Índice :

memccpy **memcpy** **memmove**	<mem.h> <string.h>

Prototipos :

```
. void *memccpy(void *dest, const void *orig, int car, size_t cant);
. void *memcpy (void *dest, const void *orig, size_t cant);
. void *memmove(void *dest, const void *orig, size_t cant);
```

Observaciones :

Cada una de estas funciones copia un bloque de **cant** bytes de **orig** a **dest** .

. **memccpy** : la copia termina tan pronto como se copió el byte **car** de **orig** a **dest**, o se han copiado **cant** bytes de **orig** a **dest**, lo que ocurra primero .

. **memcpy**, **memccpy** : si orig y dest se superponen su comportamiento es indefinido.

. **memmove** : aunque los bloques **orig** y **dest** estén superpuestos los bytes en las areas superpuestas se copian correctamente a **dest** .

Devuelven :

. **memccpy** : Si se copió **car** devuelve la dirección del byte inmediato siguiente al byte **car** en el bloque destino, de lo contrario devuelve NULL .

. **memcpy y memmove** devuelven **dest** .

Compatibilidad	DOS	UNIX	ANSI C
memccpy	Si	Si	
memcpy	Si	Si	Si
memmove	Si	Si	Si

Ejemplo :

```
void *mem_ccpy(void *dest, const void *orig, int car, size_t cant)
{
   int marca = 0;

   while(cant-- && !marca)
   {
      *(char *)dest = *(char *)orig;
      if(*(char *)dest == (char)car)
         marca++;
      else
      {
         ((char *)dest)++;
         ((char *)orig)++;
      }
   }
   return *(char *)dest == (char)car && marca ? (char *)dest + 1 : NULL;
}
```

```
void *mem_cpy(void *dest, const void *orig, size_t cant)
{
   void *auxi = dest;
   while(cant--)
      *((char *)dest)++ = *((char *)orig)++;
   return auxi;
}
```

```
void *mem_move(void *dest, const void *orig, size_t cant)
{
   void *auxi = dest;
   if(orig >= dest)
      while(cant--)
         *((char *)dest)++ = *((char *)orig)++;
   else
   {
      (char *)dest += cant - 1;
      (char *)orig += cant - 1;
      while(cant--)
         *((char *)dest)-- = *((char *)orig)--;
   }
```

```
    return auxi;
}
```

memchr	<mem.h> <string.h>

Prototipo :
. **void *memchr(const void *psz, int car, size_t cant);**

Observaciones :
. **memchr** busca en los primeros **cant** bytes desde la dirección **psz** la primer ocurrencia del carácter **car** .

Devuelve : la dirección de la primer aparición de **car** a partir de **psz**, si no devuelve NULL .

Compatibiliodad	DOS	UNIX	ANSI C
memchr	Si	Si	Si

Ejemplo :

```
void *mem_chr(const void *psz, int car, size_t cant)
{
   while(cant && (char)car != *(char *)dest)
   {
      cant--;
      ((char *)dest)++;
   }
   return cant ? dest : NULL;
}
```

memcmp **memicmp**	<mem.h> <string.h>

Prototipos :
. **int memcmp (const void *psz1, const void *psz2, size_t cant);**
. **int memicmp(const void *psz1, const void *psz2, size_t cant);**

Observaciones :
. **memcmp** compara los primeros **cant** bytes a partir de las direcciones **psz1** y **psz2**, como unsigned char .
. **memicmp** compara los primeros **cant** bytes a partir de las direcciones **psz1** y **psz2**, como unsigned char sin tener en cuenta mayúsculas y minúsculas .

Devuelven : Debido a que comparan bytes como unsigned char devuelven un valor :
. < 0 si psz1 < psz2
. = 0 si psz1 == psz2
. > 0 si psz1 > psz2

Compatibilidad	DOS	UNIX	ANSI C
memcmp	Si	Si	Si
memicmp	Si	Si	

Ejemplos :

```
int mem_cmp(const void *psz1, const void *psz2, size_t cant)
{
   if(!cant)
      return 0;
   while(--cant && *(char *)psz1 == *(char *)psz2)
   {
      ++(char *)psz1;
      ++(char *)psz2;
   }
   return *(unsigned char *)psz1 - *(unsigned char *)psz2;
}
```

```
/* empleando <ctype.h> */
int mem_icmp(const void *psz1, const void *psz2, size_t cant)
{
   if(!cant)
      return 0;
   while(--cant && tolower(*(char *)psz1) == tolower(*(char *)psz2))
   {
      ++(char *)psz1;
```

```
      ++(char *)psz2;
   }
   return toupper(*(unsigned char *)psz1) - toupper(*(unsigned char *)psz2);
}
```

memset	<mem.h> <string.h>

Prototipo :

. **void *memset (void *psz, int car, size_t cant);**

Observaciones :

. **memset** asigna el byte **car** al bloque de memoria de **cant** bytes a partir de la dirección de memoria **psz** .

