

Programação Procedimental

Strings em C

Aula 04

Prof. Felipe A. Louza



- 1 Strings em C
- 2 Imprimindo uma String
- 3 Leitura de String
- 4 A biblioteca `string.h`
- 5 A biblioteca `stdlib.h`
- 6 Processamento de Texto
- 7 Referências

Strings em C

Cadeias de caracteres (strings):

- Em C **não existe o tipo `string`**, utilizamos **vetores de caracteres** para armazenar uma **string**.
- Strings em C **sempre terminam** com o caractere especial: `\0`

e	x	e	m	p	l	o	\0
0	1	2	3	4	5	6	7

```
char texto[8];
```

Relembrando o tipo char

Uma letra ou caractere em C é representado pelo tipo char:

- é um número inteiro
 - ocupa 1 byte (valores entre 0 e 255)
 - podemos somar, subtrair, multiplicar, dividir, etc
- Tabela ASCII:
 - cada número representa um caractere



c

```
char c = 'a';
```

Tabela ASCII

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	\$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Exemplo

Exemplo 1:

- Escreva um programa que lê um caractere e determina se ele é maiúsculo ou minúsculo:

```
1  #include <stdio.h>
2
3  int main(){
4
5      char letra;
6      scanf( "%c" , &letra);
7
8      if(letra >= 65 && letra <= 90)
9          printf ("Caractere maiúsculo!\n");
10     else if(letra >= 97 && letra <= 122)
11         printf ("Caractere minúsculo!\n");
12     else
13         printf ("Caractere desconhecido!\n");
14
15     return 0;
16 }
```

Relembrando o tipo char

```
1  #include <stdio.h>
2
3  int main(){
4      int k ;
5      char letra;
6      scanf( "%d " , &k);
7      scanf( "%c" , &letra);
8      printf ("%d %c\n", k, letra);
9
10     return 0;
11 }
```

Importante:

- Há um espaço em branco após o %d
 - consome os próximos caracteres brancos: espaço, \n e \t
 - sem isso, o scanf("%c", &letra); leria um \n
 - cuidado, o C é chato na leitura de caracteres...

Strings em C

Strings em C:

- Sempre declare uma string com **um caractere a mais** do que precisa, já que também será preciso **armazenar o caractere '\0'**.

e	x	e	m	p	l	o	\0
0	1	2	3	4	5	6	7

```
1 char texto[8];
```


Strings em C

Strings em C:

- O tamanho da string é o **número de caracteres** antes do **'\0'**
- Por exemplo:

e	x	e	m	p	l	o	\0	l	i	x	o
0	1	2	3	4	5	6	7	8	9	10	11

```
1 char texto[12]; //o tamanho da string é 7
```

Strings em C

Podemos **inicializar** uma string durante sua a declaração:

```
1 char texto1[] = {'e', 'x', 'e', 'm', 'p', 'l', 'o', '\\0'};
```

ou (melhor):

```
1 char texto2[] = "exemplo";
```

- No segundo caso, o caractere '\\0' é inserido no final automaticamente.

e	x	e	m	p	l	o	\\0
0	1	2	3	4	5	6	7

Strings em C

Podemos declarar o tamanho dos vetores:

```
1 char texto1[12] = {'e', 'x', 'e', 'm', 'p', 'l', 'o', '\0'};
```

ou (melhor):

```
1 char texto2[12] = "exemplo";
```

- No segundo caso, o caractere `'\0'` é inserido no final automaticamente.

e	x	e	m	p	l	o	\0	l	i	x	o
0	1	2	3	4	5	6	7	8	9	10	11

Strings em C

Podemos encontrar o **tamanho da string** percorrendo-a até o **'\0'**

```
1 char texto[10] = "exemplo";  
2 int tam = 0;  
3 while(texto[tam] != '\0'){  
4     tam++;  
5 }  
6 printf("%d\n", tam);
```

- Vamos ver depois uma **função pronta** para retornar o tamanho de uma string.

Exemplo

Exemplo 2:

- Escreva um programa que converte **todos os caracteres para maiúsculo**:

```
1  #include <stdio.h>
2
3  int main(){
4
5      char texto[10] = "Exemplos!!";
6
7      int i;
8      for(i=0 ; i<10; i++){
9          if(texto[i] >= 97 && texto[i] <= 122) //minúsculo
10             texto[i] -= 32; // diferença: 'a'-'A' (97-65)
11      }
12
13      return 0;
14 }
```

- 1 Strings em C
- 2 Imprimindo uma String
- 3 Leitura de String
- 4 A biblioteca `string.h`
- 5 A biblioteca `stdlib.h`
- 6 Processamento de Texto
- 7 Referências

