

Arranjos Unidimensionais

Processamento de Caracteres

Uso de estruturas de tipo de dados
indexada e homogêneas em C

Processamento de Caracteres

- Um por vez

```
char caract;
```

- Funções

```
getchar()
```

```
scanf("%c", ...)
```

```
printf("%c", ...)
```

- Cadeias de caracteres (strings)

```
char text[20];
```

- Funções

```
gets()
```

```
scanf("%s", ...)
```

```
printf("%s", ...)
```

```
puts()
```

```
strcpy(...)
```

```
strcmp(...)
```

```
...
```

Processando 1 caractere por vez

- Declaração

```
char caract;
```

Aqui se usam aspas simples

- Inicialização

- Na declaração

```
char caract = 'P';
```

- Por atribuição

```
caract = 'P';
```

- Por leitura

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

```
caract = getchar();
```

Atenção: por vezes é necessário usar um espaço antes do %c ou executar a função `fflush(stdin)` antes de ler um caracter para esvaziar o buffer do teclado.

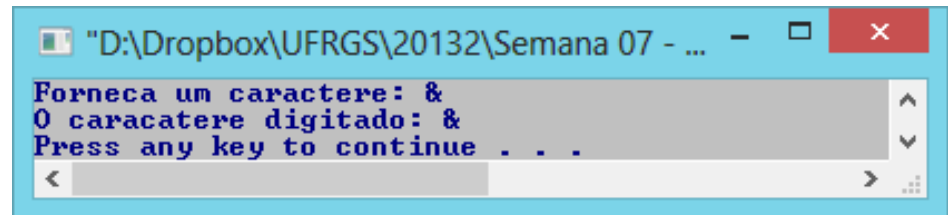
- Apresentação

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

Processando 1 caractere por vez

- Exemplo com `getchar()`

```
1 //testa funcao getchar
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <string.h>
5 int main()
6 {
7     char ch;
8     system("color 71");
9     printf("Forneca um caractere: ");
10    ch = getchar();
11    printf("O caractere digitado: %c\n", ch);
12    system("pause");
13    return 0;
14 }
```



Processando cadeias de caracteres (*strings*)

- Em C não há um tipo de dado específico para representar cadeias de caracteres ou *strings*
- As *Strings* são vetores de caracteres que têm como característica distintiva apresentar o caractere delimitador de fim `'\0'`, que corresponde ao caractere de posição zero na tabela **ASCII** (tabela na AGL)
- Então é importante salientar que toda *string* é um vetor de caracteres, mas nem todo vetor de caracteres 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		

Processando cadeias de caracteres (*strings*)

- Declaração
 - Forma geral

```
char nome[tamanho];
```

Considerar sempre que o *string* deverá conter o delimitador `'\0'`, portanto o tamanho deve ser adicionado de 1

- Exemplo:
 - Declarar duas *strings* para armazenar um dia da semana e um mês

```
char dia_semana[14];  
char mes[10];
```

Maior nome de dia da semana: segunda-feira (13)

Maior nome de mês: fevereiro (9)

Processando cadeias de caracteres (*strings*)

- Inicialização

- Na declaração (Exemplo 1)

```
char nome[9] = "Ana";
```

Aqui se usam
aspas duplas

	0	1	2	3	4	5	6	7	8
nome	A	n	a	\0					

- Na declaração (Exemplo 2)

```
char nome[9] = { 'A', 'n', 'a' };
```

	0	1	2	3	4	5	6	7	8
nome	A	n	a	\0	\0	\0	\0	\0	\0

- Na declaração (Exemplo 3)

```
char nome[] = "Ana";
```

	0	1	2	3
nome	A	n	a	\0

Processando cadeias de caracteres (*strings*)

- Inicialização

- Por leitura

```
scanf ("%s", nome) ;
```

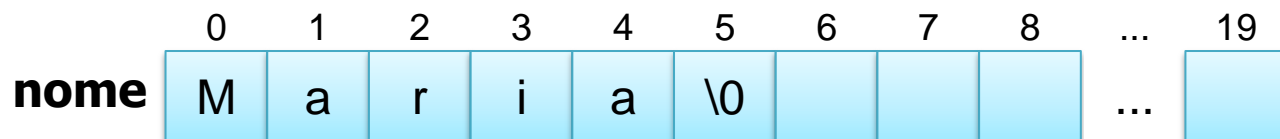
Neste caso não é necessário o &, pois **nome** já é o endereçamento do primeiro caractere da string ou seja **&nome[0]** Vamos ver isso mais tarde em ponteiros.
Obs.: Se usamos **&nome**, o C aceita.

