

## string.h

Esta livreria é responsável pela inclusão de funções de manipulação de strings, ou blocos de memória. Nomeadamente: concatenação de strings, comparação de strings, copia de strings, etc.

Funções disponibilizadas por esta livreria:

<code>_fmemccpy</code>	<code>_fmemchr</code>	<code>_fmemcmp</code>
<code>_fmemcpy</code>	<code>_fmemicmp</code>	<code>_fmemset</code>
<code>_fstrcat</code>	<code>_fstrchr</code>	<code>_fstrcmp</code>
<code>_fstrcpy</code>	<code>_fstrcspn</code>	<code>_fstrdup</code>
<code>_fstricmp</code>	<code>_fstrlen</code>	<code>_fstrlwr</code>
<code>_fstrncat</code>	<code>_fstrncmp</code>	<code>_fstrnicmp</code>
<code>_fstrncpy</code>	<code>_fstrnset</code>	<code>_fstrpbrk</code>
<code>_fstrrchr</code>	<code>_fstrrev</code>	<code>_fstrset</code>
<code>_fstrspn</code>	<code>_fstrstr</code>	<code>_fstrtok</code>
<code>_fstrupr</code>	<code>memccpy</code>	<code>memchr</code>
<code>memcmp</code>	<code>memcpy</code>	<code>memicmp</code>
<code>memmove</code>	<code>memset</code>	<code>movedata</code>
<code>movmem</code>	<code>setmem</code>	<code>stpcpy</code>
<code>strcat</code>	<code>strchr</code>	<code>strcmp</code>
<code>strcmpi</code>	<code>strcpy</code>	<code>strcspn</code>
<code>strdup</code>	<code>_strerror</code>	<code>strerror</code>
<code>stricmp</code>	<code>strlen</code>	<code>strlwr</code>
<code>strncat</code>	<code>strncmp</code>	<code>strncmpi</code>
<code>strncpy</code>	<code>strnicmp</code>	<code>strnset</code>
<code>strpbrk</code>	<code>strchr</code>	<code>strrev</code>
<code>strset</code>	<code>strspn</code>	<code>strstr</code>
<code>strtok</code>	<code>strxfrm</code>	<code>strupr</code>

Para melhor compreensão das funções descritas neste capítulo aconselho o estudo de ponteiros em C. Dada a sua pouca utilização algumas destas funções não serão descritas em pormenor neste guia. As funções iniciadas por `_f` são as versões FAR das que aqui descrevemos. Os alunos interessados na sua descrição poderão consultar as ajudas do Turbo C ou *man* do Gnu C.

## memchr

#include<mem.h>,#include <string.h>

Procura por um caracter no bloco de memória.

```
void *memchr (const void *s, int c, size_t n);
```

retorno: endereço onde foi encontrado o caracter **c**, caso contrario NULL.

argumentos: ponteiro (endereço) para o inicio do bloco a pesquisar, caracter procurado e inteiro indicando o numero de caracteres a pesquisar.

**Obs** : Procura pelo caracter **c** nos primeiros **n** caracteres do bloco **s**.

DOS	UNIX	WINDOWS	ANSI C
Sim	Sim	Sim	Sim

**Ver também**: memcpy, memmove.

Exemplo

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

```
int main(void)
{
    char str[17];
    char *ptr;

    strcpy(str, "This is a string");
    ptr = (char *) memchr(str, 'r', strlen(str));
    if (ptr)
        printf("The character 'r' is at position: %d\n", ptr - str);
    else
        printf("The character was not found\n");
    return 0;
}
```

## memcpy

#include<mem.h>,#include <string.h>

Copia um bloco de memória de endereço para outro endereço .

```
void *memcpy (void *dest, const void *src, char c, size_t n);
```

retorno: endereço para onde foi copiado o bloco de memória.

argumentos: ponteiro (endereço) para o inicio do bloco de destino, ponteiro (endereço) para inicio do bloco de origem, o **n**, numero de bytes a serem copiados e caracter **c** indicador de fim da copia.

**Obs :** Basicamente a função copia **n** caracteres do endereço **src** para o endereço **dest**. Se encontrar o caracter **c** antes interrompe a copia.

DOS	UNIX	WINDOWS	ANSI C
Sim	Sim	Sim	Sim

**Ver também:** memcpy, memmove.

Exemplo

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

```
int main(void)
{
    char *src = "This is the source string";
    char dest[50];
    char *ptr;

    ptr = (char *) memcpy(dest, src, 'c', strlen(src));

    if (ptr)
    {
        *ptr = '\0';
        printf("The character was found: %s\n", dest);
    }
    else
        printf("The character wasn't found\n");
    return 0;
}
```

<b>me memcpy</b>	<b>#include &lt;mem.h&gt;, #include &lt;string.h&gt;</b>
------------------	--

Compara N bytes de dois blocos de memória.

```
int memcmp (const void *s1, const void *s2, size_t n);
```

retorno: devolve um inteiro

```
< 0 se s1 < s2
= 0 se s1 == s2
> 0 se s1 > s2
```

argumentos: ponteiros para o inicio dos blocos a comparar e o número de caracteres a comparar.

