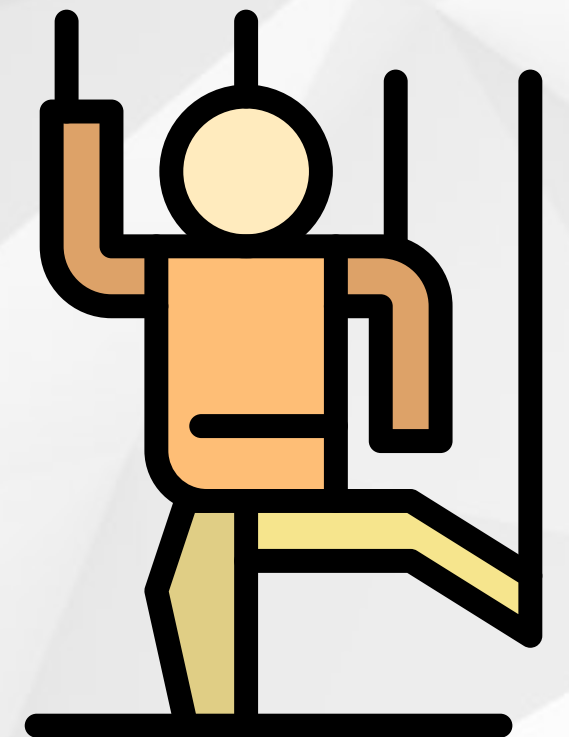
Importância das funções de manipulação de strings



1. **Segurança:** Operações diretas em **strings** podem levar a vulnerabilidades, como desbordamentos de buffer. Funções bem projetadas ajudam a mitigar esses riscos.
2. **Eficiência:** As funções padrão são geralmente otimizadas para serem rápidas e eficientes.
3. **Consistência:** Usar funções padrão garante um comportamento consistente em diferentes plataformas e compiladores.
4. **Facilidade de Uso:** Evita a reescrita de código comum, tornando o desenvolvimento mais rápido e menos propenso a erros.

Manipulação Básica

- **strcpy()**: Copia uma string para outra.
- **strncpy()**: Copia um número específico de caracteres de uma string para outra.
- **strcat()**: Concatena (anexa) uma string ao final de outra.
- **strncat()**: Concatena um número específico de caracteres de uma string ao final de outra.
- **strlen()**: Retorna o comprimento de uma string.



Manipulação Básica

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

✓ int main() {
    char origem1[] = "Olá";
    char origem2[] = "Mundo";
    char destino1[50];
    char destino2[50] = "Bom Dia, ";
    // Usando strcpy() para copiar a string origem1 para destino1
    strcpy(Dest: destino1, Source: origem1);
    printf(format: "destino1 após strcpy: %s\n", destino1);
    // Usando strncpy() para copiar os primeiros 3 caracteres de origem2 para destino1
    strncpy(Dest: destino1, Source: origem2, Count: 3);
    destino1[3] = '\0'; // Adicionando um caractere nulo após os 3 caracteres copiados
    printf(format: "destino1 após strncpy (3 caracteres): %s\n", destino1);
    // Usando strcat() para concatenar origem2 ao final de destino1
    strcat(Dest: destino1, Source: origem2);
    printf(format: "destino1 após strcat: %s\n", destino1);
    // Usando strncat() para concatenar os primeiros 3 caracteres de origem1 ao final de destino2
    strncat(Dest: destino2, Source: origem1, Count: 3);
    printf(format: "destino2 após strncat (3 caracteres): %s\n", destino2);
    // Usando strlen() para obter o comprimento de destino2
    printf(format: "Comprimento de destino2: %zu\n", strlen(Str: destino2));
    return 0;
}
```

Comparação

- **strcmp()**: Compara duas strings.
- **strncmp()**: Compara um número específico de caracteres de duas strings.
- **strcasecmp()**: Compara duas strings, ignorando a diferença entre maiúsculas e minúsculas (note que esta função não é padrão ANSI C, mas é amplamente suportada).

Comparação

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

int main() {
    char string1[] = "Olá";
    char string2[] = "olá";
    char string3[] = "Mundo";
    // Usando strcmp() para comparar string1 e string2
    if (strcmp(string1, string2) == 0) {
        printf(format: "string1 e string2 são iguais.\n");
    } else {
        printf(format: "string1 e string2 são diferentes.\n");
    }
    // Usando strncmp() para comparar os primeiros 2 caracteres de string1 e string2
    if (strncmp(string1, string2, 2) == 0) {
        printf(format: "Os primeiros 2 caracteres de string1 e string2 são iguais.\n");
    } else {
        printf(format: "Os primeiros 2 caracteres de string1 e string2 são diferentes.\n");
    }
    // Usando strcasecmp() para comparar string1 e string2 ignorando maiúsculas/minúsculas
    if (strcasecmp(string1, string2) == 0) {
        printf(format: "string1 e string2 são iguais (ignorando maiúsculas/minúsculas).\n");
    } else {
        printf(format: "string1 e string2 são diferentes (mesmo ignorando maiúsculas/minúsculas).\n");
    }
    return 0;
}
```


