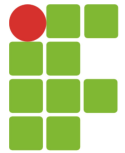
Strings

Algoritmos e Programação de Computadores – ABI/LFI/TAI

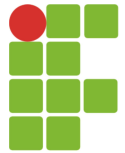
Prof. Daniel Saad Nogueira Nunes

IFB – Instituto Federal de Brasília,
Campus Taguatinga



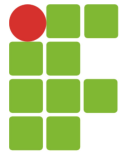
Sumário

1. Introdução
2. Strings
3. Exemplos
4. string.h



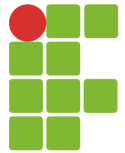
Sumário

1 Introdução



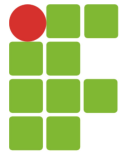
Introdução

- Já vimos como representar um caractere na linguagem `C`.
- E se quisermos representar palavras (*strings*)? Como fazemos?
- O `C` não possui um tipo *string*, mas através de um mecanismo que já vimos, podemos representar as palavras.
- Strings em `C`: **vetor de caracteres**.



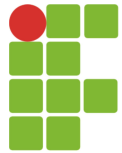
Sumário

2 Strings



Sumário

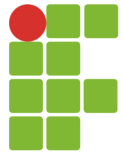
2 Strings
- Sintaxe
- Escrita
- Leitura
- Inicialização



Strings: Sintaxe

Strings

- Strings em `C` são vetores de caracteres em que o último elemento é o caractere `'\0'`.
- Este caractere indica o **fim** da string.
- Para que as funções do cabeçalho `<string.h>` funcionem, é necessário que estes vetores possuam esse caractere de terminação.
- Por conta disso, se o objetivo é armazenar n caracteres, é necessário declarar o vetor com $n + 1$ posições, considerando o caractere de terminação `'\0'`.

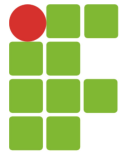


Strings: Sintaxe

- Por exemplo, caso quiséssemos armazenar uma palavra com 10 caracteres, declararíamos uma string da seguinte forma:

```
char palavra[11];
```

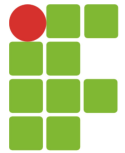
- Declaramos um espaço extra, para o armazenamento do `'\0'`, que deverá ocupar a posição 10.



Strings: Leitura

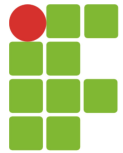
Cuidados

- A não presença do caractere `'\0'` indicando o final da string pode causar problemas.
- As funções que normalmente atuam sobre as strings contam com ele, caso não a encontrem, podemos ter erros como:
  - Buffer Overflow.
  - Falha de segmentação.



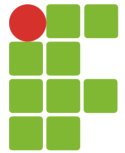
Sumário

2 Strings
- Sintaxe
- Escrita
- Leitura
- Inicialização



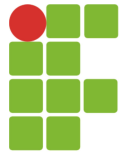
Strings: Escrita

- Para imprimir strings, podemos usar a função `printf`.
- O modificador utilizado é o `%s`.



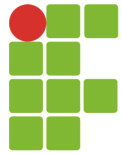
Strings: Escrita

```c
#include <stdio.h>

int main(void){
    char mensagem[] = {'o','l','a',' ','m','u','n','d','o','\0'};
    printf("%s\n",mensagem);
    return 0;
}
```



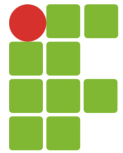
Strings: Escrita

```c
#include <stdio.h>

int main(void){
    char mensagem[10];
    mensagem[0] = 'o';
    mensagem[1] = 'l';
    mensagem[2] = 'a';
    mensagem[3] = ' ';
    mensagem[4] = 'm';
    mensagem[5] = 'u';
    mensagem[6] = 'n';
    mensagem[7] = 'd';
    mensagem[8] = 'o';
    mensagem[9] = '\0';
    printf("%s\n",mensagem);
    return 0;
}
```



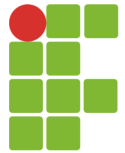
Strings: Escrita

- Caso a string não conte com o `'\0'`, podemos ter problemas.
- O exemplo a seguir utiliza um `printf` em uma string que não foi inicializada com `'\0'`.
- Desta forma o `printf` vai imprimir o conteúdo de posições de memórias mais avançadas, podendo fazer até que o programa seja abortado.



