

LINGUAGEM C: ARRAYS DE CARACTERES: STRINGS

Prof. Humberto Razente
Sala 1B144

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

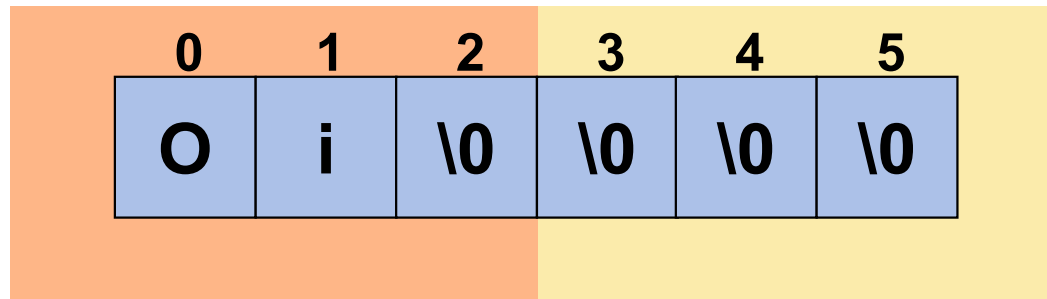
○ String

- Devemos ficar atentos para o fato de que as strings têm no elemento seguinte a última letra da palavra/frase armazenado um caracter ‘\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’



DEFINIÇÃO

○ Importante

- Ao definir o tamanho de uma string, devemos considerar o caracter ‘\0’
- Isso significa que a string **str[6]** 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 um par de “aspas”
- Ex: **char str[6] = “Teste”;**

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

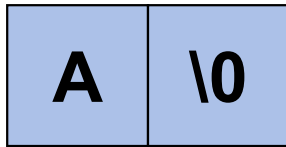
- Na atribuição de um caracter, usa-se um par de ‘apóstrofes’
- **str[0] = ‘L’;**

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

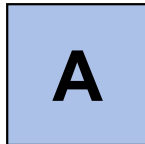
DEFINIÇÃO

○ Importante:

- “A” é diferente de ‘A’
 - “A”



- ‘A’