Devuelve :

. la misma dirección (**psz**) que recibe por argumento .

Compatibilidad	DOS	UNIX	ANSI C
memset	Si	Si	Si

Ejemplo :

```
void *mem_set (void *psz, int car, size_t cant)
{
   void *aux = psz;
   while(cant--)
      *((char *)psz)++ = (char)car;
   return aux;
}
```

strcat	<string.h>

Prototipo :

. **char *strcat(char *dest, const char *orig);**

Observaciones :

. **strcat** concatena una copia de **orig** al final de **dest**, la longitud de la cadena resultante es la suma de las longitudes, la cadena resultante tendrá su carácter terminador nulo . El terminador nulo de **dest** es pisado por el primer carácter de **orig** y así sucesivamente . Su comportamiento es indefinido si las cadenas **dest** y **orig** se superponen .

Devuelve :

. **strcat** devuelve la misma dirección **dest** que recibió por argumento .

Compatibilidad	DOS	UNIX	ANSI C
strcat	Si	Si	Si

Ejemplo :

```
char *str_cat(char *dest, const char *orig)
{
   char *auxi = dest;
   while(*dest)
      dest++;
   while((*dest++ = *((char *orig)++) != '\0')
      ;
   return auxi;
}
```

strchr	<string.h>

Prototipo :

. **char *strchr(const char *psz, int car);**

Observaciones :

. **strchr** busca a partir de la dirección **psz** hasta encontrar el carácter **car** o el carácter nulo, lo que ocurra primero . Se considera que el carácter nulo forma parte de la cadena, con lo que se puede obtener su dirección de memoria .

Devuelve :

. Si encuentra **car** antes del carácter nulo, devuelve su dirección de memoria .
. Si no encuentra **car**, pero sí encuentra el carácter nulo, devuelve NULL .

Compatibilidad	DOS	UNIX	ANSI C

strchr	Si	Si	Si

Ejemplo :

```
char *str_chr(const char *psz, int car)
{
   while(*psz && *(char *)psz != (char)car)
      ((char *)psz)++;
   return *(char *)psz == (char)car ? (char *)psz : (char *)NULL;
}
```

strcmp **strcmpi** **stricmp**	<string.h>

Prototipos :

- **int strcmp(const char *psz1, const char *psz2);**
- **int strcmpi(const char *psz1, const char *psz2);**
- **int stricmp(const char *psz1, const char *psz2);**

Observaciones :

. **strcmp** hace una comparación sin signo (unsigned char) de las cadenas **psz1 y psz2** .
. **strcmpi** hace una comparación sin signo (unsigned char) sin tener en cuenta mayúsculas y minúsculas (habitualmente declarada como macro) .
. **stricmp** hace una comparación sin signo (unsigned char) sin tener en cuenta mayúsculas y minúsculas .
La comparación de cadenas comienza con el primer carcater y termina en cuanto difieren o se llega al fin de las cadenas .

Devuelven :

. Por comparar enteros char sin signo, devuelven un valor entero :
- . < 0 si psz1 < psz2
- . == 0 si psz1 == psz2
- . > 0 si psz1 > psz2

Compatibilidad	DOS	UNIX	ANSI C
strcmp	Si	Si	Si
strcmpi	Si		
stricmp	Si		

Ejemplo :

```
int str_cmp(const char *psz1, const char *psz2)
{
   while(*psz1 && *psz1 == *psz2)
   {
      psz1++;
      psz2++;
   }
   return *(unsigned char *)psz1 - *(unsigned char *)psz2;
}
```

```
int str_icmp(const char *psz1, const char *psz2)
{
   while(*psz1 && toupper(*psz1) == toupper(*psz2))
   {
      psz1++;
      psz2++;
   }
   return toupper(*(unsigned char )psz1) - toupper((unsigned char *)psz2);
}
```

```
#define     str_cmpi(X, Y)            str_icmp(X, Y)
```

strcpy	<string.h>

Prototipo :

- **char *strcpy(char *dest, const char *orig);**

Observaciones :

. Copia la cadena desde **orig** a **dest**, termina una vez copiado el terminador nulo .