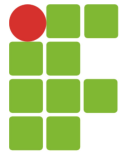
Strings: Escrita

```c
#include <stdio.h>

int main(void){
    char mensagem[1000];
    mensagem[0] = 'o';
    mensagem[1] = 'l';
    mensagem[2] = 'a';
    mensagem[3] = ' ';
    mensagem[4] = 'm';
    mensagem[5] = 'u';
    mensagem[6] = 'n';
    mensagem[7] = 'd';
    mensagem[8] = 'o';
    printf("%s\n",mensagem);
    return 0;
}
```



Sumário

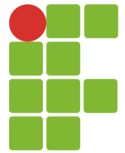
2 Strings
- Sintaxe
- Escrita
- Leitura
- Inicialização



Strings: Leitura

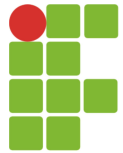
Scanf

- Podemos ler strings com o `scanf`.
- Assim que for encontrado o primeiro espaço ou quebra de linha, a leitura se encerrará.
- Não é necessário utilizar o **&**. Motivo: o nome da string já é um endereço. Em C, o nome do vetor é o endereço base dele na memória.
- Automaticamente o caractere `'\0'` é inserido após o último caractere lido (se houver espaço!).



Strings: Leitura

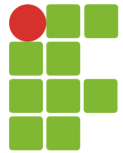
```c
#include <stdio.h>

int main(void){
    char mensagem[10];
    printf("Digite uma mensagem: ");
    scanf("%s",mensagem);
    printf("A mensagem digitada foi \"%s\"\n",mensagem);
    return 0;
}
```



Strings: Leitura

```
1  #include <stdio.h>
2
3  int main(void){
4      char mensagem[10];
5      printf("Digite uma mensagem: ");
6      scanf("%s",mensagem);
7      printf("A mensagem digitada foi \"%s\"\n",mensagem);
8      return 0;
9  }
```

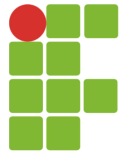
Se for digitado `"ola mundo"`, será armazenado apenas `"ola\0"` na string.



Strings: Leitura

```c
#include <stdio.h>

int main(void){
    char mensagem[10];
    printf("Digite uma mensagem: ");
    scanf("%s",mensagem);
    printf("A mensagem digitada foi \"%s\"\n",mensagem);
    return 0;
}
```

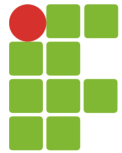
Se for digitado `"estaehumalinhamuitogrande"`, haverá um *buffer overflow*.



Strings: Leitura

Problemas com o scanf

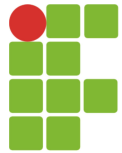
- O `scanf` puro possui diversos problemas para leitura, sendo considerado um mecanismo inseguro.
  - Caso se digite mais caracteres do que a string consiga armazenar, não será possível inserir o `'\0'` ou até mesmo o programa poderá abortar devido a um *buffer overflow!*.
- Ele também não processa espaços na sua forma mais pura.



Strings: Leitura

fgets

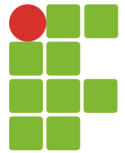
- A função `fgets` nos permite ler uma linha inteira e é capaz de processar os espaços.
- Ela também fornece uma proteção contra o *buffer overflow*, já que ela só lê o número de caracteres especificados.
- Sintaxe: `fgets(nome_string,limite,stdin)` .
- O parâmetro **stdin** diz que a leitura será feita da entrada padrão, normalmente o teclado.



Strings: Leitura

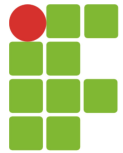
fgets

- O `fgets` lê os caracteres até o a quebra de linha ou o tamanho menos um serem atingidos, o que ocorrer primeiro.
- Automaticamente o `'\0'` é inserido no final da string.
- Se a quebra de linha ocorreu antes do limite de caracteres ser atingido, o caractere de quebra de linha é inserido na penúltima posição da string.
- Caso contrário, se foram lidos `tamanho-1` caracteres antes de encontrar uma quebra de linha, todos serão armazenados na string, juntamente com o `'\0'`. A próxima leitura continuará de onde parou.



