
Fundamentos de Programação

Aula 09 - Strings (Sequências de caracteres)

Arnaldo Barreto Vila Nova

Sumário

- Primeiro, vamos aprofundar um pouco nossos conhecimentos em caracteres, revendo também alguns conceitos já trabalhados com um pouco mais de detalhes
 - Depois, podemos começar a trabalhar com frases, nomes, entre outros tipos de strings.
-

Caracteres

- Um computador opera com números. Como ele trata letras e textos?
 - Temos diferentes codificações (encoding) possíveis para traduzir uma letra em um número e vice-versa
 - ASCII (American Standard Code for Information Interchange)
 - UNICODE
 - UTF-8
 - LATIN1 (ISO-8859-1)
-

Caracteres

- A linguagem C não tem um encoding padrão específico
 - Em geral, utiliza-se o encoding estabelecido pelo SO
 - Atualmente, o comum aqui no ocidente é o UTF-8
 - UTF-8 é compatível com ASCII, que é mais simples de entendermos e serve ao nosso propósito aqui em FUP
 - ASCII começou com 256 caracteres, mas depois foi ampliado com mais 128 caracteres (Extended ASCII)
-

Caracteres

- Extended ASCII usa 1 byte inteiro para codificação
 - char → valores de -128 a 127
 - unsigned char → valores de 0 a 255
 - 33 caracteres são de controle (0 a 31 e 127)
 - Não tem símbolos, mas tem funções específicas como o DEL, BACKSPACE, TAB, quebra de linha, entre outros.
-

Caracteres

- Caracteres de Controle

- não são visíveis, mas também são caracteres

ASCII control characters			
DEC	HEX	Símbolo ASCII	
00	00h	NULL	(carácter nulo)
01	01h	SOH	(inicio encabezado)
02	02h	STX	(inicio texto)
03	03h	ETX	(fin de texto)
04	04h	EOT	(fin transmisión)
05	05h	ENQ	(enquiry)
06	06h	ACK	(acknowledgement)
07	07h	BEL	(timbre)
08	08h	BS	(retroceso)
09	09h	HT	(tab horizontal)
10	0Ah	LF	(salto de línea)
11	0Bh	VT	(tab vertical)
12	0Ch	FF	(form feed)
13	0Dh	CR	(retorno de carro)
14	0Eh	SO	(shift Out)
15	0Fh	SI	(shift In)
16	10h	DLE	(data link escape)

17	11h	DC1	(device control 1)
18	12h	DC2	(device control 2)
19	13h	DC3	(device control 3)
20	14h	DC4	(device control 4)
21	15h	NAK	(negative acknowle.)
22	16h	SYN	(synchronous idle)
23	17h	ETB	(end of trans. block)
24	18h	CAN	(cancel)
25	19h	EM	(end of medium)
26	1Ah	SUB	(substitute)
27	1Bh	ESC	(escape)
28	1Ch	FS	(file separator)
29	1Dh	GS	(group separator)
30	1Eh	RS	(record separator)
31	1Fh	US	(unit separator)
127	20h	DEL	(delete)

Caracteres

- Caracteres gráficos do ASCII

ASCII printable characters								
DEC	HEX	Símbolo	DEC	HEX	Símbolo	DEC	HEX	Símbolo
32	20h	espacio	64	40h	@	96	60h	`
33	21h	!	65	41h	A	97	61h	a
34	22h	"	66	42h	B	98	62h	b
35	23h	#	67	43h	C	99	63h	c
36	24h	\$	68	44h	D	100	64h	d
37	25h	%	69	45h	E	101	65h	e
38	26h	&	70	46h	F	102	66h	f
39	27h	'	71	47h	G	103	67h	g
40	28h	(72	48h	H	104	68h	h
41	29h)	73	49h	I	105	69h	i
42	2Ah	*	74	4Ah	J	106	6Ah	j
43	2Bh	+	75	4Bh	K	107	6Bh	k
44	2Ch	,	76	4Ch	L	108	6Ch	l
45	2Dh	-	77	4Dh	M	109	6Dh	m
46	2Eh	.	78	4Eh	N	110	6Eh	n
47	2Fh	/	79	4Fh	O	111	6Fh	o
48	30h	0	80	50h	P	112	70h	p
49	31h	1	81	51h	Q	113	71h	q
50	32h	2	82	52h	R	114	72h	r
51	33h	3	83	53h	S	115	73h	s
52	34h	4	84	54h	T	116	74h	t
53	35h	5	85	55h	U	117	75h	u
54	36h	6	86	56h	V	118	76h	v
55	37h	7	87	57h	W	119	77h	w
56	38h	8	88	58h	X	120	78h	x
57	39h	9	89	59h	Y	121	79h	y
58	3Ah	:	90	5Ah	Z	122	7Ah	z
59	3Bh	;	91	5Bh	[123	7Bh	{
60	3Ch	<	92	5Ch	\	124	7Ch	
61	3Dh	=	93	5Dh]	125	7Dh	}
62	3Eh	>	94	5Eh	^	126	7Eh	~
63	3Fh	?	95	5Fh	_			

