C Programming for Engineers

UNIVERSITY AT ALBANY
State University of New York

Characters & Strings

ICEN 360– Spring 2017

Prof. Dola Saha

Fundamentals

- The value of a character constant is the integer value of the character in the machine's character set.
  - Example: 'z' represents the integer value of z, and '\n' the integer value of newline (122 and 10 in ASCII, respectively).
- A character is written in single quotes.

- A string is a series of characters treated as a single unit.
- A string may include letters, digits and various special characters such as +, -, *, / and $.
- String literals, or string constants, in C are written in double quotation marks.

String

- An array of characters ending in the null character ('\0').
- Accessed via a *pointer* to the first character in the string.
- Value is the address of its first character.
- In C, a string is a pointer—or, a pointer to the string's first character.
- Are like arrays → an array is also a pointer to its first element.
- A *character array* or a *variable of type char* * can be initialized with a string in a definition.
  - `char color[] = "blue"; // Array`
  - `const char *colorPtr = "blue"; // Pointer to`
    `// somewhere in memory`
  - `char color[] = {'b','l','u','e','\0'}; // Explicit`

Scanning string

- Function `scanf` will read characters until a *space*, *tab*, *newline* or *end-of-file indicator* is encountered.
- The `string2` should be no longer than 19 characters to leave room for the terminating null character.
- If the user types 20 or more characters, your program may crash or create a security vulerability.
- For this reason, we used the conversion specifier `%19s` so that `scanf` reads a maximum of 19 characters and does not write characters into memory beyond the end of the array `string2`.

Character Handling Library <ctype.h>

Prototype	Function description
`int isblank(int c);`	Returns a true value if `c` is a *blank character* that separates words in a line of text and 0 (false) otherwise. [*Note:* This function is not available in Microsoft Visual C++.]
`int isdigit(int c);`	Returns a true value if `c` is a *digit* and 0 (false) otherwise.
`int isalpha(int c);`	Returns a true value if `c` is a *letter* and 0 (false) otherwise.
`int isalnum(int c);`	Returns a true value if `c` is a *digit* or a *letter* and 0 (false) otherwise.
`int isxdigit(int c);`	Returns a true value if `c` is a *hexadecimal digit character* and 0 (false) otherwise. (See Appendix C for a detailed explanation of binary numbers, octal numbers, decimal numbers and hexadecimal numbers.)
`int islower(int c);`	Returns a true value if `c` is a *lowercase letter* and 0 (false) otherwise.
`int isupper(int c);`	Returns a true value if `c` is an *uppercase letter* and 0 (false) otherwise.
`int tolower(int c);`	If `c` is an *uppercase letter*, `tolower` returns `c` as a *lowercase letter*. Otherwise, `tolower` returns the argument unchanged.
`int toupper(int c);`	If `c` is a *lowercase letter*, `toupper` returns `c` as an *uppercase letter*. Otherwise, `toupper` returns the argument unchanged.

Character Handling Library <ctype.h>

Prototype	Function description
`int isspace(int c);`	Returns a true value if `c` is a *whitespace character*—newline (`'\n'`), space (`' '`), form feed (`'\f'`), carriage return (`'\r'`), horizontal tab (`'\t'`) or vertical tab (`'\v'`)—and 0 (false) otherwise.
`int iscntrl(int c);`	Returns a true value if `c` is a *control character*—horizontal tab (`'\t'`), vertical tab (`'\v'`), form feed (`'\f'`), alert (`'\a'`), backspace (`'\b'`), carriage return (`'\r'`), newline (`'\n'`) and others—and 0 (false) otherwise.
`int ispunct(int c);`	Returns a true value if `c` is a *printing character other than a space, a digit, or a letter*—such as `$`, `#`, `(`, `)`, `[`, `]`, `{`, `}`, `;`, `:` or `%`—and returns 0 otherwise.
`int isprint(int c);`	Returns a true value if `c` is a *printing character* (i.e., a character that's visible on the screen) *including a space* and returns 0 (false) otherwise.
`int isgraph(int c);`	Returns a true value if `c` is a *printing character other than a space* and returns 0 (false) otherwise.

