

Arranjos

- *1 dimensão* 
- *várias dimensões*

Ex: Programar, na seguinte sequência:

1. Preencher arranjo (50) por leitura (inteiros).
2. Escrever os valores de arranjo em uma só linha.
3. Ler um valor v.
4. Informar quantas vezes este valor lido aparece em arranjo.
5. Imprimir os valores de arranjo superiores a v.
6. Ordenar arranjo (ordem crescente).
7. Imprimir novamente arranjo.

//Testando uso de arranjo

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

```
#define LIM_ARRANJO 50
```

```
int main ( )
```

```
{
```

```
int arranjo[LIM_ARRANJO];
```

```
int ind_arranjo;
```

```
for (ind_arranjo = 0; ind_arranjo < LIM_ARRANJO; ind_arranjo++) // 1  
    scanf("%d", &arranjo[ind_arranjo]);
```

```
for (ind_arranjo = 0; ind_arranjo < LIM_ARRANJO; ind_arranjo++) // 2  
    printf("%d    ", arranjo[ind_arranjo]);  
printf("\n\n");
```

```
...
```

```
}
```

Ex (cont):

3. Ler um valor v.
4. Informar quantas vezes este valor lido aparece em arranjo.
5. Imprimir os valores de arranjo superiores a v.
6. Ordenar arranjo (ordem crescente).
7. Imprimir novamente arranjo.

```
//Testando uso de arranjo
#define LIM_ARRANJO 50
#include <stdio.h>
int main ( )
{
    int arranjo[LIM_ARRANJO]; // arranjo
    int ind_arranjo; // indice para percorrer arranjo
    int v; // valor lido
    int cont; // contador
    scanf("%d", &v); // 3
    cont = 0; // 4
    for (ind_arranjo = 0; ind_arranjo < LIM_ARRANJO; ind_arranjo ++){
        if (arranjo[ind_arranjo] == v)
            cont = cont + 1;
    }
    printf("%d", cont);
    printf("\n\n");
    ...
}
```

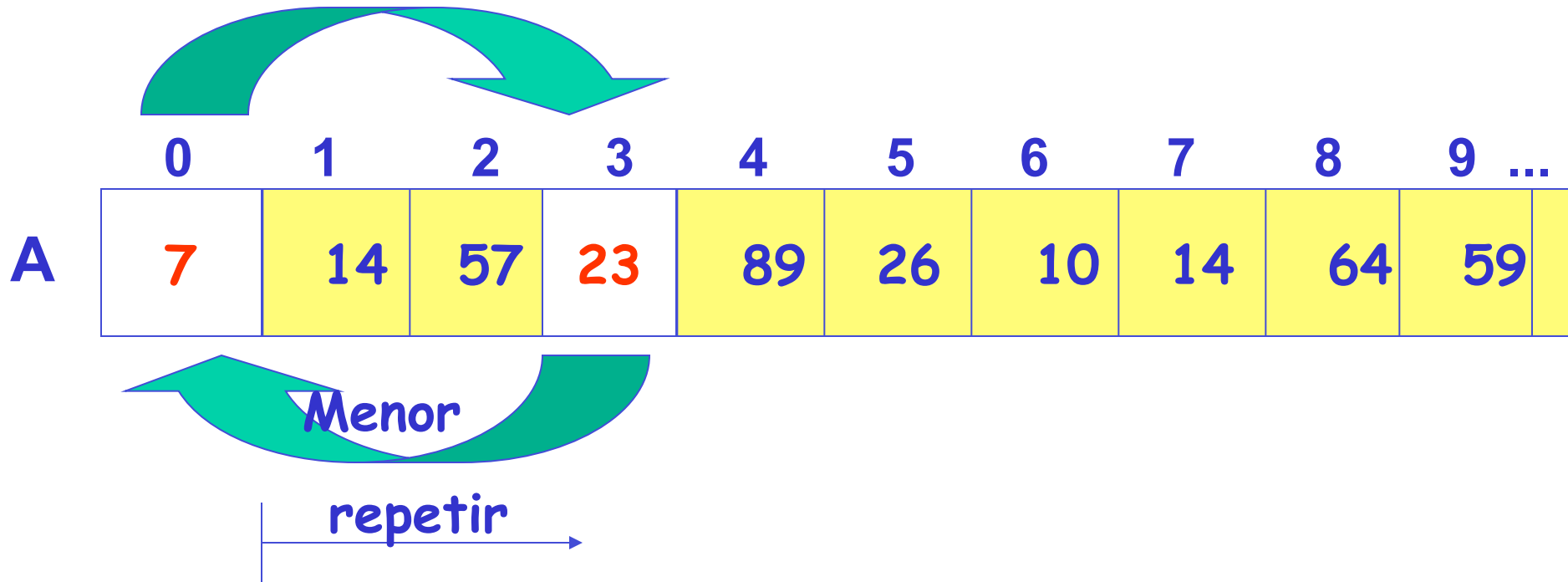
Ex (cont):