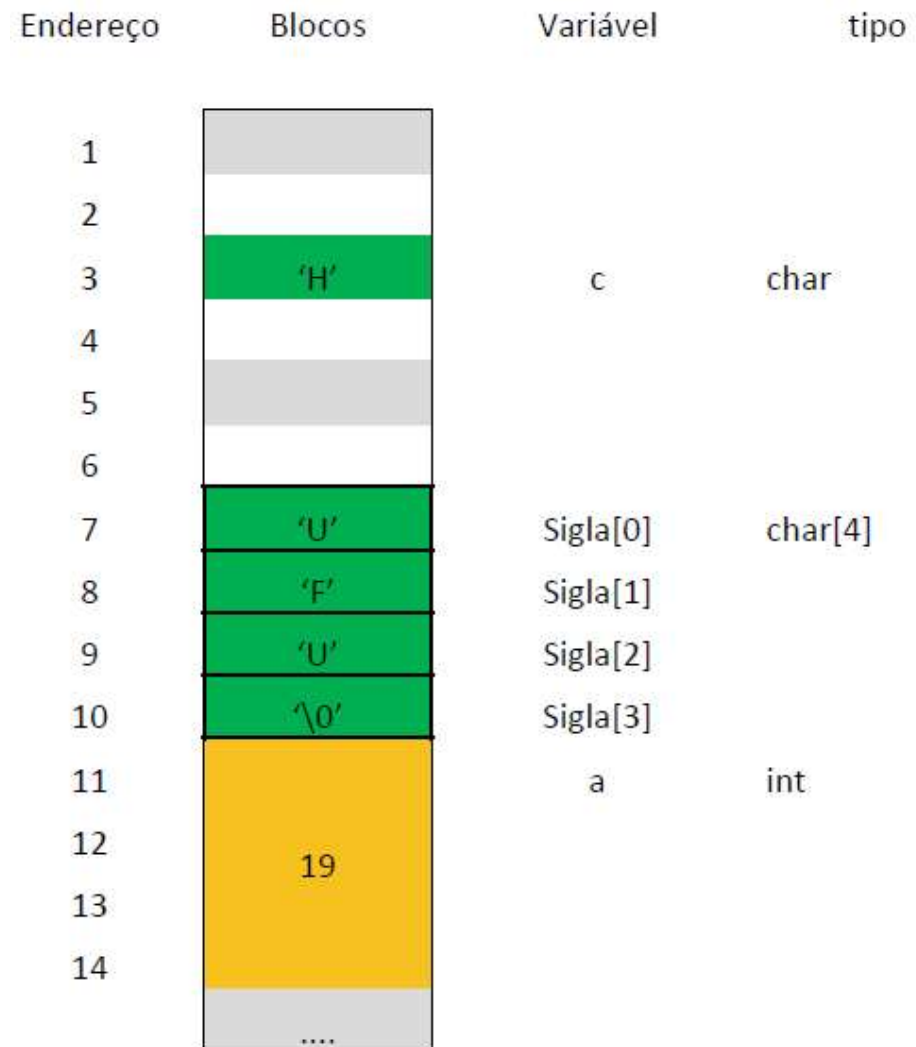
DEFINIÇÃO

○ 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';
```



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 anexada ao final da string de **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. Retorno:
 - ZERO se as strings forem iguais
- Exemplo:

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

MANIPULANDO STRINGS - COMPARAR

- **strcmp(str1, str2):** compara duas strings. Retorno:
 - um valor negativo se str1 ocorrer antes de str2
 - um valor positivo se str1 ocorrer depois de str2
- Ordem lexicográfica
- Valor: diferença do código do caractere onde não houve casamento
- ASCII 'c' é 99 e 'C' é 67

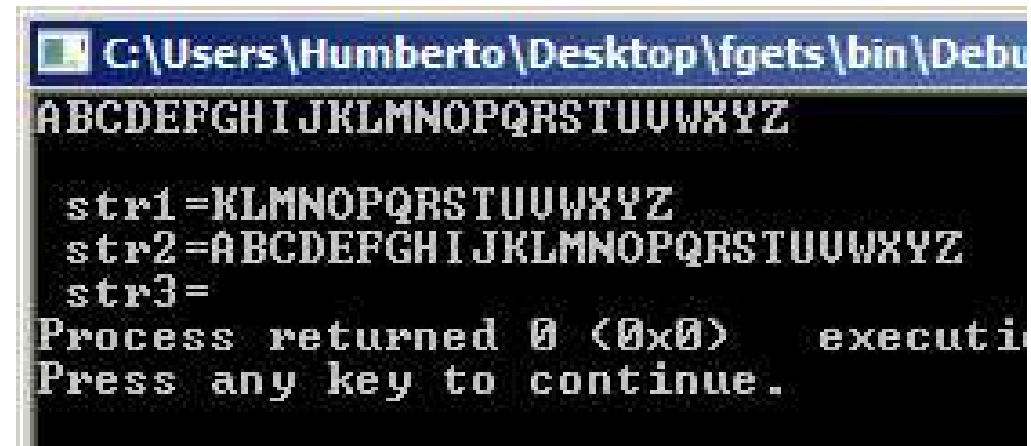
```
char str1[] = "abcd", str2[] = "abCd";  
int result;
```

```
result = strcmp(str1, str2);  
printf("strcmp(str1, str2) = %d\n", result);
```

MANIPULANDO STRINGS

- Basicamente, para se ler uma string do teclado utilizamos a função **gets()**
- A função gets não limita a leitura de caracteres, podendo invadir a memória de outras variáveis
 - entrada: "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 int main() {
6     char str1[10] = "";
7     char str2[10] = "";
8     char str3[10] = "";
9     gets(str2);
10    printf("\n str1=%s ", str1);
11    printf("\n str2=%s ", str2);
12    printf("\n str3=%s ", str3);
13    return 0;
14 }
```



```
C:\Users\Humberto\Desktop\fgets\bin\Debug
ABCDEFGHIJKLMNOPQRSTUVWXYZ

str1=KLMNOPQRSTUVWXYZ
str2=ABCDEFGHIJKLMNOPQRSTUVWXYZ
str3=
Process returned 0 (0x0)   execution time: 0.000 s
Press any key to continue.
```

MANIPULANDO STRINGS

- 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);
```

MANIPULANDO STRINGS

- A função **fgets** recebe 3 argumentos
 - a string a ser lida, **str**
 - o limite máximo de caracteres a serem lidos, **tamanho (incluindo o '\0')**
 - ou seja, ela lê no máximo tamanho-1 caracteres
 - 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);
```