Strings: Leitura

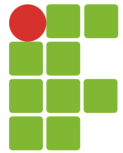
```c
#include <stdio.h>

int main(void){
    char mensagem[10];
    printf("Digite uma mensagem: ");
    fgets(mensagem,10,stdin);
    printf("A mensagem digitada foi \"%s\"\n",mensagem);
    return 0;
}
```



Strings: Leitura

```c
#include <stdio.h>

int main(void){
    char mensagem[10];
    printf("Digite uma mensagem: ");
    fgets(mensagem,10,stdin);
    printf("A mensagem digitada foi \"%s\"\n",mensagem);
    return 0;
}
```

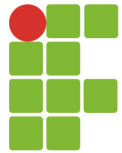
Se for digitado `"ola"`, será armazenado `"ola"` na string, juntamente com os caracteres de quebra de linha e `'\0'`.



Strings: Leitura

```c
#include <stdio.h>

int main(void){
    char mensagem[10];
    printf("Digite uma mensagem: ");
    fgets(mensagem,10,stdin);
    printf("A mensagem digitada foi \"%s\"\n",mensagem);
    return 0;
}
```

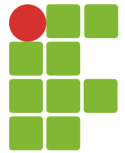
Se for digitado `"ola mundo"`, será armazenado apenas `"ola mundo\0"` na string.



Strings: Leitura

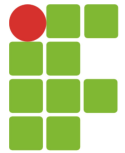
```c
#include <stdio.h>

int main(void){
    char mensagem[10];
    printf("Digite uma mensagem: ");
    fgets(mensagem,10,stdin);
    printf("A mensagem digitada foi \"%s\"\n",mensagem);
    return 0;
}
```

Se for digitado `"estaehumalinhamuitogrande"`, será armazenado apenas `"estaehuma\0"` na string. A próxima leitura de `fgets` continuará de `"linhamuitogrande"`.



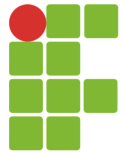
Sumário

2 Strings
- Sintaxe
- Escrita
- Leitura
- Inicialização



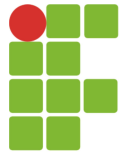
Strings: Inicialização

- É possível inicializar strings com valores pré-determinados, sem que seja necessário uma leitura.
- A maneira mais fácil é utilizar a seguinte sintaxe:
  `char string[10] = "ola mundo"` .
- Todos os caracteres serão armazenados, bem como o `'\0'` no final da string.
- Também é possível omitir o tamanho da string, que é inferido pelo compilador.



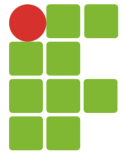
Strings: Inicialização

```
1  #include <stdio.h>
2
3  int main(void){
4      char mensagem[10] = "Ola mundo";
5      printf("A mensagem digitada foi \"%s\"\n",mensagem);
6      return 0;
7  }
```



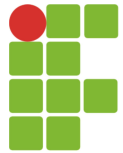
Strings: Inicialização

```
1  #include <stdio.h>
2
3  int main(void){
4      char mensagem[] = "Ola mundo";
5      printf("A mensagem digitada foi \"%s\"\n",mensagem);
6      return 0;
7  }
```



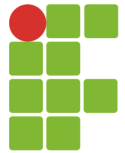
Sumário

3 Exemplos



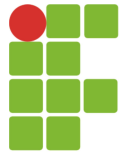
Exemplos

- Serão apresentados agora uma série de exemplos sobre strings para fixação dos conceitos.



Sumário

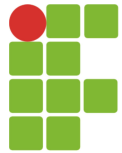
3 Exemplos
- Inversão
- Palíndromo
- Exemplos: Casamento de Padrões



Exemplos: Inversão

Problema

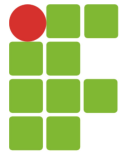
- Dada uma string, deverá ser computar a string invertida, isto é, de trás para frente.
- Exemplo: a inversão de `"abracadabra"` é `"arbadacarba"`.
- O `'\0'` deve permanecer no final da string.



Exemplos: Inversão

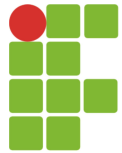
Estratégia

- A estratégia é utilizar uma string auxiliar, que armazenará os caracteres da string original do final para o início.



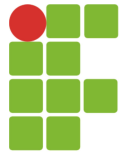
Exemplos: Inversão

```c
#include <stdio.h>

int main(void) {
    char string[100];
    char string_invertida[100];
    int i, j, tamanho;
    printf("Digite a string a ser invertida: ");
    fgets(string, 100, stdin);
    for (tamanho = 0; string[tamanho] != '\0' && string[tamanho] != '\n';
         tamanho++)
        ;
    for (i = 0, j = tamanho - 1; i < tamanho; i++, j--) {
        string_invertida[i] = string[j];
    }
    string_invertida[tamanho] = '\0';
    printf("A string invertida é: %s\n", string_invertida);
    return 0;
}
```



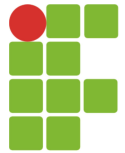
Exemplos: Inversão

- É possível inverter a string sem usar uma string auxiliar.
- Podemos trocar os valores do final com os do começo! Basta usar uma variável auxiliar do tipo `char`.
- O nosso contador só precisa ir até a metade do vetor.