Caracteres

- Caracteres gráficos do Extended ASCII

Extended ASCII characters											
DEC	HEX	Símbolo	DEC	HEX	Símbolo	DEC	HEX	Símbolo	DEC	HEX	Símbolo
128	80h	Ç	160	A0h	á	192	C0h	Ł	224	E0h	Ó
129	81h	ü	161	A1h	í	193	C1h	ł	225	E1h	ô
130	82h	é	162	A2h	ó	194	C2h	Ł	226	E2h	Ô
131	83h	â	163	A3h	ú	195	C3h	ł	227	E3h	Õ
132	84h	ä	164	A4h	ñ	196	C4h	Ł	228	E4h	ö
133	85h	à	165	A5h	Ñ	197	C5h	ł	229	E5h	Ö
134	86h	â	166	A6h	ª	198	C6h	Ł	230	E6h	µ
135	87h	ç	167	A7h	º	199	C7h	ł	231	E7h	þ
136	88h	ê	168	A8h	¿	200	C8h	Ł	232	E8h	Þ
137	89h	ë	169	A9h	®	201	C9h	ł	233	E9h	Ú
138	8Ah	è	170	AAh	¬	202	CAh	Ł	234	EAh	Û
139	8Bh	ï	171	ABh	½	203	CBh	ł	235	EBh	Ü
140	8Ch	î	172	ACH	¼	204	CCh	Ł	236	ECh	Ý
141	8Dh	ï	173	ADh	í	205	CDh	ł	237	EDh	Ý
142	8Eh	Ä	174	A Eh	«	206	CEh	Ł	238	EEh	ÿ
143	8Fh	Å	175	AFh	»	207	CFh	ł	239	EFh	ÿ
144	90h	É	176	B0h	⋮	208	D0h	Ł	240	F0h	
145	91h	æ	177	B1h	⋮	209	D1h	ł	241	F1h	±
146	92h	Æ	178	B2h	⋮	210	D2h	Ł	242	F2h	¼
147	93h	ô	179	B3h	⋮	211	D3h	ł	243	F3h	½
148	94h	ò	180	B4h	⋮	212	D4h	Ł	244	F4h	¾
149	95h	ò	181	B5h	⋮	213	D5h	ł	245	F5h	§
150	96h	ù	182	B6h	⋮	214	D6h	Ł	246	F6h	÷
151	97h	ù	183	B7h	⋮	215	D7h	ł	247	F7h	
152	98h	ÿ	184	B8h	⋮	216	D8h	Ł	248	F8h	°
153	99h	Ö	185	B9h	⋮	217	D9h	ł	249	F9h	°
154	9Ah	Ü	186	BAh	⋮	218	DAh	Ł	250	FAh	°
155	9Bh	ø	187	BBh	⋮	219	DBh	ł	251	FBh	°
156	9Ch	£	188	BCh	⋮	220	DCh	Ł	252	FBh	°
157	9Dh	Ø	189	BDh	⋮	221	DDh	ł	253	FDh	°
158	9Eh	x	190	BEh	⋮	222	DEh	Ł	254	FEh	°
159	9Fh	f	191	BFh	⋮	223	DFh	ł	255	FFh	°

Caracteres

```
void main()
{
    char c;
    printf("Cod\tChar\n");
    printf("---\t----\n");
    for (c=-128;c<127;c++)
    {
        printf("%d\t%c\n",c,c);
    }
}
```

```
void main()
{
    unsigned char c;
    printf("Cod\tChar\n");
    printf("---\t----\n");
    for (c=0;c<255;c++)
    {
        printf("%d\t%c\n",c,c);
    }
}
```

