

Conteúdo

Revisão: revisão de aula 5 (arrays, strings, matrizes)

Structures: juntar variaveis numa novo tipo

Teste TecWeb: enunciado TecWeb, deadline 24/3/2025@9hrs

Test Aula 5: 5 perguntas escolha multipla, 5 "programas" de ~5 linhas

Vectores (Arrays)

Um vector é um conjunto de elementos do mesmo tipo.

Ocupam posições contíguas na memória.

Índices começam em 0!

```
char socos[5] = {100, 120, 90, 130, 110};  
printf("Soco mais forte: %d\n", socos[3]); // 130
```

A tabela exemplifica o modelo de memória.
Os Endereços são definidos no momento em que o programa corre.

Modelo da memória RAM:

Endereço	Conteudo	Identificador
1023	100	socos[0]
1024	120	socos[1]
1025	90	socos[2]
1026	130	socos[3]
1027	110	socos[4]
	1023	socos
...		

A variável `socos` contém o endereço de memória do elemento `socos[0]` (primeiro elemento do vector)

Vetores como Parâmetros de Função

Em C, os vetores são passados por referência para funções

Isso significa que qualquer alteração dentro da função afeta o vetor original.

Exemplo: Modificando um vetor dentro de uma função

```
#include <stdio.h>

void modificar(int v[], int tamanho) {
    for (int i = 0; i < tamanho; i++) {
        v[i] *= 2; // Dobra cada elemento
    }
}
```

```
int main() {
    int valores[] = {1, 2, 3, 4, 5};
    int n = 5;

    modificar(valores, n);

    for (int i = 0; i < n; i++) {
        printf("%d ", valores[i]); // Imprime: 2 4 6 8 10
    }
    return 0;
}
```

✓ O vetor **valores** foi modificado dentro da função **modificar()** !

Cadeias de Caracteres (Strings) em C

Strings em C são **vetores de caracteres** terminados pelo caractere especial `\0` (caractere nulo).

Devemos sempre reservar espaço para o caractere `\0` ao declarar uma string.

```
char nome[10]; // Permite até 9 caracteres + '\0'
```

Declaração e Inicialização de Strings

```
char nome[20] = "oscar";
```

```
char nome[20] = {'o','s','c','a','r', '\0'};
```

```
char nome[] = "oscar";
```

// 0 compilador define o tamanho automaticamente. Incluindo espaço para o \0

```
char *nome = "oscar";
```

⚠ O `\0` deve ser sempre considerado, pois indica o fim da string.

Comprimento de uma String

```
int n = strlen(nome);  
printf("A string tem %d caracteres", n);
```

⚠ `strlen()` retorna apenas o número de caracteres **antes** do `\0`.

`printf()` e `scanf()` com Strings

```
printf("%s\n", var); // Imprime a string normalmente
printf("%10s\n", var); // Alinha à direita com espaço mínimo de 10 caracteres
printf("%-10s\n", var); // Alinha à esquerda

puts(var); // Similar ao printf("%s\n", var), sempre adiciona o '/n'

scanf("%s", nome); // Lê até encontrar espaço ou \n
scanf("%5s", nome); // Lê até 5 caracteres

scanf("%[^\\n]s", nome); // Lê até \n

fgets(nome, 128, stdin); // Alternativa segura
```

Matrizes: Vetores de Vetores

Uma **matriz** é um **vetor multidimensional**, i.e. **vector de vetores**

São declaradas com **duas dimensões ou mais**.

Em C, declaramos assim:

```
int socos[2][3] = {  
    {100, 120, 110}, // Linha 0  
    {90, 130, 105}  // Linha 1  
};
```

 **Dica:** A primeira dimensão é **linhas**, a segunda é **colunas**.

Estrutura de Matrizes

Primeira dimensão → Número de linhas

Segunda dimensão → Número de colunas

```
tipo nome_matriz[num_linhas][num_colunas];
```

✓ Exemplo:

```
char Galo[3][3]; // Matriz 3x3
Galo[0][0] = 'X';
Galo[0][2] = '0';
Galo[1][1] = 'X';
Galo[2][2] = '0';
```

✓ Galo[2][2] armazena '0'.

Percorrer Matrizes

 **Dica:** percorre linha a linha!

```
for (int i = 0; i < 2; i++) {  
    for (int j = 0; j < 3; j++) {  
        printf("Soco[%d][%d]: %d\n", i, j, socos[i][j]);  
    }  
}
```

Inicialização Automática

Podemos inicializar uma matriz no momento da declaração:

```
char soup[5][5] = {  
    {'f', 'e', 'k', 'u', 'l'},  
    {'u', 'o', 'x', 's', 'n'},  
    {'t', 'n', 'r', 'e', 'r'},  
    {'y', 'h', 'e', 'c', 'j'},  
    {'v', 'q', 'e', 'w', 'e'}  
};
```

✅ Podemos omitir o número de linhas (o compilador infere):

```
char soup[][5] = {  
    {'e', 'e', 'k', 'u', 'l'},  
    {'u', 'c', 'x', 'q', 'n'},  
    {'t', 's', 'r', 'd', 'r'},  
    {'y', 'h', 'e', 'o', 'j'},  
    {'v', 'q', 'e', 'w', 'f'}  
};
```

⚠ Não podemos omitir o número de colunas!

