Aula 9:
Strings

Disciplina: Fundamentos de Programação

Prof. Luiz Olmes
olmes@unifei.edu.br

Nas aulas anteriores...

- **O QUE JÁ ESTUDAMOS?**
  - Algoritmos.
  - Linguagem C.
  - Variáveis, operadores e tipos.
  - Estruturas de controle condicionais.
  - Estruturas de controle iterativas.
  - Vetores.
  - Matrizes.
- **OBJETIVOS:**
  - Definição.
  - Leitura e manipulação.
  - Conversão:
    - Strings em números.
    - Números em strings.
  - Funções de manipulação de caracteres.
  - Funções de manipulação de strings.

Definição

- `String` é o nome dado a uma sequência de caracteres dispostos de modo sequencial na memória do computador.

- Também chamadas de arrays (vetores) de caracteres.

- Declaração:

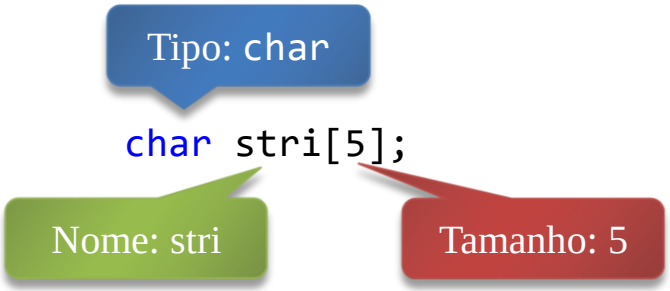
```
char stri[5];
```

Definição

- `String` é o nome dado a uma sequência de caracteres dispostos de modo sequencial na memória do computador.

- Também chamadas de arrays (vetores) de caracteres.

- Declaração:



Definição

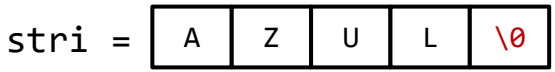
- `String` é o nome dado a uma sequência de caracteres dispostos de modo sequencial na memória do computador.

- Também chamadas de arrays (vetores) de caracteres.

- Declaração:

```
char stri[5];
```

- O último caractere de uma string é um símbolo especial que indica sua terminação: `\0`.



Inicialização de strings

- Strings podem ter seu valor inicializado da mesma forma que vetores de tipos numéricos, como `int` ou `float`.

```
char s[10] = { ‘A’, ‘Z’, ‘U’, ‘L’, ‘\0’ };
```

Inicialização de strings

- Strings podem ter seu valor inicializado da mesma forma que vetores de tipos numéricos, como `int` ou `float`.

```
char s[10] = { ‘A’, ‘Z’, ‘U’, ‘L’, ‘\0’ };
```

- Ou através da utilização de aspas duplas:

```
char s[10] = “AZUL”;
```

Inicialização de strings

- Strings podem ter seu valor inicializado da mesma forma que vetores de tipos numéricos, como `int` ou `float`.

```
char s[10] = { ‘A’, ‘Z’, ‘U’, ‘L’, ‘\0’ };
```

- Ou através da utilização de aspas duplas:

```
char s[10] = “AZUL”;
```

O terminador `\0` é inserido automaticamente.

Leitura de strings

- A leitura de strings é realizada através da função `scanf()`.

- Existem dois especificadores (ou máscaras) distintos para realizar a leitura, dependendo do conteúdo da string:
  - Strings que não contém espaços: `%s`
  - Strings contendo espaços: `%[^\n]` (leia tudo o que vier na entrada, até encontrar um `\n`)

Leitura de strings

- A leitura de strings é realizada através da função `scanf()`.

- Existem dois especificadores (ou máscaras) distintos para realizar a leitura, dependendo do conteúdo da string:
  - Strings que não contém espaços: `%s`
  - Strings contendo espaços: `%[^\n]` (leia tudo o que vier na entrada, até encontrar um `\n`)

- Exceção no uso do `scanf()`: na leitura de strings, a variável (vetor de `char`) ***NÃO*** deve vir acompanhada do `&`.
  - *Por quê? Veremos na aula de ponteiros e funções!*

- Independente da string conter ou não espaços, o especificador usado no comando `printf()` é sempre o `%s`.

