BAKI ALİ NEFT MƏKTƏBİ
BAKU HIGHER OIL SCHOOL

Programming and Computer Applications-1

Characters and Strings

Instructor : PhD, Associate Professor Leyla Muradkhanli

String-Manipulation Functions of the String-Handling Library

- The string-handling library (`<string.h>`) provides many useful functions for manipulating string data (copying strings and concatenating strings), comparing strings, searching strings for characters and other strings, tokenizing strings (separating strings into logical pieces) and determining the length of strings.
- This section presents the string-manipulation functions of the string-handling library.
- The functions are summarized in Fig. 8.17.
- Every function—except for `strncpy`—appends the null character to its result.

Function prototype	Function description
`char *strcpy( char *s1, const char *s2 )`	Copies string `s2` into array `s1`. The value of `s1` is returned.
`char *strncpy( char *s1, const char *s2, size_t n )`	Copies at most `n` characters of string `s2` into array `s1`. The value of `s1` is returned.
`char *strcat( char *s1, const char *s2 )`	Appends string `s2` to array `s1`. The first character of `s2` overwrites the terminating null character of `s1`. The value of `s1` is returned.
`char *strncat( char *s1, const char *s2, size_t n )`	Appends at most `n` characters of string `s2` to array `s1`. The first character of `s2` overwrites the terminating null character of `s1`. The value of `s1` is returned.

Fig. 8.17 | String-manipulation functions of the string-handling library.

String-Manipulation Functions of the String-Handling Library (Cont.)

- Function `strcpy` copies its second argument (a string) into its first argument (a character array that must be large enough to store the string and its terminating null character, which is also copied).
- Function `strncpy` is equivalent to `strcpy`, except that `strncpy` specifies the number of characters to be copied from the string into the array.
- Function `strncpy` does not necessarily copy the terminating null character of its second argument.

String-Manipulation Functions of the String-Handling Library (Cont.)

- A terminating null character is written only if the number of characters to be copied is at least one more than the length of the string.
- For example, if `"test"` is the second argument, a terminating null character is written only if the third argument to `strncpy` is at least `5` (four characters in `"test"` plus a terminating null character).
- If the third argument is larger than `5`, null characters are appended to the array until the total number of characters specified by the third argument are written.

String-Manipulation Functions of the String-Handling Library (Cont.)

- Figure 8.18 uses `strcpy` to copy the entire string in array `x` into array `y` and uses `strncpy` to copy the first `14` characters of array `x` into array `z`.
- A null character (`'\0'`) is appended to array `z`, because the call to `strncpy` in the program does not write a terminating null character (the third argument is less than the string length of the second argument).

```c
/* Fig. 8.18: fig08_18.c
   Using strcpy and strncpy */
#include <stdio.h>
#include <string.h>

int main( void )
{
   char x[] = "Happy Birthday to You"; /* initialize char array x */
   char y[ 25 ]; /* create char array y */
   char z[ 15 ]; /* create char array z */

   /* copy contents of x into y */
   printf( "%s%s\n%s%s\n",
      "The string in array x is: ", x,
      "The string in array y is: ", strcpy( y, x ) );

   /* copy first 14 characters of x into z. Does not copy null
      character */
   strncpy( z, x, 14 );

   z[ 14 ] = '\0'; /* terminate string in z */
   printf( "The string in array z is: %s\n", z );
   return 0; /* indicates successful termination */
} /* end main */
```

Fig. 8.18 | Using `strcpy` and `strncpy`. (Part 1 of 2.)

```
The string in array x is: Happy Birthday to You
The string in array y is: Happy Birthday to You
The string in array z is: Happy Birthday
```

Fig. 8.18 | Using `strcpy` and `strncpy`. (Part 2 of 2.)

String-Manipulation Functions of the String-Handling Library (Cont.)

