

# Aula 15: Variáveis compostas - Strings ou array de caracteres\*

---

Delano Beder

\*Adaptado do livro de Andre Backes

# Strings

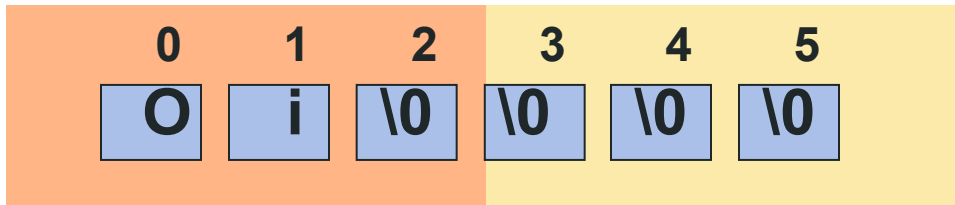
# String - Definição

- Sequência de caracteres adjacentes na memória.
- Essa sequência de caracteres, que pode ser uma palavra ou frase
- Strings são arrays do tipo **char**.
- Ex:
  - **char str[6];**

# String - Definição

- As strings têm no elemento seguinte à última letra da palavra/frase armazenado um caractere `'\0'` (barra invertida + zero).
  - O caracter `'\0'` indica o fim da sequência de caracteres.
- Exemplo
  - `char str[6] = "Oi";`

Região inicializada:  
2 letras + 1  
caractere  
terminador `'\0'`



# String - Definição

- **Importante**

- Ao definir o tamanho de uma string, devemos considerar o caractere **'\0'**.
- Isso significa que a string **str** comporta uma palavra de no máximo 5 caracteres.

- Exemplo:

- `char str[6] = "Teste";`

T	e	s	t	e	\0
---	---	---	---	---	----

# Definição

- Por se tratar de um array, cada caractere pode ser acessado individualmente por meio de um índice
- Exemplo
  - `char str[6] = "Teste";`

T	e	s	t	e	\0
---	---	---	---	---	----

- `str[0] = 'L';`

L	e	s	t	e	\0
---	---	---	---	---	----

# Definição

- IMPORTANTE:

- Na inicialização de palavras, usa-se “**aspas duplas**”.

- Ex: `char str[6] = "Teste";`

T	e	s	t	e	\0
---	---	---	---	---	----

- Na atribuição de um caractere, usa-se ‘**aspas simples**’

- `str[0] = 'L';`

L	e	s	t	e	\0
---	---	---	---	---	----

# Definição

- **Importante:**

- “A” é diferente de ‘A’

- “A” (string)

A \0

- ‘A’ (char)

A



# Manipulação de strings

# Manipulando strings - Leitura

□ **scanf:** lê uma string do teclado e armazena em uma string.

● Exemplo:

```
char s1[5];  
int i;  
scanf("%s",s1);
```

## Problemas:

**Não permite a entrada de strings com espaço em branco.**

**Não é um método seguro, pois não limita o número de caracteres lidos.**

# Manipulando strings - Leitura

□ **gets(str):** lê uma string do teclado e armazena em uma string.

● Exemplo:

```
char str[10];  
gets(str);
```

**Problema::** não é um método seguro, pois não limita o número de caracteres lidos.

**A versão mais recente da especificação da linguagem C (2011) removeu definitivamente esta função de sua especificação.**

# Manipulando strings - Leitura

- Função recomendada para a leitura de strings: **fgets()**,
  - permite a leitura de strings do teclado ou de arquivo
  - A função **fgets** recebe 3 argumentos
    - a string a ser lida, **str**;
    - o limite máximo de caracteres a serem lidos, **tamanho**;
    - A variável FILE **\*fp**, que está associado ao arquivo de onde a string será lida (ou **stdin** para leitura do teclado)
  - E retorna
    - NULL em caso de erro ou fim do arquivo;
    - O ponteiro para o primeiro caractere recuperado em **str**.

# Exemplo

- Função fgets com leitura de entrada pelo teclado.

```
int main(){  
    char nome[30];  
    printf("Digite um nome: ");  
    fgets(nome, 30, stdin);  
    printf("O nome digitado foi: %s", nome);  
  
    return 0;  
}
```

# Manipulando strings - Leitura

- Funcionamento da função **fgets**
  - A função **lê a string até que um caractere de nova linha** seja lido **ou *tamanho-1* caracteres** tenham sido lidos.
  - **A string resultante sempre terminará com '\0'** (por isto somente *tamanho - 1* caracteres, no máximo, serão lidos).
  - **Se o caractere de nova linha ('\n') for lido, ele fará parte da string.**
  - Se ocorrer algum erro, a função devolve um ponteiro nulo (**NULL**) em **str**.