Leitura de strings

- Exemplo 1: faça um programa que leia uma palavra e mostre a palavra digitada.

- Exemplo 2: faça um programa que leia uma frase (com espaços) e mostre a palavra digitada.

- Exemplo 3: faça um programa que leia uma palavra ou frase e a mostre. Não deixe o usuário inserir mais que 10 caracteres.

Leitura de strings: `scanf()`

- Lendo uma palavra (sem espaços):

```
1. #include <stdio.h> // printf(), scanf()
2.
3. int main()
4. {
5.     char palavra[25];
6.
7.     printf(“Digite a palavra: ”);
8.
9.     scanf(“%s”, palavra);
10.
11.    printf(“Palavra digitada = %s \n”, palavra);
12.
13.    return 0;
14.}
```

Leitura de strings: `scanf()`

- Lendo uma palavra (sem espaços):

```
1.  #include <stdio.h> // printf(), scanf()
2.
3.  int main()
4.  {
5.      char palavra[25];
6.
7.      printf("Digite a palavra: ");
8.
9.      scanf("%s", palavra);
11.     printf("Palavra digitada = %s \n", palavra);
12.
13.     return 0;
14. }
```

Declaração da string

Variável sem &

Máscara de leitura

Leitura de strings: `scanf()`

- Lendo uma palavra (sem espaços):

```
1.  #include <stdio.h> // printf(), scanf()
2.
3.  int main()
4.  {
5.      char palavra[25];
6.
7.      printf("Digite a palavra: ");
8.
9.      scanf("%24s", palavra);
10.
11.     printf("Palavra digitada = %s \n", palavra);
12.
13.     return 0;
14. }
```

Garantindo que o usuário não entre com mais caracteres do que o espaço máximo.

Manipulação de cadeias de caracteres

- A manipulação de cadeias de caracteres (vetor de `char`) é realizada de modo análogo à manipulação de vetores de tipos numéricos (`int`, `float`, ...).

Manipulação de cadeias de caracteres

- A manipulação de cadeias de caracteres (vetor de `char`) é realizada de modo análogo à manipulação de vetores de tipos numéricos (`int`, `float`, ...).

- Exemplo 4: fazer um programa que leia uma frase (vetor de `char`) e copie a frase lida para outro vetor de `char`.

- Exemplo 5: fazer um programa que leia uma frase e a imprima sem suas vogais.

- Exemplo 6: fazer um programa que leia uma frase e conte o número de espaços.

- Exemplo 7: fazer um programa que leia uma frase e a codifique da seguinte forma: substitua A por 1, E por 2, I por 3, O por 4, U por 5. Mostre o resultado.

Conversão de strings em números

- Ao se desenvolver software, muitas vezes os dados são recebidos pelo programa no formato de strings.
  - Parâmetros da função `main()`.
  - Leitura de dados de um arquivo.

- Entretanto, para que dados representando grandezas numéricas possam ser manipulados como tal, eles devem ser convertidos do formato de string para um formato numérico.

- As bibliotecas `stdio.h` e `stdlib.h` definem funções que realizam essa conversão.

Conversão de strings em números

- Função `sscanf()`:
  - Está definida na biblioteca `stdio.h`.
  - Realiza a leitura formatada de dados a partir de uma string recebida.

- Sintaxe: `sscanf(str, "Expressão", Argumentos);`
  - `str[]`: a string (vetor de char) que a função processa como fonte de entrada de dados.
  - Expressão: são códigos de formatação, precedidos pelo símbolo de porcento (%). Especificam o formato (tipagem) dos dados de entrada. O formato segue o mesmo padrão definido para a função `scanf()`.
  - Argumentos: consistem nos endereços das variáveis.

- A função devolve o número de itens na lista de argumentos que foram lidos com sucesso.