**Obs :** considera os blocos de memória como sendo de *unsigned char*.

DOS	UNIX	WINDOWS	ANSI C
Sim	Sim	Sim	Sim

**Ver também:** strcmp, memcmp.

Exemplo

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

```
int main(void)
```

```
{  
    char *buf1 = "aaa";  
    char *buf2 = "bbb";  
    char *buf3 = "ccc";
```

```
  
    int stat;
```

```
  
    stat = memcmp(buf2, buf1, strlen(buf2));
```

```
    if (stat > 0)
```

```
        printf("buffer 2 is greater than buffer 1\n");
```

```
    else
```

```
        printf("buffer 2 is less than buffer 1\n");
```

```
  
    stat = memcmp(buf2, buf3, strlen(buf2));
```

```
    if (stat > 0)
```

```
        printf("buffer 2 is greater than buffer 3\n");
```

```
    else
```

```
        printf("buffer 2 is less than buffer 3\n");
```

```
  
    return 0;
```

```
}
```

## memcpy

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

Copia um bloco de memória de endereço para outro endereço .

```
void *memcpy (void *dest, const void *src, size_t n);
```

retorno: endereço para onde foi copiado o bloco de memória.

argumentos: ponteiro (endereço) para o início do bloco de destino, ponteiro (endereço) para início do bloco de origem e o  $n$  número de bytes a serem copiados.

**Obs :** Se os dois blocos estão sobrepostos o comportamento da função é indefinido

DOS	UNIX	WINDOWS	ANSI C
Sim	Sim	Sim	Sim

**Ver também:** memcpy, memmove.

Exemplo

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

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

```

int main(void)
{
    char src[] = "*****";
    char dest[] = "abcdefghijklmnpqrstuvwxy0123456709";
    char *ptr;

    printf("destination before memcpy: %s\n", dest);
    ptr = (char *) memcpy(dest, src, strlen(src));
    if (ptr)
        printf("destination after memcpy: %s\n", dest);
    else
        printf("memcpy failed\n");
    return 0;
}

```

## memicmp

#include <mem.h>, #include <string.h>

Compara dois blocos de memória ignorando se são maiúsculas ou minúsculas.

int memcmp(const void \*s1, const void \*s2, size\_t n);

retorno: devolve um inteiro

< 0 se s1 < s2  
 = 0 se s1 == s2  
 > 0 se s1 > s2

argumentos: ponteiros para o início dos blocos a comparar e o número de caracteres a comparar.

**Obs :** Ignora se o tipo de caracteres. Não estabelece diferença entre maiúsculas e minúsculas.

DOS	UNIX	WINDOWS	ANSI C
Sim	Sim	Sim	

**Ver também:** strcmp, stricmp, strcmpi.

Exemplo

```

#include <stdio.h>
#include <string.h>

```

```

int main(void)
{
    char *buf1 = "ABCDE123";
    char *buf2 = "abcde456";
    int stat;
    stat = memcmp(buf1, buf2, 5);
    printf("As strings ate à posição 5 são ");
    if (stat)
        printf("desiguais\n ");
    printf("iguais\n");
    return 0;
}

```

```
}
```

<b>memmove</b>	<code>#include &lt;mem.h&gt;, #include &lt;string.h&gt;</code>
----------------	--

Copia bloco de um memória de um endereço para outro.

```
void *memmove(void *dest, void *src, size_t n);
```

retorno: devolve um ponteiro para o endereço para onde foi movido o bloco de memória.

argumentos: ponteiro para o endereço onde vai ser colocado o bloco, ponteiro para o bloco a copiar e o número de caracteres a comparar.

**Obs :**

DOS	UNIX	WINDOWS	ANSI C
Sim	Sim	Sim	Sim

**Ver também:** memset, movmem.

Exemplo

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

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

```
int main(void)
```

```
{
```

```
    char *dest = "abcdefghijklmnopqrstuvwxy0123456789";
```

```
    char *src = "*****";
```

```
    printf("Conteudo do dest antes de memmove: %s\n", dest);
```

```
    memmove(dest, src, 26);
```

```
    printf("Conteudo do dest depois memmove:  %s\n", dest);
```

```
    return 0;
```

```
}
```

<b>memset</b>	<code>#include &lt;mem.h&gt;, #include &lt;string.h&gt;</code>
---------------	--

Coloca **n** caracteres **c** no bloco de memória .

```
void *memset(void *s, int c, size_t n);
```

retorno: devolve um ponteiro para o endereço onde foram colocados os caracteres **c**.

argumentos: ponteiro para o endereço onde vão ser colocados os caracteres **c**, caracter a ser colocado e o número de caracteres a colocar.