5. Imprimir os valores de A superiores a V.
6. Ordenar A (ordem crescente).
7. Imprimir novamente A.

```
//Testando uso de arranjo
#include <stdio.h>
#define LIM_ARRANJO 50
int main ( )
{
    int arranjo[LIM_ARRANJO];
    int ind_arranjo;
    int v;      // valor lido
    int cont;   // contador
    scanf("%d", &v);                // 3
    cont = 0;                          // 4
    for (ind_arranjo = 0; ind_arranjo < LIM_ARRANJO; ind_arranjo ++ )
        if (arranjo[ind_arranjo] == v)
            cont = cont + 1;
    printf("%d", cont);
    printf("\n\n");
    for (ind_arranjo = 0; ind_arranjo < LIM_ARRANJO; ind_arranjo ++ ) // 5
        if (arranjo[ind_arranjo] > v)
            printf("%d", arranjo[ind_arranjo] );
    printf("\n\n");
    ...
}
```

Ex (cont):

6. Ordenar arranjo (ordem crescente).
7. Imprimir novamente arranjo.



Classificação por Seleção

1. Procurar o menor elemento
2. Colocar este menor na primeira posição
3. Procurar o menor da segunda em diante
4. Colocar na segunda posição
5. ... até o penúltimo

Ex (cont):

6. Ordenar A (ordem crescente).

7. Imprimir novamente A.

```
//Testando uso de arranjo
#include <stdio.h>
#define LIM_ARRANJO 50
int main ( )
{
    int arranjo[LIM_ARRANJO];
    int v;      // valor lido
    int cont;   // contador
    int ind_arranjo, ind2; // índices para percorrer elementos de arranjo
    int menor,posmenor; // auxiliar para a ordenação }
    int aux ; // variável auxiliar para a troca de 2 elementos

    ...
    for (ind_arranjo = 0; ind_arranjo < LIM_ARRANJO; ind_arranjo ++){ // 6
        // procura o menor do elemento ind_arranjo em diante e o coloca na posição ind_arranjo
        menor = arranjo[ind_arranjo]; //considera o primeiro elemento como o menor,
        // alterando o primeiro a cada laço
        posmenor = ind_arranjo // guarda a posicao do primeiro
        for (ind2 = ind_arranjo + 1; ind2 < LIM_ARRANJO; ind2 ++){ // Laço de busca do menor de
        //todo arranjo
            if (arranjo[ind2] < menor) // Procura um menor que o primeiro
            {
                menor = arranjo[ind2];
                posmenor = ind2;
            }
        }
        if (posmenor != ind_arranjo) // Se posição diferente do primeiro,
        //tem um menor que deve ser colocado na 1a posição
        {
            aux = arranjo [ind_arranjo]; //Troca o menor de posição com o primeiro
            arranjo [ind_arranjo] = arranjo [posmenor];
            arranjo [posmenor] = aux;
        }
    }

    for (ind_arranjo = 0; ind_arranjo < LIM_ARRANJO; ind_arranjo ++){ //Imprime arranjo
        printf ("%d ",arranjo[ind_arranjo]); // 7
        printf("\n\n");
    }
    return 0;
}
```

Vetores de Caracteres

Strings

Ex: 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.