- Por exemplo:

```
char nome[20] ;
```

```
scanf ("%s", nome) ;
```

```
//usuário digitou Maria do Socorro
```



Processando cadeias de caracteres (*strings*)

- Inicialização

- Por atribuição, durante a execução do programa NÃO é permitida

~~nome = "Pedro" ;~~

- Nem é possível copiar valores de variáveis de cadeias de caracteres (*strings*)

~~nome1 = nome2 ;~~

Processando cadeias de caracteres (*strings*)

- Apresentação

- Depois de declarada e inicializada a *string*, imprime-se normalmente com **printf** e **%s**

printf("Seu nome eh: %s\n", nome);

```
1 //testa funcao getchar
2 #include <stdio.h>
3 #include <stdlib.h>
4 int main()
5 {
6     char nome[] = "Dennis Ritchie";
7     printf("Seu nome eh: %s\n", nome);
8     system("pause");
9     return 0;
10 }
```



Funções para manipular *strings*

- A biblioteca `string.h` compila uma série de funções úteis para manipulação de *strings*

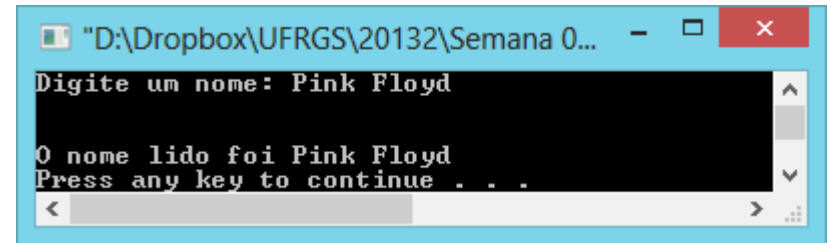
```
#include <string.h>
```

- ❖ `gets`
- ❖ `puts`
- ❖ `strcpy`
- ❖ `strcat`
- ❖ `strlen`
- ❖ `strcmp`

gets

- Lê uma *string* do teclado (similar ao `scanf` com `%s`)
- Forma geral: `gets (nome_da_string) ;`

```
1 //testa funcao gets
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <string.h>
5 int main()
6 {
7     char nome[11];
8     printf("Digite um nome: ");
9     gets(nome);
10    printf("\n\nO nome lido foi %s\n", nome);
11    system("pause");
12    return 0;
13 }
```



Aqui o nome completo (com os espaços) será lido

gets

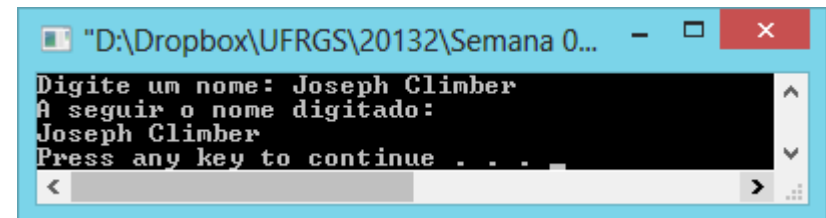
ATENÇÃO: muito cuidado ao usar esta função.
Ela não controla se os caracteres fornecidos ultrapassam a capacidade da *string*!