**Obs :**

DOS	UNIX	WINDOWS	ANSI C
Sim	Sim	Sim	Sim

**Ver também:** memset, movmem.

Exemplo

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

int main(void)
{
    char buffer[] = "Hello world\n";

    printf("Buffer antes de memset: %s\n", buffer);
    memset(buffer, '*', strlen(buffer) - 1);
    printf("Buffer depois de memset: %s\n", buffer);
    return 0;
}
```

**movedata** `#include <mem.h>, #include <string.h>`

Copia **n** bytes de um endereço para outro. movedata copia **n** bytes de um endereço fonte (srcseg:srcoff) para o endereço de destino (destseg:destoff).

```
void movedata(unsigned srcseg, unsigned srcoff,
              unsigned destseg, unsigned destoff, size_t n);
```

retorno:

argumentos: 4 inteiros não sinalizados representado respectivamente: segmento de origem, offset de origem, segmento destino e offset destino e **n** número de bytes a serem movidos.

**Obs :** movedata é independente do modelo de memória.

DOS	UNIX	WINDOWS	ANSI C
Sim		Sim	

**Ver também:** FP\_OFF, memcpy, MK\_FP, movmem, segread.

Exemplo

```
#include <mem.h>
```

```
#define MONO_BASE 0xB000 /* endereço da base da memória de video monocromático */

char buf[80*25*2];

/* grava o conteúdo de um screen monocromático em buffer */
void save_mono_screen(char near *buffer)
{
    movedata(MONO_BASE, 0, _DS, (unsigned)buffer, 80*25*2);
}

int main(void)
{
    save_mono_screen(buf);
    return 0;
}
```

## movmem

#include <mem.h>, #include <string.h>

Move um bloco de **n** bytes do endereço origem (src) para o endereço de destino (dest).

```
void movmem(void *src, void *dest, unsigned length);
```

retorno:

argumentos: 2 ponteiros, apontado respectivamente para o bloco de origem (src) e para o bloco de destino (dest) e o numero de bytes a serem movidos.

**Obs :**

DOS	UNIX	WINDOWS	ANSI C
Sim		Sim	

**Ver também:** memcpy , memmove, movedata.

Exemplo

```
#include <mem.h>
#include <alloc.h>
#include <stdio.h>
#include <string.h>

int main(void)
{
    char *source = "Borland International";
    char *destination;
    int length;
```



```

length = strlen(source);
destination = (char *) malloc(length + 1);
movmem(source, destination, length);
printf("%s\n", destination);

return 0;
}

```

## setmem

#include <mem.h>, #include <string.h>

Coloca no interior de um bloco de memória um determinado caracter.

```
void *memset( void *dest, int c, size_t count );
```

retorno:

argumentos: um ponteiro, apontado para o bloco de destino (dest), o caracter a ser colocado e o numero de caracteres a serem colocados.

**Obs :**

DOS	UNIX	WINDOWS	ANSI C
Sim		Sim	

**Ver também:** memset, strset.

Exemplo

```

#include <string.h>
#include <stdio.h>
#include <mem.h>

int main(void)
{
    char buffer[] = "Hello world\n";

    printf("Buffer antes de memset: %s\n", buffer);
    memset(buffer, '*', strlen(buffer) - 1);
    printf("Buffer depois de memset: %s\n", buffer);
    return 0;
}

```

## strcpy

#include <string.h>

Copia uma string (src) em outra (dest).

```
char *strcpy(char *dest, const char *src);
```

retorno: endereço da string destino.

argumentos: 2 ponteiros, apontado respectivamente para a string de destino (dest) e para a string de destino (dest).

**Obs :** Termina quando encontre o caracter '\0' indicando fim de string. O programador deve garantir que a string destino tenha espaço para acolher a nova string.

DOS	UNIX	WINDOWS	ANSI C
Sim	Sim	Sim	

### Ver também:

Exemplo

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

int main(void)
{
    char string[10];
    char *str1 = "abcdefghi";

    strcpy(string, str1);
    printf("%s\n", string);
    return 0;
}
```

## strcat

#include <string.h>

Une duas strings (src).

```
char *strcat(char *dest, const char *src);
```

retorno: endereço da string destino.

argumentos: 2 ponteiros, apontado respectivamente para a string de destino (dest) e para a string de destino (dest).

**Obs :** Termina quando encontre o caracter '\0' indicando fim de string. O programador deve garantir que a string destino tenha espaço para acolher a nova string.

DOS	UNIX	WINDOWS	ANSI C
Sim	Sim	Sim	Sim

### Ver também:

Exemplo

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

int main(void)
{
    char destination[25];
    char *blank = " ", *c = "C++", *turbo = "Turbo";

    strcpy(destination, turbo);
    strcat(destination, blank);
    strcat(destination, c);

    printf("%s\n", destination);
    return 0;
}
```

## strchr

#include <string.h>

Pesquisa uma string por uma dado caracter.