# Manipulando strings - Leitura

- A função **fgets** possui as seguintes vantagens:
  - pode fazer a leitura a partir de um arquivo de dados e incluir o caractere de nova linha “\n” na string;
  - especifica o tamanho máximo da string de entrada. **Evita estouro no buffer.**

# Manipulando strings - Escrita

- Para se escrever uma string na tela utilizamos a função **printf()**.

```
printf("%s", str);
```

- A função **puts** imprime uma string na saída padrão, incluindo uma nova linha ('\n') ao final da impressão.

```
puts(str);
```

- Outra função que permite a escrita de strings é a **fputs()**

```
fputs(str, stdout);
```



# Manipulando strings

- Strings são arrays. Portanto, **não** podemos atribuir uma string a outra!

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

```
int main() {  
    char str1[20]="CAP 2019 - Turma D";  
    char str2[20];  
    str2 = str1;  
    return 0;  
}
```

**X**

# Copiando uma string

- O correto é copiar a string elemento por elemento.

```
#include <stdio.h>

int main() {
    int i;
    char str1[20]="CAP 2019 - Turma D";
    char str2[20];
    for (i=0;i<20;i++){
        str2[i]=str1[i];
    }
    printf("%s\n",str1);
    printf("%s\n",str2);
    return 0;
}
```

# Manipulando strings

- A biblioteca padrão C possui funções especialmente desenvolvidas para esse tipo de tarefa
  - `#include <string.h>`

# Manipulando strings - Tamanho

- `strlen(str)`: retorna o tamanho da string `str`. Ex:

```
char str[15] = "teste";  
printf("%d", strlen(str));
```

- Neste caso, a função retornará 5, que é o número de caracteres na palavra “teste” e não 15, que é o tamanho do array.
  - O ‘\0’ também não é considerado pela `strlen`, mas vale lembrar que ele está escrito na posição `str[5]` do vetor.

# Manipulando strings - Cópia

- **strcpy(dest, fonte)**: copia a string contida na variável **fonte** para **dest**.

- Exemplo

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

int main() {
    char s1[101], s2[101];
    scanf("%s", s1);

    strcpy(s2, s1);
    printf("%s\n", s2);

    return 0;
}
```

# Manipulando strings - Concatenação

- **strcat(dest, fonte)**: concatena duas strings.
- Neste caso, a string contida em **fonte** permanecerá inalterada e será anexada ao final da string de **dest**.
- Exemplo

```
char str1[15] = "bom ";  
char str2[15] = "dia";  
strcat(str1, str2);  
printf("%s", str1);
```

# Manipulando strings - Comparação

- **strcmp(str1, str2):** compara duas strings e
  - retorna VALOR NEGATIVO se **str1** “menor”\* que **str2**
  - retorna ZERO se as strings forem iguais.
  - retorna VALOR POSITIVO se **str1** “maior”\* que **str2**

\*ordem lexicográfica

- Exemplo

```
if(strcmp(str1, str2) == 0)
    printf("Strings iguais");
else
    printf("Strings diferentes");
```

# Leitura - término com EOF

A entrada termina com fim-de-arquivo (EOF).

```
int main() {  
  
    char nome[11];  
  
    while (scanf("%s", nome) != EOF) {  
        printf("String lida = %s\n", nome);  
    }  
  
    return 0;  
}
```

```
int main() {  
  
    char nome[11];  
  
    while (fgets(nome, 11, stdin)) {  
        printf("String lida = %s", nome);  
    }  
  
    return 0;  
}
```



# Observação final

- Ao inicializar uma string em sua declaração as regiões do vetor que não foram utilizadas pela string são preenchidas com zeros ('\0')
- Entretanto, esse comportamento não ocorre com o **strcpy**, **scanf** e **fgets**. Nessas funções as posições não usadas não são alteradas (ficam **lixos**).
  - Ex: `char str[6] = "Oi";`

O	i	\0	\0	\0	\0
---	---	----	----	----	----

# Observação final

- Exemplos

- `char str[6] = "Teste";`

T	e	s	t	e	\0
---	---	---	---	---	----

- `scanf("%s", str); //digite "Ola" no prompt`

O	I	a	\0	e	\0
---	---	---	----	---	----

- `strcpy(str, "Oi");`

O	i	\0	\0	e	\0
---	---	----	----	---	----

# Referências

- BACKES, André. **Linguagem C: completa e descomplicada**. Rio de Janeiro: Elsevier, 2013. 371 p. ISBN 978-85-352-6855-3. Disponível na Biblioteca.