

# Strings

Rafael Beserra Gomes

UFRN

Material compilado em 19 de março de 2018.

Licença desta apresentação:



<http://creativecommons.org/licenses/>

# Strings em C

# Strings em C

- **Strings** é o conceito abstrato de uma sequência de caracteres.
- Representação em C: vetor de caracteres ou literais (entre aspas duplas)
- O caractere `'\0'` indica o final da string

# Representação da String na memória

Exemplo de dados na memória:

Endereço									valor	tipo	identificação
0xbffff22c	0	0	0	0	0	1	0	1	5	inteiro curto	valorIndice
0xbffff22d	0	1	0	0	0	0	1	0	B	caractere	letra1
0xbffff22e	0	1	0	0	0	0	1	1	C	caractere	letra2
0xbffff22f	1	1	0	1	1	1	0	1	3.2	real	precoGasolina
0xbffff230	1	1	0	0	1	1	0	0			
0xbffff231	0	1	0	0	1	1	0	0			
0xbffff232	0	1	0	0	0	0	0	0			
0xbffff233	0	1	0	0	1	0	0	1	I	caractere	frase[0]
0xbffff234	0	1	0	0	1	1	0	1	M	caractere	frase[1]
0xbffff235	0	1	0	0	0	1	0	0	D	caractere	frase[2]
0xbffff236	0	0	1	1	0	0	0	0	0	caractere	frase[3]
0xbffff237	0	0	1	1	0	0	0	0	0	caractere	frase[4]
0xbffff238	0	0	1	1	0	0	0	1	1	caractere	frase[5]
0xbffff239	0	0	1	1	0	0	1	0	2	caractere	frase[6]
0xbffff23a	0	0	0	0	0	0	0	0	\0	caractere	frase[7]
0xbffff23b	0	1	1	0	0	1	1	0	f	caractere	frase[8]

# Strings em C

Lembre-se de que os caracteres são codificados como números de acordo com a tabela ascii

```
1 #include <stdio.h>
2
3 int main() {
4
5     char nome[] = "Eu tenho uma casa";
6     int i;
7
8     printf("%s\n", nome);
9
10    nome[15] = 'm';
11
12    printf("%s\n", nome);
13
14    return 0;
15 }
```

## Escrita de strings na tela

# Escrita de strings na tela

- **printf**: use %s no especificador de formato, a escrita é feita até o caractere nulo ('\0')

```
1 #include <stdio.h>
2
3 int main() {
4
5     char frase[300] = "Ouviram do Ipiranga as margens placidas";
6
7     printf("%s\n", frase);
8
9     frase[19] = '\0';
10
11     printf("%s\n", frase);
12
13     return 0;
14 }
```



## Leitura de strings



# Leitura de strings

## ■ Leitura de strings:

```
1 char palavra[100];  
2  
3 scanf("%s", palavra);  
4 printf("%s\n", palavra);
```

-  **cuidado:** lembre-se do espaço para o `\0`, nesse exemplo palavra deve ter no máximo 99 caracteres
-  **cuidado:** não há verificação de buffer overflow (exceder o tamanho da área de armazenamento)

# Leitura de strings

buffer	h	i	n	o	\n		
próximo					↑		

- **Cuidado** com o buffer de entrada! Quando apertamos enter em um scanf, o \n continua lá

```
1 char palavra[100];
2 char t;
3 scanf("%s", palavra);
4 printf("%s\n", palavra);
5 scanf("%c", &t);
6 printf("( %c )\n", t);
```

- getchar() retorna o próximo caractere do buffer de entrada

# Leitura de strings

No scanf há um matching entre o buffer de entrada e a primeira string

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

string do scanf	O	u	v	i	r	a	m
buffer	O	u	v	x	y	z	\n
próximo				↑			

# Leitura de strings

Cada especificador de formato espera algo diferente, podendo consumir vários caracteres do buffer

```
1 #include <stdio.h>
2
3 int main() {
4
5     int h, m, s;
6
7     scanf("%d:%d:%d", &h, &m, &s);
8     printf("%c\n", getchar());
9     printf("%c\n", getchar());
10    printf("%c\n", getchar());
11    printf("%d:%d:%d\n", h, m, s);
12
13    return 0;
14 }
```

string do scanf	%d	:	%d	:	%d
buffer					

# Leitura de strings

## ■ Opções para ler uma string com espaços:

```
1 #include <stdio.h>
2
3 int main() {
4
5     char frase[100];
6     scanf("%[^\n]", frase);
7     printf("%s\n", frase);
8
9     return 0;
10 }
```

- usar `[^\n]` significa consumir qualquer coisa exceto `\n`
- o `\n` continua no buffer
- ⚠ **cuidado:** não há verificação de buffer overflow

# Leitura de strings

## ■ Opções para ler uma string com espaços:

```
1 #include <stdio.h>
2
3 int main() {
4
5     char frase[100];
6     gets(frase);
7     printf("%s\n", frase);
8
9     return 0;
10 }
```

- consome o `\n`, substituindo-o pelo `\0`
- 💣 **cuidado:** não há verificação de buffer overflow

# Leitura de strings

## ■ Opções para ler uma string com espaços:

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

- lê no máximo o segundo argumento - 1 caracteres da entrada (evita buffer overflow)
- o que ultrapassar esse máximo, continua no buffer de entrada
- 💣 **cuidado:** fgets pode armazenar o \n na string!
- **stdin** é uma macro definida em stdio.h, veremos mais na frente no tópico de **arquivos**

## Percorrendo uma string



# Percorrendo uma string

- A função `strlen` (de `string.h`) retorna a quantidade de caracteres
- Ex: ler uma string e trocar todos os espaços por -

```
1 #include <stdio.h>
2 #include <string.h>
3
4 int main() {
5
6     char frase[100];
7     int i;
8
9     fgets(frase, 100, stdin);
10
11     for(i = 0; i < strlen(frase); i++) {
12         if(frase[i] == ' ') {
13             frase[i] = '-';
14         }
15     }
16     printf("%s\n", frase);
17
18     return 0;
19 }
```

# Percorrendo uma string

## ■ Resolvendo o problema do possível \n no fgets

```
1 #include <stdio.h>
2
3 int main() {
4
5     char frase[5];
6     fgets(frase, 5, stdin);
7     if(frase[strlen(frase)-1] == '\n') {
8         frase[strlen(frase)-1] = '\0';
9     }
10    printf("%s\n", frase);
11
12    return 0;
13 }
```



## Exercício em sala

Uma das regras mais conhecidas da ortografia no português é que 'n' não deve preceder 'p' e 'b'. Escreva um programa que leia uma string de até 20 caracteres e realize o conserto de acordo com essa regra. Por exemplo, a string “inpedido no canto do canpo” é transformada em “impedido no canto do campo”.



## Exercício em sala

Escreva um programa para ler uma string qualquer e, em seguida, escrever a mesma string em caixa alta.