MANIPULANDO STRINGS

← → ↻ ⓘ Not secure | www.cplusplus.com/reference/cstdio/fgets/

function

fgets

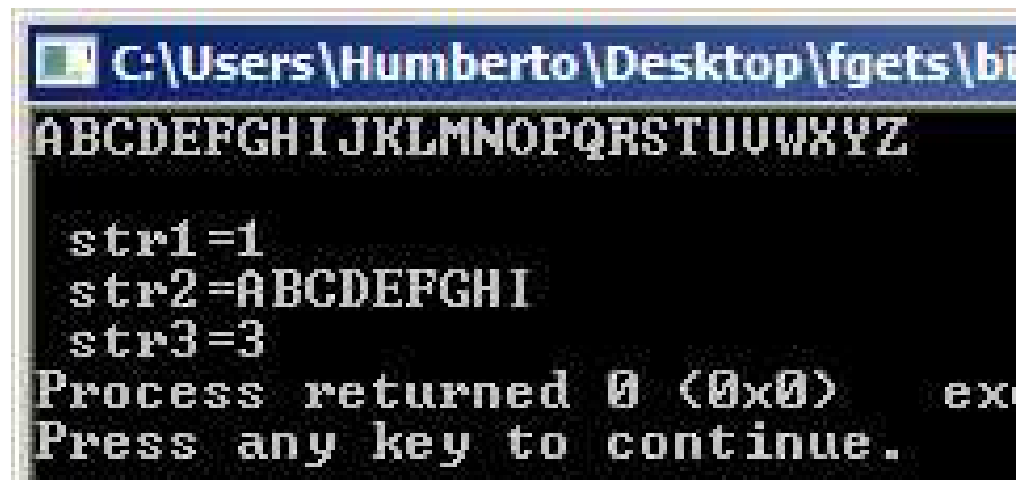
<stdio>

```
char * fgets ( char * str, int num, FILE * stream );
```

Get string from stream

Reads characters from *stream* and stores them as a C string into *str* until (*num*-1) characters have been read or either a newline or the *end-of-file* is reached, whichever happens first.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 int main() {
6     char str1[10] = "1";
7     char str2[10] = "2";
8     char str3[10] = "3";
9     fgets(str2, 10, stdin);
10    printf("\n str1=%s ", str1);
11    printf("\n str2=%s ", str2);
12    printf("\n str3=%s ", str3);
13    return 0;
14 }
```



```
C:\Users\Humberto\Desktop\fgets\bi
ABCDEFGH IJKLMNOPQRSTUVWXYZ

str1=1
str2=ABCDEFGH
str3=3
Process returned 0 (0x0)   ex
Press any key to continue.
```

MANIPULANDO 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;  
}
```


MANIPULANDO 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**

MANIPULANDO STRINGS

- A função **fgets** é semelhante à função **gets**, porém, com a seguinte vantagem:
 - especifica o tamanho máximo da string de entrada → evita estouro no buffer

FGETS E GETS

- Tanto **gets()** quanto **fgets()** lêem do teclado até encontrar um caracter de nova linha (ENTER → ‘\n’)
- Na função **fgets()** o caracter de nova linha (‘\n’) fará parte da string, e um terminador de linha ‘\0’ será acrescentado em seguida
- Na função **gets()** o caracter de nova linha (‘\n’) não fará parte da string

MANIPULANDO 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);
```

MANIPULANDO 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

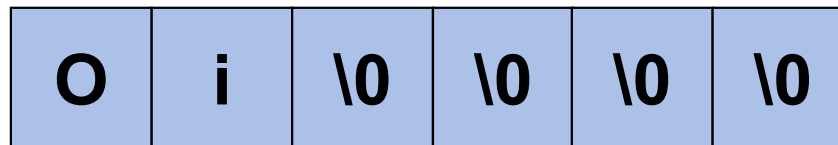
MANIPULANDO STRINGS

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



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

BUSCA EM STRINGS

- A função **strstr** retorna um ponteiro para a **primeira** ocorrência de **str2** em **str1**

```
char * strstr (char * str1, char * str2 );
```

- ou retorna NULL se **str2** não for encontrado em **str1**



strstr

<cstring>