Devuelve :

. la misma dirección **dest** .

Compatibilidad	DOS	UNIX	ANSI C
strcpy	Si	Si	Si

Ejemplo :

```
char *str_cpy(char *dest, const char *orig)
{
   char *auxi = dest;
   while((*dest++ = *((char *)orig)++) != '\0')
      ;
   return auxi;
}
```

strcspn	<string.h>
strspn	

Prototipos :

```
. size_t strcspn(const char *psz1, const char *psz2);
. size_t strspn(const char *psz1, const char *psz2);
```

Observaciones :

. **strcspn** encuentra el tamaño de la porción inicial de la cadena **psz1** que no contiene ningún carácter de la cadena **psz2** .

. **strspn** encuentra el tamaño de la porción inicial de la cadena **psz1** que consiste enteramente de carácteres de la cadena **psz2** .

Devuelven :

. **strcspn** y **strspn** : la longitud de la subcadena encontrada .

Compatibilidad	DOS	UNIX	ANSI C
strcspn	Si	Si	Si
strspn	Si	Si	Si

Ejemplos :

```
size_t str_cspn(const char *psz1, const char *psz2)
{
   size_t cant = 0;
   while(*psz1 && !strchr(psz2, *psz1))
   {
      cant++;
      psz1++;
   }
   return cant;
}
```

```
size_t str_spn(const char *psz1, const char *psz2)
{
   size_t cant = 0;
   while(*psz1 && strchr(psz2, *psz1))
   {
      cant++;
      psz1++;
   }
   return cant;
}
```

strdup	<string.h>

Prototipo :

```
. char *strdup(const char *psz);
```

Observaciones :

. **strdup** copia una cadena en un espacio de almacenamiento generado dinámicamente (empleando malloc) . El espacio obtenido es el ocupado por la cadena original, incluyendo el terminador nulo . El programador es responsable de liberar el espacio de memoria obtenido por **strdup** cuando ya no es necesario .

Valor devuelto :

. Si es exitoso, devuelve la dirección de memoria de la cadena copiada .
. De lo contrario, si no pudo generar espacio de almacenamiento, devuelve NULL.

Compatibilidad	DOS	UNIX	ANSI C

strdup	Si	Si	

Ejemplo :

```
char *str_dup(const char *psz)
{
   char *auxi = (char *)malloc(strlen(psz) + 1);
   if(auxi)
      strcpy(auxi, psz);
   return auxi;
}
```

strerror	<stdio.h> <string.h>

Prototipo :

. **char *strerror(int errnum);**

Observaciones :

. **strerror** recibe por argumento un número de error y devuelve la dirección de comienzo de un mensaje de error relacionada con tal número de error .

Valor devuelto :

. **strerror** devuelve la dirección de comienzo del mensaje de error vinculado con **errnum.**

Compatibilidad	DOS	UNIX	ANSI C
strerror	Si		Si

Ejemplo :

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

char *str_error(int errnum)
{
   static char msg_error[40];
   sprintf(msg_error,
           "%.*s\n",
           sizeof(msg_error) - 2,
           errnum >= 0 && errnum < sys_nerr ?
                  sys_errlist[errnum] :
                  "Unknown error");
   return msg_error;
}

void main(void)
{
   int ciclo = 0;

   while(ciclo <= sys_nerr)
   {
      printf("Error %2d : %s", ciclo, strerror(ciclo));
      printf("Error %2d : %s", ciclo, str_error(ciclo));
      ciclo++;
      if(ciclo % 10 == 0)
      {
         fflush(stdin);
         scanf("%*c");
      }
   }
}
```

strlen	<string.h>

Prototipo :

. **size_t strlen(const char *cade);**

Observaciones :

. **strlen** Devuelve cuántos carácteres tiene una cadena antes del terminador nulo .

Valor devuelto :

. Devuelve un size_t (unsigned) que indica el número de carácteres sin contar el terminador carácter nulo .

Compatibilidad	DOS	UNIX	ANSI C
strlen	Si	Si	Si

Ejemplo