Imprimindo uma String

Para imprimir uma **string** usamos o formatador **%s**.

```
1 #include <stdio.h>
2
3 int main(){
4
5     char texto[80] = "Exemplo 1!";
6
7     printf("%s\n", texto);
8
9     return 0;
10 }
```

- Observe que precisamos do '**\n**' para pular linha.

```
1 Exemplo 1!
```

Imprimindo uma String

Podemos determinar a **quantidade mínima** de caracteres:

```
1 #include <stdio.h>
2
3 int main(){
4
5     char texto[80] = "Exemplo 1!";
6
7     printf("%13s\n", texto);
8
9     return 0;
10 }
```

- Saída:

```
1   _Exemplo_1!
```


Imprimindo uma String

Lembre-se: o símbolo `'\0'` identifica o final da string.

```
1  #include <stdio.h>
2
3  int main(){
4      char texto[80];
5
6      texto[0] = 'o';
7      texto[1] = 'l';
8      texto[2] = 'á';
9
10     printf("%s\n", texto);
11
12     return 0;
13 }
```

- Pode acontecer de ser impresso uma palavra diferente, como "ola8uj", pois não temos especificado o final da string.

- 1 Strings em C
- 2 Imprimindo uma String
- 3 Leitura de String**
- 4 A biblioteca `string.h`
- 5 A biblioteca `stdlib.h`
- 6 Processamento de Texto
- 7 Referências

Leitura de Strings

Para ler uma **string** com o `scanf()` também usamos o formatador `%s`.

```
1  #include <stdio.h>
2
3  int main(){
4      char texto[80];
5      int idade;
6
7      scanf("%s", texto);
8      scanf("%d", &idade);
9
10     printf("Digitado: %s (%d)\n", texto, idade);
11
12     return 0;
13 }
```

- Note que no `scanf` não colocamos o `&` antes do nome da variável.

Leitura de Strings

O comando `scanf` lê até o primeiro espaço, '`\n`' ou '`\t`'.

```
1 scanf("%s", texto);  
2 scanf("%d", &idade);
```

- Se digitarmos:

```
1 Joao da Silva  
2 20
```

- será salvo apenas "Joao" em `texto`, e um valor diferente de 19 em `idade`.
 - Isto ocorre pois o `scanf` lê a string até o primeiro espaço, e converte o próximo dado (que é a string "da") em um inteiro.

Função fgets

Lendo strings **com espaços** usando a função **fgets**:

```
1 char texto[80];  
2 fgets(texto, 80, stdin);
```

- **primeiro parâmetro**: nome da variável
- **segundo parâmetro**: **tamanho máximo** da string
 - contando com o **'\0'**
- **terceiro parâmetro**: de qual **arquivo** devemos ler
 - **entrada padrão** (teclado) ⇒ **stdin**

Função fgets

A função `fgets` lê apenas até o primeiro `'\n'`

```
1 char texto[80];  
2 fgets(texto, 80, stdin);
```

- incluindo o `'\n'` na string (pode depender da `versão compilador`).

Função fgets

Exemplo:

```
1  #include <stdio.h>
2
3  int main(){
4
5      char texto[80];
6      int idade;
7
8      printf("Entre com o nome e idade:");
9      fgets(texto, 80, stdin);
10     scanf("%d", &idade);
11
12     printf("Digitado: %s(%d)\n", texto, idade);
13
14     return 0;
15 }
```

Função scanf

Existe uma forma de ler uma string com espaços usando `scanf`:

```
1 char texto[80];  
2 scanf("%[^\n]", texto);
```

- lê caracteres enquanto não fizerem parte do conjunto `{ '\n' }`.

Função scanf

Exemplo:

```
1 abacate_abacaxi '\n'
```

- Resultado

Formatador	string
<code>%[^\\n]</code>	abacate abacaxi\\0
<code>%[^e]</code>	abacat\\0
<code>%[abc]</code>	abaca\\0
<code>%s</code>	abacate\\0
<code>%10c</code>	abacate ab

Importante:

- `fgets` é **mais seguro** do que o `scanf` pois o primeiro especifica o **tamanho máximo** da string a ser lida.
- Se um usuário digitar uma string maior do que o vetor declarado, o `scanf` pode **sobreescrever posições inválidas** da memória.
- Existe uma ataque conhecido como **buffer overflow** que explora justamente essa questão.

- 1 Strings em C
- 2 Imprimindo uma String
- 3 Leitura de String
- 4 A biblioteca `string.h`
- 5 A biblioteca `stdlib.h`
- 6 Processamento de Texto
- 7 Referências

A biblioteca `string.h`

A biblioteca `string.h` tem várias funções úteis:

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

`strlen` devolve o tamanho da string

`strcpy` copia uma string

`strcat` concatena duas strings

`strcmp` compara duas strings já que não podemos usar
<, <=, >, >=, == e !=

entre outras...

