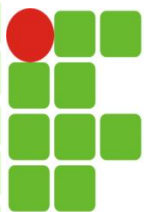


INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SUL-RIO-GRANDENSE
Campus Passo Fundo

ALGORITMOS II

Prof. Adilso Nunes de Souza



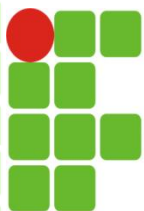
STRING

- Um caractere é uma letra, um numeral, uma pontuação ou um símbolo.
- Uma string consiste em um conjunto de caracteres, sendo, portanto, utilizado para armazenar textos em geral.
- Como um char armazena apenas um caractere, é necessário ter uma maneira de armazenar uma sequência de caracteres



STRING

- Na linguagem C++ existem diferentes formas de manipular string (texto), uma delas é utilizado uma matriz de caracteres. Uma string é uma matriz tipo char que termina com '\0'. Por essa razão uma string deve conter uma posição a mais do que o número de caracteres que se deseja.
- Constantes strings são uma lista de caracteres que aparecem entre aspas;



STRING

- Definição de uma *string*:

`char nome[7]`

- última posição ocupada pelo `'\0'`

- Atribuição entre strings

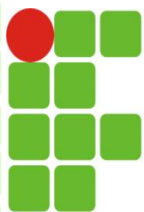
- Uma das maneiras de fazer atribuição é de posição por posição, strings sempre começam na posição zero (0)

`nome[0] = 'A';`

`nome[1] = 'd';`

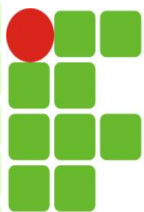
nome

A	d	i	l	s	o	\0
---	---	---	---	---	---	----



STRING

- **Função gets()**
 - É utilizada para leitura de uma string através do dispositivo padrão, até que o ENTER seja pressionado. A função gets() não testa limites na matriz em que é chamada.
 - Sintaxe: *gets(nome_matriz);*



STRING

- *Exemplo:*

```
char nome[50];
```

```
main()
```

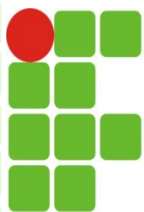
```
{
```

```
    cout<<"Digite seu nome: ";
```

```
    gets(nome);
```

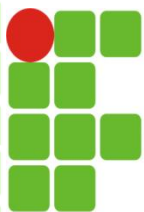
```
    getchar();
```

```
}
```



STRING

- **Método getline**
 - É utilizada para leitura de uma string através do dispositivo padrão. Diferente da função `gets()` o método `getline` lê somente o número de caracteres especificado.
 - Sintaxe: *`cin.getline(vetor_char, tamanho);`*



STRING

- *Exemplo:*

```
char nome[50];
```

```
main()
```

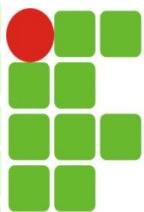
```
{
```

```
    cout<<"Digite seu nome: ";
```

```
    cin.getline(nome, sizeof(nome))
```

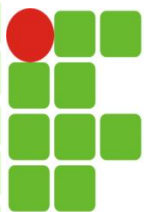
```
    getchar();
```

```
}
```

STRING

- **Função puts()**
 - Escreve o seu argumento no dispositivo padrão de saída (vídeo), coloca um '\n' no final.
 - Sintaxe: `puts(nome_do_vetor);`
- Exemplo:
`puts(nome);`



STRING

- **Função strcpy()**

- Copia o conteúdo de uma string.
- Sintaxe: `strcpy(destino, origem);`

- Ex:

```
char nome[50], a[50];
```

```
main()
```

```
{
```

```
    cout<<"Digite seu nome: ";
```

```
    gets(nome);
```

```
    strcpy(a, nome);
```

```
    puts(a);
```

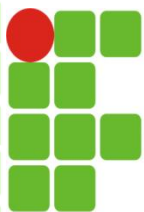
```
    getchar();
```

```
}
```



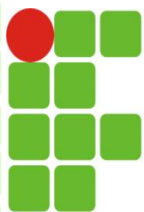
STRING

- **Função strcat()**
 - Concatena duas strings agrupando o resultado no primeiro argumento. Não verifica tamanho.
 - Sintaxe: `strcat(string1,string2);`



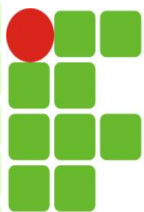
STRING

```
char nome[50], a[50];  
main()  
{  
    cout<<"Digite seu nome: ";  
    gets(nome);  
    strcpy(a,"nome: ");  
    strcat(a,nome);  
    puts(a);  
    getchar();  
}
```



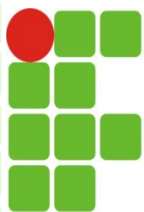
STRING

- **Função strcmp()**
 - Compara duas strings, se forem iguais devolve 0. Geralmente usado em combinações com testes condicionais.
 - Sintaxe: `strcmp(s1,s2);`



