



UNIVERSIDADE  
FEDERAL DO CEARÁ



## **CK0179 – Programação Computacional para Engenharia:** String

Prof. Maurício Moreira Neto

# Objetivo

- Aprender a utilizar Strings na linguagem C
- Aprender as diversas funções associadas a strings

# Definição

- O que é String?
  - Sequência de caracteres adjacentes na memória
  - Essa sequência de caracteres, que pode ser uma palavra ou frase
  - Em outras palavras, strings são arrays do tipo **char**
- **Ex:**
  - `char str[10];`

# Definição

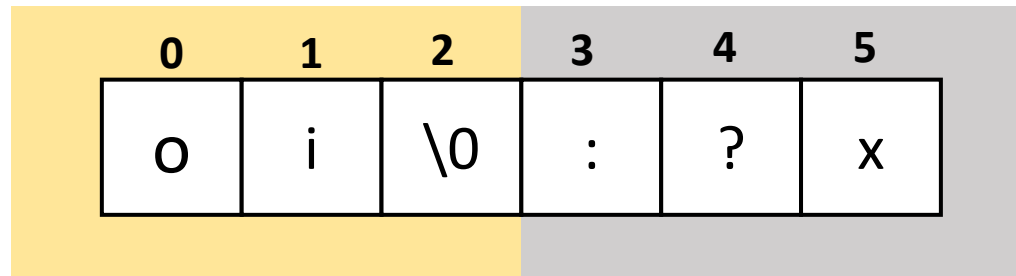
- String

- Devemos ficar atentos para o fato de que as strings tem no elemento seguinte a última letra da palavra armazenada um caractere `'\0'`
- O caractere `'\0'` indica o fim da sequência de caracteres

- Exemplo:

- `char str[10] = "oi";`

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



Lixo de memória  
(região não  
inicializada)

# 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 9 caracteres!

- **Exemplo:**

- `Char str[10] = "perspicaz"`

p	e	r	s	p	i	c	a	z	\0
---	---	---	---	---	---	---	---	---	----

# Definição

- Como uma string é um array, logo, cada caractere pode ser acessada individualmente por meio de um índice

- Exemplo:

- `char str[10] = "perspicaz"`

p	e	r	s	p	i	c	a	z	\0
---	---	---	---	---	---	---	---	---	----

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

R	e	r	s	p	i	c	a	z	\0
---	---	---	---	---	---	---	---	---	----

# Definição

- Importante
  - Na inicialização de palavras, usa-se **“aspas duplas”**
  - Ex: `char str[10] = “perspicaz”`

p	e	r	s	p	i	c	a	z	\0
---	---	---	---	---	---	---	---	---	----

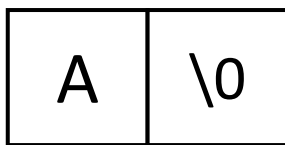
- Na atribuição de um caractere, usa-se **‘aspas simples’**
- `str[0] = ‘R’`

R	e	r	s	p	i	c	a	z	\0
---	---	---	---	---	---	---	---	---	----

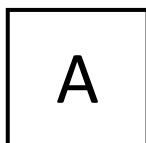
# Definição

- **Importante**

- “A” é diferente de ‘A’
  - “A”



- ‘A’





# Definição

- Observações sobre a memória

```
char c;  
c = 'h';
```

```
int a;  
a = 19;
```


```
char Sigla[4];  
Sigla[0] = 'U';  
Sigla[1] = 'F';  
Sigla[2] = 'U';  
Sigla[3] = '\0';
```

Endereço	Blocos	Variável	tipo
1			
2			
3	'H'	c	char
4			
5			
6			
7	'U'	Sigla[0]	char[4]
8	'F'	Sigla[1]	
9	'U'	Sigla[2]	
10	'\0'	Sigla[3]	
11			
12	19	a	int
13			
14			
	....		

# Manipulando as Strings

- Strings são arrays, conseqüentemente, **não** se pode atribuir uma string para outra!