Veja o manual para a documentação

```
1 $ man strlen
```

⁰Não confunda com a biblioteca `strings.h`

A biblioteca `string.h`

`strlen(s)`: devolve o tamanho da string

- A função recebe **uma string** como parâmetro e devolve o **número de caracteres** até o `'\0'`.

```
1  #include <stdio.h>
2  #include <string.h>
3
4  int main(){
5      char s[80]="ola pessoal";
6
7      printf("%d\n", strlen(s));
8
9      return 0;
10 }
```

Saída:

```
1  11
```

A biblioteca `string.h`

Exemplo:

- Podemos percorrer todos os caracteres de uma string:

```
1 #include <stdio.h>
2 #include <string.h>
3
4 int main(){
5     char s[80]="ola pessoal";
6
7     int i, tam = strlen(s);
8     for(i=0; i <= tam; i++)
9         printf("%d\n", s[i]);
10
11 return 0;
12 }
```

Saída:

```
1 ola pessoal
```

A biblioteca `string.h`

`strcat(s1, s2)`: concatena duas strings.

- A função recebe **duas strings** como parâmetro, e concatena os caracteres de `s2` em `s1` (com o `'\0'` de `s2` no final).

```
1 #include <stdio.h>
2 #include <string.h>
3
4 int main(){
5     char s1[80]="ola ", s2[80]="pessoal";
6
7     //concatena s2 no final de s1
8     strcat(s1, s2);
9
10    printf("%s\n", s1);
11
12    return 0;
13 }
```

Saída:

```
1 ola pessoal
```

A biblioteca `string.h`

Importante:

- se não houver espaço suficiente na **primeira string** ocorrerá um erro.

```
1 #include <stdio.h>
2 #include <string.h>
3
4 int main(){
5     char s1[5]="ola ", s2[80]="pessoal";
6
7     //concatena s2 no final de s1
8     strcat(s1, s2);
9
10    printf("%s\n", s1);
11
12    return 0;
13 }
```

Saída:

```
1 ola *@pessoal
2 Segmentation fault (core dumped)
```


A biblioteca `string.h`

`strncat(s1, s2, n)`: concatena os `n` primeiros caracteres.

- A função recebe **duas strings** como parâmetro e um **inteiro**, e concatena os `n` primeiros caracteres de `s2` em `s1`.

```
1  #include <stdio.h>
2  #include <string.h>
3
4  int main(){
5      char s1[80]="ola ", s2[80]="pessoal";
6
7      //concatena s2 no final de s1
8      strncat(s1, s2, 2);
9      s1[6]='\0'; //precisamos garantir o símbolo '\0'
10
11     printf("%s\n", s1);
12
13     return 0;
14 }
```

Saída:

```
1  ola pe
```

A biblioteca `string.h`

`strcpy(s1, s2)`: copia uma string, incluindo o `'\0'`.

- A função recebe **duas strings** como parâmetro, e copia a **segunda** na string do **primeiro** parâmetro.

```
1 #include <stdio.h>
2 #include <string.h>
3
4 int main(){
5     char s1[80]="oi", s2[80]="ola pessoal";
6
7     strcpy(s1, s2);
8     printf("%s\n", s1);
9
10    return 0;
11 }
```

Saída:

```
1 ola pessoal
```

A biblioteca `string.h`

Importante:

- se não houver espaço suficiente na primeira string ocorrerá um erro.

```
1 #include <stdio.h>
2 #include <string.h>
3
4 int main(){
5     char s1[5], s2[80]="ola pessoal";
6
7     strcpy(s1, s2);
8     printf("%s\n", s1);
9
10    return 0;
11 }
```

Saída:

```
1 ola *@pessoal
2 Segmentation fault (core dumped)
```

A biblioteca `string.h`

`strncpy(s1, s2, n)`: copia os `n` primeiros caracteres.

- A função recebe **duas strings** como parâmetro e um **inteiro**, e copia os `n` primeiros caracteres de `s2` em `s1`.

```
1 #include <stdio.h>
2 #include <string.h>
3
4 int main(){
5     char s1[80], s2[80]="ola pessoal";
6
7     strncpy(s1, s2, 3);
8     s1[3]='\0'; //precisamos garantir o símbolo '\0'
9     printf("%s\n", s1);
10
11 return 0;
12 }
```

Saída:

```
1 ola
```

A biblioteca `string.h`

Comparando (lexicograficamente) `strings`:

- `strcmp`: compara duas strings já que não podemos usar `<`, `<=`, `>`, `>=`, `==` e `!=`

```
1 strcmp(s1, s2); //compara as strings s1 e s2
```

- A função retorna:
 - **0** caso as duas strings sejam iguais.
 - um valor **menor que 0** caso **s1** seja **menor** que **s2**.
 - um valor **maior que 0** caso **s1** seja **maior** que **s2**.