- Function `strcat` appends its second argument (a string) to its first argument (a character array containing a string).
- The first character of the second argument replaces the null (`'\0'`) that terminates the string in the first argument.
- You must ensure that the array used to store the first string is large enough to store the first string, the second string and the terminating null character copied from the second string.
- Function `strncat` appends a specified number of characters from the second string to the first string.
- A terminating null character is appended to the result.
- Figure 8.19 demonstrates functions `strcat` and `strncat`.

```c
/* Fig. 8.19: fig08_19.c
   Using strcat and strncat */
#include <stdio.h>
#include <string.h>

int main( void )
{
   char s1[ 20 ] = "Happy "; /* initialize char array s1 */
   char s2[] = "New Year "; /* initialize char array s2 */
   char s3[ 40 ] = ""; /* initialize char array s3 to empty */

   printf( "s1 = %s\ns2 = %s\n", s1, s2 );

   /* concatenate s2 to s1 */
   printf( "strcat( s1, s2 ) = %s\n", strcat( s1, s2 ) );

   /* concatenate first 6 characters of s1 to s3. Place '\0'
      after last character */
   printf( "strncat( s3, s1, 6 ) = %s\n", strncat( s3, s1, 6 ) );

   /* concatenate s1 to s3 */
   printf( "strcat( s3, s1 ) = %s\n", strcat( s3, s1 ) );
   return 0; /* indicates successful termination */
} /* end main */
```

Fig. 8.19 | Using `strcat` and `strncat`. (Part 1 of 2.)

```
s1 = Happy
s2 = New Year
strcat( s1, s2 ) = Happy New Year
strncat( s3, s1, 6 ) = Happy
strcat( s3, s1 ) = Happy Happy New Year
```

Fig. 8.19 | Using `strcat` and `strncat`. (Part 2 of 2.)

Comparison Functions of the String-Handling Library

- This section presents the string-handling library's string-comparison functions, `strcmp` and `strncmp`.
- Fig. 8.20 contains their prototypes and a brief description of each function.

Function prototype	Function description
`int strcmp( const char *s1, const char *s2 );`	Compares the string `s1` with the string `s2`. The function returns `0`, less than `0` or greater than `0` if `s1` is equal to, less than or greater than `s2`, respectively.
`int strncmp( const char *s1, const char *s2, size_t n );`	Compares up to `n` characters of the string `s1` with the string `s2`. The function returns `0`, less than `0` or greater than `0` if `s1` is equal to, less than or greater than `s2`, respectively.

Fig. 8.20 | String-comparison functions of the string-handling library.

Comparison Functions of the String-Handling Library (Cont.)

- Figure 8.21 compares three strings using `strcmp` and `strncmp`.
- Function `strcmp` compares its first string argument with its second string argument, character by character.
- The function returns 0 if the strings are equal, a negative value if the first string is less than the second string and a positive value if the first string is greater than the second string.
- Function `strncmp` is equivalent to `strcmp`, except that `strncmp` compares up to a specified number of characters.
- Function `strncmp` does not compare characters following a null character in a string.
- The program prints the integer value returned by each function call.

```c
/* Fig. 8.21: fig08_21.c
   Using strcmp and strncmp */
#include <stdio.h>
#include <string.h>

int main( void )
{
   const char *s1 = "Happy New Year"; /* initialize char pointer */
   const char *s2 = "Happy New Year"; /* initialize char pointer */
   const char *s3 = "Happy Holidays"; /* initialize char pointer */

   printf("%s%s\n%s%s\n%s%s\n\n%s%2d\n%s%2d\n%s%2d\n\n",
      "s1 = ", s1, "s2 = ", s2, "s3 = ", s3,
      "strcmp(s1, s2) = ", strcmp( s1, s2 ),
      "strcmp(s1, s3) = ", strcmp( s1, s3 ),
      "strcmp(s3, s1) = ", strcmp( s3, s1 ) );

   printf("%s%2d\n%s%2d\n%s%2d\n",
      "strncmp(s1, s3, 6) = ", strncmp( s1, s3, 6 ),
      "strncmp(s1, s3, 7) = ", strncmp( s1, s3, 7 ),
      "strncmp(s3, s1, 7) = ", strncmp( s3, s1, 7 ) );
   return 0; /* indicates successful termination */
} /* end main */
```

