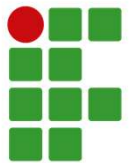


# Caractere e String



**INSTITUTO FEDERAL**  
Sul-rio-grandense  
Câmpus Pelotas





# Tipo caractere

Utilizado para armazenar uma **letra**, **dígito** ou **símbolo**.

```
#include <stdio.h>
main(){
    char letra, digito, simbolo;
```

Especifica o tipo caractere

```
    letra = 'A';
```

```
    digito = '1';
```

```
    simbolo = '!';
```

Constante caractere

```
    printf("Letra: %c\n", letra);
```

```
    printf("Dígito: %c\n", digito);
```

```
    printf("Símbolo: %c\n", simbolo);
```

```
}
```

Código de formatação para caractere



# Tipo caractere - detalhes

---

**%c** - Código de formatação para caractere

**'A'** - Constante caractere

## Comandos para **leitura** de caracteres

```
ch = getchar();
```

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

## Comandos para **escrita** de caracteres

```
putchar(ch);
```

```
printf("%c", ch);
```

# Exemplos de utilização

---



```
#include <stdio.h>
main(){
    char carac;

    printf("Digite um caractere: ");
    carac = getchar();
    printf("Caractere digitado: %c\n", carac);
}
```

```
#include <stdio.h>
main(){
    char carac;

    printf("Digite um caractere: ");
    scanf("%c", &carac);
    putchar(carac);
}
```



# Problemas com o scanf

**Contexto:** Ao utilizar o scanf para ler caracteres isolados em C, podem ocorrer problemas devido à presença de caracteres de nova linha (\n) no buffer de entrada.

**Como funciona:** scanf("%c", &c); tenta ler o próximo caractere do buffer de entrada.

```
main() {  
    int num;  
    char c;  
    printf("Digite um número: ");  
    scanf("%d", &num);  
    printf("Digite um caractere: ");  
    scanf("%c", &c); // Pode ler um caractere de nova linha residual  
    printf("O caractere digitado foi: %c\n", c);  
}
```

**Problema:** Se houver um caractere de nova linha (\n) residual no buffer (por exemplo, após pressionar Enter), scanf pode interpretar esse caractere de nova linha como parte da próxima entrada.

**O que pode acontecer?** Após inserir o número e pressionar Enter, o \n fica no *buffer*. scanf("%c", &c); pode ler esse \n em vez do caractere esperado.



# Como solucionar esse problema do scanf?

Utilizar um espaço antes do %c no scanf para ignorar caracteres de espaço em branco, incluindo novas linhas.

```
main() {  
    int num;  
    char c;  
    printf("Digite um número: ");  
    scanf("%d", &num);  
    printf("Digite um caractere: ");  
  
    scanf(" %c", &c); // O espaço antes de %c ignora espaços em branco e novas linhas  
  
    printf("O caractere digitado foi: %c\n", c);  
}
```



# O mesmo acontece com o `getchar()`

**Contexto:** função em C usada para ler um único caractere da entrada padrão (teclado).

Assim como `scanf`, `getchar()` também pode encontrar problemas devido a caracteres de nova linha (`\n`) no *buffer* de entrada.

**Como funciona?** `getchar()` lê o próximo caractere do buffer de entrada.

```
main() {  
    int num;  
    char c;  
    printf("Digite um número: ");  
    scanf("%d", &num);  
    printf("Digite um caractere: ");  
    c = getchar(); // Pode ler o caractere de nova linha residual  
    printf("O caractere digitado foi: %c\n", c);  
}
```

**Problema:** Se houver um caractere de nova linha (`\n`) residual no buffer (por exemplo, após pressionar Enter), `getchar()` pode interpretar esse caractere de nova linha como parte da próxima entrada.

**O que pode acontecer?** Após inserir o número e pressionar Enter, o `\n` fica no *buffer*. `getchar()` pode ler esse `\n` em vez do caractere esperado.



