

---

# LINGUAGEM C: ARRAYS DE CARACTERES - STRINGS

PROF. ANDRÉ BACKES



# DEFINIÇÃO

- String
  - Sequência de caracteres adjacentes na memória.
  - Essa sequência de caracteres, que pode ser uma palavra ou frase
  - Em outras palavras, strings são arrays do tipo char.
- Exemplo
  - `char str[6];`

# DEFINIÇÃO

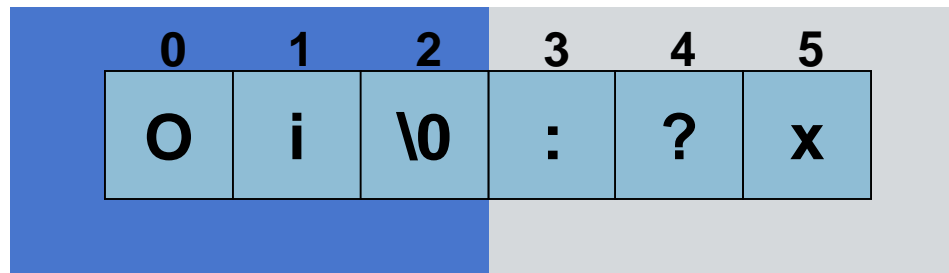
- String

- Devemos ficar atentos para o fato de que as strings têm no elemento seguinte a última letra da palavra/frase armazenado um caractere '\0' (barra invertida + zero).
- O caractere '\0' indica o fim da sequência de caracteres.

- Exemplo

- `char str[6] = "Oi";`

Região inicializada: 2  
letras + 1 caractere  
terminador '\0'



Lixo de memória  
(região não  
inicializada)

# DEFINIÇÃO

- Importante
  - Ao definir o tamanho de uma string, devemos considerar o caractere '\0'.
  - Isso significa que a string str comporta uma palavra de no máximo 5 caracteres.
- Exemplo:
  - `char str[6] = "Teste";`

|   |   |   |   |   |    |
|---|---|---|---|---|----|
| T | e | s | t | e | \0 |
|---|---|---|---|---|----|

# DEFINIÇÃO

- Por se tratar de um array, cada caractere podem ser acessados individualmente por meio de um índice
- Exemplo
  - `char str[6] = "Teste";`

|   |   |   |   |   |    |
|---|---|---|---|---|----|
| T | e | s | t | e | \0 |
|---|---|---|---|---|----|

- `str[0] = 'L';`

|   |   |   |   |   |    |
|---|---|---|---|---|----|
| L | e | s | t | e | \0 |
|---|---|---|---|---|----|

# DEFINIÇÃO

- IMPORTANTE:
  - Na inicialização de palavras, usa-se "aspas duplas".
  - Exemplo: `char str[6] = "Teste";`

|   |   |   |   |   |    |
|---|---|---|---|---|----|
| T | e | s | t | e | \0 |
|---|---|---|---|---|----|

- Na atribuição de um caractere, usa-se 'aspas simples'
- `str[0] = 'L';`

|   |   |   |   |   |    |
|---|---|---|---|---|----|
| L | e | s | t | e | \0 |
|---|---|---|---|---|----|

# DEFINIÇÃO

- Importante:
  - "A" é diferente de 'A'
    - "A"

|   |    |
|---|----|
| A | \0 |
|---|----|

- 'A'

|   |
|---|
| A |
|---|

# OBSERVAÇÕES SOBRE A MEMÓRIA

```
char c;  
c = 'h';
```

```
int a;  
a = 19;
```

```
char Sigla[4];  
Sigla[0] = 'U';  
Sigla[1] = 'F';  
Sigla[2] = 'U';  
Sigla[3] = '\0';
```


| Endereço | Blocos | Variável | tipo    |
|----------|--------|----------|---------|
| 1        |        |          |         |
| 2        |        |          |         |
| 3        | 'H'    | c        | char    |
| 4        |        |          |         |
| 5        |        |          |         |
| 6        |        |          |         |
| 7        | 'U'    | Sigla[0] | char[4] |
| 8        | 'F'    | Sigla[1] |         |
| 9        | 'U'    | Sigla[2] |         |
| 10       | '\0'   | Sigla[3] |         |
| 11       | 19     | a        | int     |
| 12       |        |          |         |
| 13       |        |          |         |
| 14       |        |          |         |
|          | ....   |          |         |



# MANIPULANDO STRINGS

- Strings são arrays. Portanto, não se pode atribuir uma string para outra!
- O correto é copiar a string elemento por elemento.

