

# Computer Programming I

Ming-Feng Tsai (Victor Tsai)

Dept. of Computer Science  
National Chengchi University

# C Characters and Strings

- 8.1** Introduction
- 8.2** Fundamentals of Strings and Characters
- 8.3** Character-Handling Library
- 8.4** String-Conversion Functions
- 8.5** Standard Input/Output Library Functions
- 8.6** String-Manipulation Functions of the String-Handling Library
- 8.7** Comparison Functions of the String-Handling Library
- 8.8** Search Functions of the String-Handling Library
- 8.9** Memory Functions of the String-Handling Library
- 8.10** Other Functions of the String-Handling Library

# 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**.

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

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

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

- Functions **strncpy** and **strncat** specify a parameter of type **size\_t**, which is a type defined by the C standard as the integral type of the value returned by operator **sizeof**.

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

- Example: [fig08\\_18.c](#)

```
7 char x[] = "Happy Birthday to You"; /* initialize char array x */
8 char y[ 25 ]; /* create char array y */
9 char z[ 15 ]; /* create char array z */
10
11 /* copy contents of x into y */
12 printf( "%s%s\n%s%s\n",
13         "The string in array x is: ", x,
14         "The string in array y is: ", strcpy( y, x ) );
15
16 /* copy first 14 characters of x into z. Does not copy null.
17    character */
18 strncpy( z, x, 14 );
19
20 z[ 14 ] = '\0'; /* terminate string in z */
21 printf( "The string in array z is: %s\n", z );
```

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

- Example: [fig08\\_18.c](#)

```
7 char x[] = "Happy Birthday to You"; /* initialize char array x */
8 char y[ 25 ]; /* create char array y */
9 char z[ 15 ]; /* create char array z */
10
11 /* copy contents of x into y */
12 printf( "%s%s\n%s%s\n",
13         "The string in array x is: ", x,
14         "The string in array y is: ", strcpy( y, x ) );
15
16 /* copy first 14 characters of x into z. Does not copy null.
17    character */
18 strncpy( z, x, 14 );
19
20 z[ 14 ] = '\0'; /* terminate string in z */
21 printf( "The string in array z is: %s\n", z );
```



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

- Example: [fig08\\_18.c](#)

```
7 char x[] = "Happy Birthday to You"; /* initialize char array x */
8 char y[ 25 ]; /* create char array y */
9 char z[ 15 ]; /* create char array z */
10
11 /* copy contents of x into y */
12 printf( "%s%s\n%s%s\n",
13         "The string in array x is: ", x,
14         "The string in array y is: ", strcpy( y, x ) );
15
16 /* copy first 14 characters of x into z. Does not copy null.
17    character */
18 strncpy( z, x, 14 );
19
20 z[ 14 ] = '\0'; /* terminate string in z */
21 printf( "The string in array z is: %s\n", z );
```

**strcpy()** copies the entire string in array **x** into array **y**

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

- Example: [fig08\\_18.c](#)

```
7 char x[] = "Happy Birthday to You"; /* initialize char array x */
8 char y[ 25 ]; /* create char array y */
9 char z[ 15 ]; /* create char array z */
10
11 /* copy contents of x into y */
12 printf( "%s%s\n%s%s\n",
13         "The string in array x is: ", x,
14         "The string in array y is: ", strcpy( y, x ) );
15
16 /* copy first 14 characters of x into z. Does not copy null.
17    character */
18 strncpy( z, x, 14 );
19
20 z[ 14 ] = '\0'; /* terminate string in z */
21 printf( "The string in array z is: %s\n", z );
```

**strcpy()** copies the entire string in array **x** into array **y**

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

- Example: [fig08\\_18.c](#)

```
7 char x[] = "Happy Birthday to You"; /* initialize char array x */
8 char y[ 25 ]; /* create char array y */
9 char z[ 15 ]; /* create char array z */
10
11 /* copy contents of x into y */
12 printf( "%s%s\n%s%s\n",
13         "The string in array x is: ", x,
14         "The string in array y is: ", strcpy( y, x ) );
15
16 /* copy first 14 characters of x into z. Does not copy null.
17    character */
18 strncpy( z, x, 14 );
19
20 z[ 14 ] = '\0'; /* terminate string in z */
21 printf( "The string in array z is: %s\n", z );
```