Fig. 8.21 | Using `strcmp` and `strncmp`. (Part 1 of 2.)

```
s1 = Happy New Year
s2 = Happy New Year
s3 = Happy Holidays

strcmp(s1, s2) =  0
strcmp(s1, s3) =  1
strcmp(s3, s1) = -1

strncmp(s1, s3, 6) =  0
strncmp(s1, s3, 7) =  1
strncmp(s3, s1, 7) = -1
```

Fig. 8.21 | Using `strcmp` and `strncmp`. (Part 2 of 2.)

Comparison Functions of the String-Handling Library (Cont.)

- To understand just what it means for one string to be “greater than” or “less than” another string, consider the process of alphabetizing a series of last names.
- The reader would, no doubt, place “Jones” before “Smith,” because the first letter of “Jones” comes before the first letter of “Smith” in the alphabet.

Comparison Functions of the String-Handling Library (Cont.)

- But the alphabet is more than just a list of 26 letters—it is an ordered list of characters.
- Each letter occurs in a specific position within the list.
- "Z" is more than merely a letter of the alphabet; "Z" is specifically the 26^{th} letter of the alphabet.
- How does the computer know that one particular letter comes before another?
- All characters are represented inside the computer as numeric codes; when the computer compares two strings, it actually compares the numeric codes of the characters in the strings.

Comparison Functions of the String-Handling Library (Cont.)

- In an effort to standardize character representations, most computer manufacturers have designed their machines to utilize one of two popular coding schemes—ASCII or EBCDIC.
- ASCII stands for “American Standard Code for Information Interchange,” and EBCDIC stands for “Extended Binary Coded Decimal Interchange Code.”
- There are other coding schemes, but these two are the most popular.

Search Functions of the String-Handling Library

- This section presents the functions of the string-handling library used to search strings for characters and other strings.
- The functions are summarized in Fig. 8.22.

Function prototype and description
`char *strchr( const char *s, int c );` Locates the first occurrence of character `c` in string `s`. If `c` is found, a pointer to `c` in `s` is returned. Otherwise, a `NULL` pointer is returned.
`size_t strcspn( const char *s1, const char *s2 );` Determines and returns the length of the initial segment of string `s1` consisting of characters *not* contained in string `s2`.
`size_t strspn( const char *s1, const char *s2 );` Determines and returns the length of the initial segment of string `s1` consisting only of characters contained in string `s2`.
`char *strpbrk( const char *s1, const char *s2 );` Locates the first occurrence in string `s1` of any character in string `s2`. If a character from string `s2` is found, a pointer to the character in string `s1` is returned. Otherwise, a `NULL` pointer is returned.
`char *strrchr( const char *s, int c );` Locates the last occurrence of `c` in string `s`. If `c` is found, a pointer to `c` in string `s` is returned. Otherwise, a `NULL` pointer is returned.

Fig. 8.22 | String-manipulation functions of the string-handling library. (Part 1 of 2.)

Function prototype and description
`char *strstr( const char *s1, const char *s2 );` Locates the first occurrence in string `s1` of string `s2`. If the string is found, a pointer to the string in `s1` is returned. Otherwise, a `NULL` pointer is returned.
`char *strtok( char *s1, const char *s2 );` A sequence of calls to `strtok` breaks string `s1` into "tokens"—logical pieces such as words in a line of text—separated by characters contained in string `s2`. The first call contains `s1` as the first argument, and subsequent calls to continue tokenizing the same string contain `NULL` as the first argument. A pointer to the current token is returned by each call. If there are no more tokens when the function is called, `NULL` is returned.

Fig. 8.22 | String-manipulation functions of the string-handling library. (Part 2 of 2.)

Search Functions of the String-Handling Library (Cont.)