```
const char * strstr ( const char * str1, const char * str2 );  
char * strstr (      char * str1, const char * str2 );
```

Locate substring

Returns a pointer to the first occurrence of *str2* in *str1*, or a null pointer if *str2* is not part of *str1*.

The matching process does not include the terminating null-characters, but it stops there.

Parameters

str1

C string to be scanned.

str2

C string containing the sequence of characters to match.

Return Value

A pointer to the first occurrence in *str1* of the entire sequence of characters specified in *str2*, or a null pointer if the sequence is not present in *str1*.



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

int main() {
    // poema de Carlos Drummond de Andrade
    char poema[250] = "E agora, Jose? A festa acabou, a luz apagou, o povo
sumiu, a noite esfriou, e agora, Jose? e agora, voce? voce que e sem nome, que
zomba dos outros, voce que faz versos, que ama, protesta? e agora, Jose?";
    printf("Considere o seguinte poema: \"%s\"", poema);

    char busca[20];
    do {
        printf("\n\nTermo busca (digite 'sair' para sair): ");
        gets(busca);
        int quantidade = 0;
        char *ocorrencia = poema;
        do {
            ocorrencia = strstr(ocorrencia, busca);
            if (ocorrencia != NULL) {
                printf("\n--> %s\n", ocorrencia);
                ocorrencia++;
                quantidade++;
            }
        } while (ocorrencia != NULL);
        printf("\n O termo %s foi encontrado %d vezes", busca, quantidade);
    } while(strcmp(busca, "sair") != 0);
}
```

ASCII ISO/IEC 8859-1 (LATIN1)

ISO 8859-1 (Latin-1)															
–0	–1	–2	–3	–4	–5	–6	–7	–8	–9	–A	–B	–C	–D	–E	–F
SP 0020 32	! 0021 33	" 0022 34	# 0023 35	\$ 0024 36	% 0025 37	& 0026 38	' 0027 39	(0028 40) 0029 41	* 002A 42	+ 002B 43	, 002C 44	- 002D 45	. 002E 46	/ 002F 47
0 0030 48	1 0031 49	2 0032 50	3 0033 51	4 0034 52	5 0035 53	6 0036 54	7 0037 55	8 0038 56	9 0039 57	: 003A 58	; 003B 59	< 003C 60	= 003D 61	> 003E 62	? 003F 63
@ 0040 64	A 0041 65	B 0042 66	C 0043 67	D 0044 68	E 0045 69	F 0046 70	G 0047 71	H 0048 72	I 0049 73	J 004A 74	K 004B 75	L 004C 76	M 004D 77	N 004E 78	O 004F 79
P 0050 80	Q 0051 81	R 0052 82	S 0053 83	T 0054 84	U 0055 85	V 0056 86	W 0057 87	X 0058 88	Y 0059 89	Z 005A 90	[005B 91	\ 005C 92] 005D 93	^ 005E 94	_ 005F 95
` 0060 96	a 0061 97	b 0062 98	c 0063 99	d 0064 100	e 0065 101	f 0066 102	g 0067 103	h 0068 104	i 0069 105	j 006A 106	k 006B 107	l 006C 108	m 006D 109	n 006E 110	o 006F 111
p 0070 112	q 0071 113	r 0072 114	s 0073 115	t 0074 116	u 0075 117	v 0076 118	w 0077 119	x 0078 120	y 0079 121	z 007A 122	{ 007B 123	 007C 124	}	~ 007E 126	

ASCII ISO/IEC 8859-1 (LATIN1)

	ı	ç	£	¤	¥	¦	§	¨	©	ª	«	¬		®	¯
00A0 160	00A1 161	00A2 162	00A3 163	00A4 164	00A5 165	00A6 166	00A7 167	00A8 168	00A9 169	00AA 170	00AB 171	00AC 172	00AD 173	00AE 174	00AF 175
	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
00B0 176	00B1 177	00B2 178	00B3 179	00B4 180	00B5 181	00B6 182	00B7 183	00B8 184	00B9 185	00BA 186	00BB 187	00BC 188	00BD 189	00BE 190	00BF 191
À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
00C0 192	00C1 193	00C2 194	00C3 195	00C4 196	00C5 197	00C6 198	00C7 199	00C8 200	00C9 201	00CA 202	00CB 203	00CC 204	00CD 205	00CE 206	00CF 207
Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
00D0 208	00D1 209	00D2 210	00D3 211	00D4 212	00D5 213	00D6 214	00D7 215	00D8 216	00D9 217	00DA 218	00DB 219	00DC 220	00DD 221	00DE 222	00DF 223
à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
00E0 224	00E1 225	00E2 226	00E3 227	00E4 228	00E5 229	00E6 230	00E7 231	00E8 232	00E9 233	00EA 234	00EB 235	00EC 236	00ED 237	00EE 238	00EF 239
ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ
00F0 240	00F1 241	00F2 242	00F3 243	00F4 244	00F5 245	00F6 246	00F7 247	00F8 248	00F9 249	00FA 250	00FB 251	00FC 252	00FD 253	00FE 254	00FF 255

EXERCÍCIO 1

- Escreva um programa que leia uma string e converta todos os seus caracteres em caracteres maiúsculos. Imprima a string convertida. Dica: subtrair 32 dos caracteres cujo código ASCII estiverem entre 97 e 122.



