

INF101202

Algoritmos e Programação

Modalidade Ead – Turma H

Material de apoio: *Processamento de caracteres*



Processamento de 1 (um) caractere

Leitura e escrita de um caractere

Seja a variável **caract**:

char caract;

Leitura

⇒ Com scanf (função genérica de leitura)

```
scanf("%c", &caract);
```

Atenção!!!: por vezes é necessário antes do %c colocar-se um espaço em branco, para que um caractere de fim de linha, de espaço ou de tabulação, ainda existente no *buffer* do dispositivo, não interfira na leitura que se deseja realizar.

⇒ Com getchar (função específica para leitura de caracteres)

```
ch = getchar( ); (ver exemplo mais completo a seguir)
```

Escrita

⇒ Com printf

```
printf("O caractere lido: %c\n", caract);
```

Função `getchar()`

Função `getchar()`: específica para leitura de caracteres.
Lê um caractere.

Ex.:

```
//testa funcao getchar
```

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

```
#include <stdlib.h>
```

```
#define MAXIMO 20
```

```
int main( )
```

```
{
```

```
    char ch;
```

```
    system("color 71");
```

```
    printf("Forneca um caractere: ");
```

```
    ch = getchar( );
```

```
    printf("O caractere digitado: %c\n",ch);
```

```
    system("pause");
```

```
    return 0;
```

```
}
```

Forma geral:

`getchar()`



```
C:\ G:\ARRAYSUNI\stringgetchar.exe
```

```
Forneca um caractere: &
```

```
O caractere digitado: &
```

```
Pressione qualquer tecla para continuar. . . _
```



*Processamento de **n** caracteres
(cadeias de caracteres ou *strings*)*

Problema: Ler o nome de 30 alunos e a nota correspondente. Calcular e informar a média da turma, seguido dos nomes dos alunos com nota inferior à média da turma.

Pergunta: Como obter e armazenar nomes ?

Resposta: Em cadeias de caracteres ou *strings*.

Em C, conforme visto, existe um tipo para armazenar explicitamente caracteres, o tipo *char*, que armazena 1 (um) caractere a cada vez, mas **NÃO EXISTE UM TIPO BÁSICO PARA ARMAZENAR CADEIAS DE CARACTERES** (ou seja, *strings*).

Para armazenar-se múltiplos caracteres em C, deve-se usar *vetores* de caracteres (2 a *n* caracteres), com um caractere terminador '\0' como último elemento válido.

STRINGS em C

Strings são cadeias ou sequências de caracteres.

Como em C não há um tipo de dado específico para **strings** elas são representadas como vetores de caracteres que têm como característica distintiva apresentar o caractere delimitador de fim '\0', que corresponde ao caractere de posição 0 na tabela ASCII.

Então é importante salientar que toda *string* em C é um vetor de caracteres, mas nem todo vetor de caracteres em C é uma *string*.

Exemplo de um vetor de caracteres **vet** de 9 posições, apenas com as posições de 0 a 6 ocupadas:

	0	1	2	3	4	5	6	7	8
vet	B	r	a	s	i	l	\0	?	?

STRINGS em C (cont.)

Forma geral de declaração:

`char nome_da_string[tamanho];`

Ex.: `char primeiro_nome[15], ultimo_sobrenome[25];`

O tamanho de uma string deve sempre prever a inclusão do caractere delimitador `'\0'`.

Assim, antes de declarar-se uma variável para armazenar strings, deve-se somar os caracteres do maior texto a ser nela armazenado e acrescentar-se um(1) ao total obtido.

Exemplo: sejam duas variáveis tipo char `dia_da_semana [?]` e `mes [?]`.

O dia da semana com maior número de caracteres é `segunda-feira` (13) e o mês com maior número de caracteres é `fevereiro` (9), logo essas variáveis devem ser declaradas no mínimo como:

`char dia_da_semana[14], mes[10];`

Inicialização de *strings* em C

A inicialização de *strings* segue as mesmas regras da inicialização de arranjos em geral e pode ser feita de várias maneiras conforme exemplos a seguir.