```
char *strchr(const char *s, int c);
```

retorno: em caso de sucesso endereço da primeira ocorrência do caracter dentro da string. Caso contrário NULL.

argumentos: um ponteiro, apontando a string a pesquisar e o caracter a localizar-se.

**Obs :** O caracter '\0' é considerado como fazendo parte da string. A instrução `strchr(str, 0)` devolve um ponteiro para o fimda string.

DOS	UNIX	WINDOWS	ANSI C
Sim	Sim	Sim	Sim

### Ver também:

Exemplo

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

```

int main(void)
{
    char string[15];
    char *ptr, c = 'r';

    strcpy(string, "Isto é uma string");
    ptr = strchr(string, c);
    if (ptr)
        printf("O caracter %c está na posição: %d\n", c, ptr-string);
    else
        printf("O caracter não foi encontrado\n");
    return 0;
}

```

## strcmp

#include <string.h>

Compara duas strings, caracter a caracter.

```
int strcmp(const char *s1, const char*s2);
```

retorno: Um inteiro que terá:

```

< 0 if s1 < s2
== 0 if s1 == s2
> 0 if s1 > s2

```

argumentos: dois ponteiros, apontando as strings a comparar.

**Obs :** A comparação é efectuada enquanto os caracteres forma iguais.

DOS	UNIX	WINDOWS	ANSI C
Sim	Sim	Sim	Sim

**Ver também:** strncmp, strncmpi, strnicmp.

Exemplo

```

#include <string.h>
#include <stdio.h>

int main(void)
{
    char *buf1 = "aaa", *buf2 = "bbb", *buf3 = "ccc";
    int ptr;

    ptr = strcmp(buf2, buf1);
    if (ptr > 0)

```

```

    printf("buffer 2 é maior que o buffer 1\n");
else
    printf("buffer 2 é menor que o buffer 1\n");

ptr = strcmp(buf2, buf3);
if (ptr > 0)
    printf("buffer 2 é maior que o buffer 3\n");
else
    printf("buffer 2 é menor que o buffer 3\n");

return 0;
}

```

## strcmpi

#include <string.h>

Macro que compara duas strings, sem atender ao facto dos caracteres serem maiúsculas ou minúsculas.

```
int strcmpi(const char *s1, const char *s2);
```

retorno: Um inteiro que terá:

```

< 0 if s1 < s2
== 0 if s1 == s2
> 0 if s1 > s2

```

argumentos: dois ponteiros, apontando as strings a comparar.

**Obs :** A comparação é efectuada enquanto os caracteres forma iguais.

DOS	UNIX	WINDOWS	ANSI C
Sim	Sim	Sim	Sim

**Ver também:** strcmp, strcmpi, strnicmp.

Exemplo

```

#include <string.h>
#include <stdio.h>

int main(void)
{
    char *buf1 = "BBB", *buf2 = "bbb";
    int ptr;

    ptr = strcmpi(buf2, buf1);

```

```

if (ptr > 0)
    printf("buffer 2 é maior que o buffer 1\n");

if (ptr < 0)
    printf("buffer 2 é menor que o buffer 1\n");

if (ptr == 0)
    printf("buffer 2 é igual buffer 1\n");

return 0;
}

```

## strcpy

#include <string.h>

Copia a string (src) para outra string (dest).

```
char *strcpy(char *dest, const char *src);
```

retorno: ponteiro para a string final (dest).

argumentos: dois ponteiros, apontando respectivamente para a string destino e para a string a copiar.

**Obs :** A comparação é efectuada enquanto os caracteres forma iguais.

DOS	UNIX	WINDOWS	ANSI C
Sim	Sim	Sim	Sim

**Ver também:** strncpy.

Exemplo

```

#include <stdio.h>
#include <string.h>

int main(void)
{
    char string[10];
    char *str1 = "abcdefghi";

    strcpy(string, str1);
    printf("%s\n", string);
    return 0;
}

```

## strcspn

#include <string.h>

Pesquisa uma string em busca de um segmento que não contenha um subconjunto de um conjunto de caracteres.

```
size_t strcspn(const char *s1, const char *s2);
```

retorno: comprimento do segmento encontrado,.

argumentos: dois ponteiros, apontando respectivamente para a string (s1) a pesquisar e para a string (s2) contendo o conjunto de caracteres.

**Obs :** procura o inicio do segmento da string s1, que NÃO CONTENHA caracteres da string s2.

DOS	UNIX	WINDOWS	ANSI C
Sim	Sim	Sim	Sim

**Ver também:** strspn .

Exemplo

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

int main(void)
{
    char *string1 = "1234567890";
    char *string2 = "747DC8";
    int length;

    length = strcspn(string1, string2);
    printf("Caracter onde a string é intersectada está na posição: %d\n",
        length);

    return 0;
}
```

## strdup

#include <string.h>

Duplica uma string para uma posição de memória recém alocada.

```
char *strdup(const char *s);
```

retorno: No caso de sucesso devolve um ponteiro para a posição onde criou a cópia.

argumentos: um ponteiro, apontando para a string (s) a duplicar. Caso não seja possível devolve um ponteiro NULL.