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    char str1[20] = "Hello World";
    char str2[20];

     str1 = str2;

    system("pause");
    return 0;
}
```

# COPIANDO UMA STRING

- O correto é copiar a string elemento por elemento.

```
#include <stdio.h>
#include <stdlib.h>
int main(){
    int i;
    char str1[20] = "Hello World";
    char str2[20];

    for(i = 0; str1[i] != '\0'; i++)
        str2[i] = str1[i];
    str2[i] = '\0';

    system("pause");
    return 0;
}
```

# MANIPULANDO STRINGS

- Felizmente, a biblioteca padrão C possui funções especialmente desenvolvidas para esse tipo de tarefa
  - `#include <string.h>`

# MANIPULANDO STRINGS | LEITURA

- Exemplo de algumas funções para manipulação de strings
- `gets(str)`: lê uma string do teclado e armazena em `str`.

```
char str[10];  
gets(str);
```

# MANIPULANDO STRINGS | LIMPEZA DO BUFFER

- Às vezes, podem ocorrer erros durante a leitura de caracteres ou strings.
- Para resolver esses pequenos erros, podemos limpar o buffer do teclado

```
char str[10];
```

```
setbuf(stdin, NULL); //limpa o buffer
```

```
gets(str);
```

# MANIPULANDO STRINGS | ESCRITA

- Basicamente, para se escrever uma string na tela utilizamos a função printf().
  - Especificador de formato: %s

```
char str[20] = "Hello World";  
printf("%s", str);
```

# MANIPULANDO STRINGS | TAMANHO

- `strlen(str)`: retorna o tamanho da string `str`.
- Neste caso, a função retornará 5, que é o número de caracteres na palavra "teste" e não 15, que é o tamanho do array.
  - O `'\0'` também não é considerado pela `strlen`, mas vale lembrar que ele está escrito na posição `str[5]` do vetor.

```
char str[15] = "teste";  
printf("%d", strlen(str));
```

# MANIPULANDO STRINGS | COPIAR

- strcpy(dest, fonte): copia a string contida na variável fonte para dest.

```
char str1[100], str2[100];  
printf("Entre com uma string: ");  
gets(str1);  
strcpy(str2, str1);  
printf("%s", str2);
```



# MANIPULANDO STRINGS | CONCATENAR

- `strcat(dest, fonte)`: concatena duas strings.
- Neste caso, a string contida em fonte permanecerá inalterada e será copiada para o final da string dest.

```
char str1[15] = "bom ";  
char str2[15] = "dia";  
strcat(str1, str2);  
printf("%s", str1);
```

# MANIPULANDO STRINGS | COMPARAR

- `strcmp(str1, str2)`: compara duas strings. Neste caso, a função retorna ZERO se as strings forem iguais.

```
if(strcmp(str1, str2) == 0)
    printf("Strings iguais");
else
    printf("Strings diferentes");
```

# MANIPULANDO STRINGS

- Basicamente, para se ler uma string do teclado utilizamos a função `gets()`.
- No entanto, existe outra função que, utilizada de forma adequada, também permite a leitura de strings do teclado. Essa função é a `fgets()`, cujo protótipo é:

```
char *fgets(char *str, int tamanho, FILE *fp);
```

# MANIPULANDO STRINGS

- A função `fgets` recebe 3 argumentos
  - a string a ser lida, `str`;
  - o limite máximo de caracteres a serem lidos, `tamanho`;
  - A variável `FILE *fp`, que está associado ao arquivo de onde a string será lida.
- E retorna
  - `NULL` em caso de erro ou fim do arquivo;
  - O ponteiro para o primeiro caractere recuperado em `str`.

