

# Fundamentos de Programação

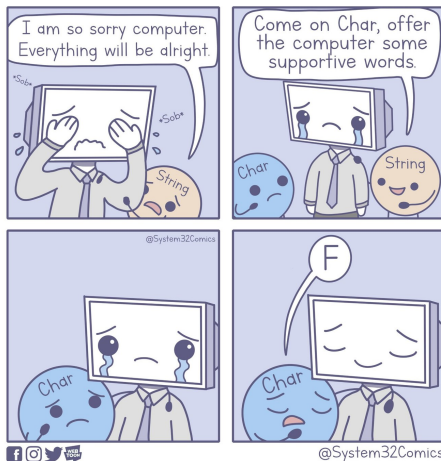
## *Strings*

Dainf - UTFPR

Profa. Leyza B. Dorini  
Prof. Bogdan T. Nassu

# Strings

Uma cadeia de caracteres, comumente chamada de *string*, é uma sequência de caracteres - por exemplo, letras, dígitos, espaços em branco e símbolos de pontuação.



# Strings

---

Já temos usado *strings* constantes...

```
printf ("Eu sou uma string.\n");  
printf ("Uma barra invertida:\t\\.\n");
```

Antes de tudo, um aviso: existem linguagens muito mais práticas que C para manipular *strings*!

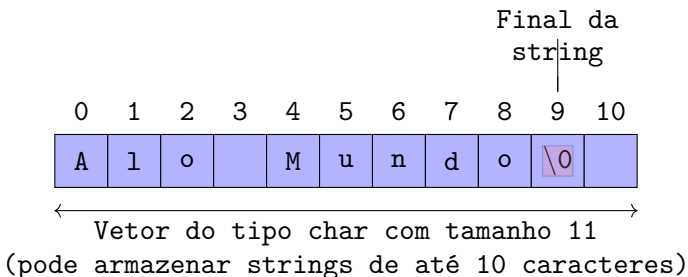
Veremos *strings* como casos particulares de vetores.

# Strings em C

A linguagem C não possui um tipo primitivo para *strings*.

Em C, uma *string* é simplesmente um vetor de char ou unsigned char, onde o número em cada posição é mapeado em uma tabela.

- Existem várias codificações/tabelas, a mais comum é a ASCII.
- *Null-terminated string*: `'\0'` (valor 0) delimita o final.
  - Isso quer dizer que precisamos de uma posição extra no vetor.
  - Esquecer disso é uma causa muito comum de *buffer overflow*!



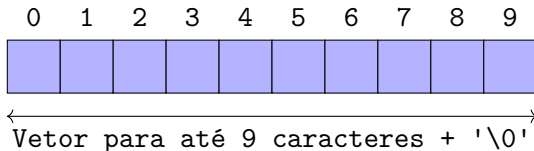
# Criando uma *string*

Para declarar uma *string* em C, criamos um vetor de caracteres:

```
char identificador [número de posições];  
unsigned char identificador [número de posições];
```

## Exemplo

```
char texto [10];
```



Não se esqueça de deixar espaço para o '\0'!!!

# Inicialização de *strings*

Inicializando com uma *string* constante, entre aspas.  
Neste caso, o '`\0`' é inserido automaticamente.

```
1 char texto[11] = "PROG UTFPR";
```

```
2
```

```
3 char texto[11] = {'P','R','O','G',' ','U','T','F','P','R','\0'};
```

Também é possível inicializar como  
um vetor, caractere a caractere.

# Inicialização de *strings*

Inicializando com uma *string* constante, entre aspas.  
Neste caso, o '\0' é inserido automaticamente.

```
1 char texto[11] = "PROG UTFPR";
```

```
2
```

```
3 char texto[11] = {'P','R','O','G',' ','U','T','F','P','R','\0'};
```

Também é possível inicializar como  
um vetor, caractere a caractere.

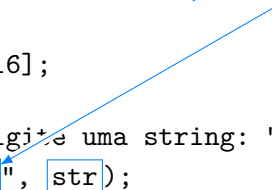
Para as duas inicializações acima, o resultado seria:

0	1	2	3	4	5	6	7	8	9	10
P	R	O	G		U	T	F	P	R	\0

Muito cuidado com o tamanho ao inicializar *strings* assim!!!