**Execute os códigos para visualizar os caracteres
de acordo com a tabela ASCII**

Caracteres

- Lembrando que os caracteres podem ser tratados tanto numericamente quanto por uma letra em aspas simples

```
char c = 'A';  
char d = 65;
```

```
if (c == d)  
    printf("iguais");  
else  
    printf("diferentes");
```

O código acima vai imprimir *iguais*

Caracteres

- Recebendo um caractere do usuário

```
char c;
```

```
scanf ("%c", &c);
```

Estamos acostumados com o scanf

Caracteres

- Recebendo um caractere do usuário

```
char c;
```

```
char c;
```

```
scanf("%c", &c);
```

```
c = getchar();
```

Mas também temos a opção do `getchar()`

Caracteres

- Um pequeno problema: a quebra de linha (o enter que o usuário pressiona para mandar um valor) também é um caractere

```
char c;  
while (c!=65)  
{  
    printf("Uma letra:\n");  
    scanf("%c", &c);  
    printf("Digitou: %c\n", c);  
}
```

```
char c;  
while (c!=65)  
{  
    printf("Uma letra:\n");  
    c = getchar();  
    printf("Digitou: %c\n", c);  
}
```

Estes trechos de código querem pegar um letra até o usuário digitar a letra A, mas algo estranho acontece...

Caracteres

- Um pequeno problema: a quebra de linha (o enter que o usuário pressiona para mandar um valor) também é um caractere

```
char c;
while (c!=65)
{
    printf("Uma letra:\n");
    scanf("%c", &c);
    printf("Digitou: %c\n", c);
}
```

```
char c;
while (c!=65)
{
    printf("Uma letra:\n");
    c = getchar();
    printf("Digitou: %c\n", c);
}
```

**Dá a impressão que o código não esperou o usuário digitar,
mas na verdade ele já digitou o enter antes.**

Caracteres

- É comum alguém sugerir usar **fflush(stdin)** para resolver

```
char c;
while (c!=65)
{
    printf("Uma letra:\n");
    fflush(stdin);
    scanf("%c", &c);
    printf("Digitou: %c\n", c);
}
```

```
char c;
while (c!=65)
{
    printf("Uma letra:\n");
    fflush(stdin);
    c = getchar();
    printf("Digitou: %c\n", c);
}
```

Muita gente acha que o propósito deste comando é limpar o buffer de entrada (stdin), mas não é bem assim...

Caracteres

- É comum alguém sugerir usar **fflush(stdin)** para resolver

```
char c;
while (c!=65)
{
    printf("Uma letra:\n");
    fflush(stdin);
    scanf("%c", &c);
    printf("Digitou: %c\n", c);
}
```

```
char c;
while (c!=65)
{
    printf("Uma letra:\n");
    fflush(stdin);
    c = getchar();
    printf("Digitou: %c\n", c);
}
```

Pode funcionar, mas o comportamento do comando neste tipo de caso é imprevisível (ou seja, nem sempre funciona)

Caracteres

- Podemos corrigir isso simplesmente colocando um espaço em branco antes do `%c` no `scanf`

```
char c;
while (c!=65 && c!='\n')
{
    printf("Uma letra:\n");
    scanf(" %c", &c);
    printf("Digitou: %c\n", c);
}
```

O espaço antes do `%c` vai fazer com que qualquer espaço em branco digitado anteriormente seja desconsiderado, incluindo quebra de linha, tabulação, entre outros caracteres especiais.

Caracteres

- Para o getchar, podemos colocá-lo em uma repetição

Aqui o segundo while vai
descartar quebras de
linha anteriores

```
char c;  
while (c!=65)  
{  
    printf("Uma letra:\n");  
    while((c = getchar()) == '\n');  
    printf("Digitou: %c\n", c);  
}
```

Caracteres

- Os caracteres referentes aos dígitos, às letras maiúsculas e às letras minúsculas são sequenciais

```
char c = getchar();
```

```
if ((c >= 'A') && (c <= 'Z'))  
    printf("digitou uma letra maiúscula");  
else  
    printf("nao digitou uma letra maiúscula");
```

Strings - Sequências de caracteres

- Em C, strings são arrays do tipo char
- O caractere nulo (`'\0'`) indica o final da string, mesmo que haja mais caracteres depois no array;
- Ou seja, um vetor char de 10 posições pode comportar uma string de até 9 caracteres

```
char str1[50];  
char str2[8] = "ola";  
char str3[] = "Saudações, humano!";  
str3[9] = '!';
```