# Como resolver o problema com o getchar?

Utilizar Utilize `getchar()` adicional para consumir o caractere de nova linha residual antes de ler o caractere desejado.

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

```
main() {  
    int num;  
    char c;  
  
    printf("Digite um número: ");  
    scanf("%d", &num);  
  
    // Consumir o newline residual deixado no buffer pelo scanf  
    while (getchar() != '\n');  
  
    printf("Digite um caractere: ");  
    c = getchar(); // Ler o próximo caractere do usuário  
  
    printf("O caractere digitado foi: %c\n", c);  
}
```

`while (getchar() != '\n');` é uma técnica para limpar o buffer de entrada após uma leitura com `scanf`, removendo qualquer caractere residual (especialmente `newline`) que possa interferir nas leituras subsequentes.



# Sintaxe e Propósito



## getchar / putchar:

- Funções projetadas especificamente para trabalhar com caracteres individuais.

## Exemplo:

```
char ch;  
ch = getchar();    // Lê um caractere  
putchar(ch);       // Exibe um caractere
```

## scanf / printf:

- Funções mais gerais que lidam com diferentes tipos de dados, incluindo caracteres.

## Exemplo:

```
char ch;  
scanf("%c", &ch);    // Lê um caractere  
printf("%c", ch);     // Exibe um caractere
```



# Quadro comparativo

Critério	<code>getchar</code> / <code>putchar</code>	<code>scanf("%c")</code> / <code>printf("%c")</code>
<b>Simplicidade</b>	Simples e direto para trabalhar com caracteres individuais.	Exige especificadores de formato ( <code>%c</code> ), mais verboso.
<b>Flexibilidade</b>	Limitado a caracteres individuais.	Flexível para múltiplos tipos de dados e formatação.
<b>Controle do Buffer</b>	Lê diretamente do buffer de entrada (incluindo <code>\n</code> ).	Pode deixar caracteres no buffer, como <code>\n</code> (exige cuidado).
<b>Desempenho</b>	Mais rápido, pois não tem sobrecarga de formatação.	Um pouco mais lento devido à flexibilidade extra.
<b>Facilidade de Uso</b>	Ideal para tarefas simples.	Melhor para entradas e saídas mais complexas.
<b>Exemplo de Uso</b>	<pre>c&lt;br&gt;char ch = getchar(); &lt;br&gt;putchar(ch);</pre>	<pre>c&lt;br&gt;char ch;&lt;br&gt;scanf("%c", &amp;ch); &lt;br&gt;printf("%c", ch);</pre>

# Quando usar?

---



Critério	<code>getchar</code> / <code>putchar</code>	<code>scanf</code> / <code>printf</code>
Quando Usar	- Entrada e saída simples de caracteres.	- Manipulação de múltiplos dados ou formatação complexa.
	- Código que prioriza simplicidade e desempenho.	- Situações que exigem maior flexibilidade e controle.
Conclusão	- Ideal para operações básicas com caracteres.	- Melhor para entradas e saídas mais completas e formatadas.

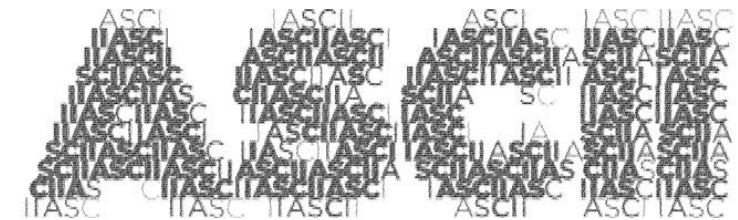
# Código ASCII

---



## Código ASCII (*American Standard Code for Information Interchange*)

Caractere	Código ASCII	Caractere	Código ASCII
'0'	48	.	
'1'	49	.	
.		'a'	97
.		'b'	98
'9'	57	.	
.		.	
.		'z'	122
'A'	65	.	
'B'	66	.	
.			
.			
'Z'	90		
.			
.			