Exemplo 1: `char primeiro_nome[15] = "Ana";`

Com esse tipo de inicialização, o sistema insere os caracteres indicados entre as aspas duplas no vetor `primeiro_nome`, a partir da posição 0, e ao final dos mesmos coloca, na próxima posição do arranjo, o caractere `'\0'`.

Exemplo 2: `char primeiro_nome[15] = {'A', 'n', 'a'};`

O sistema insere os caracteres entre chaves, a partir da posição 0. Se o tamanho do vetor for superior ao número de caracteres nele armazenados, as posições não ocupadas serão preenchidas com zero, ou seja, nesse caso o arranjo terá um caractere terminador `'\0'`, pois será preenchido com zeros a partir da terceira posição.

Inicialização de *strings* em C

Exemplo 3: `char primeiro_nome[] = "Ana";`

O sistema determina o número de caracteres entre as aspas duplas, soma um para o caractere terminador, e cria uma *string* com o tamanho igual a tamanho da string + 1.

Leitura e escrita de strings com *scanf* e *printf*

Leitura

Não se deve colocar o & antes do nome da variável.

E é importante lembrar que o *scanf* encerrará a leitura do *string* assim que um espaço for encontrado no *string* de entrada, ou um caractere de fim de linha ou de tabulação.

```
Ex.: char nome_cliente[20];  
      scanf("%s", nome_cliente);  
      //usuário digitou Maria do Socorro  
      //em nome do cliente apenas Maria foi armazenada
```

Escrita

É possível apresentar o conteúdo de *strings* no *printf*.

Nesse caso o formato usado também é o %s

```
Ex.: char nome_cliente[20];  
      (...)  
      printf("O nome do cliente eh %s\n", nome_cliente);
```

Leitura e escrita de strings com *gets* e *puts*

Leitura

Gets

```
Ex.: char nome_cliente[20];  
      gets(nome_cliente);  
      // se usuário digitou Maria do Socorro,  
      // todo o nome estará armazenado em nome_cliente
```

Escrita

Puts

```
Ex.: char nome_cliente[20];  
      (...)  
      puts("Informe nome da cliente:");  
      gets(nome_cliente);
```

Função `gets()`

Lê uma *string* do teclado.

Permite colocar em uma variável todos os caracteres introduzidos, sem estar limitada a 1 palavra.

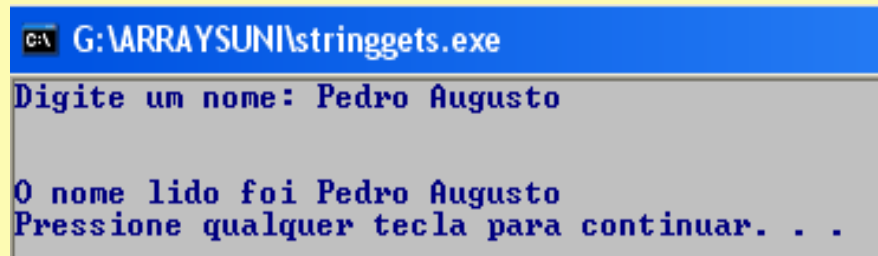
Ex.:

```
//testa funcao gets
#include <stdio.h>
#include <stdlib.h>
```

```
int main( )
{
    char nome[10];
    printf("Digite um nome: ");
    gets(nome);
    printf("\n\nO nome lido foi %s\n",
nome);
    system("pause");
    return 0;
}
```

Forma geral:

```
gets(nome_da_string);
```



```
C:\ G:\ARRAYSUNI\stringgets.exe
Digite um nome: Pedro Augusto

O nome lido foi Pedro Augusto
Pressione qualquer tecla para continuar. . .
```

Função `gets()` (cont.)

ATENÇÃO:

muito cuidado ao usar esta função.