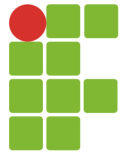
Exemplos: Inversão

```c
#include <stdio.h>

int main(void) {
    char string[100];
    int i, j, tamanho;
    printf("Digite a string a ser invertida: ");
    fgets(string, 100, stdin);
    for (tamanho = 0; string[tamanho] != '\0' && string[tamanho] != '\n';
         tamanho++)
        ;

    // Substitui o \n por \0 se for o caso
    if(string[tamanho] == '\n')
        string[tamanho] = '\0';
    for (i = 0, j = tamanho - 1; i < tamanho/2; i++, j--) {
        char aux = string[i];
        string[i] = string[j];
        string[j] = aux;
    }
    printf("A string invertida é: %s\n", string);
    return 0;
}
```



Sumário

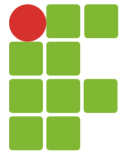
3 Exemplos
- Inversão
- Palíndromo
- Exemplos: Casamento de Padrões



Exemplos: Palíndromo

Problema

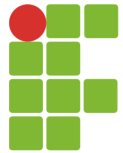
- Dada uma string, deve ser verificado se a string é um palíndromo.
- Um palíndromo é uma string é igual se lida da esquerda para direita ou da direita para esquerda.
- Exemplo de palíndromos: ovo, ele, socorrammesubinoonibusemmarrocos.



Exemplos: Inversão

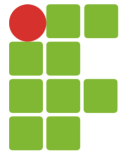
Estratégia

- A estratégia é comparar os caracteres do início com os do final: o primeiro é comparado com o último, o segundo com o penúltimo e assim por diante.
- Caso todos os caracteres casem, então a string é um palíndromo.



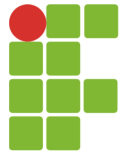
Exemplos: Inversão

```c
#include <stdio.h>

int main(void) {
    char string[100];
    int i, j, tamanho;
    printf("Digite a string a ser checada: ");
    fgets(string, 100, stdin);
    for (tamanho = 0; string[tamanho] != '\0' && string[tamanho] != '\n';
         tamanho++)
        ;
    // Substitui o \n por \0 se for o caso
    if (string[tamanho] == '\n')
        string[tamanho] = '\0';

    int palindromo = 1;
    for (i = 0, j = tamanho - 1; i < tamanho / 2 && palindromo; i++, j--) {
        if (string[i] != string[j])
            palindromo = 0;
    }

```



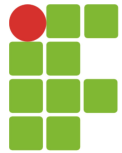
Exemplos: Inversão

```
21      if(palindromo)
22          printf("A string %s é um palíndromo.\n", string);
23      else
24          printf("A string %s NÃO é um palíndromo.\n", string);
25      return 0;
26  }
```



Sumário

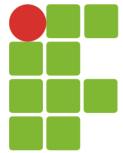
3 Exemplos
- Inversão
- Palíndromo
- Exemplos: Casamento de Padrões



Casamento de Padrões

Problema

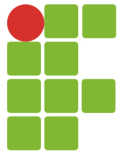
- O problema do casamento de padrões é um problema clássico em Ciência da Computação.
- Dada duas strings, T, o texto, e P, o padrão, identificar todas as posições de T em que P ocorre.
- Exemplo, se $T = abracadabra$ e $P = abra$, temos que P ocorre em T nas posições 0 e 7.



Exemplos: Casamento de Padrões

Estratégia

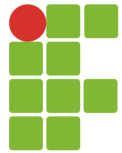
- Uma estratégia é tentar casar P com T a partir de cada posição do texto.
- Caso a comparação falhe em algum caractere, consideramos a próxima posição do texto.
- Se houve o casamento do padrão com o texto a partir da posição i, então achamos uma ocorrência iniciando de i.



Casamento de Padrões

Considere $T = xyxxyxyxyyxyxyxyyyxyxyxx$ e $P = xyxyyyxyxyxx$.