Código ASCII é baseado no alfabeto romano e sua função é padronizar a forma como os computadores representam letras, números, acentos, sinais diversos e alguns códigos de controle.

# Código 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]

O ASCII é um código que foi proposto por Robert W. Bemer como **uma solução para unificar a representação de caracteres alfanuméricos em computadores.**

Antes de 1960 cada computador utilizava uma regra diferente para representar estes caracteres e o **código ASCII nasceu para se tornar comum entre todas as máquinas.**

No ASCII existem **95 caracteres que podem ser impressos.**

Eles são numerados de **32 a 126** sendo os caracteres de 0 a 31 reservados para funções de controle.

O código ASCII é um padrão de codificação de caracteres que **atribui números inteiros a letras, dígitos, sinais de pontuação e outros caracteres.**

No C, o uso de caracteres ASCII é bastante comum, pois o C trata caracteres como valores inteiros de acordo com a tabela ASCII.



# Determinando o próximo caractere

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

```
main(){
```

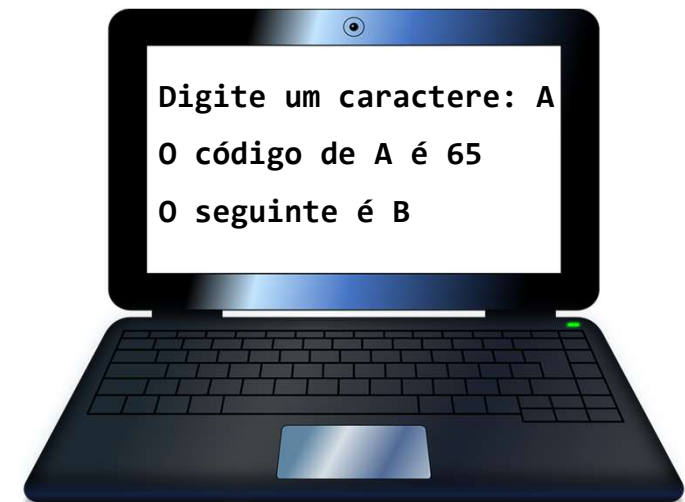
```
    char ch, x;
```

```
    printf("Digite um caractere: ");  
    scanf("%c", &ch);
```

```
    printf("O código de %c é %d\n", ch, ch);
```

```
    x = ch + 1;  
    printf("O seguinte é %c\n", x);
```

```
}
```



# Comparação de caracteres



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

```
main() {
```

```
    char x, y;
```

```
    x = 'A';
```

```
    y = 'B';
```

```
    if (x > y)
```

```
        printf("%c é maior que %c\n", x, y);
```

```
    else
```

```
        printf("%c não é maior que %c\n", x, y);
```

```
}
```

A não é maior que B

A: 65

B: 66

Utiliza a tabela ASCII  
para a comparação.





# Verificando se é uma letra maiúscula

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

```
main() {
```

```
    char ch;
```

```
    printf("Digite uma letra: ");
```

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

```
    if (ch>='A' && ch<='Z')
```

```
        printf("%c é uma letra maiúscula\n", ch);
```

```
    else
```

```
        printf("%c não é uma letra maiúscula\n", ch);
```

```
}
```

Digite uma letra: R

R é uma letra maiúscula.

