



UNIVERSIDADE  
FUMEC

# Estrutura de Dados

```
1 #include <stdio.h>
2 #include <string.h>
3 int main () {
4     char origem[100],
5         destino[100];
6     printf ("Digite o nome: ");
7     fgets (origem, 100, stdin);
8
9     printf ("Digite o sobrenome: ");
10    fgets (destino, 100, stdin);
11
12    strcat (destino, origem, 5);
13    printf ("\nO nome é %s.", destino);
14    return 0;
15 }
```

## MANIPULAÇÃO DE STRINGS

## APRESENTAÇÃO

---

**P**rezado(a) aluno(a), a partir de agora falaremos sobre manipulação de *strings*. Você descobrirá seu conceito, como declará-la e utilizá-la em seus códigos.

Conhecerá também algumas funções que permitem manipulação nas *strings* declaradas.

---

## OBJETIVOS DE APRENDIZAGEM

Ao final desse módulo você deverá ser capaz de:

- Declarar e utilizar uma *string*;
- Diferenciar uma *string* de um caractere e de um vetor de caracteres comum;
- Aplicar funções da biblioteca `string.h` em seu código.



# MANIPULAÇÃO DE STRINGS

Em computação, *strings* são compostas por conteúdos alfanuméricos com mais de uma posição, ou seja, é um conjunto de caracteres. Mas você sabe exatamente o que isto significa?



Para compreendermos melhor, antes de falarmos sobre as *strings*, primeiro falaremos sobre os caracteres comuns.

Para armazenarmos um caractere na linguagem C, utilizamos o tipo `char`, neste tipo você pode armazenar letras, números, caracteres especiais, etc.

Uma coisa importante, que você deve saber, é que ao receber o valor alfanumérico, a linguagem também armazena seu valor decimal, ou seja, o valor correspondente na Tabela ASCII, assim você poderá acessar qualquer um destes conteúdos durante o código.

Veja o seguinte exemplo:

```
1 #include <stdio.h>
2 int main () {
3     char letra = 'A';
4     printf ("A letra é %c e seu valor decimal é %d.", letra , letra );
5     return 0;
6 }
```

Nó código você pode perceber que, no valor de retorno, foi utilizada a mesma variável, sem nenhuma alteração, porém, a *string* de controle foi `%c` para imprimir o caractere, neste caso 'A', e depois `%d` para imprimir o decimal, neste caso 65, que corresponde a letra A na tabela ASCII.



Agora que já conhecemos a forma de utilização de um variável caractere, vamos voltar a falar sobre *strings*!

Na linguagem C, não existe o tipo de variável *string*, como conhecemos em outras linguagens. Esta linguagem não possui isto como definição, por isso, para declararmos uma *string*, temos que, na verdade, declarar um vetor de caracteres.

```
char vetorString[10] = "teste";
```

Veja que esta é uma declaração de vetor comum, ou seja, no caso você está declarando um vetor que armazena até 10 caracteres.



Mas como saberemos que este vetor é uma *string*? Afinal, este vetor poderia servir para inserção de caracteres comuns, sem necessariamente ser uma *string*.

Para te explicar isto, vou primeiro te explicar sobre o caractere `\0`, ele, na linguagem C, corresponde ao nulo (NULL), e em um vetor de caracteres indica o final de uma *string*. Esta é a forma que a linguagem utiliza para diferenciar vetores de caracteres comuns de *strings*.

Portanto, ao final da frase, ou palavra, basta incluir o caractere `\0`, e com esta definição, "procurar" o caractere nulo para saber que a *string* acabou, mesmo que não utilize todos os *bytes* separados para ela.

Para a declaração da *string* `vetorString` o preenchimento da memória seria:

```
char vetorString[10]
```

0	1	2	3	4	5	6	7	8	9
t	e	s	t	e	\0				

Pode ser que a sua *string* tenha vários `\0`, mas o código sempre procurará o primeiro, para determinar o final da sua *string*.

```
char vetorString[10]
```

0	1	2	3	4	5	6	7	8	9
o	l	á	\0	e	\0				

Para este exemplo, os caracteres que vierem depois da posição 4 (índice 3) são considerados "lixos", mesmo que exista outros `\0` estes serão ignorados, e apenas o primeiro será considerado terminador válido da *string*.





## Inicialização de uma string

Durante a declaração de uma *string* é possível inicializá-la, mas atenção, isto deve ser feito apenas na declaração e utilizando aspas duplas (""), indicando que corresponde a uma *string*.

Para estes casos o terminador \0 já será inserido automaticamente.

Para os casos onde precisaremos receber os valores do usuário, utilizaremos funções de entrada.