- Function `strchr` searches for the first occurrence of a character in a string.
- If the character is found, `strchr` returns a pointer to the character in the string; otherwise, `strchr` returns `NULL`.
- Figure 8.23 uses `strchr` to search for the first occurrences of `'a'` and `'z'` in the string `"This is a test"`.

```c
/* Fig. 8.23: fig08_23.c
   Using strchr */
#include <stdio.h>
#include <string.h>

int main( void )
{
   const char *string = "This is a test"; /* initialize char pointer */
   char character1 = 'a'; /* initialize character1 */
   char character2 = 'z'; /* initialize character2 */

   /* if character1 was found in string */
   if ( strchr( string, character1 ) != NULL ) {
      printf( "\'%c\' was found in \"%s\".\n",
         character1, string );
   } /* end if */
   else { /* if character1 was not found */
      printf( "\'%c\' was not found in \"%s\".\n",
         character1, string );
   } /* end else */

```

Fig. 8.23 | Using `strchr`. (Part 1 of 2.)

```
22      /* if character2 was found in string */
23      if ( strchr( string, character2 ) != NULL ) {
24         printf( "\'%c\' was found in \"%s\".\n",
25            character2, string );
26      } /* end if */
27      else { /* if character2 was not found */
28         printf( "\'%c\' was not found in \"%s\".\n",
29            character2, string );
30      } /* end else */
31
32      return 0; /* indicates successful termination */
33   } /* end main */
```

```
'a' was found in "This is a test".
'z' was not found in "This is a test".
```

Fig. 8.23 | Using `strchr`. (Part 2 of 2.)

Search Functions of the String-Handling Library (Cont.)

- Function `strcspn` (Fig. 8.24) determines the length of the initial part of the string in its first argument that does not contain any characters from the string in its second argument.
- The function returns the length of the segment.

```c
/* Fig. 8.24: fig08_24.c
   Using strcspn */
#include <stdio.h>
#include <string.h>

int main( void )
{
   /* initialize two char pointers */
   const char *string1 = "The value is 3.14159";
   const char *string2 = "1234567890";

   printf( "%s%s\n%s%s\n\n%s\n%s%u\n",
      "string1 = ", string1, "string2 = ", string2,
      "The length of the initial segment of string1",
      "containing no characters from string2 = ",
      strcspn( string1, string2 ) );
   return 0; /* indicates successful termination */
} /* end main */
```

Fig. 8.24 | Using `strcspn`. (Part 1 of 2.)

```
string1 = The value is 3.14159
string2 = 1234567890

The length of the initial segment of string1
containing no characters from string2 = 13
```

Fig. 8.24 | Using `strcspn`. (Part 2 of 2.)

Search Functions of the String-Handling Library (Cont.)

- Function `strpbrk` searches its first string argument for the first occurrence of any character in its second string argument.
- If a character from the second argument is found, `strpbrk` returns a pointer to the character in the first argument; otherwise, `strpbrk` returns `NULL`.
- Figure 8.25 shows a program that locates the first occurrence in `string1` of any character from `string2`.

```c
/* Fig. 8.25: fig08_25.c
   Using strpbrk */
#include <stdio.h>
#include <string.h>

int main( void )
{
   const char *string1 = "This is a test"; /* initialize char pointer */
   const char *string2 = "beware"; /* initialize char pointer */

   printf( "%s\"%s\"\n'%c'%s\n\"%s\"\n",
      "Of the characters in ", string2,
      *strpbrk( string1, string2 ),
      " appears earliest in ", string1 );
   return 0; /* indicates successful termination */
} /* end main */
```

```
Of the characters in "beware"
'a' appears earliest in
"This is a test"
```

Fig. 8.25 | Using `strpbrk`.

Search Functions of the String-Handling Library (Cont.)