Como obter e armazenar nomes ?

Nome : cadeia de caracteres ou *string*

No C:

- caractere : **char** → 1 caractere
- *strings*: **vetor** de caracteres → pelo menos um caracter, seguido da marca `\0`.

Um caractere

Variável caractere: `char caract;`

Leitura:

1) `scanf` - função genérica de leitura:

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

A colocação de um espaço em branco antes do `%c` faz com que sejam descartados caracteres previamente escritos no buffer do dispositivo.

O mesmo efeito pode ser obtido com a função

`fflush(stdin);` //funcao "limpa" o buffer de leitura antes do `scanf`

2) `getchar` - função que retorna um char lido do buffer do teclado:

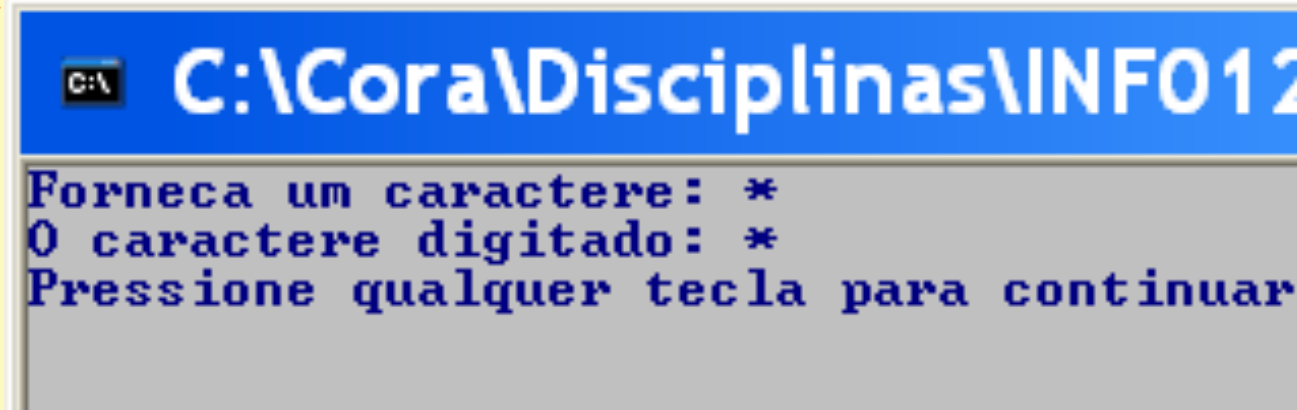
`caract = getchar();`

Função getchar

```
// Leitura de 1 caractere:  
// Forma geral: getchar( )
```

```
//testa funcao getchar  
#include <stdio.h>
```

```
int main( )  
{  
    char ch;  
    int i;  
    printf("Forneca um caractere: ");  
    ch = getchar( );  
    printf("O caractere digitado: %c\n", ch);  
    return 0;  
}
```



```
C:\Cora\Disciplinas\INF012  
Forneca um caractere: *  
O caractere digitado: *  
Pressione qualquer tecla para continuar
```

N caracteres = string

- Strings são cadeias ou sequências de caracteres, que tem delimitada a área significativa por uma marca '\0'.
- O '\0' é o caractere da posição 0 da tabela ASCII e não o dígito 0).
- Exemplo:
 - char vet[9] é vetor de caracteres que contem um string de tamanho 6
 - Sete posições do arranjo são utilizadas e as demais contém lixo.

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

STRINGS

Declaração:

```
char nome_da-string [tamanho+1];
```

- O tamanho de uma string deve sempre prever a inclusão do caractere delimitador '\0'.
- Tamanho de uma variável **STRING** = número máximo de caracteres + 1
- Exemplo:
 - sejam 2 variáveis 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

A inicialização de *strings*, como de arranjos em geral, pode ser feita de várias maneiras conforme exemplos a seguir.

