7.4 - Strings

7.4.1 - Características básicas

Supor a declaração abaixo de um vetor de caracteres.

```
char  v[] = {'a', 'b', 'c'};
```

O vetor armazena texto, mas ficar digitando os caracteres entre apóstrofes não é necessário se usarmos o texto entre aspas, tal como usamos no comando printf. Este texto é uma string. Veja abaixo uma alternativa usando string.

```
char  v[] = “abc” ;
```

- Uma string corresponde a um vetor de caracteres terminada pelo caracter ‘\0’ ou NULL que tem valor 0. Para usar o NULL ao invés de '\0’ pode ser necessária a definição

```
#define NULL '\0'
```

Quando usamos aspas queremos dizer string, e automaticamente o '\0' é colocado pelo compilador

No exemplo acima, embora não tenhamos escrito explicitamente o caracter NULL, o compilador automaticamente colocou este caracter como o último elemento do vetor v.

0	1	2	3
‘a’	‘b’	‘c’	‘\0’

Portanto, o tamanho de v é 4: 3 para os caracteres 'a', ' b' e 'c' e 1 para o caractere NULL que o compilador introduziu automaticamente.

- O caracter '\0' diferencia um vetor de char comum de uma string.

As definições abaixo são equivalentes para definir uma string.

```
char  v[ ] = {'a', 'b', 'c', '\0'};
char  v[ ] = “abc” ;
```

0	1	2	3
‘a’	‘b’	‘c’	‘\0’

No entanto:

```
char  v[ ] = {'a', 'b', 'c'};   /* não é tratado como uma string */
```

0	1	2
‘a’	‘b’	‘c’

Lembre-se sempre de reservar uma posição no vetor para o ‘\0’. Não há problemas em ter posições a mais. O problema é ter a menos.

```
char v[5] = “abc”;
```

0	1	2	3	4
‘a’	‘b’	‘c’	‘\0’	‘\0’

char v[3] = “abc”; → PROBLEMA.

0	1	2
‘a’	‘b’	‘c’

- A seguinte matriz de strings está sendo declarada e inicializada no mesmo comando:

Exemplo:

```
#define num_nomes 4 /* número de nomes no vetor nomes*/
#define tam 10          /* tamanho do vetor de cada nome */

main(
{
     char  nomes [num_nomes] [tam] = {    “Carmen”,
                                          “Ricardo”,
                                          “Alexandre”,
                                          “Ana Lucia”
                                       };
       int i;
        for (i = 0; i < 4; i ++)
            printf("%s\n", nomes[i]);
 }
```

A saída deste programa é:

```
Carmen
Ricardo
Alexandre
Ana Lucia
```

a declaração acima é equivalente a:

```
char  nomes [4] [10] = { {‘C ‘, ‘a’, ‘r’, ‘m’, ‘e’, ‘n’, ‘\0’},
                         {‘R’, ’i’, ‘c’, ‘a’, ‘r’, ‘d’, ‘o’, ‘\0’};
                         {‘A’, ‘l’, ‘e’, ‘x’, ‘a’, ‘n’, ‘d’, ‘r’, ‘e’, ‘\0’};
                         {‘A’, ‘n’, ‘a’, ‘ ‘, ‘L’, ‘u’, ‘c’, ‘i’, ‘a’,  ‘\0’} };
```

7.4.2 - Leitura e escrita de strings em E/S padrão

- scanf( ): Esta função lê os caracteres até achar um separador (branco, tabs ou enter).

- Exemplo: char nome [15];

  scanf (“% s”, nome); /* não é necessário &, porque nome já é um apontador para a string */

Entrada digitada	0	1	2	3	4	. . .	14
ANA ⇒	A	N	A	\0	\0	. . .	\0

Entrada digitada	0	1	2	3	4	. . .	14
ANA LUCIA⇒	A	N	A	\0	\0	. . .	\0

obs.: armazena somente ANA, pois encontrou um branco !!!

- gets( ) : Lê uma linha inteira. Os caracteres são lidos até que seja encontrado o caracter ‘\n’ (nova linha), gerado pela tecla <enter>.

- Exemplo:

```
main( )
{
   char nome[81];
   printf (“Digite seu nome:   “);
   gets(nome);
   printf (“Saudações,  %s ! “, nome);
}
```

A saída desse programa é a seguinte:

Digite seu nome: Ana Maria Bonita <*enter*>
Saudações, Ana Maria Bonita !