**strcpy()** copies the entire string in array **x** into array **y**

**strncpy()** copies the first 14 characters of array **x** into array **z**

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

- Example: [fig08\\_18.c](#)

```
7 char x[] = "Happy Birthday to You"; /* initialize char array x */
8 char y[ 25 ]; /* create char array y */
9 char z[ 15 ]; /* create char array z */
10
11 /* copy contents of x into y */
12 printf( "%s%s\n%s%s\n",
13         "The string in array x is: ", x,
14         "The string in array y is: ", strcpy( y, x ) );
15
16 /* copy first 14 characters of x into z. Does not copy null.
17    character */
18 strncpy( z, x, 14 );
19
20 z[ 14 ] = '\0'; /* terminate string in z */
21 printf( "The string in array z is: %s\n", z );
```

**strcpy()** copies the entire string in array **x** into array **y**

**strncpy()** copies the first 14 characters of array **x** into array **z**

```
|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
```

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

- Example: [fig08\\_19.c](#)

```
7 char s1[ 20 ] = "Happy "; /* initialize char array s1 */
8 char s2[] = "New Year "; /* initialize char array s2 */
9 char s3[ 40 ] = ""; /* initialize char array s3 to empty */
10
11 printf( "s1 = %s\ns2 = %s\n", s1, s2 );
12
13 /* concatenate s2 to s1 */
14 printf( "strcat( s1, s2 ) = %s\n", strcat( s1, s2 ) );
15
16 /* concatenate first 6 characters of s1 to s3. Place '\0'
17    after last character */
18 printf( "strncat( s3, s1, 6 ) = %s\n", strncat( s3, s1, 6 ) );
19
20 /* concatenate s1 to s3 */
21 printf( "strcat( s3, s1 ) = %s\n", strcat( s3, s1 ) );
22 return 0; /* indicates successful termination */
```



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

- Example: [fig08\\_19.c](#)

```
7 char s1[ 20 ] = "Happy "; /* initialize char array s1 */
8 char s2[] = "New Year "; /* initialize char array s2 */
9 char s3[ 40 ] = ""; /* initialize char array s3 to empty */
10
11 printf( "s1 = %s\ns2 = %s\n", s1, s2 );
12
13 /* concatenate s2 to s1 */
14 printf( "strcat( s1, s2 ) = %s\n", strcat( s1, s2 ) );
15
16 /* concatenate first 6 characters of s1 to s3. Place '\0'
17    after last character */
18 printf( "strncat( s3, s1, 6 ) = %s\n", strncat( s3, s1, 6 ) );
19
20 /* concatenate s1 to s3 */
21 printf( "strcat( s3, s1 ) = %s\n", strcat( s3, s1 ) );
22 return 0; /* indicates successful termination */
```

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

- Example: [fig08\\_19.c](#)

```
7 char s1[ 20 ] = "Happy "; /* initialize char array s1 */
8 char s2[] = "New Year "; /* initialize char array s2 */
9 char s3[ 40 ] = ""; /* initialize char array s3 to empty */
10
11 printf( "s1 = %s\ns2 = %s\n", s1, s2 );
12
13 /* concatenate s2 to s1 */
14 printf( "strcat( s1, s2 ) = %s\n", strcat( s1, s2 ) );
15
16 /* concatenate first 6 characters of s1 to s3. Place '\0'
17    after last character */
18 printf( "strncat( s3, s1, 6 ) = %s\n", strncat( s3, s1, 6 ) );
19
20 /* concatenate s1 to s3 */
21 printf( "strcat( s3, s1 ) = %s\n", strcat( s3, s1 ) );
22 return 0; /* indicates successful termination */
```