Strings - Sequências de caracteres

- Em C, strings são arrays do tipo char
- O caractere nulo (`'\0'`) indica o final da string, mesmo que haja mais caracteres depois no array;
- Ou seja, um vetor char de 10 posições pode comportar uma string de até 9 caracteres

```
char str1[50];  
char str2[8] = "ola";  
char str3[] = "Saudações, humano!";  
str3[9] = '!';
```

**Este array pode ter
uma frase de até 49
caracteres**

Strings - Sequências de caracteres

- Em C, strings são arrays do tipo char
- O caractere nulo (`'\0'`) indica o final da string, mesmo que haja mais caracteres depois no array;
- Ou seja, um vetor char de 10 posições pode comportar uma string de até 9 caracteres

```
char str1[50];  
char str2[8] = "ola";  
char str3[] = "Saudações, humano!";  
str3[9] = '!';
```



**Este array está
recebendo 'o', 'l' e
'a' nas 3 primeiras
posições e '\0' no
índice 3.**

Strings - Sequências de caracteres

- Em C, strings são arrays do tipo char
- O caractere nulo (`'\0'`) indica o final da string, mesmo que haja mais caracteres depois no array;
- Ou seja, um vetor char de 10 posições pode comportar uma string de até 9 caracteres

```
char str1[50];  
char str2[8] = "ola";  
char str3[] = "Saudações, humano!";  
str3[9] = '!';
```

O tamanho deste array
será a quantidade de
caracteres da frase
somado ao `'\0'` no final
(18, neste caso)

Strings - Sequências de caracteres

- Em C, strings são arrays do tipo char
- O caractere nulo (`'\0'`) indica o final da string, mesmo que haja mais caracteres depois no array;
- Ou seja, um vetor char de 10 posições pode comportar uma string de até 9 caracteres

```
char str1[50];  
char str2[8] = "ola";  
char str3[] = "Saudações, humano!";  
str3[9] = '!';
```

Nesta linha, o 10º
caractere do array
anterior está sendo
modificado para '!'

Strings - Sequências de caracteres

- Recebendo uma string do usuário
 - Podemos usar o scanf com o código %s

```
char nome[50];
```

```
printf("Seu nome?\n");  
scanf("%s", &nome);  
printf("Olá, %s", nome);
```

Irá aparecer na tela:

Seu nome?

Strings - Sequências de caracteres

- Recebendo uma string do usuário
 - Podemos usar o scanf com o código %s

```
char nome[50];
```

```
printf("Seu nome?\n");  
scanf("%s", &nome);  
printf("Olá, %s", nome);
```

Digitando "Arnaldo":

Seu nome?

Arnaldo

Olá, Arnaldo

Strings - Sequências de caracteres

- Recebendo uma string do usuário
 - Podemos usar o scanf com o código %s

```
char nome[50];
```

```
printf("Seu nome?\n");  
scanf("%s", &nome);  
printf("Olá, %s", nome);
```

**Este formato %s vai
pegar tudo que o
usuário digitar até um
espaço em branco
aparecer**

Strings - Sequências de caracteres

- Recebendo uma string do usuário
 - Podemos usar o scanf com o código %s

```
char nome[50];
```

```
printf("Seu nome?\n");  
scanf("%s", &nome);  
printf("Olá, %s", nome);
```

Por isso, quando digito
"Arnaldo Barreto", ele
só pega "Arnaldo"

Strings - Sequências de caracteres

- Recebendo uma string do usuário
 - Podemos usar o scanf com o código %s

```
char nome[50];
```

```
printf("Seu nome?\n");  
scanf("%[^\\n]s", &nome);  
printf("Olá, %s", nome);
```



**Precisamos então
informar para ele pegar
até um determinado
caractere: o '\\n'**

Strings - Sequências de caracteres

- Recebendo uma string do usuário
 - Podemos usar o scanf com o código %s

```
char nome[50];
```

```
printf("Seu nome?\n");  
scanf("%[^\\n]s", &nome);  
printf("Olá, %s", nome);
```



Os colchetes com acento circunflexo ([^]) indicam até qual caractere desejamos pegar.