**PROBLEMA:** Escreva um programa para ler um inteiro N. A seguir ler N caracteres. Escrever cada caractere lido com o seu respectivo sucessor.



```
#include <stdio.h>
main() {
    int i, n;
    char x, y;

    printf("Informe a quantidade de caracteres:");
    scanf("%d", &n);

    for (i=1; i<=n; i++) {
        printf("Informe o caractere:");
        scanf("%c", &x);
        y = x + 1;
        printf("%c --> %c \n", x, y);
    }
}
```

# Strings

# String



## Constante **string**

`"IF-SUL"`

`"Um exemplo de string"`

`"A"`

`" "`

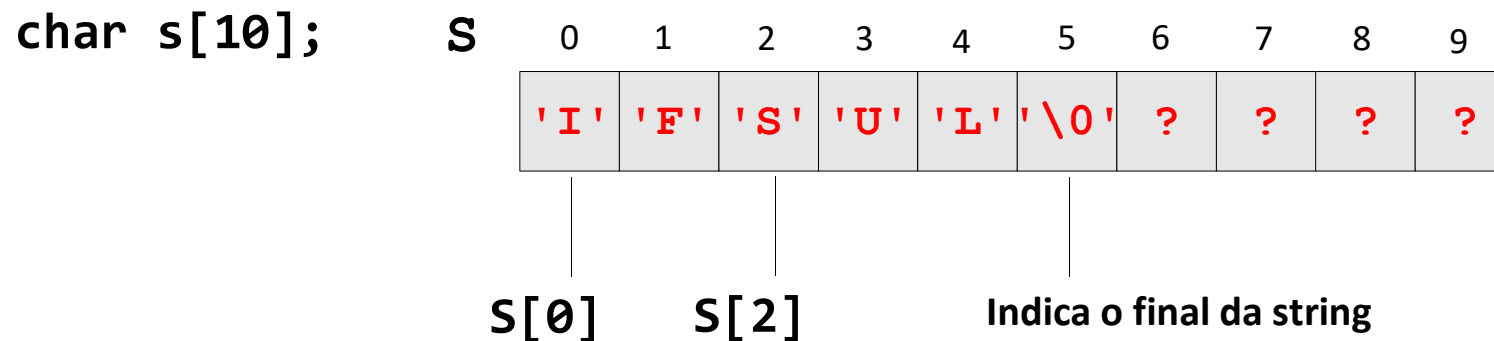
Uma string é armazenada  
em um **vetor de caracteres**.

**OBS:**

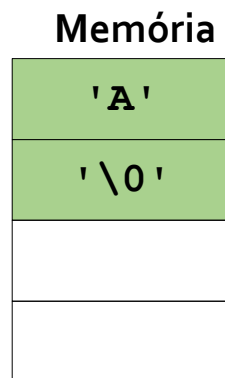
<code>'A'</code>	é diferente de	<code>"A"</code>
Caractere		String



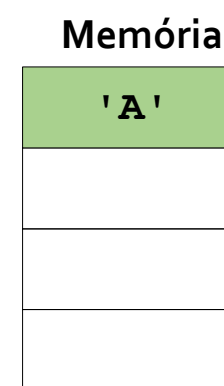
# Uma String na memória



OBS: Deve ser previsto espaço para o terminador da string.



"A" é diferente de 'A'





# Leitura e escrita de String

## Leitura

`scanf("%s", s);` ←

Não faz a leitura de espaços em branco.

`gets(s);`

Aceita espaços em branco.

A função receber apenas **nome do vetor**.

Não é preciso usar o **&** no **scanf**.

## Escrita

`printf("%s", s);` ←

`puts(s);`



# Leitura e escrita de String

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

```
main(){
```

```
    char nome[30];
```

Permite o armazenamento de uma **string** de até 29 caracteres.

```
    printf("Informe o seu nome: ");
```

```
    scanf("%s", nome);
```

```
    printf("Olá %s\n", nome);
```

Apenas o **nome do vetor**

```
}
```

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

```
main(){
```

```
    char nome[30];
```

```
    printf("Informe o seu nome: ");
```

```
    gets(nome);
```

Usando o **gets**.

```
    printf("Olá %s\n", nome);
```

```
}
```

## E a leitura de strings? Acontece o mesmo que ocorria com caracteres?



Sim, o mesmo princípio que acontecia ao se ler um caractere, se aplica a leitura de strings usando funções como `gets`, `fgets` ou `scanf` com o especificador de formato `%s`.