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.

puts

- Escreve *strings* e inclui uma quebra de linha automaticamente (similar ao `printf` com `%s\n`)
- Forma geral: `puts (nome_da_string) ;`

```
1 //testa funcao puts
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <string.h>
5 int main( )
6 {
7     char nome[15];
8     printf("Digite um nome: ");
9     gets(nome);
10    puts("A seguir o nome digitado:");
11    puts (nome);
12    system("pause");
13    return 0;
14 }
```

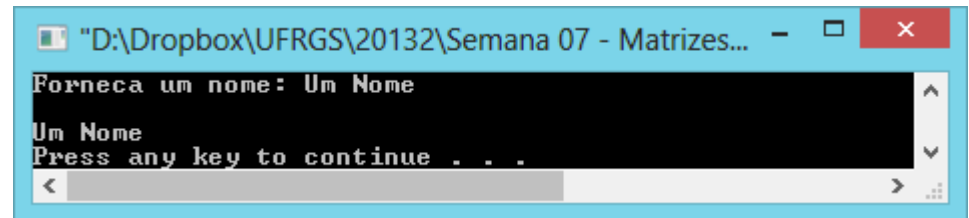


strcpy

- Copia a *string* de origem para a *string* destino
- Forma geral:

strcpy(string_dest, string_orig);

```
1 //testa funcao strcpy
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <string.h>
5 int main()
6 {
7     char string_origem[10], string_destino[10];
8     printf("Forneca um nome: ");
9     gets(string_origem);
10    strcpy(string_destino, string_origem);
11    printf("\n%s\n", string_destino);
12    system("pause");
13    return 0;
14 }
```

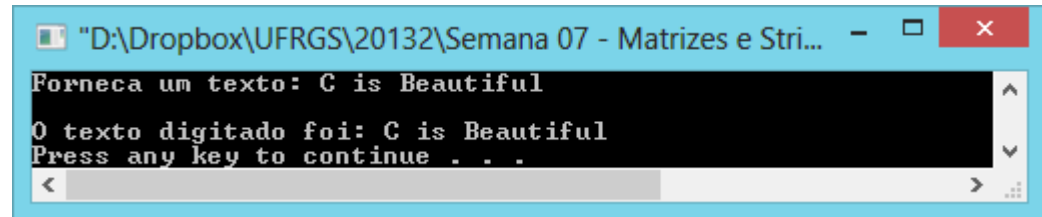


strcat

- A *string* de origem é anexada ao final da *string* destino (concatenação)
- Forma geral:

strcat(string_dest, string_orig);

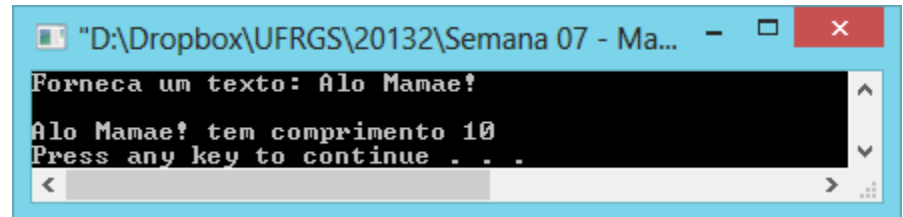
```
1 //testa funcao strcat
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <string.h>
5 int main()
6 {
7     char string_origem[20], string_destino[40];
8     printf("Forneca um texto: ");
9     gets(string_origem);
10    strcpy(string_destino, "O texto digitado foi: ");
11    strcat(string_destino, string_origem);
12    printf("\n%s\n", string_destino );
13    system("pause");
14    return 0;
15 }
```



strlen

- Retorna o comprimento de uma *string*, sem contar o terminador `'\0'`
- Forma geral: `strlen(string)` ;

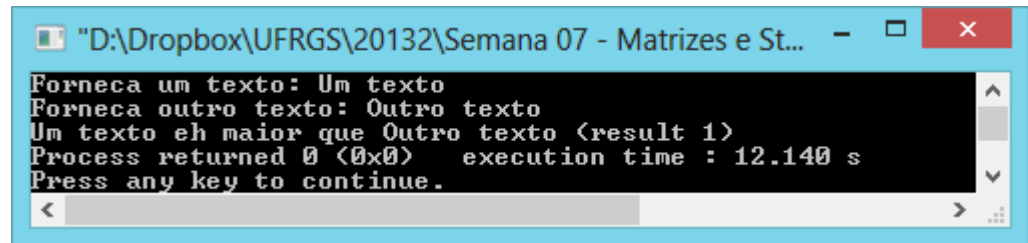
```
1 //testa funcao strlen
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <string.h>
5 int main( )
6 {
7     char texto[40];
8     printf("Forneca um texto: ");
9     gets(texto);
10    printf("\n%s tem comprimento %d\n",
11           texto, strlen(texto) );
12    system("pause");
13    return 0;
14 }
```