STRING

```
char nome[50];  
main()  
{  
    cout<<"Digite seu nome: ";  
    gets(nome);  
    if((strcmp(nome,"Adilso"))==0)  
        cout<<"Nome correto";  
    else  
        cout<<"Erro";  
    getchar();  
}
```

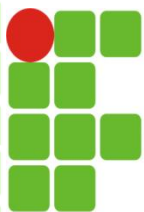


STRING

- **Função strlen()**

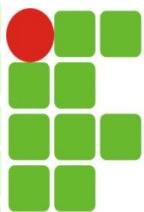
- Retorna o tamanho de uma string
- Sintaxe: strlen(string)
- Exemplo:

```
cout << "seu nome tem: " << strlen(nome) << "
    caracteres";
```



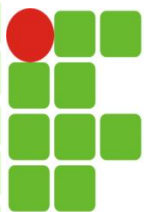
BIBLIOTECA STRING

- Também é possível utilizar `cin` e `cout` para ler e escrever strings, assim como fazemos com variáveis simples. Basicamente, `cin` e `cout` tratam strings como se fossem variáveis simples, facilitando sua manipulação e seu uso em programas.



BIBLIOTECA STRING

- Inicialmente é necessário incluir a biblioteca do C++ que possibilita esta manipulação:
`#include <cstring>`
- Declaramos strings da mesma maneira que declaramos variáveis, explicitando o tipo da variável (no caso, string) e seu nome.
`string <nome da string>;`



BIBLIOTECA STRING

- C++ possui uma série de facilidades para a inicialização de strings. Cada um desses diferentes métodos é chamado de “inicializador” de uma string.

`string s1;`

Cria uma string vazia, chamada s1. Esta é a inicialização default de uma string, toda string criada dessa forma está vazia.



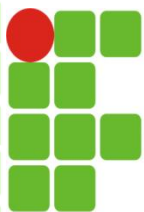
BIBLIOTECA STRING

`string s2 (s1);`

Cria a string s2 como uma cópia de outra string (nesse caso, s1).

`string s2 (“Esta é uma string literal”);`

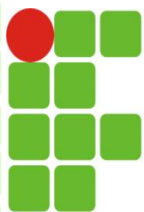
Cria a string s2 como uma cópia da string literal entre os parênteses. Cuidar para sempre usar aspas duplas neste caso.



BIBLIOTECA STRING

`string s2 (x, 'b');`

Cria a string s2, que contém x cópias do caractere entre aspas (no caso, b).



BIBLIOTECA STRING

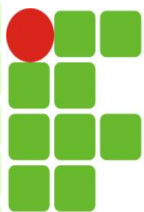
- Para ler uma entrada de dados do usuário utilizamos do método `getline`, que em se tratando de string apresenta uma sintaxe diferenciada:

```
string nome;
```

```
cout << "Informe o nome: ";
```

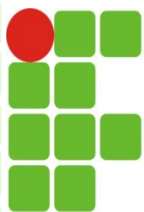
```
getline(cin, nome);
```

OBS: desta forma é possível ler sem ser interrompido quando encontrar um espaço em branco.



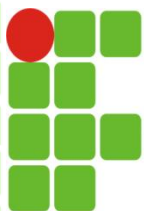
OPERAÇÕES COM STRING

- C++ possui uma série de funções e operações próprias para strings. A tabela abaixo resume as operações mais utilizadas (s é uma string qualquer):
- `s.empty()`
 - Função que retorna verdadeiro se a string está vazia, e falso caso contrário.



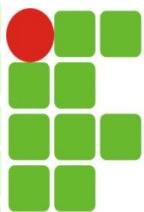
OPERAÇÕES COM STRING

- `s.size ()`
 - Função que retorna o tamanho em caracteres da string.
- `s [n]`
 - Acessa um elemento da string. Funciona exatamente com um elemento de uma matriz.
- `s1 + s2`
 - Concatena duas strings.



OPERAÇÕES COM STRING

- `s1 = s2`
 - Atribui o conteúdo de `s2` na string `s1`.
- `s1 == s2`
 - Testa a igualdade entre `s1` e `s2` (retorna verdadeiro se as duas strings forem iguais). Duas strings são consideradas iguais se elas tiverem o mesmo número de caracteres e seus caracteres forem iguais, sensível a maiúsculas e minúsculas.