## FUNÇÃO DE ENTRADA PARA STRINGS

Para entrada de *strings*, o ideal é utilizarmos outra função (ao invés do scanf), que permitirá um tratamento mais eficiente.

Utilizaremos então a função fgets.

A sintaxe desta função é:

```
fgets(<nome da variável>, <tamanho da variável>, stdin);
```

Onde o:

- **Nome da variável** corresponde ao nome do vetor de *strings*
- **Tamanho da variável** corresponde ao tamanho máximo permitido para inclusão (aquele que foi predefinido em sua declaração de vetor).
- **Stdin** que corresponde ao espaço de memória (*buffer*) utilizado.

Veja uma aplicação desta função:

```
1 #include <stdio.h>
2 int main () {
3     char str[100];
4     printf ("\nDigite uma string: ");
5     fgets (str , 100 , stdin);
6     printf ("\nVocê digitou %s. ", str);
7     return 0;
8 }
```

A função fgets finaliza a *string* incluindo o caractere \0.



## FUNÇÃO DE SAÍDA PARA STRINGS

Para imprimir o conteúdo de um *string* você pode utilizar o mesmo comando das anteriores, que será o `printf`. Basta utilizar a *string* de controle `%s`.

Por se tratar de um vetor, você pode acessar qualquer caractere dele, bastando indicar o índice, mas ao acessar o índice este conteúdo será utilizado como caractere comum.



Sabendo que *strings* possuem tratamentos especiais, já existe uma biblioteca com algumas funções preparadas para manipulação deste conteúdo. Vamos conhecer?

## FUNÇÕES DA BIBLIOTECA `STRING.H`

Já existem algumas funções que nos permite manipularmos *strings* na linguagem C, elas pertencem a biblioteca `string.h` e, a partir de agora, veremos algumas.

### Strlen

A primeira que conheceremos é a função `strlen`, que informa a quantidade de caracteres válidos em uma *string*.

Ela retorna um valor inteiro, com a quantidade de caracteres existentes na *string*, anteriores ao primeiro `\0` encontrado.

A sintaxe desta função é:

```
int strlen(<nome da variável>);
```

Veja uma aplicação desta função:

```
1 #include <stdio.h>
2 #include <string.h>
3 int main () {
4     char str[100];
5     printf ("\nDigite uma string: ");
6     fgets (str, 100, stdin);
7     printf ("\nA string %s possui %d caracteres. ", str, strlen(str));
8     return 0;
9 }
```

Como o retorno é inteiro, poderá ser utilizado para preencher variável, imprimir no `printf` ou passagem de parâmetro de uma função. Tudo o que você faria com uma variável inteira, você pode fazer com o retorno desta função.

### Strcpy

A função `strcpy` efetua a cópia de uma variável para outra.

Ao copiar, a variável destino será substituída pela variável origem, portanto, ao utilizar esta função, você perderá o conteúdo originalmente existente na *string* de destino.

Em sua sintaxe ele não possui retorno, a alteração será feita diretamente em um dos parâmetros.

Possui dois parâmetros que efetua a cópia da variável de origem para a variável de destino.

A sintaxe é:

```
void strcpy(<variável de destino>, <variável de origem>);
```



## ATENÇÃO

Não se esqueça de que, o conteúdo da variável de destino será completamente substituído.



Veja uma aplicação desta função:

```
1 #include <stdio.h>
2 #include <string.h>
3 int main () {
4     char origem[100],
5         destino[100];
6     printf ("\nDigite uma string : ");
7     fgets (origem, 100 , stdin);
8     strcpy ( destino , origem );
9     printf ("\nO nome é %s. ", destino );
10    return 0;
11 }
```

## Strncpy

A função strncpy é análoga à função strcpy, porém, ao invés de copiar totalmente a variável origem para a variável destino, ela permite determinar quantos caracteres serão copiados.

Em sua sintaxe ele não possui retorno, a alteração será feita diretamente em um dos parâmetros.

Possui três parâmetros que efetua a cópia da variável de origem para a variável de destino, na quantidade indicada.

A sintaxe é:

```
void strncpy(<variável de destino>, <variável de origem>, <quantidade>);
```

## ATENÇÃO

O conteúdo da variável de destino será completamente substituído, na quantidade de caracteres indicados.



```
1 #include <stdio.h>
2 #include <string.h>
3 int main () {
4     char origem[100],
5         destino[100];
6     printf ("\nDigite uma string : ");
7     fgets (origem, 100 , stdin);
8     strncpy (destino , origem , 5);
9     printf ("\nO nome é %s. ", destino);
10    return 0;
11 }
```



## Strcmp

A função `strcmp` efetua uma comparação entre duas *strings*, indicando se existem diferenças entre elas (avalia caractere por caractere).