Exemplo 1:

```
char primeiro_nome[15] = "Ana";
```

O sistema insere os caracteres indicados entre as aspas duplas no vetor `primeiro_nome`, a partir da posição 0, e insere na posição 3 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, será inserido o caractere terminador `'\0'` após os caracteres válidos e as posições não ocupadas serão preenchidas com zero.

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

Formato: **%s**, mas não se deve colocar o **&** antes do nome da variável. O *scanf* encerrará a leitura do *string* assim que um branco 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);
// se usuário digitou Maria do Socorro
//em nome do cliente, apenas Maria foi armazenada

Escrita

O formato usado também é o **%s**.

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

Concatenação de Strings

```
#define TAM 8    // tamanho do string
int main ()
{
    char pal1[TAM], pal2[TAM], palfinal[TAM+TAM-1];
    int i, j ;
    printf("Escreve o primeiro string\n"); scanf ("%s", pal1);
    printf("Escreve o segundo string\n");  scanf ("%s", pal2);
    i=0; // indice do primeiro vetor
    j=0; // indice do segundo vetor
    while (pal1[i]!='\0' && i<TAM)
    {
        palfinal[i]= pal1[i];
        i++;
    }
    do
    {
        palfinal[i]= pal2[j];
        i++; j++;
    }
    while (pal2[j]!='\0' || j<TAM) ;
    palfinal[i]= '\0' ;
    printf ("%s\n", palfinal);
    return 0;
}
```

Comprimento do String

```
#define TAM 8 //definindo tamanho do string
int main ()
{
    char pal[TAM];
    int i;
    printf("Escreve o string\n");
    scanf ("%s", pal);
    i=0; // indice do primeiro vetor
    while (pal[i]!='\0' && i<TAM)
        i++;

    printf ("Tamanho da String %d\n", i);
    return 0;
}
```


Funções de Manipulação de STRINGS

- O C oferece funções pré-definidas para tratamento de strings,
- Essas funções utilizam o ‘\0’ para tratar apenas a parte significativa dos vetores de caracteres

Leitura e escrita de strings com gets e puts

Leitura

O gets permite colocar na variável todos os caracteres introduzidos, sem estar limitado a 1 palavra.

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

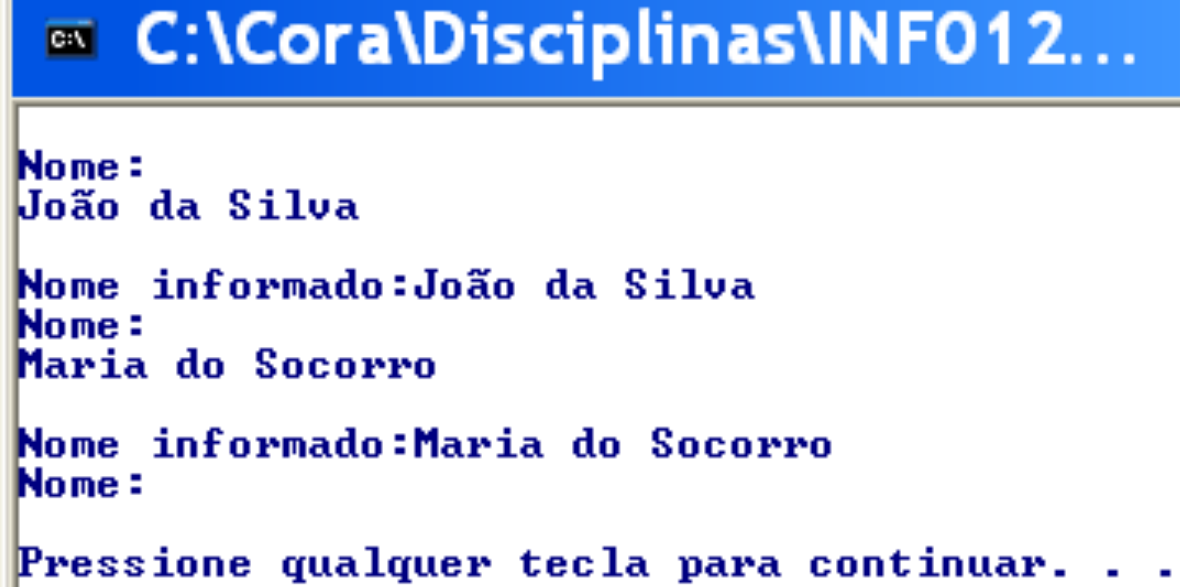
Escrita

Permite a escrita de 1 única string (constante literal ou variável), mudando automaticamente de linha após.