- Function `strrchr` searches for the last occurrence of the specified character in a string.
- If the character is found, `strrchr` returns a pointer to the character in the string; otherwise, `strrchr` returns `NULL`.
- Figure 8.26 shows a program that searches for the last occurrence of the character `'z'` in the string `"A zoo has many animals including zebras."`

```c
/* Fig. 8.26: fig08_26.c
   Using strrchr */
#include <stdio.h>
#include <string.h>

int main( void )
{
   /* initialize char pointer */
   const char *string1 = "A zoo has many animals including zebras";

   int c = 'z'; /* character to search for */

   printf( "%s\n%s'%c'%s\"%s\"\n",
      "The remainder of string1 beginning with the",
      "last occurrence of character ", c,
      " is: ", strrchr( string1, c ) );
   return 0; /* indicates successful termination */
} /* end main */
```

```
The remainder of string1 beginning with the
last occurrence of character 'z' is: "zebras"
```

Fig. 8.26 | Using `strrchr`.

Search Functions of the String-Handling Library (Cont.)

- Function `strspn` (Fig. 8.27) determines the length of the initial part of the string in its first argument that contains only characters from the string in its second argument.
- The function returns the length of the segment.

```c
/* Fig. 8.27: fig08_27.c
   Using strspn */
#include <stdio.h>
#include <string.h>

int main( void )
{
   /* initialize two char pointers */
   const char *string1 = "The value is 3.14159";
   const char *string2 = "aehi lsTuv";

   printf( "%s%s\n%s%s\n\n%s\n%s%u\n",
      "string1 = ", string1, "string2 = ", string2,
      "The length of the initial segment of string1",
      "containing only characters from string2 = ",
      strspn( string1, string2 ) );
   return 0; /* indicates successful termination */
} /* end main */
```

Fig. 8.27 | Using `strspn`. (Part 1 of 2.)

```
string1 = The value is 3.14159
string2 = aehi lsTuv

The length of the initial segment of string1
containing only characters from string2 = 13
```

Fig. 8.27 | Using `strspn`. (Part 2 of 2.)

Search Functions of the String-Handling Library (Cont.)

- Function `strstr` searches for the first occurrence of its second string argument in its first string argument.
- If the second string is found in the first string, a pointer to the location of the string in the first argument is returned.
- Figure 8.28 uses `strstr` to find the string `"def"` in the string `"abcdefabcdef"`.

```c
/* Fig. 8.28: fig08_28.c
   Using strstr */
#include <stdio.h>
#include <string.h>

int main( void )
{
   const char *string1 = "abcdefabcdef"; /* string to search */
   const char *string2 = "def"; /* string to search for */

   printf( "%s%s\n%s%s\n\n%s\n%s%s\n",
      "string1 = ", string1, "string2 = ", string2,
      "The remainder of string1 beginning with the",
      "first occurrence of string2 is: ",
      strstr( string1, string2 ) );
   return 0; /* indicates successful termination */
} /* end main */
```

```
string1 = abcdefabcdef
string2 = def

The remainder of string1 beginning with the
first occurrence of string2 is: defabcdef
```

Fig. 8.28 | Using `strstr`.

Search Functions of the String-Handling Library (Cont.)

- Function `strtok` (Fig. 8.29) is used to break a string into a series of tokens.
- A token is a sequence of characters separated by delimiters (usually spaces or punctuation marks, but a delimiter can be any character).
- For example, in a line of text, each word can be considered a token, and the spaces and punctuation separating the words can be considered delimiters.

```c
/* Fig. 8.29: fig08_29.c
   Using strtok */
#include <stdio.h>
#include <string.h>

int main( void )
{
   /* initialize array string */
   char string[] = "This is a sentence with 7 tokens";
   char *tokenPtr; /* create char pointer */

   printf( "%s\n%s\n\n%s\n",
      "The string to be tokenized is:", string,
      "The tokens are:" );

   tokenPtr = strtok( string, " " ); /* begin tokenizing sentence */

   /* continue tokenizing sentence until tokenPtr becomes NULL */
   while ( tokenPtr != NULL ) {
      printf( "%s\n", tokenPtr );
      tokenPtr = strtok( NULL, " " ); /* get next token */
   } /* end while */

   return 0; /* indicates successful termination */
} /* end main */
```

Fig. 8.29 | Using `strtok`. (Part 1 of 2.)

```
The string to be tokenized is:
This is a sentence with 7 tokens

The tokens are:
This
is
a
sentence
with
7
tokens
```

Fig. 8.29 | Using `strtok`. (Part 2 of 2.)