Ela não controla se os caracteres fornecidos ultrapassam a capacidade da *string*!

Por exemplo, se em nome o usuário digitar 10 ou mais caracteres, como a *string* nome só tem 10 posições disponíveis, e nelas tem que estar incluído o '\0', os caracteres a mais serão colocados na área de memória subsequente à ocupada por nome, escrevendo uma região de memória que não está reservada à *string*. Este efeito é conhecido como "estouro de buffer" e pode causar problemas imprevisíveis na execução de um programa.

Função puts()

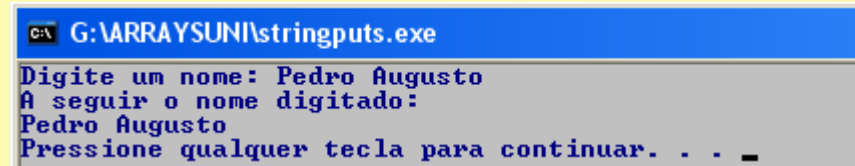
Escreve *strings*, constantes ou conteúdo de variáveis e muda automaticamente de linha.

Forma geral:

puts(nome_da_string);

Ex.: `//testa funcao puts`
`#include <stdio.h>`
`#include <stdlib.h>`

```
int main( )  
{  
    char nome[10];  
    system("color 71");  
    printf("Digite um nome: ");  
    gets(nome);  
    puts("A seguir o nome digitado:");  
    puts (nome);  
    system("pause");  
    return 0;  
}
```



```
G:\ARRAYS\UNIlstringputs.exe  
Digite um nome: Pedro Augusto  
A seguir o nome digitado:  
Pedro Augusto  
Pressione qualquer tecla para continuar. . . _
```

Exemplo de uso de funções **gets** (getstring) e **puts** (putstring)

// Leitura de n caracteres, sem parar no espaço

//testa funcoes puts e gets

#include <stdio.h>

#include <stdlib.h>

int main()

{

int seguir;

char nome[30];

seguir = 1;

while (seguir)

{

puts("\nNome:");

gets(nome);

if (nome[0] == '\0') // para quando sem conteúdo - enter direto

seguir = 0;

else

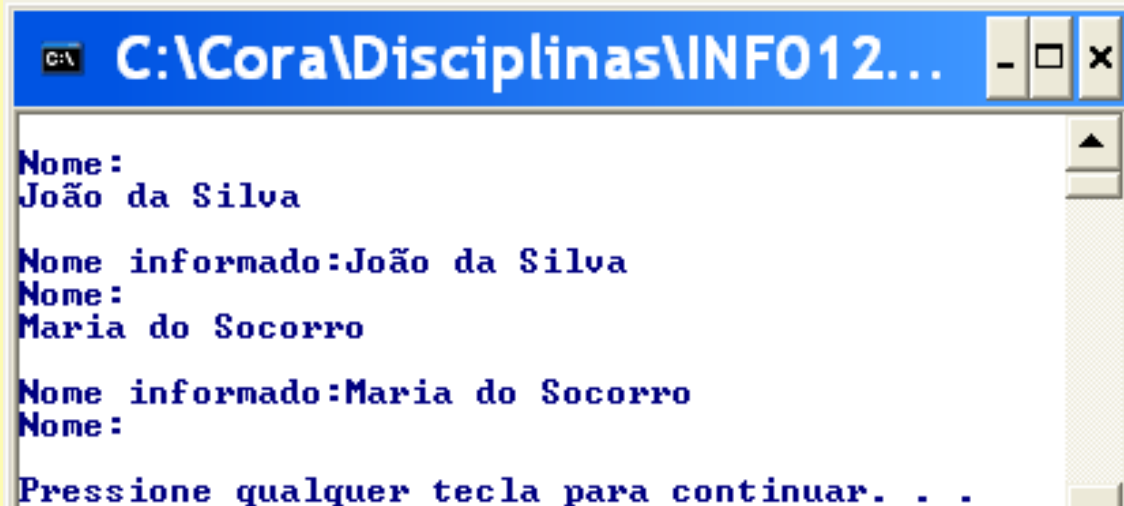
printf("\nNome informado:%s", nome); // não pode puts:2 conteúdos