strcmp

- Compara duas *strings*, s1 e s2, caractere a caractere com base na tabela ASCII
 - Se s1 e s2 forem **iguais**, retorna **zero**
 - Se s1 for **maior** que s2, retorna um valor **positivo**
 - Se s1 for **menor** que s2, retorna um valor **negativo**
- Forma geral: **strcmp**(s1, s2);

```
1 //testa funcao strcmp
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <string.h>
5 int main(){
6     int result;
7     char s1[40],s2[40];
8     printf("Forneca um texto: ");
9     gets(s1);
10    printf("Forneca outro texto: ");
11    gets(s2);
12    result = strcmp(s1, s2);
13    if (result == 0)
14        printf("Strings iguais (result %d)", result);
15    else if (result > 0)
16        printf("%s eh maior que %s (result %d)", s1, s2, result);
17    else
18        printf("%s eh maior que %s (result %d)", s2, s1, result);
19    return 0;
20 }
```

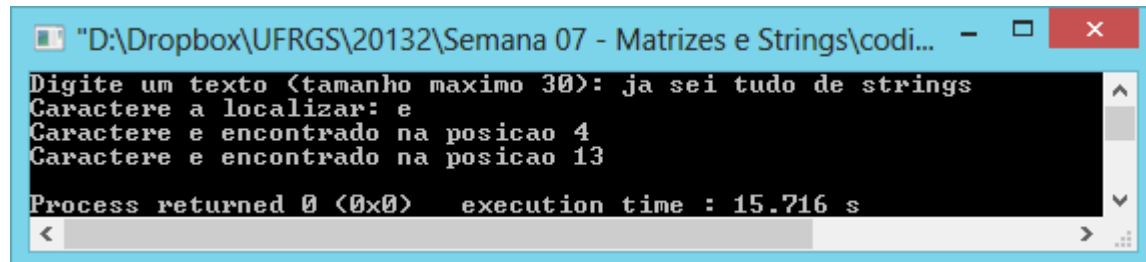


```
"D:\Dropbox\UFRGS\20132\Semana 07 - Matrizes e St...
Forneca um texto: Um texto
Forneca outro texto: Outro texto
Um texto eh maior que Outro texto (result 1)
Process returned 0 (0x0) execution time : 12.140 s
Press any key to continue.
```

Lembrar que as maiúsculas vêm antes das minúsculas na tabela ASCII

Um exemplo completo

```
1 //encontra um caractere qualquer dentro de um string
2 #include <stdio.h>
3 #include <string.h>
4 #define MAXIMO 30
5 int main() {
6     char nome[MAXIMO + 1], caract;
7     int tamanho, i, achou;
8     printf("Digite um texto (tamanho maximo %d): ", MAXIMO);
9     gets(nome);
10    printf("Caractere a localizar: ");
11    scanf("%c", &caract);
12    achou = 0;
13    for (i = 0; i < strlen(nome); i++){
14        if(nome[i] == caract){
15            printf("Caractere %c encontrado na posicao %d\n", caract, i);
16            achou = 1;
17        }
18    }
19    if(!achou)
20        printf("Caractere %c nao encontrado\n", caract);
21    return 0;
22 }
```



```
"D:\Dropbox\UFRGS\20132\Semana 07 - Matrizes e Strings\codi... - [X]
Digite um texto (tamanho maximo 30): ja sei tudo de strings
Caractere a localizar: e
Caractere e encontrado na posicao 4
Caractere e encontrado na posicao 13
Process returned 0 (0x0)   execution time : 15.716 s
< [ ] >
```

Lembretes

- As *strings* são representadas entre aspas duplas e os caracteres entre apóstrofos
- Por definição toda *string* tem o caractere terminador `'\0'` ao final

Sendo 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*!