```
char *fgets(char *str, int tamanho, FILE *fp);
```

# MANIPULANDO STRINGS

- Note que a função `fgets` utiliza uma variável `FILE *fp`, que está associado ao arquivo de onde a string será lida.
- Para ler do teclado, basta substituir `FILE *fp` por `stdin`, o qual representa o dispositivo de entrada padrão (geralmente o teclado):

```
int main() {  
    char nome[30];  
    printf("Digite um nome: ");  
    fgets(nome, 30, stdin);  
    printf("O nome digitado foi: %s", nome);  
  
    return 0;  
}
```

# MANIPULANDO STRINGS

- Funcionamento da função fgets
  - A função lê a string até que um caractere de nova linha seja lido ou tamanho-1 caracteres tenham sido lidos.
  - Se o caractere de nova linha ('\n') for lido, ele fará parte da string, o que não acontecia com gets.
  - A string resultante sempre terminará com '\0' (por isto somente tamanho-1 caracteres, no máximo, serão lidos).
  - Se ocorrer algum erro, a função devolverá um ponteiro nulo (NULL) em str.

# MANIPULANDO STRINGS

- A função fgets é semelhante à função gets, porém, com as seguintes vantagens:
  - pode fazer a leitura a partir de um arquivo de dados e incluir o caractere de nova linha “\n” na string;
  - especifica o tamanho máximo da string de entrada. Evita estouro no buffer;

# MANIPULANDO STRINGS

- Basicamente, para se escrever uma string na tela utilizamos a função printf().

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

- No entanto, existe outra função que, utilizada de forma adequada, também permite a escrita de strings. Essa função é a fputs(), cujo protótipo é:

```
int fputs(char *str, FILE *fp);
```



# MANIPULANDO STRINGS

- A função `fputs()` recebe como parâmetro um array de caracteres e a variável `FILE *fp` representando o arquivo no qual queremos escrever.
- Retorno da função
  - Se o texto for escrito com sucesso um valor inteiro diferente de zero é retornado.
  - Se houver erro na escrita, o valor EOF (em geral, `-1`) é retornado.

# MANIPULANDO STRINGS

- Note que a função `fputs` utiliza uma variável `FILE *fp`, que está associado ao arquivo de onde a string será escrita.
- Para escrever no monitor, basta substituir `FILE *fp` por `stdout`, o qual representa o dispositivo de saída padrão (geralmente a tela do monitor):

```
int main() {  
    char texto[30] = "Hello World\n";  
    fputs(texto, stdout);  
  
    return 0;  
}
```

## OBSERVAÇÃO

- Ao inicializar uma string em sua declaração, ao contrário do que dizia os slides anteriores, as regiões do vetor que não foram utilizadas pela string são preenchidas com zeros ('\0')
- Entretanto, esse comportamento não ocorre com o strcpy e gets. Nessas funções as posições não usadas são lixos.
- Exemplo: `char str[6] = "Oi";`

|   |   |    |    |    |    |
|---|---|----|----|----|----|
| O | i | \0 | \0 | \0 | \0 |
|---|---|----|----|----|----|

# OBSERVAÇÃO

- Exemplos

- `char str[6] = "Oi";`

|   |   |    |    |    |    |
|---|---|----|----|----|----|
| O | i | \0 | \0 | \0 | \0 |
|---|---|----|----|----|----|

- `gets(str);` //digite "Oi" no prompt

|   |   |    |   |   |   |
|---|---|----|---|---|---|
| O | i | \0 | : | ? | x |
|---|---|----|---|---|---|

- `strcpy(str, "Oi");`

|   |   |    |   |   |   |
|---|---|----|---|---|---|
| O | i | \0 | X | ? | @ |
|---|---|----|---|---|---|

# TRABALHANDO COM CARACTERE AMPLO (WIDE CHAR)

- O tipo `char` não permite a codificação de grandes conjuntos de caracteres
- Outra opção é usar o tipo de caractere amplo `wchar_t`, definido na biblioteca `wchar.h`
  - Caractere representado por mais de 8 bits
  - Permite o uso de conjuntos de caracteres estendidos, como as codificações UTF-16 e UTF-32

# TRABALHANDO COM CARACTERE AMPLO (WIDE CHAR)

- Duas formas de representar um caractere
  - \uhhhh, onde hhhh é uma sequência de 4 caracteres hexadecimal
  - \Uhhhhhhhh, onde hhhhhhhh é uma sequência de 8 caracteres hexadecimal

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

int main() {

    wchar_t caractere = L'\u00F5';
    wchar_t texto[30] = L"meu caractere longo = \u00F1";

    wprintf(L"Marca registrada: \u00A9 \n");
    wprintf(L"Caractere: %lc \n", caractere);
    wprintf(L"%ls \n", texto);

    return 0;
}
```