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

Função *gets* - getstring e *puts* - putstring

```
// Leitura de n caracteres, sem parar no branco:
//testa funcao 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
        // conteúdo
    }
    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...
```

Comparação e atribuição de Strings

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

~~string1 = string2;~~

Para isso: comparar cada um dos índices do vetor ou utilizar funções para manipulação de *strings*.

Comparação e atribuição de Strings

- Utilização da biblioteca `#include <string.h>`
- Funções:
 - `strcpy`
 - `strcat`
 - `strlen`
 - `strcmp`

Função strcpy

- Copia *string_origem* para *string_destino*.
- Formato:

```
strcpy(string_destino, string_origem);
```

- Exemplo:

```
...  
#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;  
}
```



G:\ARRAYSUNI\stringcpy.exe

Forneca um nome: Mariana

Mariana

Pressione qualquer tecla para conti

Função strcat

- A **string_origem**, sem alteração, é anexada ao final da **string_destino**.
- Formato: `strcat(string_destino, string_origem);`
- Exemplo:

```
#include <string.h>
int main( )
{
    char string_origem[20], string_destino[40];
    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**!

C:\ 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 '**0**'.
- Formato: `strlen(string);`
- Exemplo:

```
#include <string.h>
int main( )
{
    char string_primeiro[40], string_segundo[40];
    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;
}
```

 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 contin

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**.
- Formato: `strcmp(s1, s2);`
- Exemplo:

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

```
#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) );
    }
    return 0;
}
```

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**.
- Formato: `strcmp(s1, s2);`
- Exemplo:

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

```
#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 comparação de %s com %s: ",
               string_primeiro, string_segundo);
        printf("%d\n", strcmp(string_primeiro, string_segundo));
    }
    return 0;
}
```

C:\G:\ARRAYSUNI\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. . .
```

1 - Escreva um programa que lê um texto contando até MAXIMO caracteres (consistindo), lê 1 caractere e informa a 1ª posição do texto onde este caractere ocorre ou que não existe tal caractere no texto.

// Exemplos de uso de funções:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h> // para usar funções de manipulação de strings
#define MAXIMO 20
int main( )
```

```
{
    char caract, texto[MAXIMO + 1], palavra[15], aux[15];
    int tamanho, i;
    system("color f1");
```

// leitura, consistindo se não ultrapassa limite de caracteres:

```
do
{
    printf("Digite um texto (tamanho maximo %d): ", MAXIMO);
    gets(texto);
    tamanho = strlen(texto);
    if (tamanho > MAXIMO)
        printf("Tamanho maximo deve ser %d!\n", MAXIMO);
}
while ( tamanho > MAXIMO);
```

// obtém caractere e procura primeira ocorrência dele no texto:

```
// obtém caractere e procura primeira ocorrência dele no texto:
printf("Informe caractere a localizar: ");
caract = getchar();
i = 0;
while (texto[i] != caract && texto[i] != '\0')
    i++;
if (texto[i] // se não encontrou, é '/0' (false)
    printf("Primeira ocorrência de %c em %d\n", caract, i);
else
    printf("%c não encontrado! \n", caract);

// obtém sequência de caracteres a serem buscado:

/* para a próxima aula: ler palavra e verificar se a sequência de
caracteres da palavra existe no texto */

return 0;
}
```

String

ATENÇÃO

- As *strings* são representadas entre aspas duplas e os caracteres entre apóstrofos.
- Por definição, toda *string* tem o caractere terminador ‘\0’ ao final.
- Assim, “A” e ‘A’ **NÃO** são a mesma coisa!!
 - “A” : vetor de 2 caracteres- ‘A’ e ‘\0’.
 - ‘A’ : 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* !