Conversão de strings em números: `sscanf()`

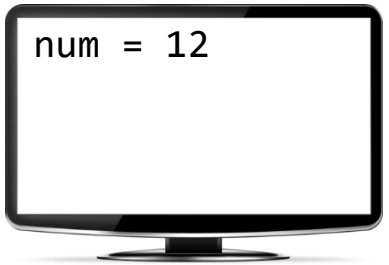
- Conversão de string para inteiro:

```
1. #include <stdio.h> // printf(), sscanf()
2.
3. int main()
4. {
5.     char strNum[10] = “12”;
6.     int num;
7.
8.     sscanf(strNum, “%d”, &num);
9.     printf(“num = %d \n”, num);
10.
11.    return 0;
12.}
```

Conversão de strings em números: sscanf()

- Conversão de string para inteiro:

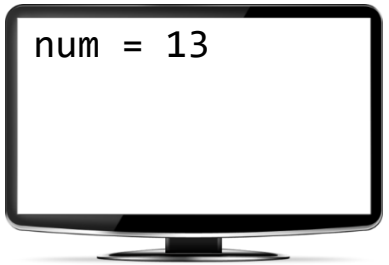
```
1. #include <stdio.h> // printf(), sscanf()
2.
3. int main()
4. {
5.     char strNum[10] = “12”;
6.     int num;
7.
8.     sscanf(strNum, “%d”, &num);
9.     printf(“num = %d \n”, num);
10.
11.    return 0;
12.}
```



Conversão de strings em números: `sscanf()`

- Conversão de string para inteiro longo:

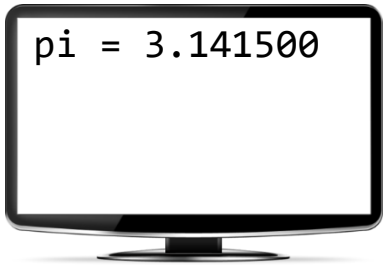
```
1. #include <stdio.h> // printf(), sscanf()
2.
3. int main()
4. {
5.     char strNum[10] = “13”;
6.     long long int num;
7.
8.     sscanf(strNum, “%lld”, &num);
9.     printf(“num = %lld \n”, num);
10.
11.    return 0;
12.}
```



Conversão de strings em números: sscanf()

- Conversão de string para ponto flutuante:

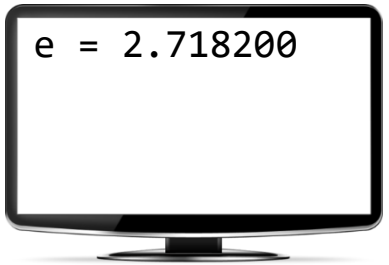
```
1. #include <stdio.h> // printf(), sscanf()
2.
3. int main()
4. {
5.     char strPi[10] = “3.1415”;
6.     float pi;
7.
8.     sscanf(strPi, “%f”, &pi);
9.     printf(“pi = %f \n”, pi);
10.
11.    return 0;
12.}
```



Conversão de strings em números: sscanf()

- Conversão de string para ponto flutuante de precisão dupla:

```
1.  #include <stdio.h> // printf(), sscanf()
2.
3.  int main()
4.  {
5.      char strE[10] = “2.7182”;
6.      double e;
7.
8.      sscanf(strE, “%lf”, &e);
9.      printf(“e = %lf \n”, e);
10.
11.     return 0;
12. }
```



Conversão de strings em números

- Funções `atoi()`, `atoll()`, e `atof()`:
  - Estão definidas na biblioteca `stdlib.h`.
  - Realizam a conversão de strings para tipos numéricos.
  - Recebem como parâmetro a string a ser convertida.
  - Devolvem o valor no formato numérico do respectivo tipo.

- `atoi(char[])`:
  - Converte uma string para um número inteiro.

- `atoll(char[])`:
  - Converte uma string para um número inteiro longo.

- `atof(char[])`:
  - Converte uma string para um número de ponto flutuante de precisão dupla.