Search Functions of the String-Handling Library (Cont.)

- Multiple calls to `strtok` are required to tokenize a string—i.e., break it into tokens (assuming that the string contains more than one token).
- The first call to `strtok` contains two arguments: a string to be tokenized, and a string containing characters that separate the tokens.
- In Fig. 8.29, the statement
  - ```
    /* begin tokenizing sentence */
    tokenPtr = strtok( string, " " );
    ```

  assigns `tokenPtr` a pointer to the first token in `string`.
    ```

Search Functions of the String-Handling Library (Cont.)

- The second argument, `" "`, indicates that tokens are separated by spaces.
- Function `strtok` searches for the first character in `string` that is not a delimiting character (space).
- This begins the first token.
- The function then finds the next delimiting character in the string and replaces it with a null (`'\0'`) character to terminate the current token.
- Function `strtok` saves a pointer to the next character following the token in `string` and returns a pointer to the current token.

Search Functions of the String-Handling Library (Cont.)

- Subsequent `strtok` calls in line 21 continue tokenizing `string`.
- These calls contain `NULL` as their first argument.
- The `NULL` argument indicates that the call to `strtok` should continue tokenizing from the location in `string` saved by the last call to `strtok`.
- If no tokens remain when `strtok` is called, `strtok` returns `NULL`.

Search Functions of the String-Handling Library (Cont.)

- You can change the delimiter string in each new call to `strtok`.
- Figure 8.29 uses `strtok` to tokenize the string `"This is a sentence with 7 tokens"`.
- Each token is printed separately.
- Function `strtok` modifies the input string by placing `\0` at the end of each token; therefore, a copy of the string should be made if the string will be used again in the program after the calls to `strtok`.

Memory Functions of the String-Handling Library

- The string-handling library functions presented in this section manipulate, compare and search blocks of memory.
- The functions treat blocks of memory as character arrays and can manipulate any block of data.
- Figure 8.30 summarizes the memory functions of the string-handling library.
- In the function discussions, “object” refers to a block of data.

Function prototype	Function description
`void *memcpy( void *s1, const void *s2, size_t n );`	Copies `n` characters from the object pointed to by `s2` into the object pointed to by `s1`. A pointer to the resulting object is returned.
`void *memmove( void *s1, const void *s2, size_t n );`	Copies `n` characters from the object pointed to by `s2` into the object pointed to by `s1`. The copy is performed as if the characters were first copied from the object pointed to by `s2` into a temporary array and then from the temporary array into the object pointed to by `s1`. A pointer to the resulting object is returned.
`int memcmp( const void *s1, const void *s2, size_t n );`	Compares the first `n` characters of the objects pointed to by `s1` and `s2`. The function returns `0`, less than `0` or greater than `0` if `s1` is equal to, less than or greater than `s2`.

Fig. 8.30 | Memory functions of the string-handling library. (Part 1 of 2.)

Function prototype	Function description
`void *memchr( const void *s, int c, size_t n );`	Locates the first occurrence of `c` (converted to `unsigned char`) in the first `n` characters of the object pointed to by `s`. If `c` is found, a pointer to `c` in the object is returned. Otherwise, `NULL` is returned.
`void *memset( void *s, int c, size_t n );`	Copies `c` (converted to `unsigned char`) into the first `n` characters of the object pointed to by `s`. A pointer to the result is returned.

Fig. 8.30 | Memory functions of the string-handling library. (Part 2 of 2.)

Memory Functions of the String-Handling Library (Cont.)

- The pointer parameters to these functions are declared `void *` so they can manipulate memory for any data type.
- In Pointer topic, we saw that a pointer to any data type can be assigned directly to a pointer of type `void *`, and a pointer of type `void *` can be assigned directly to a pointer to any data type.
- So, these functions can receive pointers to any data type.
- Because a `void *` pointer cannot be dereferenced, each function receives a size argument that specifies the number of characters (bytes) the function will process.
- For simplicity, the examples in this section manipulate character arrays (blocks of characters).

Memory Functions of the String-Handling Library (Cont.)

- Function `memcpy` copies a specified number of characters from the object pointed to by its second argument into the object pointed to by its first argument.
- The function can receive a pointer to any type of object.
- The result of this function is undefined if the two objects overlap in memory (i.e., if they are parts of the same object)—in such cases, use `memmove`.
- Figure 8.31 uses `memcpy` to copy the string in array `s2` to array `s1`.