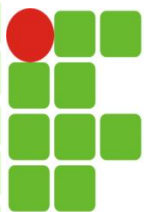
CONVERTER PARA MAIÚSCULO

❖ **toupper(x)**

Transforma o caractere para maiúsculo.

```
string teste("ADILSO Souza");  
for(int x = 0; x <= teste.size();x++)  
    teste[x] = toupper(teste[x]);  
cout << teste;
```

Exibe: ADILSO SOUZA



CONVERTER PARA MINÚSCULO

❖ **tolower(x)**

Transforma o caractere para minúsculo.

```
string teste("ADILSO Souza");  
for(int x = 0; x <= teste.size();x++)  
    teste[x] = tolower(teste[x]);  
cout << teste;
```

Exibe: adilso souza



VERIFICAR SE É LETRA OU NÚMERO

❖ **isalnum (x)**

Retorna verdadeiro (1) caso x for uma letra ou um número.

```
string teste("ab12#@");
```

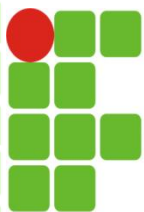
```
if(isalnum(teste[4]))
```

```
    cout << "Letra ou número";
```

```
else
```

```
    cout << "Caracter especial";
```

Exibe: Caracter especial.



VERIFICAR SE É UMA LETRA

❖ isalpha (x)

Retorna verdadeiro (1) caso x for uma letra.

```
string teste("ab12#@");
```

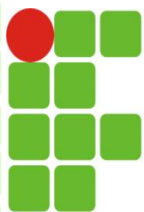
```
if(isalpha(teste[0]))
```

```
    cout << "É uma Letra";
```

```
else
```

```
    cout << "Outro Character";
```

Exibe: É uma letra.



VERIFICAR SE É UM NÚMERO

❖ isdigit (x)

Retorna verdadeiro (1) caso x for um número.

```
string teste("ab12#@");
```

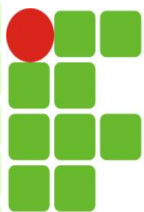
```
if(isdigit(teste[3]))
```

```
    cout << "É um número";
```

```
else
```

```
    cout << "Não é um número";
```

Exibe: É um número



VERIFICAR SE É MINÚSCULO

❖ **islower (x)**

Retorna verdadeiro (1) caso x for uma letra minúscula.

```
string teste("abAB");
```

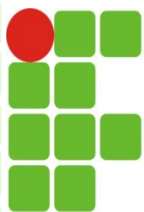
```
if(islower(teste[0]))
```

```
    cout << "É minúscula";
```

```
else
```

```
    cout << "Não é minúscula";
```

Exibe: É minúscula



VERIFICAR SE É MAIÚSCULO

❖ **isupper (x)**

Retorna verdadeiro (1) caso x for uma letra maiúscula.

```
string teste("abAB");
```

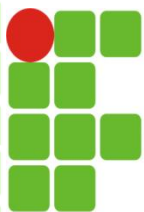
```
if(isupper(teste[3]))
```

```
    cout << "É maiúscula";
```

```
else
```

```
    cout << "Não é maiúscula";
```

Exibe: É maiúscula



VERIFICAR SE É UM ESPAÇO

❖ **isspace (x)**

Retorna verdadeiro (1) caso x for um espaço em branco.

```
string teste("Adilso Souza");
```

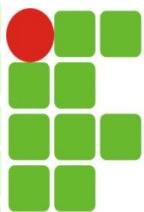
```
if(isspace(teste[6]))
```

```
    cout << "É um espaço em branco";
```

```
else
```

```
    cout << "Não é um espaço em branco";
```

Exibe: É um espaço em branco



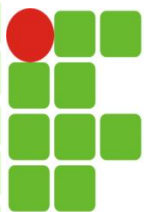
CONVERTER PARA INTEIRO

❖ atoi (vetor char)

Função que converte um vetor de char em inteiro.

OBS:

- Os espaços em branco no início da string são ignorados.
- Se a string é composta por conjuntos de caracteres separados por espaços em branco, apenas o primeiro conjunto de caracteres é considerado na conversão.
- Se a string é um conjunto vazio ou possui caracteres não numéricos, a função retorna o valor zero.
- Geralmente utilizado para transformar valores lidos em arquivos texto.



CONVERTER PARA INTEIRO

Exemplos:

```
int v;
```

```
v = atoi("1.455"); // v terá o valor 1
```

```
v = atoi(" 3 1 4 8 9"); // v terá o valor 3
```

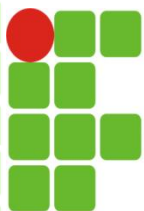
```
v = atoi("A b4 89"); // v terá o valor 0
```



CONVERTER PARA FLOAT

❖ **atof (vetor char)**

Função que converte um vetor de char em um valor float seguindo as mesmas características do atoi.



REFERÊNCIAS

- ARAÚJO, Jáiro – Dominando a Linguagem C. Editora Ciência Moderna.
- PEREIRA, Silvio do Lago. Estrutura de Dados Fundamentais: Conceitos e Aplicações, 12. Ed. São Paulo, Érica, 2008.
- LORENZI, Fabiana. MATTOS, Patrícia Noll de. CARVALHO, Tanisi Pereira de. Estrutura de Dados. São Paulo: Ed. Thomson Learning, 2007.
- VELOSO, Paulo. SANTOS, Celso dos. AZEVEDO, Paulo. FURTADO, Antonio. Estrutura de dados. Rio de Janeiro: Ed. Elsevier, 1983 27ª reimpressão.