- printf( ): Imprime segundo a formatação especificada

  Ex.: printf (“% s “, nome);

- puts( ) : Imprime uma string por vez e pula para a próxima linha.
- Exemplo:

```
char nome[15], nome2[15];
    .
    .
gets(nome);
gets(nome2);
printf ( “Os nomes são:   \n “);
puts (nome);
puts (nome2);
    .
    .
```

Supondo que os seguintes dados sejam digitados:

Ana Paula <enter>

Luiz Carlos <enter>

A saída do trecho de programa acima é a seguinte:

Os nomes são:

Ana Paula

Luiz Carlos

- Os seguintes comandos são equivalentes:
  - printf (“%s \n “, nome);
  - puts (nome);

- Observações:
  - nome é o endereço do primeiro elemento da string, sendo assim, as seguintes expressões são equivalentes:

    nome e &nome[0]

  nome está armazenado da seguinte maneira:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	N	A		P	A	U	L	A	\0	\0	\0	\0	\0	\0

  - O comando

    puts(&nome[4])

    imprime a partir de nome[4], produzindo a saída : Paula

&nome[4] é um apontador para a string a partir do quarto elemento.

7.4.3 - Algumas funções para manipular strings

- A biblioteca <string.h> contém declarações de funções para manipular strings. Para usá-las coloque no início do programa.

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

- strlen (<nome>):

  Retorna o número de caracteres sem o '\0', isto é, o tamanho da string <nome>.

- strcat (<nome1> , <nome2>):
  - Concatena duas strings, colocando a segunda no final da primeira
    - <nome1>: armazenará o resultado da concatenação (deve ter tamanho suficiente)
    - <nome2>: ficará como estava.
  - Exemplo: Seja o seguinte trecho de programa:

```
salute[20] = "Bom Dia, ";
nome = "Barney !";
strcat (salute, nome);
puts(salute);
puts(nome);
```

A saída do trecho de programa acima é:

```
Bom Dia, Barney!
Barney!
```

- strcmp(<nome1>,<nome2>):
  - Compara se duas strings são iguais
  - Devolve uma das seguintes opções:

    inteiro < 0 se nome1 < nome2;

    inteiro = 0 se nome1 = = nome2;

    inteiro > 0 se nome1 > nome2.

  - Maior ou menor não se refere a tamanho, mas sim à ordem alfabética.

    Ex.: a < b , A < a , Adriana < Ana , etc …

  - Exemplo:

    strcmp (“fat”, “fat”) = 0
    strcmp (“Fat”, “fat”) < 0
    strcmp (“fatima”, “fatorial”) < 0
    strcmp (“fatorial”, “fatima”) > 0

    A função strcmp() faz distinção entre letras maiúsculas e minúsculas.

    Se você não quer que a função faça esta distinção, você pode modificar o seu string para ter apenas letras minúsculas (ou maiúsculas) antes de passá-lo como argumento para a função strcmp(). Utilize a função tolower que transforma um caracter do tipo letra para minúscula.

- memcmp(<nome1> , <nome2>, n):
  - Compara os primeiros <n> caracteres das strings apontadas por <nome1> e <nome2>.
  - Devolve uma das seguintes opções:

    inteiro < 0 se nome1 < nome2;

    inteiro = 0 se nome1 = = nome2;

    inteiro > 0 se nome1 > nome2.

  - Exemplo:

    memcmp (“fatima”, “fatorial”,3) = 0

- Observação:
  - Seja o seguinte trecho de programa:

    ```
    char strorigem[20],  strdestino[15];      /* duas strings */
    strdestino = strorigem;
    ```

  - O comando acima está conceitualmente errado, pois o nome da string é um ponteiro com valor constante.
  - Se o objetivo é copiar o conteúdo de uma string em outra, tem-se as seguintes soluções:
    - substitir o comando anterior por:

      ```
      for (i = 0;  i < 15;  i + + )
          strdestino[i] = strorigem[i];
      ```

    - utilizar as funções para esse fim definidas em <string.h>

- strcpy (< destino >, < origem >):
    - Copia conteúdo da string <origem> na string <destino>.

        Ex.: strcpy (strdestino, strorigem);

- strncpy (< destino >, < origem >, n): Copia até n caracteres da string <origem> na string <destino>, onde n é um inteiro.

    Ex.: strncpy (strdestino, strorigem, 14); /* coloca \n no fim */

- memchr(<string> , <car> , <n>):
    - Procura em <string>, pela primeira ocorrência do caracter <car>, nos <n> primeiros caracteres.
    - Devolve um ponteiro para a primeira ocorrência de <ch> ou NULL, se não for encontrado.
    - Ex.: : Seja o seguinte trecho de programa:

```
char  *p,  nome[10];
scanf("%s", nome);
p = memchr(nome, 's',  5);
printf("%s", p);
```