```c
/* Fig. 8.31: fig08_31.c
   Using memcpy */
#include <stdio.h>
#include <string.h>

int main( void )
{
   char s1[ 17 ]; /* create char array s1 */
   char s2[] = "Copy this string"; /* initialize char array s2 */

   memcpy( s1, s2, 17 );
   printf( "%s\n%s\"%s\"\n",
      "After s2 is copied into s1 with memcpy,",
      "s1 contains ", s1 );
   return 0; /* indicates successful termination */
} /* end main */
```

```
After s2 is copied into s1 with memcpy,
s1 contains "Copy this string"
```

Fig. 8.31 | Using `memcpy`.

Memory Functions of the String-Handling Library (Cont.)

- Function `memmove`, like `memcpy`, copies a specified number of bytes from the object pointed to by its second argument into the object pointed to by its first argument.
- Copying is performed as if the bytes were copied from the second argument into a temporary character array, then copied from the temporary array into the first argument.
- This allows characters from one part of a string to be copied into another part of the same string.
- Figure 8.32 uses `memmove` to copy the last 10 bytes of array `x` into the first 10 bytes of array `x`.

```c
/* Fig. 8.32: fig08_32.c
   Using memmove */
#include <stdio.h>
#include <string.h>

int main( void )
{
   char x[] = "Home Sweet Home"; /* initialize char array x */

   printf( "%s%s\n", "The string in array x before memmove is: ", x );
   printf( "%s%s\n", "The string in array x after memmove is: ",
      memmove( x, &x[ 5 ], 10 ) );
   return 0; /* indicates successful termination */
} /* end main */
```

```
The string in array x before memmove is: Home Sweet Home
The string in array x after memmove is: Sweet Home Home
```

Fig. 8.32 | Using memmove.

Memory Functions of the String-Handling Library (Cont.)

- Function `memcmp` (Fig. 8.33) compares the specified number of characters of its first argument with the corresponding characters of its second argument.
- The function returns a value greater than 0 if the first argument is greater than the second, returns 0 if the arguments are equal and returns a value less than 0 if the first argument is less than the second.

```c
/* Fig. 8.33: fig08_33.c
   Using memcmp */
#include <stdio.h>
#include <string.h>

int main( void )
{
   char s1[] = "ABCDEFG"; /* initialize char array s1 */
   char s2[] = "ABCDXYZ"; /* initialize char array s2 */

   printf( "%s%s\n%s%s\n\n%s%2d\n%s%2d\n%s%2d\n",
      "s1 = ", s1, "s2 = ", s2,
      "memcmp( s1, s2, 4 ) = ", memcmp( s1, s2, 4 ),
      "memcmp( s1, s2, 7 ) = ", memcmp( s1, s2, 7 ),
      "memcmp( s2, s1, 7 ) = ", memcmp( s2, s1, 7 ) );
   return 0; /* indicate successful termination */
} /* end main */
```

Fig. 8.33 | Using `memcmp`. (Part 1 of 2.)

```
s1 = ABCDEFG
s2 = ABCDXYZ

memcmp( s1, s2, 4 ) =  0
memcmp( s1, s2, 7 ) = -1
memcmp( s2, s1, 7 ) =  1
```

Fig. 8.33 | Using `memcmp`. (Part 2 of 2.)

Memory Functions of the String-Handling Library (Cont.)

- Function `memchr` searches for the first occurrence of a byte, represented as `unsigned char`, in the specified number of bytes of an object.
- If the byte is found, a pointer to the byte in the object is returned; otherwise, a `NULL` pointer is returned.
- Figure 8.34 searches for the character (byte) `'r'` in the string `"This is a string"`.