**Obs :** O espaço alocado é (strlen(s)+1). A libertação do espaço alocado é da responsabilidade do programador.

DOS	UNIX	WINDOWS	ANSI C
Sim	Sim	Sim	

**Ver também:** free .

Exemplo

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

int main(void)
{
    char *dup_str, *string = "abcde";

    dup_str = strdup(string);
    printf("%s\n", dup_str);
    free(dup_str);

    return 0;
}
```

## **\_strerror**

#include <string.h>

Permite a construção de uma mensagem de erro.

char \*\_strerror(const char \*s);

**retorno:** No caso da string (s) não ser nula devolve um ponteiro para uma string termina com caracter '\0'. A string devolvida contém a mensagem de erro.

**argumentos:** um ponteiro, apontando para a string (s) com a mensagem de erro. A string deverá <= 94 caracteres.

**Obs :** .

DOS	UNIX	WINDOWS	ANSI C
Sim		Sim	



**Ver também:** perror.

Exemplo

```
#include <stdio.h>

int main(void)
{
    FILE *fp;

    /* abre um ficheiro para escrita */
    fp = fopen("TEST.$$$", "w");

    /* forca uma situação de erro */
    if (!fp) fgetc(fp);

    if (ferror(fp))
        /* visualiza a mensagem de erro */
        printf("%s", _strerror("Custom"));

    fclose(fp);
    return 0;
}
```

## strerror

#include <string.h>

Permite a construção de um sistema de mensagens de erro.

char \*strerror(int errnum);

retorno: Devolve uma string de mensagem de erro associado ao numero de erro.

argumentos: um inteiro.

**Obs :**

DOS	UNIX	WINDOWS	ANSI C
Sim		Sim	Sim

**Ver também:** perror, errno.h.

Exemplo

```
#include <stdio.h>
#include <errno.h> /* definição das mensagens de erro */

int main(void)
{
```

```

char *buffer;
buffer = strerror(errno);
printf("Error: %s\n", buffer);
return 0;
}

```

## stricmp

#include <string.h>

Compara duas strings sem atender ao factos de serem caracteres maiúsculos ou minúsculos.

```
int stricmp(const char *s1, const char *s2);
```

retorno: um inteiro que será:

```

< 0 if s1 < s2
== 0 if s1 == s2
> 0 if s1 > s2.

```

argumentos: dois ponteiros que apontam respectivamente para as duas strings a comparar.

**Obs :**

DOS	UNIX	WINDOWS	ANSI C
Sim		Sim	

**Ver também:** strcmp, strcmpi e strnicmp.

Exemplo

```

#include <string.h>
#include <stdio.h>

```

```

int main(void)
{
    char *buf1 = "BBB", *buf2 = "bbb";
    int ptr;

    ptr = stricmp(buf2, buf1);

    if (ptr > 0)
        printf("buffer 2 é maior que buffer 1\n");

    if (ptr < 0)
        printf("buffer 2 é menor que buffer 1\n");

    if (ptr == 0)
        printf("buffer 2 é igual buffer 1\n");

    return 0;
}

```

```
}
```

## **strlen**

#include <string.h>

Determina o numero de caracteres que compõem uma string.

```
size_t strlen(const char *s);
```

retorno: um inteiro que indica o numero de caracteres que tem a string s.

argumentos: ponteiro para char, referenciado uma string.

**Obs :**

DOS	UNIX	WINDOWS	ANSI C
Sim	Sim	Sim	Sim

**Ver também:**

Exemplo

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

int main(void)
{
    char *string = "Escola Superior de Tecnologia";

    printf("%d\n", strlen(string));
    return 0;
}
```

## **strlwr**

#include <string.h>

Converte os caracteres de uma string em minúsculas.

```
char *strlwr(char *s);
```

retorno: devolve ponteiro para s já convertida em minúsculas.

argumentos: ponteiro para char, referenciando uma string a converter.