EXERCÍCIO 2

- Escreva um programa que faça uso das funções `gets`, `fgets`, `strcpy`, `strcat`, `strlen` e `strstr`. Execute o programa em modo de depuração (debug).



EXERCÍCIO 3

- Faça um programa que leia uma string e a inverta. A string invertida deve ser armazenada na mesma variável (ou seja, no mesmo vetor de caracteres). Em seguida, imprima a string invertida.



EXERCÍCIO 4

- Faça um programa que leia uma string e imprima uma mensagem dizendo se ela é um palíndromo ou não. Um palíndromo é uma palavra ou frase que tem a propriedade de poder ser lida tanto da direita para a esquerda como da esquerda para a direita. Exemplos:
 - ovo
 - arara
 - rever
 - osso
 - socorrammesubinoonibusemmarrocoss



EXERCÍCIO 5

- Escreva um programa que leia 3 strings e as imprima em ordem alfabética.
- Utilize a função strcmp para comparar strings.

```
int main() {  
    char M[20] = "Joao Paulo";  
    char N[20] = "Bento";  
    char R[20] = "Francisco";  
}
```



EXERCÍCIO 6

- Escreva um programa que leia uma matriz de caracteres (um vetor de strings) com nomes e apresente quem seria o primeiro da lista (pela ordem alfabética).
- Utilize a função strcmp para comparar strings.

```
int main() {  
    char M[20][30];  
  
    // leia cada string M[i]  
  
    // localize o elemento que seria primeiro  
    // de uma lista ordenada  
}
```



EXERCÍCIO 7

- Escreva laços para obter o mesmo resultado das seguintes funções:
 - strlen
 - strcpy
 - strcat

```
int main() {  
    char M[100] = "The quick brown fox jumps over the lazy dog";  
    char N[100] = "A ligeira raposa marrom saltou sobre o cachorro preguiçoso";  
    char R[100];  
  
    // 1) imprimir o tamanho das strings M e N  
    // 2) copiar M para R  
    // 3) concatenar N em R  
}
```



EXERCÍCIO 8

- O código de César é uma das técnicas de criptografia mais simples e conhecidas. É um tipo de substituição no qual cada letra do texto é substituída por outra, que se apresenta n posições após ela no alfabeto. Por exemplo, com uma troca de três posições, a letra A seria substituída por D, B se tornaria E e Z se tornaria C. Escreva um programa que faça uso desse código de César para um número de posições informado pelo usuário. Entre com uma string e imprima a string codificada. Exemplo:
 - a ligeira raposa marrom saltou sobre o cachorro cansado xyz
 - d oljhlud udsrvd pduurp vdowrx vreuh r fdfkruur fdqvdgr abc

EXERCÍCIO 9


- Escreva um programa que ordene a seguinte lista de presença
- Utilize a função strcmp para comparar strings.

```
#include <stdio.h>
#include <stdlib.h>
#define QUANTIDADE 21
int main() {
    char temp[12];
    char lista[QUANTIDADE][12] = { "Vinicius", "Guilherme", "Vitor", "Lucas", "Igor", "Joao",
                                     "Pedro", "Abel", "Luiz", "Wemerson", "Rafael", "Pablo",
                                     "Saint", "Thais", "Matheus", "Douglas", "Gabriel",
                                     "Viviane", "Reginaldo", "Jose", "Leonardo" };

    printf("Antes de ordenar:\n=====\\n");
    for (int i = 0; i < QUANTIDADE; i++)
        printf("%s\\n", lista[i]);

    ...

    printf("\\n\\nApos ordenar:\\n=====\\n");
    for (int i = 0; i < QUANTIDADE; i++)
        printf("%s\\n", lista[i]);
    return 0;
}
```



MATERIAL COMPLEMENTAR

○ Vídeo Aulas

- Aula 31: Strings: Conceitos Básicos
 - Aula 32: Strings: Biblioteca string.h
 - Aula 33: Strings: Invertendo uma String
 - Aula 34: Strings: Contando Caracteres Específicos
 - Aula 81: Limpando o buffer do teclado
-
- <https://programacaodescomplicada.wordpress.com/index/linguagem-c/>



LINGUAGEM C: ARRAYS DE CARACTERES: STRINGS

48

Contém slides originais gentilmente
disponibilizados pelo Prof. André R. Backes (UFU)