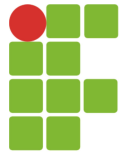
		0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2
	T :	x	y	x	x	y	x	y	x	y	y	x	y	x	y	x	y	y	x	y	x	y	x	x
0	P :	x	y	x	y																			
1	P :		x																					
2	P :			x	y																			
3	P :				x	y	x	y	y															
4	P :					x																		
5	P :						x	y	x	y	y	x	y	x	y	x	x							
6	P :							x																
7	P :								x	y	x													
8	P :									x														
9	P :										x													
10	P :											x	y	x	y	y								
11	P :												x											
12	P :													x	y	x	y	y	x	y	x	y	x	x



Exemplos: Casamento de Padrões

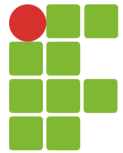
Estratégia

- Se T tem n caracteres e P tem m caracteres, com $n \geq m$, é preciso testar $n - m + 1$ posições do texto.
- Para cada uma destas posições, tentamos casar P com T.



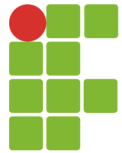
Exemplos: Casamento de Padrões

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

int main(void) {
    char texto[100];
    char padrao[100];
    int i, j;
    int tam_t, tam_p;
    printf("Digite o texto: ");
    fgets(texto, 100, stdin);
    printf("Digite o padrão: ");
    fgets(padrao, 100, stdin);
```



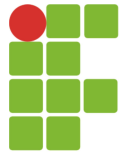
Exemplos: Casamento de Padrões

```
13
14      for (tam_t = 0; texto[tam_t] != '\0' && texto[tam_t] != '\n'; tam_t++)
15          ;
16      for (tam_p = 0; padrao[tam_p] != '\0' && padrao[tam_p] != '\n'; tam_p++)
17          ;
```



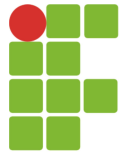
Exemplos: Casamento de Padrões

```
18
19      for (i = 0; i < tam_t - tam_p + 1; i++) {
20          int igual = 1;
21          for (j = 0; j < tam_p; j++) {
22              if (texto[i + j] != padrao[j]) {
23                  igual = 0;
24                  break;
25              }
26          }
27          if (igual) {
28              printf("O padrão ocorre na posição %d\n", i);
29          }
30      }
31      return 0;
32  }
```



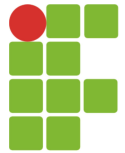
Sumário

4 string.h



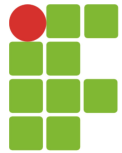
string.h

- O cabeçalho `<string.h>` dispõe de várias funções úteis na manipulação de strings.
- Abordaremos algumas destas funções.
- Naturalmente, todas elas dependem da existência do caractere `'\0'` ao final da string.



Sumário

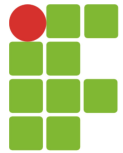
4 string.h
- strlen
- strcat e strncat
- strcpy e strncpy
- strcmp



Funções: strlen

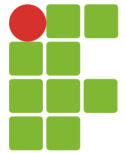
strlen

- A função `strlen` recebe uma string e nos dá o tamanho dela.
- Em outras palavras, ela retorna o número de caracteres até o primeiro `'\0'`.



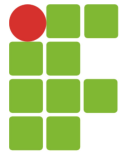
Funções: strlen

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

int main(void){
    char mensagem[10];
    printf("Digite uma mensagem: ");
    fgets(mensagem,10,stdin);
    int tamanho = strlen(mensagem);
    printf("O tamanho da string lida é %d\n",tamanho);
    return 0;
}
```



Sumário

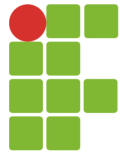
4 string.h
- strlen
- strcat e strncat
- strcpy e strncpy
- strcmp



Funções: strcat

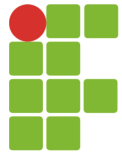
strcat

- A função `strcat` recebe duas strings e concatena a primeira com a segunda, armazenando o resultado concatenado na primeira string.
- Deve haver espaço suficiente, é preciso que a primeira string comporte a concatenação.



Funções: strcat

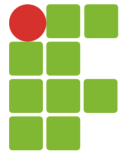
```c
#include <stdio.h>
#include <string.h>

int main(void){
    char string1[100] = "ola";
    char string2[100] = " mundo";
    strcat(string1,string2);
    printf("O resultado da concatenação é \"%s\".\n",string1);
    return 0;
}
```



Funções: strncat

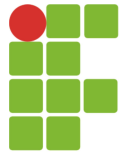
strncat

- Além de receber as duas strings, a função `strncat` também recebe o número de caracteres da segunda string que devem ser anexados à primeira string.