```c
/* Fig. 8.34: fig08_34.c
   Using memchr */
#include <stdio.h>
#include <string.h>

int main( void )
{
   const char *s = "This is a string"; /* initialize char pointer */

   printf( "%s\'%c\'%s\"%s\"\n",
      "The remainder of s after character ", 'r',
      " is found is ", memchr( s, 'r', 16 ) );
   return 0; /* indicates successful termination */
} /* end main */
```

```
The remainder of s after character 'r' is found is "ring"
```

Fig. 8.34 | Using `memchr`.

Memory Functions of the String-Handling Library (Cont.)

- Function `memset` copies the value of the byte in its second argument into the first *n* bytes of the object pointed to by its first argument, where *n* is specified by the third argument.
- Figure 8.35 uses `memset` to copy `'b'` into the first `7` bytes of `string1`.

```c
/* Fig. 8.35: fig08_35.c
   Using memset */
#include <stdio.h>
#include <string.h>

int main( void )
{
   char string1[ 15 ] = "BBBBBBBBBBBBBB"; /* initialize string1 */

   printf( "string1 = %s\n", string1 );
   printf( "string1 after memset = %s\n", memset( string1, 'b', 7 ) );
   return 0; /* indicates successful termination */
} /* end main */
```

```
string1 = BBBBBBBBBBBBBB
string1 after memset = bbbbbbbBBBBBBB
```

Fig. 8.35 | Using `memset`.

Other Functions of the String-Handling Library

- The two remaining functions of the string-handling library are `strerror` and `strlen`.
- Figure 8.36 summarizes the `strerror` and `strlen` functions.

Function prototype	Function description
`char *strerror( int errornum );`	Maps `errornum` into a full text string in a compiler- and locale-specific manner (e.g. the message may appear in different languages based on its location). A pointer to the string is returned.
`size_t strlen( const char *s );`	Determines the length of string `s`. The number of characters preceding the terminating null character is returned.

Fig. 8.36 | Other functions of the string-handling library.

Other Functions of the String-Handling Library (Cont.)

- Function `strerror` takes an error number and creates an error message string.
- A pointer to the string is returned.
- Figure 8.37 demonstrates `strerror`.

```c
/* Fig. 8.37: fig08_37.c
   Using strerror */
#include <stdio.h>
#include <string.h>

int main( void )
{
   printf( "%s\n", strerror( 2 ) );
   return 0; /* indicates successful termination */
} /* end main */
```

```
No such file or directory
```

Fig. 8.37 | Using `strerror`.

Other Functions of the String-Handling Library (Cont.)

- Function `strlen` takes a string as an argument and returns the number of characters in the string—the terminating null character is not included in the length.
- Figure 8.38 demonstrates function `strlen`.

```c
/* Fig. 8.38: fig08_38.c
   Using strlen */
#include <stdio.h>
#include <string.h>

int main( void )
{
   /* initialize 3 char pointers */
   const char *string1 = "abcdefghijklmnopqrstuvwxyz";
   const char *string2 = "four";
   const char *string3 = "Boston";

   printf("%s\"%s\"%s%lu\n%s\"%s\"%s%lu\n%s\"%s\"%s%lu\n",
      "The length of ", string1, " is ",
      ( unsigned long ) strlen( string1 ),
      "The length of ", string2, " is ",
      ( unsigned long ) strlen( string2 ),
      "The length of ", string3, " is ",
      ( unsigned long ) strlen( string3 ) );
   return 0; /* indicates successful termination */
} /* end main */
```

Fig. 8.38 | Using `strlen`. (Part 1 of 2.)

```
The length of "abcdefghijklmnopqrstuvwxyz" is 26
The length of "four" is 4
The length of "Boston" is 6
```

Fig. 8.38 | Using `strlen`. (Part 2 of 2.)