**Obs :** Converte só os caracteres de **A a Z** em **a a z**. Os restante caracteres mantém-se inalterados.

DOS	UNIX	WINDOWS	ANSI C
Sim	Sim		

**Ver também:** tolower, toupper

Exemplo

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

int main(void)
{
    char *string = "ESCOLA SUPERIOR DE TECNOLOGIA";

    printf("string antes de strlwr(): %s\n", string);
    strlwr(string);
    printf("string depois de strlwr():  %s\n", string);
    return 0;
}
```

## strncat

#include <string.h>

Une uma porção de uma string a outra.

char \*strncat(char \*dest, const char \*src, size\_t maxlen);

retorno: devolve ponteiro para a string final (dest).

argumentos: dois ponteiros apontado respectivamente para as duas string (dest e src) e o número de caracteres (maxlen) da string src que devem ser adicionados à string dest.

**Obs :** A função adiciona o caracter NULL à string dest depois de adicionados os maxlen caracteres. A string resultante dest terá: strlen(dest)+maxlen caracteres.

DOS	UNIX	WINDOWS	ANSI C
Sim	Sim	Sim	Sim

**Ver também:**

Exemplo

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

```
int main(void)
```

```

{
    char destination[25];
    char *source = " Algarve";

    strcpy(destination, "Universidade do ");
    strncat(destination, source, 7);
    printf("%s\n", destination);
    return 0;
}

```

## strncmp

#include <string.h>

Compara *n* caracteres de duas strings.

int strncmp (const char \*s1, const char \*s2, size\_t maxlen);

retorno: um inteiro que será:

```

< 0 if s1 < s2
== 0 if s1 == s2
> 0 if s1 > s2.

```

argumentos: dois ponteiros apontado respectivamente para as duas string (s1 e s2) e o número de caracteres (maxlen) a serem comparados.

**Obs :**

DOS	UNIX	WINDOWS	ANSI C
Sim	Sim	Sim	Sim

**Ver também:** strcmp, strcmpi, and stricmp.

Exemplo

```

#include <string.h>
#include <stdio.h>

int main(void)
{
    char *buf1 = "aaabbb", *buf2 = "bbbccc", *buf3 = "ccc";
    int ptr;

    ptr = strncmp(buf2, buf1, 3);
    if (ptr > 0)
        printf("buffer 2 é maior que buffer 1\n");
    else
        printf("buffer 2 é menor que buffer 1\n");

    ptr = strncmp(buf2, buf3, 3);

```

```

if (ptr > 0)
    printf("buffer 2 é maior que buffer 3\n");
else
    printf("buffer 2 é menor que buffer 3\n");

return(0);
}

```

## strncmpi

#include <string.h>

Macro que compara  $n$  caracteres de duas strings sem atender ao facto dos caracteres sem maiúsculas ou minúsculas.

int strncmpi (const char \*s1, const char \*s2, size\_t maxlen);

retorno: um inteiro que será:

```

< 0 if s1 < s2
== 0 if s1 == s2
> 0 if s1 > s2.

```

argumentos: dois ponteiros apontado respectivamente para as duas string (s1 e s2) e o número de caracteres (maxlen) a serem comparados.

**Obs :**

DOS	UNIX	WINDOWS	ANSI C
Sim		Sim	

**Ver também:** strcmp, strempi, and stricmp.

Exemplo

```

#include <string.h>
#include <stdio.h>

int main(void)
{
    char *buf1 = "BBBccc", *buf2 = "bbbccc";
    int ptr;

    ptr = strncmpi(buf2,buf1,3);

    if (ptr > 0)
        printf("buffer 2 é maior que buffer 1\n");

    if (ptr < 0)
        printf("buffer 2 é menor que buffer 1\n");

    if (ptr == 0)

```

```

    printf("buffer 2 é igual buffer 1\n");

    return 0;
}

```

## strncpy

#include <string.h>

Copia até  $n$  caracteres de uma string noutra.

```
char *strncpy(char *dest, const char *src, size_t maxlen);
```

retorno: devolve um ponteiro para char referenciando a string final.

argumentos: dois ponteiros apontado respectivamente para as duas string (dest e src) e o número de caracteres (maxlen) a serem copiados. O programador é responsável por garantir espaço suficiente na string dest.

**Obs :**

DOS	UNIX	WINDOWS	ANSI C
Sim	Sim	Sim	Sim

**Ver também:** strcpy.

Exemplo

```

#include <stdio.h>
#include <string.h>

int main(void)
{
    char string[10];
    char *str1 = "abcdefghi";

    strncpy(string, str1, 3);
    string[3] = '\0';
    printf("%s\n", string);
    return 0;
}

```

## strnicmp

#include <string.h>

Compara  $n$  caracteres de duas strings sem atender ao facto dos caracteres serem maiúsculas ou minúsculas.