Strings - Sequências de caracteres

- Recebendo uma string do usuário
 - Podemos usar o scanf com o código %s

```
char nome[50];
```

```
printf("Seu nome?\n");  
scanf("%[^\\n]s", &nome);  
printf("Olá, %s", nome);
```



Dessa forma, o scanf vai
pegar tudo o que for
escrito até a quebra de
linha (ENTER)

Strings - Sequências de caracteres

- Recebendo uma string do usuário
 - Podemos usar o scanf com o código %s

```
char nome[50];
```

```
printf("Seu nome?\n");  
scanf("%[^d]s", &nome);  
printf("Olá, %s", nome);
```



Neste outro caso, o scanf vai pegar tudo o que for escrito até a letra 'd' aparecer.

Strings - Sequências de caracteres

- Recebendo uma string do usuário
 - Podemos usar o scanf com o código %s

```
char nome[50];
```

```
printf("Seu nome?\n");  
scanf("%5s", &nome);  
printf("Olá, %s", nome);
```



**Também é possível
especificar quantos
caracteres queremos
receber (ou até um espaço
em branco aparecer)**

Strings - Sequências de caracteres

- Recebendo uma string do usuário
 - Podemos usar o scanf com o código %s

```
char nome[50];
```

```
printf("Seu nome?\n");  
scanf("%10[^\\n]s", &nome);  
printf("Olá, %s", nome);
```

Neste caso, estou pegando
até 10 caracteres ou até
aparecer a quebra de
linha

Strings - Sequências de caracteres

- Recebendo uma string do usuário
 - Também podemos utilizar a função `gets`

```
char nome[50];  
  
printf("Seu nome?\n");  
gets(&nome);  
printf("Olá, %s", nome);
```

O `gets` faz o mesmo que o `scanf` com `%[^\n]s` sem outras opções

```
Seu nome?  
Arnaldo Barreto  
Olá, Arnaldo Barreto
```

Strings - Sequências de caracteres

- Recebendo uma string do usuário
 - O que é lido tem que caber no vetor

```
char nome[5];
```

```
printf("Seu nome?\n");  
scanf("%[^\\n]s", &nome);  
printf("Olá, %s", nome);
```

Obviamente, só é possível
pegar o que cabe no array
(considerando o '\\0').

Seu nome?
Arnaldo Barreto
Olá, Arna

Um array de tamanho 5 só
consegue guardar "Arna"

Strings - Sequências de caracteres

- Imprimindo uma string
 - O **printf** com **%s** irá imprimir o conteúdo da string até o **'\0'**

```
char nome[50];
```

```
printf("Seu nome?\n");  
scanf("%[^\\n]s", &nome);  
printf("Olá, %s", nome);
```

**No final de uma string
sempre tem **'\0'**.**

Seu nome?

Arnaldo Barreto

Olá, Arnaldo Barreto

Strings - Sequências de caracteres

- Imprimindo uma string
 - O **printf** com **%s** irá imprimir o conteúdo da string até o **'\0'**

```
char nome[50];  
  
printf("Seu nome?\n");  
scanf("%[^\\n]s", &nome);  
nome[7] = '\\0';  
printf("Olá, %s", nome);
```

Se tiver outro '\\0', nada
depois do primeiro será
impresso.

Seu nome?
Arnaldo Barreto
Olá, Arnaldo

Strings - Sequências de caracteres

- Imprimindo uma string
 - Podemos imprimir uma string usando também a função **puts**

```
char nome[50];
```

```
printf("Seu nome?\n");  
gets(nome);  
puts(nome);
```

**Ele fará a mesma coisa do
printf com o código %s.**

Seu nome?
Arnaldo Barreto
Olá, Arnaldo Barreto

Operações com Strings

- A biblioteca `<string.h>` tem diversas funções úteis
 - `strlen(str)` → retorna o tamanho da string (nº de caracteres até o `'\0'`)

```
char nome[50];
```

```
printf("Seu nome? ");
```

```
gets(&nome);
```

```
printf("Seu nome tem %d letras", strlen(nome));
```

Seu nome? Arnaldo Barreto

Seu nome tem 15 letras

Operações com Strings

- A biblioteca <string.h> tem diversas funções úteis
 - strcpy(str1, str2) → copia str2 em str1

```
char nome1[50], nome2[50];
```

```
printf("Seu nome? ");
```

```
gets(nome1);
```

```
strcpy(nome2, nome1);
```

```
printf("nome2 copiado: %s", nome2);
```

```
Seu nome? Arnaldo Barreto
```

```
nome2 copiado: Arnaldo Barreto
```