Conversão de strings em números

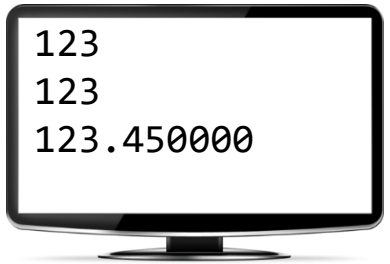
- Funções `atoi()`, `atoll()`, e `atof()`:

```
1. #include <stdio.h> // printf()
2. #include <stdlib.h> // atoi(), atoll(), atof()
3. int main()
4. {
5.     char strNum[10] = “123.45”;
6.
7.     int numI = atoi(strNum);
8.     long long int numL = atoll(strNum);
9.     double numD = atof(strNum);
10.
11.    printf(“%d \n%lld \n%lf \n”, numI, numL, numD);
12.
13.    return 0;
14. }
```

Conversão de strings em números

- Funções `atoi()`, `atoll()`, e `atof()`:

```
1. #include <stdio.h> // printf()
2. #include <stdlib.h> // atoi(), atoll(), atof()
3. int main()
4. {
5.     char strNum[10] = “123.45”;
6.
7.     int numI = atoi(strNum);
8.     long long int numL = atoll(strNum);
9.     double numD = atof(strNum);
10.
11.    printf(“%d \n%lld \n%lf \n”, numI, numL, numD);
12.
13.    return 0;
14. }
```

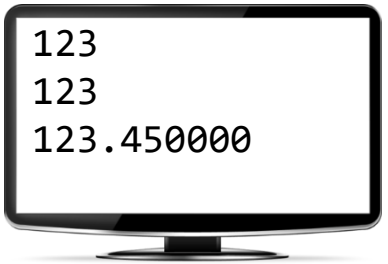


Conversão de strings em números

- Funções `atoi()`, `atoll()`, e `atof()`:

```
1.  #include <stdio.h> // printf()
2.  #include <stdlib.h> // atoi(), atoll(), atof()
3.  int main()
4.  {
5.      char strNum[10] = “123.45”;
6.
7.      int numI = atoi(strNum);
8.      long long int numL = atoll(strNum);
9.      double numD = atof(strNum);
10.
11.     printf(“%d \n%lld \n%lf \n”, numI, numL, numD);
12.
13.     return 0;
14. }
```

Perdem a parte decimal



Conversão de números em strings

- Uma vez que é possível converter strings em tipos numéricos, a operação inversa também deve ser possível de ser realizada.

- Existem diversas funções específicas que realizam esta conversão:
  - `itoa()`: converte uma variável inteira para string.

- Entretanto, funções como `itoa()`, `strToFloat()`, etc., não fazem parte da biblioteca padrão da linguagem C, e sim de implementações de compiladores específicos.

- A função pertencente ao padrão da linguagem C que realiza este tipo de conversão é `sprintf()`.

Conversão de números em strings

- Função `sprintf()`:
  - Está definida na biblioteca `stdio.h`.
  - Realiza a escrita de dados em uma string recebida como parâmetro.
  - É a função dual a `sscanf()`.

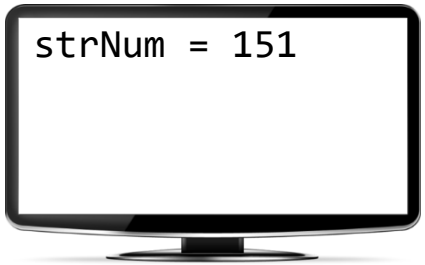
- Sintaxe: `sprintf(str, "Expressão", Argumentos);`
  - `str[]`: a string que receberá os dados. Deve ter tamanho suficiente.
  - Expressão: os dados que serão escritos. Pode conter instruções que especificam o formato dos dados. O formato segue o mesmo padrão definido para a função `printf()`.
  - Argumentos: consistem nos valores repassados para a Expressão.

- A função devolve o número de caracteres que foram escritos.