Example using ctype.h (1)

```
1  // Fig. 8.2: fig08_02.c
2  // Using functions isdigit, isalpha, isalnum, and isxdigit
3  #include <stdio.h>
4  #include <ctype.h>
5
6  int main(void)
7  {
8     printf("%s\n%s%s\n%s%s\n\n", "According to isdigit: ",
9        isdigit('8')   ? "8 is a " : "8 is not a ", "digit",
10       isdigit('#')   ? "# is a " : "# is not a ", "digit");
11
12    printf("%s\n%s%s\n%s%s\n%s%s\n%s%s\n\n",
13       "According to isalpha:",
14       isalpha('A')   ? "A is a " : "A is not a ", "letter",
15       isalpha('b')   ? "b is a " : "b is not a ", "letter",
16       isalpha('&')   ? "& is a " : "& is not a ", "letter",
17       isalpha('4')   ? "4 is a " : "4 is not a ", "letter");
18
```

Example using ctype.h (2)

```
19      printf("%s\n%s%s\n%s%s\n%s%s\n\n",
20         "According to isalnum:",
21         isalnum('A')   ? "A is a " : "A is not a ",
22         "digit or a letter",
23         isalnum('8')   ? "8 is a " : "8 is not a ",
24         "digit or a letter",
25         isalnum('#')   ? "# is a " : "# is not a ",
26         "digit or a letter");
27
28      printf("%s\n%s%s\n%s%s\n%s%s\n%s%s\n%s%s\n",
29         "According to isxdigit:",
30         isxdigit('F')   ? "F is a " : "F is not a ",
31         "hexadecimal digit",
32         isxdigit('J')   ? "J is a " : "J is not a ",
33         "hexadecimal digit",
34         isxdigit('7')   ? "7 is a " : "7 is not a ",
35         "hexadecimal digit",
36         isxdigit('$')   ? "$ is a " : "$ is not a ",
37         "hexadecimal digit",
38         isxdigit('f')   ? "f is a " : "f is not a ",
39         "hexadecimal digit");
40   }
```

Output of using `ctype.h`

```
According to isdigit:
8 is a digit
# is not a digit

According to isalpha:
A is a letter
b is a letter
& is not a letter
4 is not a letter

According to isalnum:
A is a digit or a letter
8 is a digit or a letter
# is not a digit or a letter

According to isxdigit:
F is a hexadecimal digit
J is not a hexadecimal digit
7 is a hexadecimal digit
$ is not a hexadecimal digit
f is a hexadecimal digit
```

String conversions from <stdlib.h>

Function prototype	Function description
`double strtod(const char *nPtr, char **endPtr);`	Converts the string `nPtr` to `double`.
`long strtol(const char *nPtr, char **endPtr, int base);`	Converts the string `nPtr` to `long`.
`unsigned long strtoul(const char *nPtr, char **endPtr, int base);`	Converts the string `nPtr` to `unsigned long`.

strtod()

- The function uses the char ** argument to modify a char * in the calling function (stringPtr) so that it points to the *location of the first character after the converted portion of the string* or to the entire string if no portion can be converted.
- `d = strtod(string, &stringPtr);`

indicates that d is assigned the double value converted from `string`, and `stringPtr` is assigned the location of the first character after the converted value in string.

strtod() Example

```
1  // Fig. 8.6: fig08_06.c
2  // Using function strtod
3  #include <stdio.h>
4  #include <stdlib.h>
5
6  int main(void)
7  {
8     const char *string = "51.2% are admitted"; // initialize string
9     char *stringPtr; // create char pointer
10
11    double d = strtod(string, &stringPtr);
12
13    printf("The string \"%s\" is converted to the\n", string);
14    printf("double value %.2f and the string \"%s\"\n", d, stringPtr);
15 }
```

```
The string "51.2% are admitted" is converted to the
double value 51.20 and the string "% are admitted"
```

ASCII Character Set

	0	1	2	3	4	5	6	7	8	9
0	nul	soh	stx	etx	eot	enq	ack	bel	bs	ht
1	lf	vt	ff	cr	so	si	dle	dc1	dc2	dc3
2	dc4	nak	syn	etb	can	em	sub	esc	fs	gs
3	rs	us	sp	!	"	#	$	%	&	‘
4	(	)	*	+	,	-	.	/	0	1
5	2	3	4	5	6	7	8	9	:	;
6	<	=	>	?	@	A	B	C	D	E
7	F	G	H	I	J	K	L	M	N	O
8	P	Q	R	S	T	U	V	W	X	Y
9	Z	[	\	]	^	_	’	a	b	c
10	d	e	f	g	h	i	j	k	l	m
11	n	o	p	q	r	s	t	u	v	w
12	x	y	z	{	|	}	~	del		

Fig. B.1 | ASCII Character Set.