```
size_t strlen(const char *cade)
{
   size_t cont = 0;
   while(*((char *)cade)++)
      cont++;
   return cont;
}
```

strlwr **strupr**	<string.h>

Prototipo :

- **. char *strlwr(char *psz);**
- **. char *strupr(char *psz);**

Observaciones :

. **strlwr** transforma los carácteres en mayúscula de la cadena, cuya dirección de comienzo recibe, a minúscula (A..Z -> a..z) .

. **strupr** transforma los carácteres en minúscula de la cadena, cuya dirección de comienzo recibe, a mayúscula (a..z -> A..Z) .

Los restantes carácteres permanecen sin cambios .

Valor devuelto :

. La dirección de comienzo de la cadena **psz** .

Compatibilidad	DOS	UNIX	ANSI C
strlwr	Si		
strupr	Si		

Ejemplo :

```
char *str_lwr(char *psz)
{
   char *aux = psz;
   while(*psz)
      psz = tolower(psz);
   return aux;
}
```

```
char *str_upr(char *psz)
{
   char *aux = psz;
   while(*psz)
      psz = toupper (psz);
   return aux;
}
```

strncat	<string.h>

Prototipo :

- **. char *strncat(char *dest, const char *orig, size_t cant);**

Observaciones :

. **strncat** copia a lo sumo **cant** carácteres de la cadena **orig** al final de la cadena **dest** o hasta que encuentre un carácter nulo en **orig** . Siempre agrega un carácter terminador nulo al terminar su cometido . Es responsabilidad del programador que el espacio de almacenamiento de **dest** sea suficiente para contener la cadena concatenada .

Valor devuelto :

. **dest** o sea, la misma dirección de memoria que recibe en su primer argumento .

Compatibilidad	DOS	UNIX	ANSI C
strncat	Si	Si	Si

Ejemplo :

```
char *str_ncat(char *dest, const char *orig, size_t cant)
{
```

```
   char *auxi = dest;
   while(*dest)
      dest++;
   while(*orig && cant--)
      *dest++ = *((char *)orig)++;
   *dest = '\0';
   return auxi;
}
```

strncmp **strncmpi** **strnicmp**	<string.h>

Prototipos :

- **int strncmp (const char *pss1, const char *psz2, size_t cant);**
- **int strncmpi(const char *pss1, const char *psz2, size_t cant);**
- **int strnicmp(const char *pss1, const char *psz2, size_t cant);**

Observaciones :

Cada una de estas funciones compara a lo sumo los primeros **cant** carácteres de las cadenas **psz1** y **psz2** o hasta el final de las mismas, lo que ocurra primero .

. **strncmp** hace una comparación sin signo entre **psz1** y **psz2** .

. **strncmpi** (habitualmente una macro) hace una comparación con signo entre psz1 y psz2 sin importarle las minúsculas .

. **strnicmp** hace una comparación con signo entre psz1 y psz2 sin importarle las minúsculas .

La comparación de cadenas comienza con el primer carácter de cada cadena y continúa hasta que las cadenas difieren o se alcanza el final de alguna de ellas o hasta comparar **cant** carácteres .

Valor devuelto :

Devuelven un valor entero, resultado de la comparación sin o con signo (según corresponda) dentro del ámbito de la comparación de psz1 con psz2 .

- . < 0 si psz1 < psz2
- . == 0 if psz1 == psz2
- . > 0 if psz1 > psz2

Compatibilidad	DOS	UNIX	ANSI C
strncmp	Si	Si	Si
strncmpi	Si		
strnicmp	Si		

Ejemplos :

```
int str_ncmp (const char *psz1, const char *psz2, size_t cant)
{
   if(cant == 0)
      return 0;
   while(*psz1 && --cant && *psz1 == *psz2)
   {
      psz1++;
      psz2++;
   }
   return *(unsigned char *)psz1 - *(unsigned char *)psz2;
}
```

```
int str_nicmp(const char *psz1, const char *psz2, size_t cant)
{
   if(cant == 0)
      return 0;
   while(*psz1 && --cant && tolower(*psz1) == tolower(*psz2))
   {
      psz1++;
      psz2++;
   }
   return toupper(*psz1) - toupper(*psz2);
}
```

```
#define  str_ncmpi         str_nicmp
```

strncpy	<string.h>

Prototipo :

. **char *strncpy(char *dest, const char *orig, size_t cant);**

Observaciones :

. **strncpy** copia a lo sumo **cant** carácteres de **orig** a **dest** . La cadena **dest** puede no resultar terminada con el carácter nulo si la longitud de la cadena **orig** es mayor o igual que **cant**, de lo contrario se completará **dest** con tantos carácteres nulos como sean necesarios para completar **cant** carácteres en **dest** .

Valor devuelto :

dest, o sea que devuelve la misma dirección que recibe en su primer argumento .

Compatibilidad	DOS	UNIX	ANSI C
strncpy	Si	Si	Si

Ejemplo :