Se houver um caractere de nova linha (`\n`) deixado no buffer após uma entrada anterior, ele pode afetar a leitura subsequente de strings.

Para evitar problemas, você pode seguir a mesma abordagem de adicionar um espaço em branco na string de controle do `scanf` ou, no caso do `gets`, incluir um `getchar()` adicional para consumir o caractere de nova linha.



# Problemas com scanf("%s") e gets()



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

```
main(){  
    char nome[30];  
  
    printf("Informe o seu nome: ");  
    scanf("%s", nome);  
    printf("Olá %s\n", nome);  
}
```

## scanf("%s")

### Como funciona:

- Lê uma string até encontrar um espaço em branco (espaço, tabulação ou nova linha).

### Problemas:

- **Inseguro:** Não verifica o tamanho do buffer, o que pode levar a estouro de buffer se a entrada for maior do que o tamanho do buffer.
- **Limitação:** Não pode ler strings com espaços em branco.

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

```
main(){  
    char nome[30];  
  
    printf("Informe o seu nome: ");  
    gets(nome);  
    printf("Olá %s\n", nome);  
}
```

## gets()

### Como funciona:

- Lê uma linha inteira de entrada até encontrar um nova linha (\n).

### Problemas:

- **Muito inseguro:** Não verifica o tamanho do buffer, o que pode causar estouros de buffer.
- **Descontinuado:** Foi removido do padrão C11 devido aos problemas de segurança.

# fgets



Função utilizada para ler uma linha de texto de um arquivo ou do stdin (entrada padrão).

Útil para capturar strings de maneira segura, evitando problemas comuns com o uso de funções como `gets`, que pode causar estouro de buffer.

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

Ponteiro para array de caracteres  
onde a string lida será armazenada.

Número máximo de caracteres a serem  
lidos (incluindo o caractere nulo \0)

Ponteiro para o fluxo de entrada,  
que pode ser um arquivo ou stdin.

# fgets - funcionamento

---



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

- fgets lê até n-1 caracteres da entrada especificada (stream) e os armazena no array str.
- A leitura para quando:
  - a) um *newline* (\n) é encontrado;
  - b) quando n-1 caracteres são lidos; ou
  - c) quando o fim do arquivo (EOF – *end of file*) é alcançado, **o que ocorrer primeiro**.
- O caractere lido de *newline* (\n) também é armazenado no array str.
- Um caractere nulo (\0) é automaticamente adicionado ao final da string lida para garantir que seja uma string C válida.



# fgets - exemplo

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

```
main() {  
    char buffer[100];
```

```
    printf("Digite uma linha de texto: ");
```

```
    if (fgets(buffer, sizeof(buffer), stdin) != NULL) {  
        printf("Você digitou: %s", buffer);
```

```
    } else {  
        printf("Erro ao ler a linha de texto.\n");  
    }
```

```
}
```

Array onde a string será armazenada.

Lê uma linha da entrada padrão (stdin) e armazena no buffer.

Se a leitura for bem-sucedida, a função retorna o ponteiro 'buffer', caso contrário, retorna NULL/

# Manipulação de Strings

---

# Problema



Ler uma palavra. Escrever cada letra em uma linha.

```
#include <stdio.h>
main(){
    char palavra[20];
    int i;

    printf("Informe uma palavra:");
    scanf("%s", palavra);

    for (i=0; palavra[i] != '\0'; i++)
        printf("%c\n", palavra[i]);
}
```





# Funções para manipulação de Strings

Utilizam o arquivo **"string.h"**

**strlen(string);** Retorna a quantidade de caracteres de uma String.

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

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

```
main(){
```

```
    char palavra[20];
```

```
    int tam;
```

```
    printf("Informe uma palavra:");
```

```
    scanf("%s",palavra);
```

```
    tam = strlen(palavra);
```

```
    printf("Quantidade de caracteres: %d\n",tam);
```

```
}
```

Pelotas





# Funções para manipulação de strings

**strcpy(destino, fonte);** Atribuição de strings. A string destino deve possuir espaço para armazenar a string fonte.

```
#include <stdio.h>
#include <string.h>
main(){
    char a[20], b[20];

    strcpy(a, "IFSul");
    strcpy(b, a);
    printf("%s %s\n", a, b);
}
```







# Funções para manipulação de strings

---

## Comparação de strings

Qual o critério utilizado para comparar strings?