Os retornos possíveis são:

<b>Menor que zero</b>	-	Se a primeira variável for menor que a segunda.
<b>Igual a zero</b>	-	Se as duas variáveis forem iguais
<b>Maior que zero</b>	-	Se a primeira variável for maior que a segunda.

Por exemplo, entre os nomes "Mario" e "Maria" qual seria o resultado? Neste caso seria -1 pois "Maria" vem antes de "Mario", sendo, portanto, menor que este.

A sintaxe desta função é:

```
int strcmp(<primeira variável>, <segunda variável>);
```

Veja uma aplicação desta função:

```
1 #include <stdio.h>
2 #include <string.h>
3 int main () {
4     char frase [20] = "Mario", frase2[20] = "Maria";
5     printf ("\n%s - %s = %d\n", frase , frase2 , strcmp (frase2 , frase ));
6     return 0;
7 }
```

## Strcat

A função `strcat` permite concatenar a variável de origem à variável destino.

Ela não possui retorno, a alteração será feita diretamente em um dos parâmetros.

Possui dois parâmetros que efetua a concatenação da variável de origem na variável de destino.

A sintaxe:

```
void strcat(<variável de destino>, <variável de origem>);
```

### ATENÇÃO

O conteúdo da variável de destino será **complementado** com o conteúdo da variável origem, portanto, verifique se o tamanho da variável destino suportará.





Veja uma aplicação desta função:

```
1  #include <stdio.h>
2  #include <string.h>
3  int main () {
4      char origem[100],
5          destino[100];
6      printf ("Digite o nome: ");
7      fgets (origem, 100 , stdin);
8
9      printf ("Digite o sobrenome: ");
10     fgets (destino, 100 , stdin);
11
12     strcat (destino , origem);
13     printf ("\nO nome é %s. ", destino );
14     return 0;
15 }
```

## Strncat

A função `strncat` é análoga a função `strcat`, porém, permite que uma quantidade determinada seja definida para ser concatenada.

A sintaxe desta função é:

```
void strncat(<variável de destino>, <variável de origem>, <tamanho>);
```

## ATENÇÃO

O conteúdo da variável de destino será **complementado** com alguns caracteres da variável origem, portanto, verifique se o tamanho da variável suportará a quantidade definida.



Veja uma aplicação desta função:

```
1  #include <stdio.h>
2  #include <string.h>
3  int main () {
4      char origem[100],
5          destino[100];
6      printf ("Digite o nome: ");
7      fgets (origem, 100 , stdin);
8
9      printf ("Digite o sobrenome: ");
10     fgets (destino, 100 , stdin);
11
12     strncat ( destino , origem, 5);
13     printf ("\nO nome é %s. ", destino );
14     return 0;
15 }
```

As funções citadas anteriormente são apenas algumas das funções pertencentes a biblioteca `string.h`. É sempre importante estudarmos esta biblioteca antes de criar uma função de manipulação, afinal, pode ser que já exista o que você precisa.

## Síntese

---

*Strings* são importantes para a escrita de um código, pois não é possível criar conjunto de caracteres sem utilizá-las. Por isso, neste módulo, abordamos este assunto.

Você descobriu que declarar *strings* em linguagem C é o mesmo que declarar um vetor e, ao final da frase ou palavra, incluir um terminador nulo (`\0`) para indicar que sua *string* chegou ao fim.

Para inserirmos esse conteúdo em nosso programa, utilizaremos a função `fgets`, que permite limitar a quantidade e incluir o caractere nulo necessário, mas existem outras que fazem a mesma ação.

Descobriu também que a biblioteca `string.h` possui diversas funções para manipulação de *strings*, dentre elas você conheceu: `strcpy` que copia o conteúdo de uma *string* em outra; `strcat` que concatena duas *strings*; e, `strlen` que informa o tamanho total de caracteres de uma *string*. Todas nos permitem melhorar nossos programas.

A partir de agora utilizaremos mais e melhor nossos códigos.

Até mais!

## Referências

---

CELES FILHO, W. **Introdução a estrutura de dados**: com técnicas de programação em c. Rio de Janeiro: Elsevier, 2004. 294 p.

DAMAS, Luis. **Linguagem C**. 10. ed. Rio de Janeiro: Ltc, 2016.

MANZANO, José Augusto N G. **Linguagem C**: Acompanhada de uma xícara de café. São Paulo: Érica, 2015.

MIZRAHI, Victorine Viviane. **Treinamento em linguagem C**. São Paulo: Pearson Prentice Hall, 2008. 407 p.