Conversão de números em strings: sprintf()

- Conversão de inteiro para string:

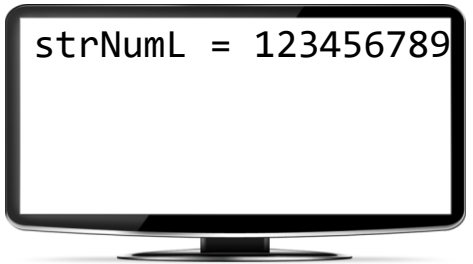
```
1.  #include <stdio.h> // printf(), sprintf()
2.  int main()
3.  {
4.      int num = 151;
5.      char strNum[10];
6.
7.      sprintf(strNum, “%d”, num);
8.
9.      printf(“strNum = %s \n”, strNum);
10.
11.     return 0;
12. }
```



Conversão de números em strings: sprintf()

- Conversão de inteiro longo para string:

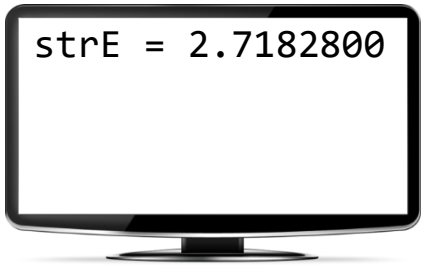
```
1.  #include <stdio.h> // printf(), sprintf()
2.  int main()
3.  {
4.      long long int num = 123456789;
5.      char strNumL[10];
6.
7.      sprintf(strNumL, “%lld”, num);
8.
9.      printf(“strNumL = %s \n”, strNumL);
10.
11.     return 0;
12. }
```



Conversão de números em strings: sprintf()

- Conversão de ponto flutuante para string:

```
1. #include <stdio.h> // printf(), sprintf()
2. int main()
3. {
4.     float e = 2.7182;
5.     char strE[10];
6.
7.     sprintf(strE, “%f”, e);
8.
9.     printf(“strE = %s \n”, strE);
10.
11.    return 0;
12.}
```



Funções de manipulação de caracteres

- A biblioteca padrão da linguagem C possui funções especialmente desenvolvidas para a manipulação de caracteres definidas no arquivo de cabeçalho `<ctype.h>`. Algumas delas são:

- `islower`: retorna `true` se o caractere é uma letra minúscula.

- `isupper`: retorna `true` se o caractere é uma letra maiúscula.

- `tolower`: retorna o caractere convertido para uma letra minúscula.

- `toupper`: retorna o caractere convertido para uma letra maiúscula.

Funções de manipulação de strings

- A biblioteca padrão da linguagem C possui funções especialmente desenvolvidas para a manipulação de strings definidas no arquivo de cabeçalho `<string.h>`. As mais utilizadas são:

- `strlen`: usada para obter o tamanho de uma string.

- `strcmp`: usada para comparar duas strings.

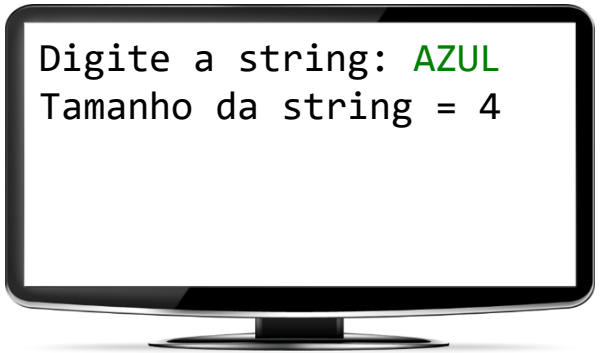
- `strcpy`: usada para copiar uma string em outra.

- `strcat`: usada para concatenar duas strings.

Funções de manipulação de strings: strlen

- A função strlen(char[]) retorna a quantidade de caracteres existentes antes do terminador \0.

```
1. #include <stdio.h> // printf()
2. #include <string.h> // strlen()
3. int main()
4. {
5.     char c[50];
6.
7.     printf("Digite a string: ");
8.     scanf("%s", c);
9.
10.    printf("Tamanho da string = %d \n", strlen(c) );
11.
12.    return 0;
13.}
```