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    char str1[20] = "Hello World";
    char str2[20];

     str1 = str2;

    system("pause");
    return 0;
}
```

- O correto é copiar a string elemento por elemento!

# Manipulando as Strings

- O correto é copiar a string elemento por elemento:

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    int i;
    char str1[20] = "Hello World";
    char str2[20];

    for(i = 0; str1[i] != '\0'; i++)
        str2[i] = str1[i];
    str2[i] = '\0';

    system("pause");
    return 0;
}
```

# Manipulação de Strings

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

# Manipulação de Strings - Leitura

- Exemplo de algumas funções para manipulação de strings:
  - **gets(str)**: esta função lê uma string do teclado e armazena em **str**
  - **Exemplo:**

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

# Manipulação de Strings – Limpando o buffer

- Podem ocorrer erros durante a leitura de caracteres ou strings
- Neste caso, para resolver esses problemas, podemos limpar o buffer do teclado

```
char str[10];
```

```
setbuf(stdin, NULL); //limpa o buffer
```

```
gets(str);
```

# Manipulação de Strings - Escrita

- Quando se quer escrever uma string na tela usamos a função **printf()**
  - Especificador de formato: %s

```
char str[20] = "Hello World";  
printf("%s", str);
```

# Manipulação de Strings - Tamanho

- **strlen(str)**: retorna o tamanho da str.
- Exemplo:

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

- Neste caso, a função retornará 5, pois 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



# Manipulação de Strings - Copiar

- **strcpy(destino, fonte)**: copia a string contida na variável **fonte** para **destino**
- Exemplo:

```
char str1[100], str2[100];  
printf("Entre com uma string: ");  
gets(str1);  
strcpy(str2, str1);  
printf("%s", str2);
```

# Manipulação de Strings - Concatenar

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

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

# Manipulação de Strings – Compara

- **strcmp(str1, str2)**: compara duas strings. Neste caso, a função retorna ZERO se as strings forem iguais
- Exemplo:

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

# Manipulação de Strings

- Basicamente, para se ler uma string do teclado utilizamos a função **gets()**
- No entanto, existe outra função que, utilizada de forma adequada, também permite a leitura de strings do teclado. Essa função é a **fgets()**, cujo protótipo é:

```
char *fgets(char *str, int tamanho, FILE *fp);
```

# Manipulação de Strings

- 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
- E retorna
  - NULL em caso de erro ou fim do arquivo
  - O ponteiro para o primeiro caractere recuperado em **str**

```
char *fgets(char *str, int tamanho, FILE *fp);
```

# Manipulação de Strings

- Note que a função **fgets** utiliza uma variável FILE **\*fp**, que está associado ao arquivo de onde a string será lida
- Para ler do teclado, basta substituir FILE **\*fp** por **stdin**, o qual representa o dispositivo de entrada padrão (geralmente o teclado):

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

# Manipulação de Strings

- 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
  - Se o caractere de nova linha ('\n') for lido, ele fará parte da string, o que não acontecia com **gets**
  - A string resultante sempre terminará com '\0' (por isto somente *tamanho-1* caracteres, no máximo, serão lidos)
  - Se ocorrer algum erro, a função devolverá um ponteiro nulo (**NULL**) em **str**

# Manipulação de Strings

- A função **fgets** é semelhante a função **gets**, porém, com 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 do buffer



# Manipulação de Strings

- Basicamente, para se escrever uma string na tela utilizamos a função **printf()**

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

- No entanto, existe outra função que, utilizada de forma adequada, também permite a escrita de strings. Essa função é a **fputs()**, cujo protótipo é:

```
int fputs(char *str, FILE *fp);
```

# Manipulação de Strings

- A função `fputs()` recebe como parâmetro um array de caracteres e a variável `FILE *fp` representando o arquivo no qual queremos escrever
- Retorno da função
  - Se o texto for escrito com sucesso um valor inteiro diferente de zero é retornado
  - Se houver erro na escrita, o valor EOF (em geral, -1) é retornado

# Manipulação de Strings

- Note que a função **fputs** utiliza uma variável **FILE \*fp**, que está associado ao arquivo de onde a string será escrita
- Para escrever no monitor, basta substituir **FILE \*fp**, por **stdout**, o qual representa o dispositivo de saída padrão (geralmente a tela do monitor)

```
int main() {  
    char texto[30] = "Hello World\n";  
    fputs(texto, stdout);  
  
    return 0;  
}
```

# Observação Final

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

o	i	\0	\0	\0	\0
---	---	----	----	----	----

# Observação Final

- Exemplos:

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

o	i	\0	\0	\0	\0
---	---	----	----	----	----

- `gets(str); // digite "oi" no prompt`

o	i	\0	:	?	x
---	---	----	---	---	---

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

o	i	\0	X	?	@
---	---	----	---	---	---

# Dúvidas?



# Referências

- André Luiz Villar Forbellone, Henri Frederico Eberspächer, Lógica de programação (terceira edição), Pearson, 2005, ISBN 9788576050247.
- Ulysses de Oliveira, Programando em C - Volume I - Fundamentos, editora Ciência Moderna, 2008, ISBN 9788573936599.
- **Slides baseados no material do site "Linguagem C Descomplicado"**
  - <https://programacaodescomplicada.wordpress.com/complementar/>