Acesso a Elementos

Podemos **acessar e modificar** elementos da matriz:

```
char soup[5][5];  
soup[0][0] = 'e';  
soup[0][1] = 'e';  
soup[0][2] = 'u';  
soup[0][3] = 'l';  
soup[1][0] = 'u';
```

✓ Cada elemento é referenciado como `matriz[linha][coluna]`.

Estruturas (**struct**) em C

As **structs** em C permitem agrupar diferentes tipos de dados numa única estrutura lógica.

```
#include <stdio.h>

struct Pessoa {
    char nome[50];
    int idade;
    float altura;
};
```

Aqui criamos uma **struct** chamada **Pessoa** que contém três membros: uma string, um inteiro e um **float**.

Utilização de `structs`

```
struct Pessoa pessoa1;  
pessoa1.idade = 25;  
pessoa1.altura = 1.75;  
strcpy(pessoa1.nome, "João");
```

Podemos declarar variáveis do tipo `struct Pessoa` e atribuir valores aos seus membros.

Uso com `typedef`

```
typedef struct Pessoa{  
    char nome[50];  
    int idade;  
} pessoa_st;  
  
pessoa_st pessoa2;
```

O `typedef` simplifica a sintaxe, permitindo criar `structs` sem necessidade de `struct` antes do nome.

Array de **structs**

```
pessoa_st pessoas[3] = {  
    {"Ana", 30},  
    {"Bruno", 25},  
    {"Carla", 28}  
};
```

Podemos criar um array de **structs** para armazenar múltiplas pessoas.

Encontrar uma pessoa pelo nome

```
#include <string.h>

int encontrar_pessoa(pessoa_t pessoas[], int len, char *nome) {
    for (int i = 0; i < len; i++) {
        if (strcmp(pessoas[i].nome, nome) == 0) {
            return i;
        }
    }
    return -1;
}
```

Esta função percorre o array de `structs` e retorna a pessoa correspondente ao nome buscado.

Exemplo

Uma maneira para retornar seguramente a pessoa é de retornar o índice no array, em vez da estrutura.

```
int idx = encontrar_pessoa(pessoas, sizeof(pessoas)/sizeof(pessoa_st), "Bruno");  
  
if (idx < 0) {  
    printf("Pessoa não encontrada\n");  
} else {  
    printf("A pessoa %s tem %d anos\n", pessoas[idx].nome, pessoas[idx].idade)  
}
```

Retornar structs é desaconselhado

```
peessoa_t encontrar_pessoa(peessoa_t pessoas[], int len, char *nome) {  
    for (int i = 0; i < len; i++) {  
        if (strcmp(pessoas[i].nome, nome) == 0) {  
            return pessoas[i];  
        }  
    }  
    peessoa_t vazia = {"", 0};  
    return vazia;  
}
```

Se o struct não tem ponteiros ou locações dinâmicas, funciona corretamente, pois os dados são copiados para o local de retorno da função.

nome: a referencia para o nome é copiado (ou seja, aponta para o mesmo string)

idade: uma copia do valor de idade


? Q&A


💬 Dúvidas?

Teste TecWeb

O teste **TecWeb** é para fazer na semana da tecweb. O teste é para fazer em casa e conta para nota: <https://moodle.deisi.ulusofona.pt/>

▼ Testes Aula 5

 **TESTE**
Teste Aula 5 Marcar como concluída

 **TRABALHO**
Teste TecWeb (semana TecWeb) Marcar como concluída

Abre: quinta-feira, 13 de março de 2025 às 10:00
Data limite: segunda-feira, 24 de março de 2025 às 09:00


O enunciado deste trabalho esta aqui e tem 4 partes. Tem de fazer upload dos 4 ficheiros com os nomes main1.c, main2.c, main3.c e main4.c


Podem trabalhar nesta tarefa depois de acabar a [Teste Aula 5](#) e tem de entregar aqui no moodle até dia 24/3/2025 @ 9hrs.

Teste Aula 5

Link: <https://moodle.deisi.ulusofona.pt/>

▼ Testes Aula 5

 **TESTE**
Teste Aula 5 Marcar como concluída

 **TRABALHO**
Teste TecWeb (semana TecWeb) Marcar como concluída

Abre: quinta-feira, 13 de março de 2025 às 10:00
Data limite: segunda-feira, 24 de março de 2025 às 09:00

O enunciado deste trabalho esta aqui e tem 4 partes. Tem de fazer upload dos 4 ficheiros com os nomes main1.c, main2.c, main3.c e main4.c

Podem trabalhar nesta tarefa depois de acabar a [Teste Aula 5](#) e tem de entregar aqui no moodle até dia 24/3/2025 @ 9hrs.

Depois de acabar, podem começar com o teste tecweb