Operações com Strings

- A biblioteca `<string.h>` tem diversas funções úteis
 - `strcpy(str1, str2)` → copia str2 em str1

```
char nome[50] = "João", snome[50] = "Siqueira";
```

Operações com Strings

- A biblioteca <string.h> tem diversas funções úteis
 - strcpy(str1, str2) → copia str2 em str1

```
char nome[50] = "João", snome[50] = "Siqueira";  
→ nome{'J', 'o', 'ã', 'o', '\0'} → snome{'S', 'i', 'q', 'u', 'e', 'i', 'r', 'a', '\0'}
```

Operações com Strings

- A biblioteca <string.h> tem diversas funções úteis
 - strcpy(str1, str2) → copia str2 em str1

```
char nome[50]= "João", snome[50] = "Siqueira";  
→ nome{'J','o','ã','o','\0'} → snome{'S','i','q','u','e','i','r','a','\0'}
```

```
strcpy(snome, nome);  
strcpy(nome, "Al");  
strcpy(snome, "Silva");
```

Operações com Strings

- A biblioteca `<string.h>` tem diversas funções úteis
 - `strcpy(str1, str2)` → copia str2 em str1

```
char nome[50] = "João", snome[50] = "Siqueira";  
    → nome{'J', 'o', 'ã', 'o', '\0'} → snome{'S', 'i', 'q', 'u', 'e', 'i', 'r', 'a', '\0'}  
  
strcpy(snome, nome);    → snome{'J', 'o', 'ã', 'o', '\0', 'i', 'r', 'a', '\0'}  
strcpy(nome, "Al");  
strcpy(snome, "Silva");
```

Operações com Strings

- A biblioteca `<string.h>` tem diversas funções úteis
 - `strcpy(str1, str2)` → copia str2 em str1

```
char nome[50] = "João", snome[50] = "Siqueira";  
→ nome{'J', 'o', 'ã', 'o', '\0'} → snome{'S', 'i', 'q', 'u', 'e', 'i', 'r', 'a', '\0'}
```

```
strcpy(snome, nome); → snome{'J', 'o', 'ã', 'o', '\0', 'i', 'r', 'a', '\0'}  
strcpy(nome, "Al"); → nome{'A', 'l', '\0', 'o', '\0'}  
strcpy(snome, "Silva");
```

Operações com Strings

- A biblioteca `<string.h>` tem diversas funções úteis
 - `strcpy(str1, str2)` → copia str2 em str1

```
char nome[50]= "João", snome[50] = "Siqueira";  
    → nome{'J','o','ã','o','\0'} → snome{'S','i','q','u','e','i','r','a','\0'}  
  
strcpy(snome, nome);      → snome{'J','o','ã','o','\0','i','r','a','\0'}  
strcpy(nome, "Al");       → nome{'A','l','\0','o','\0'}  
strcpy(snome, "Silva");   → snome{'S','i','l','v','a','\0','r','a','\0'}
```

Operações com Strings

- A biblioteca <string.h> tem diversas funções úteis
 - `strcat(str1, str2)` → copia str2 no final de str1

```
char nome[50] = "João", snome[50] = "Siqueira";  
→ nome{'J', 'o', 'ã', 'o', '\0'} → snome{'S', 'i', 'q', 'u', 'e', 'i', 'r', 'a', '\0'}
```

Operações com Strings

- A biblioteca <string.h> tem diversas funções úteis
 - `strcat(str1, str2)` → copia str2 no final de str1

```
char nome[50] = "João", snome[50] = "Siqueira";  
→ nome{'J', 'o', 'ã', 'o', '\0'} → snome{'S', 'i', 'q', 'u', 'e', 'i', 'r', 'a', '\0'}
```

```
strcat(nome, snome);
```

Operações com Strings

- A biblioteca <string.h> tem diversas funções úteis
 - `strcat(str1, str2)` → copia str2 no final de str1

```
char nome[50] = "João", snome[50] = "Siqueira";  
→ nome{'J', 'o', 'ã', 'o', '\0'} → snome{'S', 'i', 'q', 'u', 'e', 'i', 'r', 'a', '\0'}
```

```
strcat(nome, snome);  
→ nome{'J', 'o', 'ã', 'o', 'S', 'i', 'q', 'u', 'e', 'i', 'r', 'a', '\0'}
```

