

Algoritmos e Estruturas de Dados I

1º Período Engenharia da Computação

Prof. Edwaldo Soares Rodrigues

Email: edwaldo.rodrigues@uemg.br

Material adaptado do prof. André Backes

String

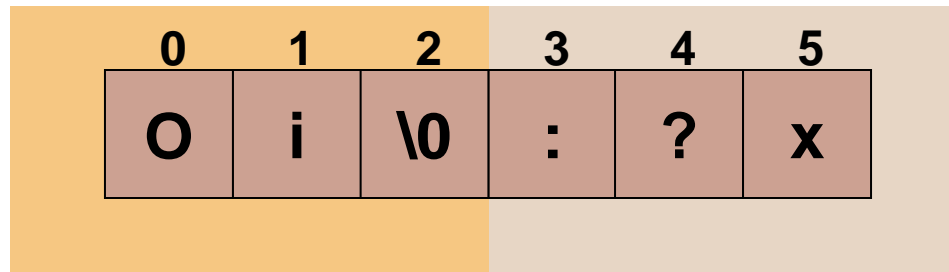
Definição

- 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**.
- Ex:
 - `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 pode ser acessado 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”**;
 - Ex: `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
---	---	---	---	---	----

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);
```


Definição

- **Importante:**

- “A” é diferente de ‘A’

- “A”

A	\0
---	----

- ‘A’

A

Definição


- Observações sobre a memória

	Endereço	Blocos	Variável	tipo
<pre>char c; c = 'h'; int a; a = 19; char Sigla[4]; Sigla[0] = 'U'; Sigla[1] = 'F'; Sigla[2] = 'U'; Sigla[3] = '\0';</pre>	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!

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

     str1 = str2;

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

- O correto é copiar a string elemento por elemento;

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**;
 - Exemplo:

```
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;
 - Exemplo:

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

Manipulando strings - Copiar

- **strcpy(dest, fonte)**: copia a string contida na variável **fonte** para **dest**;
- Exemplo

```
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 anexada ao final da string de **dest**;
 - Exemplo

```
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;

- Exemplo

```
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 final

- 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');
- Ex: `char str[6] = "Oi";`

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

- Entretanto, esse comportamento não ocorre com o **strcpy** e **gets**. Nessas funções as posições não usadas são lixos;

Observação final

- 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	?	@
---	---	----	---	---	---

Exercícios

- Faça um programa que leia o nome de três alunos e suas respectivas notas em AEDS I. Em seguida, o programa deverá apresentar o nome e a nota do aluno que obteve maior nota;
- Faça um programa que leia um nome, e em seguida calcule e retorne quantas letras esse nome tem. Obs: não use a função `strlen`;
- Faça um programa que receba uma palavra e a imprima de trás para frente;

Exercícios

```
#include <stdio.h>
#include <string.h>
int main() {
    char aluno1[20], char aluno2[20], char aluno3[20];
    float n1, n2, n3;
    printf("Digite o nome do aluno 1: ");
    setbuf (stdin, NULL);
    gets(aluno1);
    printf("Digite a nota do aluno 1: ");
    scanf("%f", &n1);
    printf("Digite o nome do aluno 2: ");
    setbuf (stdin, NULL);
    gets(aluno2);
    printf("Digite a nota do aluno 2: ");
    scanf("%f", &n2);
    printf("Digite o nome do aluno 3: ");
    setbuf (stdin, NULL);
    gets(aluno3);
    printf("Digite a nota do aluno 3: ");
    scanf("%f", &n3);

    if(n1 > n2 && n1 > n3){
        printf("O aluno de maior nota foi: %s e sua nota foi: %f. \n", aluno1, n1);
    }
    else if(n2 > n1 && n2 > n3){
        printf("O aluno de maior nota foi: %s e sua nota foi: %f. \n", aluno2, n2);
    }
    else if(n3 > n1 && n3 > n2){
        printf("O aluno de maior nota foi: %s e sua nota foi: %f. \n", aluno3, n3);
    }
    return 0;
}
```


Exercícios

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

int main() {
    char nome[20];
    printf("Entre com seu nome: ");
    gets(nome);
    int cont=0;

    for(int i=0; nome[i] != '\0'; i++){
        cont++;
    }
    printf("Este nome possui %d letras.\n", cont);
    return 0;
}
```

Exercícios

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

int main() {
    char palavra[20];
    printf("Entre com uma palavra: ");
    gets(palavra);
    int cont = 0;
    for(int i = 0; palavra[i] != '\0'; i++) {
        cont++;
    }

    for(int i = cont; cont >= 0; cont--) {
        printf("%c", palavra[cont]);
    }
    return 0;
}
```

Algoritmos e Estruturas de Dados I

- Bibliografia:

- Básica:

- CORMEN, Thomas; RIVEST, Ronald, STEIN, Clifford, LEISERSON, Charles. Algoritmos. Rio de Janeiro: Elsevier, 2002.
 - DEITEL, Paul; DEITEL, Harvey. C++ como programar. 5. ed. São Paulo: Pearson, 2006.
 - MELO, Ana Cristina Vieira de; SILVA, Flávio Soares Corrêa da. Princípios de linguagens de programação. São Paulo: Edgard Blücher, 2003.

- Complementar:

- ASCENCIO, A. F. G. & CAMPOS, E. A. V. Fundamentos da programação de computadores. 2. ed. São Paulo: Pearson Education, 2007.
 - MEDINA, Marcelo, FERTIG, Cristina. Algoritmos e programação: teoria e prática. Novatec. 2005.
 - MIZRAHI, V. V.. Treinamento em linguagem C: módulo 1. São Paulo: Makron Books, 2008.
 - PUGA, S. & RISSETTI, G. Lógica de programação e estruturas de dados com aplicações em java. São Paulo: Prentice Hall, 2004.
 - ZIVIANI, Nívio. Projeto de Algoritmos com Implementação em Pascal e C. Cengage Learning. 2010.

Algoritmos e Estruturas de Dados I