A biblioteca `string.h`

`strcmp(s1, s2)`: compara as strings `s1` e `s2`.

```
1  #include <stdio.h>
2  #include <string.h>
3
4  int main(){
5      char s1[80]="aab", s2[80]="aac";
6      int r;
7      r = strcmp(s1, s2);
8      if(r < 0)
9          printf("%s vem antes que %s\n", s1, s2);
10     else if(r > 0)
11         printf("%s vem antes que %s\n", s2, s1);
12     else // r==0
13         printf("sao iguais\n");
14
15     return 0;
16 }
```

Saída:

```
1  aab vem antes que aac
```

- 1 Strings em C
- 2 Imprimindo uma String
- 3 Leitura de String
- 4 A biblioteca `string.h`
- 5 A biblioteca `stdlib.h`
- 6 Processamento de Texto
- 7 Referências

A biblioteca `stdlib.h`

- Outras funções:

função	conversão
<code>atoi</code>	string para <code>int</code>
<code>atol/strtol</code>	string para <code>long</code>
<code>atoll/strtoll</code>	string para <code>long long</code>
<code>strtof</code>	string para <code>float</code>
<code>strtod</code>	string para <code>double</code>
<code>strtoul</code>	string para <code>unsigned long</code>
<code>strtoull</code>	string para <code>unsigned long long</code>

A biblioteca `stdlib.h`

`atoi(s)`: converte uma string para um inteiro (`int`).

- A função recebe uma string como parâmetro, e retorna um `int`:

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main(){
5
6     char s[]="12345";
7
8     int x = atoi(s);
9
10    printf("%d\n", x);
11
12    return 0;
13 }
```

Saída:

```
1 12345
```

A biblioteca `stdlib.h`

Importante:

- Se não for possível converter a `string`, a função retorna **zero (0)**:

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

Saída:

```
1 0
```

- 1 Strings em C
- 2 Imprimindo uma String
- 3 Leitura de String
- 4 A biblioteca `string.h`
- 5 A biblioteca `stdlib.h`
- 6 Processamento de Texto
- 7 Referências

Como exemplo de uso de strings vamos implementar duas funcionalidades básicas de **processadores de texto**:

- 1 **Contar** o número de palavras em um texto.
- 2 Verifica se uma **palavra ocorre** em um texto.

Contando o número de palavras

Exemplo:

Texto=Lorem ipsum dolor sit amet consectetur adipiscing elit

A resposta é 8.

Contando o número de palavras

```
1  #include <stdio.h>
2
3  int main(){
4      char s[80];
5      int i=0, count=0;
6      fgets(s, 80, stdin);
7
8      while(s[i]!='\n' && s[i] != '\0'){ //Enquanto não terminou s
9
10         while(s[i]==' ') i++; //Pula possíveis espaços
11         //Achou o começo de uma palavra ou o fim do texto
12
13         if(s[i]!='\n' && s[i]!='\0'){ //Se achou uma palavra
14             count++; //incrementa numero de palavras
15             while(s[i]!=' ' && s[i] != '\n' && s[i]!='\0') i++; //passa pela palavra
16         }
17     }
18     printf("Total de palavras: %d\n", count);
19
20     return 0;
21 }
```

¹Textos sem pontuação, sem tabs, etc..

Buscando uma palavra

Exemplo:

Texto=um teste testeste

Palavra=teste

A resposta é 3, 9 e 12.

Buscando uma palavra

Ideia do algoritmo:

- Para cada **posição** no texto onde a palavra pode iniciar, verificamos se a palavra ocorre naquela posição ou não.

Buscando uma palavra

```
1  #include <stdio.h>
2  #include <string.h>
3
4  int main(){
5      char texto[80], palavra[80];
6      printf("Digite o texto: ");
7      fgets(texto, 80, stdin);
8      printf("Digite a palavra: ");
9      fgets(palavra, 80, stdin);
10     int i, j, tamP, tamT;
11     tamP = strlen(palavra)-1;
12     tamT = strlen(texto)-1; //-1 é pelo \n
13     for(i=0; i <= tamT - tamP; i++){//Para cada i
14         j = 0;
15         while(j < tamP){
16             if(palavra[j] != texto[i+j]) break;
17             j++;
18         }
19         if(j==tamP) printf("%d\n", i);
20     }
21     return 0;
22 }
```

Dúvidas?

- 1 Strings em C
- 2 Imprimindo uma String
- 3 Leitura de String
- 4 A biblioteca `string.h`
- 5 A biblioteca `stdlib.h`
- 6 Processamento de Texto
- 7 Referências

- ① Materiais adaptados dos slides do Prof. Eduardo C. Xavier, da Universidade Estadual de Campinas.