## Exemplo: lê uma *string* e mostra seus caracteres

```
1  int main ()
2  {                                     %s é a sequência de escape para strings.
3      int i;
4      char str [16];
5
6      printf ("Digite uma string: ");
7      scanf ("%s", str);
8
9      for (i = 0; str [i] != '\0'; i++)
10         printf ("%c\n", str [i]);
11
12     return (0);
13 }
```

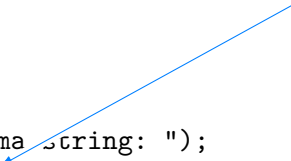




## Exemplo: lê uma *string* e mostra seus caracteres


```
1  int main ()
2  {
3      int i;
4      char str [16];
5
6      printf ("Digite uma string: ");
7      scanf ("%s", str);
8
9      for (i = 0; str [i] != '\0'; i++)
10         printf ("%c\n", str [i]);
11
12     return (0);
13 }
```

Note que não usamos o &.  
A *string* é um vetor.



## Exemplo: lê uma *string* e mostra seus caracteres

```
1  int main ()
2  {
3      int i;
4      char str [16];
5
6      printf ("Digite uma string: ");
7      scanf ("%s", str);
8
9      for (i = 0; str [i] != '\0'; i++)
10         printf ("%c\n", str [i]);
11
12     return (0);
13 }
```

 %c é a sequência de escape para um caractere. Não confunda caractere com *string*!.

A função `scanf` não é muito prática para ler *strings*.

- Lê até o primeiro divisor, que permanece no *buffer* do teclado.
- Se usada como no exemplo, pode causar um *buffer overflow*.
- Estes problemas podem ser contornados usando formatos especiais, mas o uso é um pouco desajeitado.

Uma alternativa é a função `fgets` (declarada em `stdio.h`).

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

# Lendo *strings* com `fgets`

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

- Lê até `num` caracteres e coloca em `str`.
- É para leitura de arquivos, mas podemos usar para a entrada padrão: basta passar `stdin` como `stream`.
- Lê espaços.
- Se o *buffer to teclado* tiver mais do que `num` caracteres, eles ficam no *buffer*.
- Retorna `str` em caso de sucesso, ou `NULL (0)` do contrário.

Cuidado: o `'\n'` também é guardado no vetor!

Para o código: `fgets (texto, 10, stdin)`

Ao digitar `abcde`, vai armazenar:

`abcde\n\0`

## Exemplo: lendo *strings* com `fgets`

```
1  #include <stdio.h>
2  #define BUFLen 16
3
4  int main ()
5  {
6      int i;
7      char str [BUFLen];
8
9      printf ("Digite uma string: ");
10     if (fgets (str, BUFLen, stdin))
11         for (i = 0; str [i] != '\0'; i++)
12             printf ("%c\n", str [i]);
13
14     return (0);
15 }
```

# Lendo caracteres com getchar

Outra função útil: `getchar` (declarada em `stdio.h`), que lê um único caractere da entrada padrão.

```
1  int main ()
2  {
3      char c;
4
5      while (1)
6      {
7          c = getchar ();
8          if (c != '\n')
9              printf ("Digitou '%c' (ASCII %d)\n", c, c);
10     }
11
12     return (0);
13 }
```

Fazemos este teste porque a função `getchar` captura também o `'\n'` quando pressionamos ENTER.

## Escrevendo uma *string* na tela

Podemos escrever uma *string* na tela caractere a caractere, mas é mais simples escrever utilizando a função `printf` com a sequência de escape `%s`:

```
printf ("%s", uma_string);
```

## Strings e números

Uma *string* contendo os dígitos de um número não é igual a um valor numérico!

```
"10" != 10
```

Para colocar um número em uma *string*, você pode usar a função `sprintf` (declarada em `stdio.h`). Ela tem a mesma sintaxe do `printf`, mas o vetor é passado como parâmetro.

```
sprintf (string, "%d", numero);
```

As bibliotecas-padrão da linguagem C têm uma série de funções para extrair o valor numérico de uma *string* contendo dígitos: `atof`, `atoi`, `atol`, `atoll`, `strtod`, `strtof`, `strtol`...

- Todas declaradas em `stdlib.h`.
- Se precisar usar estas funções, consulte a documentação!



## Strings: outras funções

As bibliotecas-padrão possuem várias outras funções para *strings*.

Alguns exemplos:

- Comprimento (`strlen`).
  - Cuidado! Esta função procura pelo `'\0'`, e não é “instantânea”.
- Cópia (`strcpy`).
- Comparação (`strcmp`).
- Concatenação (`strcat`).
- Inicialização de vetores (`memset`).
- Busca de caracteres e substrings.

Não vamos estudar funções específicas. Você deve saber que:

- Estas funções existem.
- Quando precisar delas consulte a documentação.
- Nosso objetivo não é memorizar cabeçalhos de funções!!!

Na dúvida, lembre-se: *strings* são vetores!

- A forma de acessar é a mesma, com `[]`.
- O primeiro índice é 0.
- Você não pode comparar strings com `==` ou `!=`.
- Você não pode copiar strings com `=`.
- Você não pode concatenar strings com `+`.