



LINGUAGEM C: ARRAYS DE CARACTERES: STRINGS

Leandro Henrique Furtado Pinto Silva

CRÉDITOS

- O material dessa aula foi gentilmente cedido pelo Professor André Ricardo Backes e, por esse motivo, o crédito é dele.



DEFINIÇÃO

- 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[6];`



DEFINIÇÃO

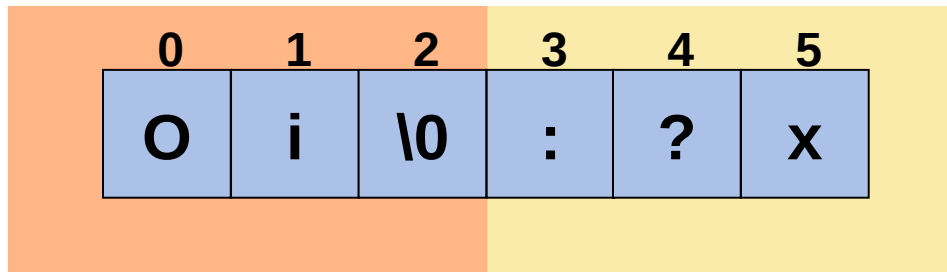
○ String

- Devemos ficar atentos para o fato de que as strings têm no elemento seguinte a última letra da palavra/frase armazenado um caractere ‘\0’ (barra invertida + zero).
- O caractere ‘\0’ indica o fim da sequência de caracteres.

○ Exemplo

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

Região inicializada:
2 letras + 1
caractere
terminador ‘\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 5 caracteres.

○ Exemplo:

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

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



DEFINIÇÃO

- Por se tratar de um array, cada caractere podem ser acessados 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
---	---	---	---	---	----

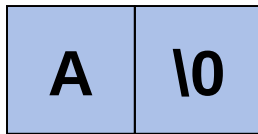


DEFINIÇÃO

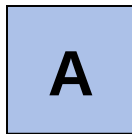
- **Importante:**

- “A” é diferente de ‘A’

- “A”



- ‘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		a	int
12	19		
13			
14			
		



MANIPULANDO STRINGS

- Strings são arrays. Portanto, **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.



COPIANDO UMA STRING

- 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;
}
```



MANIPULANDO STRINGS

- Felizmente, a biblioteca padrão C possui funções especialmente desenvolvidas para esse tipo de tarefa
 - **#include <string.h>**



MANIPULANDO STRINGS - LEITURA

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

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



MANIPULANDO STRINGS – LIMPEZA DO BUFFER

- Às vezes, podem ocorrer erros durante a leitura de caracteres ou strings.
- Para resolver esses pequenos erros, podemos limpar o buffer do teclado

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

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

```
gets(str);
```



MANIPULANDO STRINGS - ESCRITA

- Basicamente, para se escrever uma string na tela utilizamos a função **printf()**.
 - Especificador de formato: %s

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



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 - COPIAR

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

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



MANIPULANDO STRINGS - CONCATENAR

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

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



MANIPULANDO STRINGS - COMPARAR

- **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");
```



OBSERVAÇÃO

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	?	@
---	---	----	---	---	---



MATERIAL COMPLEMENTAR

○ Vídeo Aulas

- Aula 31: Strings: Conceitos Básicos:
www.youtube.com/watch?v=5mJZh_ikDaQ
- Aula 32: Strings: Biblioteca string.h:
youtu.be/MEkrf1O_CIU
- Aula 33: Strings: Invertendo uma String:
youtu.be/jNQUEpwMd_M
- Aula 34: Strings: Contando Caracteres Específicos:
youtu.be/s_V_LZX1eD0
- Aula 81: Limpando o buffer do teclado:
www.youtube.com/watch?v=ixk5RIqABjI