The digits at the left of the table are the left digits of the decimal equivalent (0–127) of the character code, and the digits at the top of the table are the right digits of the character code. For example, the character code for “F” is 70, and the character code for “&” is 38.

String Functions from <stdio.h>

Function prototype	Function description
`int getchar(void);`	Inputs the next character from the standard input and returns it as an integer.
`char *fgets(char *s, int n, FILE *stream);`	Inputs characters from the specified stream into the array `s` until a *newline* or *end-of-file* character is encountered, or until `n - 1` bytes are read. In this chapter, we specify the stream as `stdin`—the *standard input stream*, which is typically used to read characters from the keyboard. A *terminating null character* is appended to the array. Returns the string that was read into `s`. If a newline is encountered, it's included in the string stored in `s`.
`int putchar(int c);`	Prints the character stored in `c` and returns it as an integer.
`int puts(const char *s);`	Prints the string `s` followed by a *newline* character. Returns a non-zero integer if successful, or `EOF` if an error occurs.

String Functions from <stdio.h>

Function prototype	Function description
`int sprintf(char *s, const char *format, ...);`	Equivalent to `printf`, except the output is stored in the array `s` instead of printed on the screen. Returns the number of characters written to `s`, or `EOF` if an error occurs. [*Note:* We mention the more secure related functions in the Secure C Programming section of this chapter.]
`int sscanf(char *s, const char *format, ...);`	Equivalent to `scanf`, except the input is read from the array `s` rather than from the keyboard. Returns the number of items successfully read by the function, or `EOF` if an error occurs. [*Note:* We mention the more secure related functions in the Secure C Programming section of this chapter.]

sprintf() Example

```c
// Fig. 8.12: fig08_12.c
// Using function sprintf
#include <stdio.h>
#define SIZE 80

int main(void)
{
   int x; // x value to be input
   double y; // y value to be input

   puts("Enter an integer and a double:");
   scanf("%d%lf", &x, &y);

   char s[SIZE]; // create char array
   sprintf(s, "integer:%6d\ndouble:%7.2f", x, y);

   printf("%s\n%s\n", "The formatted output stored in array s is:", s);

}
```

```
Enter an integer and a double:
298 87.375
The formatted output stored in array s is:
integer:   298
double:  87.38
```

sscanf() Example

```
1  // Fig. 8.13: fig08_13.c
2  // Using function sscanf
3  #include <stdio.h>
4
5  int main(void)
6  {
7     char s[] = "31298 87.375"; // initialize array s
8     int x; // x value to be input
9     double y; // y value to be input
10
11    sscanf(s, "%d%lf", &x, &y);
12    printf("%s\n%s%6d\n%s%8.3f\n",
13       "The values stored in character array s are:",
14       "integer:", x, "double:", y);
15 }
```

```
The values stored in character array s are:
integer: 31298
double:  87.375
```

String Manipulation in <string.h>

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` and returns `s1`.
`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.

Functions: strcpy() and strncpy() (1)

```
1  // Fig. 8.15: fig08_15.c
2  // Using functions strcpy and strncpy
3  #include <stdio.h>
4  #include <string.h>
5  #define SIZE1 25
6  #define SIZE2 15
7
8  int main(void)
9  {
10    char x[] = "Happy Birthday to You"; // initialize char array x
11    char y[SIZE1]; // create char array y
12    char z[SIZE2]; // create char array z
13
14    // copy contents of x into y
15    printf("%s%s\n%s%s\n",
16       "The string in array x is: ", x,
17       "The string in array y is: ", strcpy(y, x));
18
```

Functions: strcpy() and strncpy() (2)

```
19      // copy first 14 characters of x into z. Does not copy null
20      // character
21      strncpy(z, x, SIZE2 - 1);
22
23      z[SIZE2 - 1] = '\0'; // terminate string in z
24      printf("The string in array z is: %s\n", z);
25  }
```

```
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
```

Functions: strcat() and strncat() (1)

```
1  // Fig. 8.16: fig08_16.c
2  // Using functions strcat and strncat
3  #include <stdio.h>
4  #include <string.h>
5
6  int main(void)
7  {
8     char s1[20] = "Happy "; // initialize char array s1
9     char s2[] = "New Year "; // initialize char array s2
10    char s3[40] = ""; // initialize char array s3 to empty
11
12    printf("s1 = %s\ns2 = %s\n", s1, s2);
13
14    // concatenate s2 to s1
15    printf("strcat(s1, s2) = %s\n", strcat(s1, s2) );
16
17    // concatenate first 6 characters of s1 to s3. Place '\0'
18    // after last character
19    printf("strncat(s3, s1, 6) = %s\n", strncat(s3, s1, 6));
20
21    // concatenate s1 to s3
22    printf("strcat(s3, s1) = %s\n", strcat(s3, s1) );
23 }
```

Functions: strcat() and strncat() (2)

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

String Compare

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.