}

system("PAUSE");

return 0;

}



```
C:\Cora\Disciplinas\INF012...
Nome:
João da Silva
Nome informado:João da Silva
Nome:
Maria do Socorro
Nome informado:Maria do Socorro
Nome:
Pressione qualquer tecla para continuar. . .
```

```
puts("Nome informado:");
puts (nome); //em linhas diferentes...
```


STRINGS em C

Como *strings* não são um tipo em C, não se pode atribuir uma *string* a outra:

~~string1 = string2;~~

Mas a biblioteca padrão do C possui diversas funções para manipular *strings*.

Funções para manipular *strings*

Para utilizar as funções a seguir, deve-se usar o `#include <string.h>`:

`strcpy()`

`strcat()`

`strlen()`

`strcmp()`

Função strcpy()

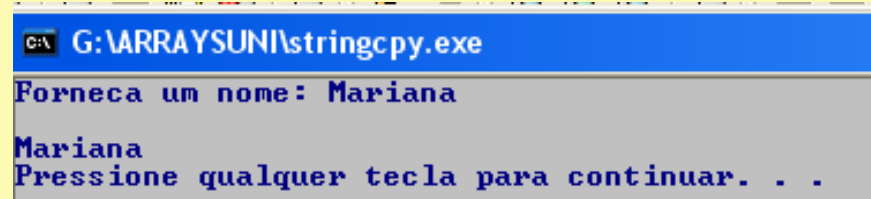
Copia a string_origem para a string_destino.

Ex.:

```
//testa funcao strcpy
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main( )
{
    char string_origem[10], string_destino[10];
    system("color 71");
    printf("Forneca um nome: ");
    gets(string_origem);
    strcpy(string_destino, string_origem);
    printf("\n%s\n", string_destino );
    system("pause");
    return 0;
}
```

Forma geral:

strcpy(string_destino, string_origem);



```
C:\ G:\ARRAYSUNI\stringcpy.exe
Forneca um nome: Mariana
Mariana
Pressione qualquer tecla para continuar. . .
```

Função `strcat()`

A `string_origem`, sem alteração, é anexada ao final da `string_destino`.

Forma geral:

`strcat(string_destino, string_origem);`

Ex.:

```
//testa funcao strcat
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main( )
{
    char string_origem[20], string_destino[40];
    system("color 71");
    printf("Forneca um texto: ");
    gets(string_origem);
    strcpy(string_destino, "O texto digitado foi: ");
    strcat(string_destino, string_origem);
    printf("\n%s\n", string_destino );
    system("pause");
    return 0;
}
```

ATENÇÃO:

A `string_destino` deve ter tamanho suficiente para armazenar o resultado de **`strcat!`**

 G:\ARRAYSUNI\stringcat.exe

Forneca um texto: Lindo dia!

O texto digitado foi: Lindo dia!

Pressione qualquer tecla para continuar. . .

Função `strlen()`

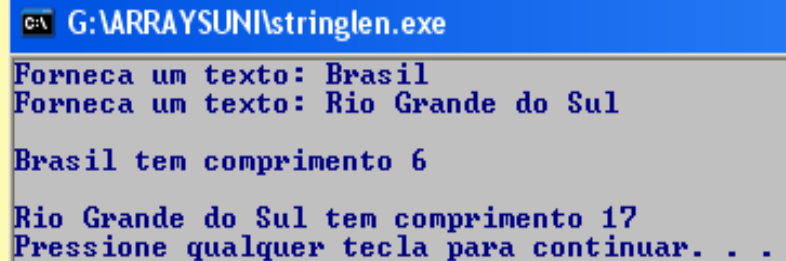
Retorna o tamanho de uma **string**, sem contar o terminador `'\0'`.