Marque cada comparação com **V** ou **F**

**"amora" < "uva"**

**"amora" < "Uva"**

**"uva" < "uvas"**

**" teste" > "amora"**

**"121" > "9"**

# Funções para manipulação de strings



"amora" < "uva"



0	1	2	3	4	5	6	7
'a'	'm'	'o'	'r'	'a'	'\0'	?	?


0	1	2	3	4	5	6	7
97	109	111	114	97	0	?	?

0	1	2	3	4	5	6	7
'u'	'v'	'a'	'\0'	?	?	?	?

0	1	2	3	4	5	6	7
117	118	97	0	?	?	?	?



# Funções para manipulação de strings

"amora" < "Uva" 

0	1	2	3	4	5	6	7
'a'	'm'	'o'	'r'	'a'	'\0'	?	?

0	1	2	3	4	5	6	7
97	109	111	114	97	0	?	?

0	1	2	3	4	5	6	7
'U'	'v'	'a'	'\0'	?	?	?	?

0	1	2	3	4	5	6	7
85	118	97	0	?	?	?	?

# Funções para manipulação de strings



"uva" < "uvas"



0	1	2	3	4	5	6	7
'u'	'v'	'a'	'\0'	?	?	?	?

0	1	2	3	4	5	6	7
117	118	97	0	?	?	?	?

0	1	2	3	4	5	6	7
'u'	'v'	'a'	's'	'\0'	?	?	?

0	1	2	3	4	5	6	7
117	118	97	115	0	?	?	?



# Funções para manipulação de strings

" teste" > "amora"



0	1	2	3	4	5	6	7
' '	't'	'e'	's'	't'	'e'	'\0'	?

0	1	2	3	4	5	6	7
32	116	101	115	116	101	0	?

0	1	2	3	4	5	6	7
'a'	'm'	'o'	'r'	'a'	'\0'	?	?

0	1	2	3	4	5	6	7
97	109	111	114	97	0	?	?

# Funções para manipulação de strings



"121" > "9"



0	1	2	3	4	5	6	7
'1'	'2'	'1'	'\0'	?	?	?	?

0	1	2	3	4	5	6	7
49	50	49	0	?	?	?	?

0	1	2	3	4	5	6	7
'9'	'\0'	?	?	?	?	?	?

0	1	2	3	4	5	6	7
57	0	?	?	?	?	?	?



# Funções para manipulação de strings

---

**Não é possível comparar 2 strings diretamente com os operadores relacionais.**

Deve ser utilizada a função **strcmp** que retorna um inteiro conforme especificado.

**strcmp(stringA, stringB);**

Compara duas strings. Retorna um dos seguintes códigos:

**0** se as strings são **iguais**.

**> 0** (positivo) se a stringA é **maior** que a stringB.

**< 0** (negativo) se a stringA é **menor** que a stringB.

# Funções para manipulação de strings



```
#include <stdio.h>
#include <string.h>
main(){
    char a[30],b[30];
    int cod;

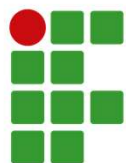
    printf("Digite uma palavra: ");
    scanf("%s",a);
    printf("Digite outra: ");
    scanf("%s",b);

    cod = strcmp(a, b);

    if (cod==0)
        printf("Iguais\n");
    else
        if (cod > 0)
            printf("%s > %s\n",a,b);
        else
            printf("%s < %s\n",a,b);
}
```



# Caractere e String



30A19B87