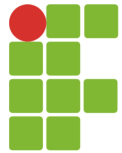
Funções: strcat

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

int main(void){
    char string1[100] = "ser";
    char string2[100] = " ou nao ser";
    strncat(string1,string2,7);
    printf("O resultado da concatenação é \"%s\".\n",string1);
    return 0;
}
```



Sumário

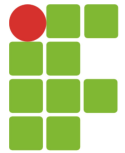
4 string.h
- strlen
- strcat e strncat
- strcpy e strncpy
- strcmp



Funções: strcpy

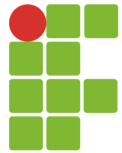
strcpy

- A função `strcpy` recebe duas strings e copia o conteúdo da segunda para a primeira.
- Deve haver espaço suficiente na primeira string para armazenar o conteúdo da segunda string.



Funções: strcpy

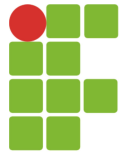
```c
#include <stdio.h>
#include <string.h>

int main(void) {
    char string1[100] = "abracadabra pe de cabra";
    char string2[100];
    strcpy(string2, string1);
    printf("O resultado da cópia é \"%s\".\n", string2);
    return 0;
}
```



Funções: strncpy

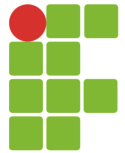
strncpy

- Funciona de maneira similar à `strcpy`, mas requer um terceiro parâmetro, que indica quantos caracteres da segunda string serão copiados para a primeira string.



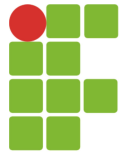
Funções: strncpy

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

int main(void) {
    char string1[100] = "abracadabra pe de cabra";
    char string2[100];
    strncpy(string2, string1,11);
    printf("O resultado da cópia é \"%s\".\n", string2);
    return 0;
}
```



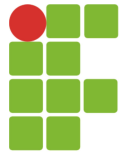
Sumário

4 string.h
- strlen
- strcat e strncat
- strcpy e strncpy
- strcmp



Funções: strcmp

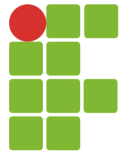
- Como fazer para comparar duas strings de acordo com a ordem estabelecida pela tabela ASCII?
- Em C, não podemos usar o operador `==` para comparar strings.
- Motivo: o nome de uma string é exatamente o endereço base do vetor, então, ao comparar dois vetores com `==`, efetivamente estamos comparando os seus endereços bases.
- Para comparar duas strings em C, devemos fazer uma comparação caractere por caractere entre elas.
- Felizmente, temos a função `strcmp` para isto.



Funções: strcmp

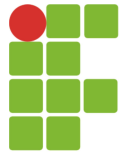
strcmp

- A função `strcmp` recebe duas strings e devolve um inteiro. O valor do inteiro devolvido corresponde ao resultado da comparação entre as strings:
  - < 0: a primeira string é menor do que a segunda.
  - $= 0$: as duas strings são iguais.
  - > 0: a segunda string é menor do que a primeira.



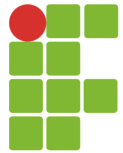
Funções: strcmp

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

int main(void) {
    char string1[100];
    char string2[100];
    printf("Digite a primeira string: ");
    fgets(string1, 100, stdin);
    printf("Digite a segunda string: ");
    fgets(string2, 100, stdin);
    int retorno = strcmp(string1, string2);
    if (retorno < 0) {
        printf("A primeira string é menor que a segunda.\n");
    }
    else if (retorno == 0) {
        printf("As strings são iguais!\n");
    }
    else {
        printf("A segunda string é menor que a primeira.\n");
    }
    return 0;
}
```



Comparação entre Strings

- O exemplo a seguir não funcionará, pois estamos comparando o endereço das strings, e não o conteúdo das mesmas.



Comparação entre Strings

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

int main(void) {
    char string1[100];
    char string2[100];
    printf("Digite a primeira string: ");
    fgets(string1, 100, stdin);
    printf("Digite a segunda string: ");
    fgets(string2, 100, stdin);
    if (string1 == string2) {
        printf("As strings são iguais.\n");
    }
    else {
        printf("As strings são diferentes.\n");
    }
    return 0;
}
```