Funções de manipulação de strings: `strcmp`

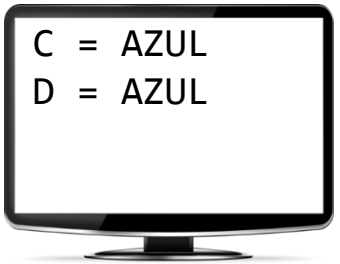
- A função `strcmp(char[], char[])` retorna um valor menor que zero se a primeira string é lexicograficamente menor que a segunda, zero se ambas são iguais ou maior que zero se a primeira é lexicograficamente maior que a segunda.

```
1. #include <stdio.h> // printf()
2. #include <string.h> // strcmp()
3. int main()
4. {
5.     char c[20] = “AZUL”, d[20] = “AZUL”, e[20] = “VERDE”;
6.
7.     if ( strcmp(c, d) == 0) printf(“C e D sao iguais \n”);
8.     else printf(“C e D nao sao iguais \n”);
9.
10.    if ( strcmp(c, e) == 0) printf(“C e E sao iguais \n”);
11.    else printf(“C e E nao sao iguais \n”);
12.
13.    return 0;
14. }
```

Funções de manipulação de strings: strcpy

- A função strcpy(char destino[], char origem[]) copia os caracteres da string de origem na string de destino. A string de destino deve ter tamanho suficiente para receber os caracteres da origem.

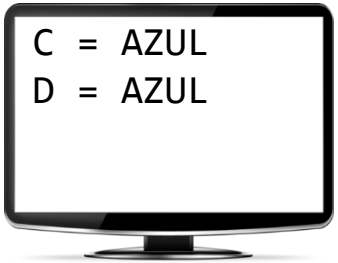
```
1.  #include <stdio.h> // printf()
2.  #include <string.h> // strcpy()
3.  int main()
4.  {
5.      char c[20] = “AZUL”, d[20];
6.
7.      strcpy(d, c);
8.
9.      printf(“C = %s \n”, c);
10.     printf(“D = %s \n”, d);
11.
12.     return 0;
13. }
```



Funções de manipulação de strings: strcpy

- A função `strcpy(char destino[], char origem[])` copia os caracteres da string de origem na string de destino. A string de destino deve ter tamanho suficiente para receber os caracteres da origem.

```
1.  #include <stdio.h> // printf()
2.  #include <string.h> // strcpy()
3.  int main()
4.  {
5.      char c[20] = “AZUL”, d[20];
6.
7.      strcpy(d, c);
8.
9.      printf(“C = %s \n”, c);
10.     printf(“D = %s \n”, d);
11.
12.     return 0;
13. }
```



Não é possível atribuir strings diretamente em C:

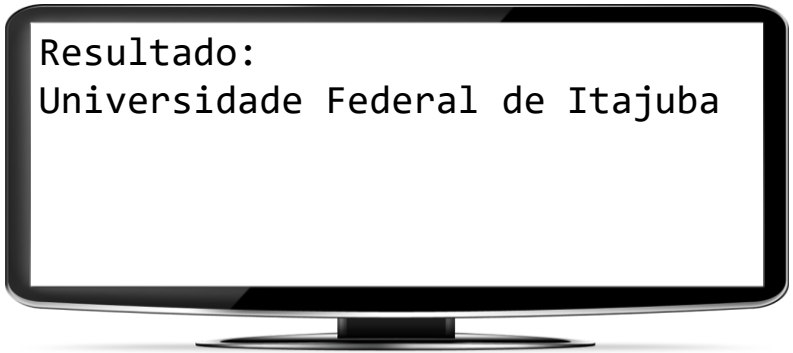
```
char c[20] = “AZUL”, d[20];
d = c; // não funciona!
```

É preciso usar `strcpy` ou fazer a cópia com laços.

Funções de manipulação de strings: strcat

- A função strcat(char destino[], char origem[]) adiciona os caracteres da string de origem ao final da string de destino (concatena).

```
1.  #include <stdio.h> // printf()
2.  #include <string.h> // strcat()
3.  int main()
4.  {
5.      char a[50] = “Universidade ”;
6.      char b[20] = “Federal ”;
7.      char c[20] = “de ”;
8.      char d[20] = “Itajuba”;
9.
10.     strcat(a, b);
11.     strcat(a, c);
12.     strcat(a, d);
16.     printf(“Resultado:\n%s \n”, a);
17.     return 0;
18. }
```



Dúvidas?

Aula 9:
Strings

Disciplina: Fundamentos de Programação

Prof. Luiz Olmes
olmes@unifei.edu.br