```
int strnicmp(const char *s1, const char *s2, size_t maxlen);
```

retorno: um inteiro que será:

< 0 if s1 < s2  
== 0 if s1 == s2  
> 0 if s1 > s2.

argumentos: dois ponteiros apontado respectivamente para as duas string (s1 e s2) e o número de caracteres (maxlen) a serem comparados.

**Obs :**

DOS	UNIX	WINDOWS	ANSI C
Sim		Sim	

**Ver também:** strcmp, strcmpi e stricmp.

Exemplo

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

int main(void)
{
    char *buf1 = "BBBccc", *buf2 = "bbbccc";
    int ptr;

    ptr = strnicmp(buf2, buf1, 3);

    if (ptr > 0)
        printf("buffer 2 é maior que buffer 1\n");

    if (ptr < 0)
        printf("buffer 2 é menor que buffer 1\n");

    if (ptr == 0)
        printf("buffer 2 é igual buffer 1\n");

    return 0;
}
```

## strnset

#include <string.h>

Coloca *n* caracteres de string com um determinado caracter.

char \*strnset(char \*s, int ch, size\_t n);

retorno: devolve um ponteiro para char, referenciando *s*.



argumentos: um ponteiro apontado para a string (*s*), caracter *ch* e o número de caracteres (*n*) a serem colocados.

**Obs :** copia o caracter *ch* para as primeiros *n* bytes da string *s*.

DOS	UNIX	WINDOWS	ANSI C
Sim	Sim		

**Ver também:** `strset`.

Exemplo

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

int main(void)
{
    char *string = "abcdefghijklmnopqrstuvwxyz";
    char letter = 'x';

    printf("string antes de strnset: %s\n", string);
    strnset(string, letter, 13);
    printf("string depois de strnset: %s\n", string);

    return 0;
}
```

## **strpbrk**

`#include <string.h>`

Pesquisa uma string pela primeira ocorrência de um caracter que esteja na segunda string.

`char *strpbrk(const char *s1, const char *s2);`

retorno: devolve um ponteiro para char, referenciando a posição onde encontrou o primeiro caracter. Ou NULL no caso de não existir caracteres de *s2* em *s1*.

argumentos: dois ponteiros apontando respectivamente para a string (*s1*) a pesquisar e para a string (*s2*) que contém os caracteres.

**Obs :**

DOS	UNIX	WINDOWS	ANSI C
-----	------	---------	--------

Sim	Sim	Sim	Sim
-----	-----	-----	-----

**Ver também:**

Exemplo

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

int main(void)
{
    char *string1 = "abcdefghijklmnopqrstuvwxyz";
    char *string2 = "onm";
    char *ptr;

    ptr = strpbrk(string1, string2);

    if (ptr)
        printf("a função strpbrk encontrou o caracter: %c\n", *ptr);
    else
        printf("a função strpbrk não encontrou nenhum caracter onm em string1\n");

    return 0;
}
```

## strchr

#include <string.h>

Pesquisa última ocorrência de um caracter numa string.

**char \*strchr(const char \*s, int c);**

**retorno:** devolve um ponteiro para char, referenciando a posição onde encontrou o último caracter. Ou NULL no caso de não existam caracteres.

**argumentos:** um ponteiro apontando a string (s) a pesquisar e um inteiro (c) indicando o caracter que se pretende localizar.

**Obs :** A função pesquisa do fim para o início da string. O caracter nulo ‘\0’ é considerado como pertencente à string.

DOS	UNIX	WINDOWS	ANSI C
Sim	Sim	Sim	Sim

**Ver também:**

Exemplo

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

int main(void)
{
    char string[15];
    char *ptr, c = 'r';

    strcpy(string, "Isto é uma string");
    ptr = strchr(string, c);
    if (ptr)
        printf("O character %c está na posição: %d\n", c, ptr-string); /* porque ?????? */
    else
        printf("O character não foi encontrado\n");
    return 0;
}
```

## strrev

#include <string.h>

Inverte os caracteres de uma string.

```
char *strrev(char *s);
```

retorno: devolve um ponteiro para char, referenciando a string revertida.

argumentos: um ponteiro apontando a string (s) a inverter.

**Obs :**

DOS	UNIX	WINDOWS	ANSI C
Sim		Sim	

**Ver também :**

Exemplo

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

int main(void)
{
    char *forward = "string";

    printf("Antes de strrev(): %s\n", forward);
```

```

    strrev(forward);
    printf("Depois strrev(): %s\n", forward);
    return 0;
}

```

## strnset

#include <string.h>

Coloca todos caracteres de string com um determinado caracter.

```
char *strset(char *s, int ch);
```

retorno: devolve um ponteiro para char, referenciando *s*.

argumentos: um ponteiros apontado para a string (*s*), caracter *ch*.

**Obs :** Enche a string *s* com o caracter *ch*.

DOS	UNIX	WINDOWS	ANSI C
Sim		Sim	

**Ver também:** strnset.

Exemplo

```

#include <stdio.h>
#include <string.h>

```

```

int main(void)
{
    char string[10] = "123456789";
    char symbol = 'c';

    printf("Antes strset(): %s\n", string);
    strset(string, symbol);
    printf("Depois strset(): %s\n", string);
    return 0;
}

```

## strspn

#include <string.h>

Pesquisa uma string em busca de um segmento que contenha um subconjunto de um conjunto de caracteres.

```
size_t strspn(const char *s1, const char *s2);
```

retorno: comprimento do segmento encontrado,.