Conversão e Formatação

- **sprintf()**: Formata e armazena uma string.
- **sscanf()**: Lê dados formatados de uma string.
- **atoi()**: Converte uma string para um inteiro.
- **atol()**: Converte uma string para um long int.
- **atof()**: Converte uma string para um número de ponto flutuante

Conversão e formatação

```
3 int main() {
4     char buffer[100];
5     char entrada[] = "123 456.789";
6     int valorInt;
7     long int valorLongInt;
8     double valorDouble;
9     // Usando sprintf() para formatar e armazenar uma string
10    int idade = 30;
11    sprintf(buffer, "Eu tenho %d anos.", idade);
12    printf("%s\n", buffer); // Imprime: Eu tenho 30 anos.
13    // Usando sscanf() para ler dados formatados de uma string
14    int num1;
15    double num2;
16    sscanf(entrada, "%d %lf", &num1, &num2);
17    printf("num1: %d, num2: %lf\n", num1, num2); // Imprime: num1: 123, num2: 456.789000
18    // Usando atoi() para converter uma string para um inteiro
19    char strInt[] = "42";
20    valorInt = atoi(strInt);
21    printf("Valor inteiro: %d\n", valorInt); // Imprime: Valor inteiro: 42
22    // Usando atol() para converter uma string para um long int
23    char strLongInt[] = "1234567890";
24    valorLongInt = atol(strLongInt);
25    printf("Valor long int: %ld\n", valorLongInt); // Imprime: Valor long int: 1234567890
26    // Usando atof() para converter uma string para um número de ponto flutuante
27    char strDouble[] = "3.14159";
28    valorDouble = atof(strDouble);
29    printf("Valor double: %lf\n", valorDouble); // Imprime: Valor double: 3.141590
30    return 0;
31 }
```



Exercícios - Aula 05

Vetor

1. Soma dos Elementos:

Escreva um programa que leia um vetor de 10 números inteiros e calcule e imprima a soma de todos os seus elementos.

2. Maior e Menor Valor:

Desenvolva um programa que leia um vetor de 8 posições e, em seguida, encontre o maior valor e a sua posição no vetor. Faça o mesmo para o menor valor.

3. Inversão de Vetor:

Crie um programa que leia um vetor de 15 números inteiros e o apresente invertido. Por exemplo, se o vetor de entrada for $[1, 2, 3, \dots, 15]$, a saída deve ser $[15, 14, 13, \dots, 1]$.

4. Contagem de Pares e Ímpares:

Desenvolva um programa que leia um vetor com 20 números. Em seguida, determine e imprima quantos valores pares e ímpares ele possui.

5. Vetor de Números Negativos:

Escreva um programa que leia um vetor de 12 posições. Substitua todas as posições que contêm um valor negativo por zero. Ao final, imprima o vetor modificado.



Exercícios - Aula 05

Matriz 2D

1. **Soma de Matrizes:** Escreva um programa que leia duas matrizes 3×3 e calcule a soma dessas matrizes. O programa deve imprimir a matriz resultante.
2. **Diagonal Principal:** Desenvolva um programa que leia uma matriz 4×4 e imprima a sua diagonal principal.
3. **Transposta de uma Matriz:** Crie um programa que leia uma matriz 3×3 e imprima a sua matriz transposta (a matriz transposta é obtida trocando linhas por colunas).
4. **Maior Valor de uma Matriz:** Elabore um programa que leia uma matriz 5×5 . O programa deve encontrar e imprimir o maior valor da matriz e sua respectiva posição (linha e coluna).
5. **Multiplicação de Matriz por um Escalar:** Escreva um programa que leia uma matriz 2×2 e um número inteiro (escalar). O programa deve multiplicar cada elemento da matriz pelo escalar e imprimir a matriz resultante.



Exercícios - Aula 05

Funções manipulação de Strings

1. **Concatenação de Strings:** Escreva um programa que leia duas strings do usuário e, usando a função `strcat()`, concatene a segunda string ao final da primeira. Imprima a string resultante.
2. **Comparação de Strings:** Desenvolva um programa que leia duas strings e use a função `strcmp()` para compará-las. Se as strings forem iguais, imprima "As strings são iguais", caso contrário, imprima "As strings são diferentes".
3. **Cópia com Limite de Caracteres:** Crie um programa que leia duas strings. A primeira é uma frase e a segunda é um número ***n***. Use a função **`strncpy()`** para copiar os primeiros ***n*** caracteres da frase para uma terceira string e imprima o resultado.



Exercícios - Aula 05

Funções manipulação de Strings

4. Conversão de String para Número: Elabore um programa que leia uma string representando um número decimal e um número de ponto flutuante. Use as funções `atoi()` e `atof()` para converter as strings para seus respectivos tipos numéricos e, em seguida, imprima a soma dos dois números.

5. Formatando e Lendo Strings: Escreva um programa que leia do usuário seu nome, idade e altura. Use a função `sprintf()` para formatar essas informações em uma única string no formato: "Nome: [nome], Idade: [idade], Altura: [altura]". Em seguida, usando `sscanf()`, extraia essas informações da string formatada e as imprima separadamente.

Referências

.....

DAMAS, L. M. D. Linguagem C. LTC, 2007.

.....

HERBERT, S. C completo e total. 3a. ed. Pearson, 1997.

.....