# TRABALHANDO COM CARACTERE AMPLO (WIDE CHAR)

- Note que não usamos `printf()` para impressão.
  - Não possui suporte a caractere amplo
- Devemos usar as funções definidas na biblioteca `wchar.h`
  - Funções de entrada de dados: `fgetwc()`, `fgetws()`, `getwc()`, `getwchar()`, `fwscanf()`, `wscanf()`, `vfwscanf()`, e `vwscanf()`.
  - Funções de saída de dados: `fputwc()`, `fputws()`, `putwc()`, `putwchar()`, `fwprintf()`, `wprintf()`, `vfwprintf()`, e `vwprintf()`.

# MELHOR SUPORTE AO PADRÃO UNICODE

- Unicode: padrão utilizado pelos computadores para codificar, representar e manipular, de forma consistente, textos produzidos por qualquer sistema de escrita existente
- Pode ser implementado usando diferentes regras de codificação de caracteres, como o UTF-8, UTF-16, e UTF-32, entre outros
  - UTF significa Formato de Transformação Unicode (do inglês, Unicode Transformation Format).



# MELHOR SUPORTE AO PADRÃO UNICODE

- No padrão C11 é possível especificar o padrão Unicode utilizado:
  - UTF-8, UTF-16 e UTF-32
- Dois novos tipos adicionados (biblioteca uchar.h)
  - `char16_t` armazena strings codificadas em UTF-16 (prefixo `u`)
  - `char32_t` armazena strings codificadas em UTF-32 (prefixo `U`)
  - O padrão UTF-8 é codificado no tipo `char` (prefixo `u8`)

# MELHOR SUPORTE AO PADRÃO UNICODE

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

int main(){
    // ASCII
    char str[] = "Exemplo de string ASCII.";

    // UTF-8
    char str_u8[] = u8"Exemplo de string UTF-8.";

    // UTF-16
    char16_t str_u16[] = u"Exemplo de string UTF-16.";

    // UTF-32
    char32_t str_u32[] = U"Exemplo de string UTF-32.";

    return 0;
}
```

# MATERIAL COMPLEMENTAR

- Vídeo Aulas

- Aula 31: Strings: Conceitos Básicos: [www.youtube.com/watch?v=5mJZh\\_ikDaQ](http://www.youtube.com/watch?v=5mJZh_ikDaQ)
- Aula 32: Strings: Biblioteca string.h: [youtu.be/MEkrf1O\\_CIU](http://youtu.be/MEkrf1O_CIU)
- Aula 33: Strings: Invertendo uma String: [youtu.be/jNQUEpwMd\\_M](http://youtu.be/jNQUEpwMd_M)
- Aula 34: Strings: Contando Caracteres Específicos: [youtu.be/s\\_V\\_LZX1eD0](http://youtu.be/s_V_LZX1eD0)
- Aula 81: Limpando o buffer do teclado: [www.youtube.com/watch?v=ixk5RlqABjI](http://www.youtube.com/watch?v=ixk5RlqABjI)