**argumentos:** dois ponteiros, apontando respectivamente para a string (s1) a pesquisar e para a string (s2) contendo o conjunto de caracteres.

**Obs :** procura o início do segmento da string s1, que NÃO CONTENHA caracteres da string s2.

DOS	UNIX	WINDOWS	ANSI C
Sim	Sim	Sim	Sim

**Ver também:** strstrn .

Exemplo

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

int main(void)
{
    char *string1 = "1234567890";
    char *string2 = "123DC8";
    int length;

    length = strstrn(string1, string2);
    printf("O caracter onde a string é diferente está na posição: %d\n", length);
    return 0;
}
```

## strstr

#include <string.h>

Pesquisa uma string em busca da primeira ocorrência de um cadeia de caracteres, isto é de uma string.

char \*strstr(const char \*s1, const char \*s2);

retorno: ponteiro referenciando a posição do primeiro caracter de s2 em s1.

**argumentos:** dois ponteiros, apontando respectivamente para a string (s1) a pesquisar e para a string (s2) contendo o conjunto de caracteres.

**Obs :**

DOS	UNIX	WINDOWS	ANSI C
Sim	Sim	Sim	Sim

**Ver também:**

Exemplo

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

int main(void)
{
    char *str1 = "Departamento Engenharia Eléctrica Electrónica", *str2 = " Engenharia", *ptr;

    ptr = strstr(str1, str2);
    printf("A sub string é : %s\n", ptr);
    return 0;
}
```

## strtok

#include <string.h>

Pesquisa uma string s1 em busca da primeira ocorrência de um cadeia de caracteres, não contida em s2.

char \*strtok(char \*s1, const char \*s2);

**retorno:** ponteiro referenciando a posição do primeiro caracter de s2 em s1. Caso não encontre devolve NULL.

**argumentos:** dois ponteiros, apontando respectivamente para a string (s1) a pesquisar e para a string (s2) contendo o conjunto de caracteres.

**Obs :** A string s2 possui normalmente um único caracter. À primeira chamada da função strtok() devolve a primeira posição em que encontra o caracter de s2. Coloca o caracter nulo logo a seguir à posição encontrada. A função pode ser chamada de novo tendo o primeiro argumento como NULL e assim retirar todas as substrings de s1 (considerando s2 como separador).

DOS	UNIX	WINDOWS	ANSI C
Sim	Sim	Sim	Sim

**Ver também :**

Exemplo

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

int main(void)
{
    char input[16] = "abc dares";
    char *p;

    /* strtok coloca o caracter NULL
    logo a seguir separador, se encontrado */ /* considere token como palavra */
```

```

p = strtok(input, " ");
if (p) printf("%s\n", p);

/* A Segunda chamada a strtok() usando NULL
como primeiro argumento devolve um ponteiro
ao caracter logo a seguir ao separador */
p = strtok(NULL, " ");
if (p) printf("%s\n", p);
return 0;
}

```

## strxfrm

#include <string.h>

Transforma uma porção da string.

```
size_t strxfrm(char *s1, char *s2, size_t n);
```

retorno: número de caracteres resultantes da transformação, excluindo o '\0'.

argumentos: dois ponteiros, apontando respectivamente para a string (s1) a transformar, para a string (s2) e o número de caracteres a transformar.

**Obs :** Transforma a string s2 na string s1.

DOS	UNIX	WINDOWS	ANSI C
Sim		Sim	Sim

### Ver também:

Exemplo

```

#include <stdio.h>
#include <string.h>
#include <alloc.h>

int main(void)
{
    char *target;
    char *source = "Frank Borland";
    int length;

    /* Aloca espaço para string transformada */
    target = (char *) calloc(80, sizeof(char));

    /* copia a string source para a recém alocada target e obtém o comprimento */
    length = strxfrm(target, source, 80);

    /* mostra o resultado */
    printf("%s tem o comprimento de %d\n", target, length);
    return 0;
}

```

## strupr

#include <string.h>

Converte os caracteres de uma string em maiúsculas.

```
char *strupr(char *s);
```

retorno: devolve ponteiro para s já convertida em maiúsculas.

argumentos: ponteiro para char, referenciando uma string a converter.

**Obs :** Converte só os caracteres de **a** a **z** em **A** a **Z**. Os restante caracteres mantém-se inalterados.

DOS	UNIX	WINDOWS	ANSI C
Sim	Sim		

**Ver também:** tolower, toupper

Exemplo

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

int main(void)
{
    char *string = "universidade do algarve 8000 faro";

    printf("string antes de strupr(): %s\n", string);
    strupr(string);
    printf("string depois de strupr():  %s\n", string);
    return 0;
}
```