-1 : if the ASCII value of first unmatched character is less than second.
1: if the ASCII value of first unmatched character is greater than second.

String Compare Example – C Code

```
1  // Fig. 8.18: fig08_18.c
2  // Using functions strcmp and strncmp
3  #include <stdio.h>
4  #include <string.h>
5
6  int main(void)
7  {
8     const char *s1 = "Happy New Year"; // initialize char pointer
9     const char *s2 = "Happy New Year"; // initialize char pointer
10    const char *s3 = "Happy Holidays"; // initialize char pointer
11
12    printf("%s%s\n%s%s\n%s%s\n\n%s%2d\n%s%2d\n%s%2d\n\n",
13       "s1 = ", s1, "s2 = ", s2, "s3 = ", s3,
14       "strcmp(s1, s2) = ", strcmp(s1, s2)  ,
15       "strcmp(s1, s3) = ", strcmp(s1, s3)  ,
16       "strcmp(s3, s1) = ", strcmp(s3, s1)  );
17
18    printf("%s%2d\n%s%2d\n%s%2d\n",
19       "strncmp(s1, s3, 6) = ", strncmp(s1, s3, 6)  ,
20       "strncmp(s1, s3, 7) = ", strncmp(s1, s3, 7)  ,
21       "strncmp(s3, s1, 7) = ", strncmp(s3, s1, 7)  );
22 }
```

String Compare Example – Output

```
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
```

More string functions (1)

Function prototypes and descriptions

```
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.

More string functions (2)

Function prototypes and descriptions

```
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.

strchr()

```
1  // Fig. 8.20: fig08_20.c
2  // Using function strchr
3  #include <stdio.h>
4  #include <string.h>
5
6  int main(void)
7  {
8     const char *string = "This is a test"; // initialize char pointer
9     char character1 = 'a'; // initialize character1
10    char character2 = 'z'; // initialize character2
11
12    // if character1 was found in string
13    if (strchr(string, character1) != NULL) { // can remove "!= NULL"
14       printf("\'%c\' was found in \"%s\".\n",
15          character1, string);
16    }
17    else { // if character1 was not found
18       printf("\'%c\' was not found in \"%s\".\n",
19          character1, string);
20    }
21
```

strchr()

```
22      // if character2 was found in string
23      if (strchr(string, character2) != NULL) { // can remove "!= NULL"
24         printf("\'%c\' was found in \"%s\".\n",
25            character2, string);
26      }
27      else { // if character2 was not found
28         printf("\'%c\' was not found in \"%s\".\n",
29            character2, string);
30      }
31   }
```

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

strcspn()

```
1  // Fig. 8.21: fig08_21.c
2  // Using function strcspn
3  #include <stdio.h>
4  #include <string.h>
5
6  int main(void)
7  {
8     // initialize two char pointers
9     const char *string1 = "The value is 3.14159";
10    const char *string2 = "1234567890";
11
12    printf("%s%s\n%s%s\n\n%s\n%s%u\n",
13       "string1 = ", string1, "string2 = ", string2,
14       "The length of the initial segment of string1",
15       "containing no characters from string2 = ",
16       strcspn(string1, string2) );
17 }
```

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

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

strpbrk()

```
1  // Fig. 8.22: fig08_22.c
2  // Using function strpbrk
3  #include <stdio.h>
4  #include <string.h>
5
6  int main(void)
7  {
8     const char *string1 = "This is a test"; // initialize char pointer
9     const char *string2 = "beware"; // initialize char pointer
10
11    printf("%s\"%s\"\n'%c'%s\n\"%s\"\n",
12       "Of the characters in ", string2,
13       *strpbrk(string1, string2) ,
14       " appears earliest in ", string1);
15 }
```