```
char *str_ncpy(char *dest, const char *orig, size_t cant)
{
   char *auxi = dest;
   while(cant && *orig)
   {
      *dest++ = *((char *)orig)++;
      cant--;
   }
   while(cant--)
      *dest++ = '\0';
   return auxi;
}
```

strpbrk	<string.h>

Busca en **cad1** la primer ocurrencia de uno de los carácteres de **cad2**, devolviendo la dirección de memoria en que lo encuentra .

Prototipo :

. **char *strpbrk(const char *cad1, const char *cad2);**

Observaciones :

. **strpbrk** Busca en **cad1** la primer ocurrencia de uno de los carácteres de **cad2**, devolviendo la dirección de memoria en que lo encuentra .

Valor devuelto :

. Si es exitoso, **strpbrk** devuelve la dirección de memoria en que encuentra en **cad1** la primer ocurrencia de uno de los carácteres de **cad2** .

. Si no, devuelve NULL .

Compatibilidad	DOS	UNIX	ANSI C
strpbrk	Si	Si	Si

Ejemplo :

```
char *str_pbrk(const char *cad1, const char *cad2)
{
   while(*cad1 && !strchr(cad2, *cad1))
      cad1++;
   return *cad1 ? (char *)cad1 : NULL;
}
```

strrchr	<string.h>

Prototipo :

. **char *strrchr(const char *cad, int car);**

Observaciones :

. **strrchr** busca en la cadena **cad** la última ocurrencia de la carácter **car** . Se considera que el terminador nulo es parte de la cadena .

Valor devuelto :

. **strrchr** devuelve la dirección de memoria en que encuentra el carácter **car** dentro de la cadena **cad**, si no lo encuentra devuelve NULL .

Compatibilidad	DOS	UNIX	ANSI C
strrchr	Si	Si	Si

Ejemplo :

```
char *str_rchr(const char *cad, char car)
{
   char *aux = (char *)cad;
   while(*aux)
      aux++;
   while(aux > cad && *aux != car)
      aux--;
   return *aux == car ? aux : NULL;
}
```

strstr	<string.h>

Prototipo :

. **char *strstr(const char *cad, const char *sub);**

Observaciones :

. **strstr** busca en **cad** en qué dirección de memoria encuentra por primera vez la subcadena **sub** . La búsqueda no incluye el terminador nulo de la subcadena **sub** .

Valor devuelto :

. **strstr** devuelve la dirección en que encuentra la subcadena apuntada por **sub** en la cadena **cad**, de no encontrarla, devuelve NULL .

Compatibilidad	DOS	UNIX	ANSI C
strstr	Si	Si	Si

Ejemplo :

```
char *str_str(const char *cad, const char *sub)
{
   size_t  tam = strlen(sub);
   while(*cad && strncmp(cad, sub, tam))
      cad++;
   return *cad ? (char *)cad : NULL;
}
```

strtok	<string.h>

Prototipo :

. **char *strtok(char *cad, const char *sep);**

Observaciones :

. **strtok** considera que **cad** consta de 0 o más subcadenas iniciadas, separadas, y/o terminadas por 0 o más carácteres de **sep** .
La primer invocación a **strtok** devuelve la dirección de la primer subcadena en **cad**, almacenando un terminador nulo en el primer carácter separador encontrado en **cad** .
Para las siguientes llamadas con NULL en el primer argumento, conserva la dirección siguiente al terminador nulo para buscar a partir de allí el comienzo de la siguiente subcadena . Las siguientes llamadas pueden hacerse con una cadena de separadores distinta de una llamada a otra .
Esta función emplea una variable estática para separar las subcadenas de **cad**, por lo que solo se la debería invocar repetidamente para una sola cadena .

Valor devuelto :

. Si la encuentra, devuelve la dirección de comienzo de la primer subcadena encontrada en **cad**, de lo contrario devuelve NULL .

Compatibilidad	DOS	UNIX	ANSI C
strtok	Si	Si	Si

Ejemplo :

```
char *str_tok(char *cad, const char *sep)
{
   static char *sig;
   char        *aux;
   if(cad)
```

```
      sig = cad;
   while(*sig && strchr(sep, *sig))
      sig++;
   aux = sig;
   while(*sig && !strchr(sep, *sig))
      sig++;
   if(*sig)
      *sig++ = '\0';
   return *aux ? aux : NULL;
}
```