Strings

Prof. Fernando Figueira
(adaptado do material do Prof. Rafael Beserra Gomes)

UFRN

Material compilado em 8 de outubro de 2025.

Strings em C

Strings em C

- **Strings**: conceito abstrato de uma sequência de caracteres
- Representação em C: vetor de caracteres ou literais (entre aspas duplas)

```
1 char palavra[100] //string como vetor de caracteres
2 "esta eh string literal" //string como literal
```

- caractere '\0' indica o final da string, é adicionado automaticamente na string literal

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	4.6	**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**	palavra[0]
0xbffff234	0	1	0	0	1	1	0	1	'M'	**caractere**	palavra[1]
0xbffff235	0	1	0	0	0	1	0	0	'D'	**caractere**	palavra[2]
0xbffff236	0	0	0	0	0	0	0	0	'\0'	**caractere**	palavra[3]
0xbffff237	0	1	1	0	0	1	1	0	'f'	**caractere**	palavra[4]
0xbffff238	0	0	0	0	0	1	0	1	5	**inteiro curto**	valorIndice

Declaração e inicialização:

```c
#include <stdio.h>

int main() {

    char palavra1[] = {'I', 'M', 'D'};
    char palavra2[10] = "IMD";

    palavra2 = "alo"; //nao funciona
    palavra2[0] = 'a';
    palavra2[1] = 'l';
    palavra2[2] = 'o';
    palavra2[3] = '\0';
    return 0;
}
```

Depois da declaração, atribuições devem ser caractere a caractere

Equivalência char $\leftrightarrow$ int:

Maiúsculas:

$'A' \leftrightarrow 65$

$'B' \leftrightarrow 66$

$'C' \leftrightarrow 67$

$\vdots$

$'Z' \leftrightarrow 90$

Minúsculas:

$'a' \leftrightarrow 97$

$'b' \leftrightarrow 98$

$'c' \leftrightarrow 99$

$\vdots$

$'z' \leftrightarrow 122$

Consulte **man ascii**

Oct	Dec	Hex	Char	Oct	Dec	Hex	Char
000	0	00	NUL '\0' (null character)	100	64	40	@
001	1	01	SOH (start of heading)	101	65	41	A
002	2	02	STX (start of text)	102	66	42	B
003	3	03	ETX (end of text)	103	67	43	C
004	4	04	EOT (end of transmission)	104	68	44	D
005	5	05	ENQ (enquiry)	105	69	45	E
006	6	06	ACK (acknowledge)	106	70	46	F
007	7	07	BEL '\a' (bell)	107	71	47	G

Equivalência char $\leftrightarrow$ int:
$'s' \leftrightarrow 115$
$'t' \leftrightarrow 116$
$'o' \leftrightarrow 111$

```c
#include <stdio.h>

int main() {

    char palavra[] = "asa";

    palavra[1] += 1;
    printf("%s\n", palavra); //escreve ata
    palavra[2] = 111;
    printf("%s\n", palavra); //escreve ato

    return 0;
}
```

Escrita de strings na tela

Escrita de strings na tela

Ao contrário de outros tipos, um **vetor de char** pode ser escrito no 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] = "IMD 2018";
6
7      printf("%s\n", frase);
8
9      frase[5] = '\0';
10
11     printf("%s\n", frase);
12
13     return 0;
14 }
```

Leitura de strings

```
1 char palavra[5];
2
3 scanf("%s", palavra);
4 gets(frase);
5 fgets(frase, 5, stdin);
```

- scanf
  - pode ocorrer **buffer overflow**
  - interrompe leitura no **enter ou espaço**
- gets
  - pode ocorrer **buffer overflow**
  - interrompe leitura no **enter**
- fgets
  - lê no máximo o segundo argumento - 1 caracteres da entrada (evita **buffer overflow**)
  - armazena o \n na string (caso caiba!)
  - interrompe leitura no **enter**

Na disciplina:

- **palavra(x)**: string sem espaços com até x caracteres → **scanf**
- **frase(x)**: string com espaços com até x caracteres → **gets**
- o gcc emite um warning sobre **buffer overflow** ao usar **gets**
- os casos de testes obedecem aos limites estabelecidos

Percorrendo uma string

Percorrendo uma string

- A função strlen (de string.h) retorna a quantidade de caracteres
- Ex: ler uma frase(100) e trocar todos os espaços por -

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

✎ 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 frase(20) 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 frase(100) e, em seguida, escrever a mesma string em caixa alta.

Funções para Strings em C

Principais funções da biblioteca string.h

strlen

```c
#include <stdio.h>
#include <string.h>

int main() {

    char str1[] = "Aviao";
    char str2[] = "Carro";

    printf("Tamanho de str1: %d\n", strlen(str1));
    printf("Tamanho de str2: %d\n", strlen(str2));

    return 0;
}
```

strcpy

```c
#include <stdio.h>
#include <string.h>

int main() {

    char str1[] = "Aviao";
    char str2[100] = "Carro";

    printf("str1: %s\n", str1);
    printf("str2: %s\n", str2);
    strcpy(str2, str1);
    printf("str1: %s\n", str1);
    printf("str2: %s\n", str2);

    return 0;
}
```

strncpy

```
1  #include <stdio.h>
2  #include <string.h>
3
4  int main() {
5
6      char str1[] = "Aviao";
7      char str2[100] = "Carro";
8
9      printf("str1: %s\n", str1);
10     printf("str2: %s\n", str2);
11     strncpy(str2, str1, 3);
12     str2[3] = '\0';
13     printf("str1: %s\n", str1);
14     printf("str2: %s\n", str2);
15
16     return 0;
17 }
```

strcmp

```c
#include <stdio.h>
#include <string.h>

int main() {

    char str1[100];
    char str2[100];

    printf("Digite duas palavras: ");
    scanf("%s %s", str1, str2);
    printf("Resultado da comparacao: %d\n", strcmp(str1, str2));

    return 0;
}
```