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

strtok()

```
1  // Fig. 8.26: fig08_26.c
2  // Using function strtok
3  #include <stdio.h>
4  #include <string.h>
5
6  int main(void)
7  {
8     // initialize array string
9     char string[] = "This is a sentence with 7 tokens";
10
11    printf("%s\n%s\n\n%s\n",
12       "The string to be tokenized is:", string,
13       "The tokens are:");
14
15    char *tokenPtr = strtok(string, " "); // begin tokenizing sentence
16
17    // continue tokenizing sentence until tokenPtr becomes NULL
18    while (tokenPtr != NULL) {
19       printf("%s\n", tokenPtr);
20       tokenPtr = strtok(NULL, " "); // get next token
21    }
22 }
```

strtok() Output

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

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

More 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 spoken languages based on the computer's locale). A pointer to the string is returned. Error numbers are defined in `errno.h`.
`size_t strlen(const char *s);`	Determines the length of string `s`. The number of characters preceding the terminating null character is returned.

strlen()

```
1  // Fig. 8.35: fig08_35.c
2  // Using function strlen
3  #include <stdio.h>
4  #include <string.h>
5
6  int main(void)
7  {
8     // initialize 3 char pointers
9     const char *string1 = "abcdefghijklmnopqrstuvwxyz";
10    const char *string2 = "four";
11    const char *string3 = "Boston";
12
13    printf("%s\"%s\"%s%u\n%s\"%s\"%s%u\n%s\"%s\"%s%u\n",
14       "The length of ", string1, " is ", strlen(string1),
15       "The length of ", string2, " is ", strlen(string2),
16       "The length of ", string3, " is ", strlen(string3));
17 }
```

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

Memory Functions

Function prototype	Function description
`void *memcpy(void *s1, const void *s2, size_t n);`	*Copies n bytes* 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 bytes* from the object pointed to by `s2` into the object pointed to by `s1`. The copy is performed as if the bytes 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 bytes* 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`.
`void *memchr(const void *s, int c, size_t n);`	*Locates the first occurrence* of `c` (converted to `unsigned char`) in the first `n` bytes 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 bytes* of the object pointed to by `s`. A pointer to the result is returned.

memcpy()

```
1  // Fig. 8.28: fig08_28.c
2  // Using function memcpy
3  #include <stdio.h>
4  #include <string.h>
5
6  int main(void)
7  {
8     char s1[17]; // create char array s1
9     char s2[] = "Copy this string"; // initialize char array s2
10
11    memcpy(s1, s2, 17);
12    printf("%s\n%s\"%s\"\n",
13       "After s2 is copied into s1 with memcpy,",
14       "s1 contains ", s1);
15 }
```

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

memmove()

```
1  // Fig. 8.29: fig08_29.c
2  // Using function memmove
3  #include <stdio.h>
4  #include <string.h>
5
6  int main(void)
7  {
8     char x[] = "Home Sweet Home"; // initialize char array x
9
10    printf("%s%s\n", "The string in array x before memmove is: ", x);
11    printf("%s%s\n", "The string in array x after memmove is: ",
12       (char *) memmove(x, &x[5], 10));
13 }
```

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

memcmp()

```
1  // Fig. 8.30: fig08_30.c
2  // Using function memcmp
3  #include <stdio.h>
4  #include <string.h>
5
6  int main(void)
7  {
8     char s1[] = "ABCDEFG"; // initialize char array s1
9     char s2[] = "ABCDXYZ"; // initialize char array s2
10
11    printf("%s%s\n%s%s\n\n%s%2d\n%s%2d\n%s%2d\n",
12       "s1 = ", s1, "s2 = ", s2,
13       "memcmp(s1, s2, 4) = ", memcmp(s1, s2, 4),
14       "memcmp(s1, s2, 7) = ", memcmp(s1, s2, 7),
15       "memcmp(s2, s1, 7) = ", memcmp(s2, s1, 7));
16 }
```

```
s1 = ABCDEFG
s2 = ABCDXYZ

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

memchr()

```
1  // Fig. 8.31: fig08_31.c
2  // Using function memchr
3  #include <stdio.h>
4  #include <string.h>
5
6  int main(void)
7  {
8     const char *s = "This is a string"; // initialize char pointer
9
10    printf("%s\'%c\'%s\"%s\"\n",
11       "The remainder of s after character ", 'r',
12       " is found is ", (char *) memchr(s, 'r', 16));
13 }
```

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

memset()

```
1  // Fig. 8.32: fig08_32.c
2  // Using function memset
3  #include <stdio.h>
4  #include <string.h>
5
6  int main(void)
7  {
8     char string1[15] = "BBBBBBBBBBBBBB"; // initialize string1
9
10    printf("string1 = %s\n", string1);
11    printf("string1 after memset = %s\n",
12       (char *) memset(string1, 'b', 7));
13 }
```

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