Operações com Strings

- A biblioteca `<string.h>` tem diversas funções úteis
 - `strcat(str1, str2)` → copia str2 no final de str1

```
char nome[50] = "João", snome[50] = "Siqueira";  
→ nome{'J', 'o', 'ã', 'o', '\0'} → snome{'S', 'i', 'q', 'u', 'e', 'i', 'r', 'a', '\0'}
```

```
strcat(nome, " ");
```

```
strcat(nome, snome);
```

Operações com Strings

- A biblioteca `<string.h>` tem diversas funções úteis
 - `strcat(str1, str2)` → copia str2 no final de str1

```
char nome[50] = "João", snome[50] = "Siqueira";  
→ nome{'J', 'o', 'ã', 'o', '\0'} → snome{'S', 'i', 'q', 'u', 'e', 'i', 'r', 'a', '\0'}
```

```
strcat(nome, " ");  
→ nome{'J', 'o', 'ã', 'o', ' ', '\0'}
```

```
strcat(nome, snome);
```

Operações com Strings

- A biblioteca `<string.h>` tem diversas funções úteis
 - `strcat(str1, str2)` → copia str2 no final de str1

```
char nome[50] = "João", snome[50] = "Siqueira";  
→ nome{'J', 'o', 'ã', 'o', '\0'} → snome{'S', 'i', 'q', 'u', 'e', 'i', 'r', 'a', '\0'}
```

```
strcat(nome, " ");  
→ nome{'J', 'o', 'ã', 'o', ' ', '\0'}
```

```
strcat(nome, snome);  
→ nome{'J', 'o', 'ã', 'o', ' ', 'S', 'i', 'q', 'u', 'e', 'i', 'r', 'a', '\0'}
```

Operações com Strings

- A biblioteca <string.h> tem diversas funções úteis
 - strcmp(str1, str2) → retorna 0 se str1 == str2 ou 1 se str1 != str2

```
char nome1[50]= "João", nome2[50] = "João";
```


```
if (nome1 == nome2)
    printf("iguais");
else
    printf("diferentes");
```

O que vai ser impresso?

Operações com Strings

- A biblioteca <string.h> tem diversas funções úteis
 - strcmp(str1, str2) → retorna 0 se str1 == str2 ou 1 se str1 != str2

```
char nome1[50]= "João", nome2[50] = "João";
```

```
if (nome1 == nome2)   
    printf("iguais");  
else  
    printf("diferentes");
```

Isso aqui está comparando o endereço de memória nome1 com o endereço de memória nome2, por isso vai imprimir "diferentes"

Operações com Strings

- A biblioteca <string.h> tem diversas funções úteis
 - `strcmp(str1, str2)` → retorna 0 se `str1 == str2` ou 1 se `str1 != str2`

```
char nome1[50]= "João", nome2[50] = "João";
```

```
if (strcmp(nome1, nome2)==0)
    printf("iguais");
else
    printf("diferentes");
```

Para comparar o conteúdo dos arrays podemos usar o `strcmp` e, neste caso, irá imprimir "iguais"

Exercícios

- Faça um programa que receba uma string e a escreva apenas com letras minúsculas
 - Dica: 'A' == 65 e 'a' == 97
 - Altere o programa acima para escrever apenas com letras maiúsculas
 - Agora, faça o programa substituir os seguintes caracteres pelos respectivos dígitos:
 - 'A' → '4'; 'E' → '3'; 'I' → '1'; 'O' → '0'; 'T' → '7'; 'S' → '5'
-

Exercícios

- Faça um programa que receba 2 nomes e os imprima em ordem alfabética.
 - Dica: Coloque todas as letras para minúsculas ou todas para maiúsculas e depois compare os caracteres
 - E se forem 3 nomes?
-

Exercícios

- Faça um programa que receba do usuário o nome de uma cidade que ele quer visitar e verifique se ela é um possível destino de uma empresa de turismo, sendo que esta empresa faz viagens para: SALVADOR, SÃO PAULO, FORTALEZA, TERESINA, NATAL, RIO DE JANEIRO, NOVA YORK, SANTIAGO, PARIS, LONDRES, BUENOS AIRES e MADRI
-