Ex.:

```
//testa funcao strlen
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main( )
{
    char string_primeiro[40], string_segundo[40];
    system("color 71");
    printf("Forneca um texto: ");
    gets(string_primeiro);
    printf("Forneca um texto: ");
    gets(string_segundo);
    printf("\n%s tem comprimento %d\n",
           string_primeiro, strlen(string_primeiro)
    );
    printf("\n%s tem comprimento %d\n",
           string_segundo, strlen(string_segundo) );
    system("pause");
    return 0;
}
```

Forma geral:

`strlen(string);`



```
G:\ARRAYSUNI\stringlen.exe
Forneca um texto: Brasil
Forneca um texto: Rio Grande do Sul

Brasil tem comprimento 6
Rio Grande do Sul tem comprimento 17
Pressione qualquer tecla para continuar. . .
```

Função `strcmp()`

Compara duas **strings**, `s1` e `s2`, caractere a caractere, com base na posição dos caracteres na tabela ASCII.

Se `s1` e `s2` forem iguais, retorna zero.

Se `s1` for maior que `s2`, retorna um valor maior que zero.

Se `s1` for menor que `s2`, retorna um valor menor que zero.

Ex.:

//têsta funcao strcmp

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main( )
{
    int i;
    char string_primeiro[40], string_segundo[40];
    for (i = 1; i <=3; i++)
    {
        printf("Forneca um texto: ");
        gets(string_primeiro);
        printf("Forneca um texto: ");
        gets(string_segundo);
        printf("Resultado da comparacao de %s com %s:
%d\n\n",
            string_primeiro, string_segundo,
            strcmp(string_primeiro, string_segundo) );
    }
    system("pause");
    return 0;
}
```

Forma geral:
`strcmp(s1, s2);`

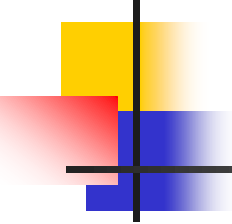
Lembrar que as maiúsculas
vêm antes das minúsculas
na tabela ASCII.

```
G:\VARRAYSUNI\stringcompara.exe
Forneca um texto: <...0
Forneca um texto: <...0
Resultado da comparacao de <...0 com <...0: 0

Forneca um texto: ANA
Forneca um texto: ana
Resultado da comparacao de ANA com ana: -1

Forneca um texto: ana
Forneca um texto: ANA
Resultado da comparacao de ana com ANA: 1

Pressione qualquer tecla para continuar. . .
```



Exemplo de uso de funções de *string*

//devolve indice da primeira ocorrencia de um caractere procurado

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAXIMO 20
int main( )
{
    char nome[MAXIMO + 1];
    char caract;
    int tamanho, i;
    system("color 71");
    do
    {
        printf("Digite um texto (tamanho maximo %d): ", MAXIMO);
        gets(nome);
        tamanho = strlen(nome);
        if (tamanho>20)
            printf("Tamanho maximo deve ser %d!\n", MAXIMO);
    }
    while ( tamanho >20);
    printf("Caractere a localizar: ");
    scanf("%c", &caract);
    i = 0;
    while (nome[i] != caract && nome[i] != '\0')
        i++;
    if (nome[i])
        printf("primeira ocorrencia de %c em %d\n", caract, i);
    else
        printf("%c nao encontrado\n", caract);
    system("pause");
    return 0;
}
```

STRINGS: importante

ATENÇÃO

→ As *strings* são representadas entre aspas duplas e os caracteres entre apóstrofos.

E por definição toda *string* tem o caractere terminador '\0' ao final.

Assim "A" e 'A' NÃO são a mesma coisa!!

"A" é na realidade um vetor de caracteres, com dois caracteres, sendo o segundo caractere '\0'.

'A' é um único caractere.

→ Uma *string* é sempre um vetor de caracteres (com '\0' ao final), mas um vetor de caracteres nem sempre é uma *string*!