**strcat()** appends the string **s2** into **s1**

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

- Example: [fig08\\_19.c](#)

```
7 char s1[ 20 ] = "Happy "; /* initialize char array s1 */
8 char s2[] = "New Year "; /* initialize char array s2 */
9 char s3[ 40 ] = ""; /* initialize char array s3 to empty */
10
11 printf( "s1 = %s\ns2 = %s\n", s1, s2 );
12
13 /* concatenate s2 to s1 */
14 printf( "strcat( s1, s2 ) = %s\n", strcat( s1, s2 ) );
15
16 /* concatenate first 6 characters of s1 to s3. Place '\0'
17    after last character */
18 printf( "strncat( s3, s1, 6 ) = %s\n", strncat( s3, s1, 6 ) );
19
20 /* concatenate s1 to s3 */
21 printf( "strcat( s3, s1 ) = %s\n", strcat( s3, s1 ) );
22 return 0; /* indicates successful termination */
```

**strcat()** appends the string **s2** into **s1**



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

- Example: [fig08\\_19.c](#)

```
7 char s1[ 20 ] = "Happy "; /* initialize char array s1 */
8 char s2[] = "New Year "; /* initialize char array s2 */
9 char s3[ 40 ] = ""; /* initialize char array s3 to empty */
10
11 printf( "s1 = %s\ns2 = %s\n", s1, s2 );
12
13 /* concatenate s2 to s1 */
14 printf( "strcat( s1, s2 ) = %s\n", strcat( s1, s2 ) );
15
16 /* concatenate first 6 characters of s1 to s3. Place '\0'
17    after last character */
18 printf( "strncat( s3, s1, 6 ) = %s\n", strncat( s3, s1, 6 ) );
19
20 /* concatenate s1 to s3 */
21 printf( "strcat( s3, s1 ) = %s\n", strcat( s3, s1 ) );
22 return 0; /* indicates successful termination */
```

**strcat()** appends the string **s2** into **s1**

**strncat()** appends the first 6 characters of **s1** into **s3**

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

- Example: [fig08\\_19.c](#)

```
7 char s1[ 20 ] = "Happy "; /* initialize char array s1 */
8 char s2[] = "New Year "; /* initialize char array s2 */
9 char s3[ 40 ] = ""; /* initialize char array s3 to empty */
10
11 printf( "s1 = %s\ns2 = %s\n", s1, s2 );
12
13 /* concatenate s2 to s1 */
14 printf( "strcat( s1, s2 ) = %s\n", strcat( s1, s2 ) );
15
16 /* concatenate first 6 characters of s1 to s3. Place '\0'
17    after last character */
18 printf( "strncat( s3, s1, 6 ) = %s\n", strncat( s3, s1, 6 ) );
19
20 /* concatenate s1 to s3 */
21 printf( "strcat( s3, s1 ) = %s\n", strcat( s3, s1 ) );
22 return 0; /* indicates successful termination */
```

**strcat()** appends the string **s2** into **s1**

**strncat()** appends the first 6 characters of **s1** into **s3**

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

# Comparison Functions of the String-Handling Library

- This section presents the string-handling library's **string-comparison functions: `strcmp` and `strncmp`**.

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

Function prototype	Function description
<code>int strcmp( const char *s1, const char *s2 );</code>	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.
<code>int strncmp( const char *s1, const char *s2, size_t n );</code>	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.

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

- Example: [fig08\\_21.c](#)

```
7  const char *s1 = "Happy New Year"; /* initialize char pointer */
8  const char *s2 = "Happy New Year"; /* initialize char pointer */
9  const char *s3 = "Happy Holidays"; /* initialize char pointer */
10
11  printf("%s%s\n%s%s\n%s%s\n\n%s%2d\n%s%2d\n%s%2d\n\n",
12         "s1 = ", s1, "s2 = ", s2, "s3 = ", s3,
13         "strcmp(s1, s2) = ", strcmp( s1, s2 ),
14         "strcmp(s1, s3) = ", strcmp( s1, s3 ),
15         "strcmp(s3, s1) = ", strcmp( s3, s1 ) );
16
17  printf("%s%2d\n%s%2d\n%s%2d\n",
18         "strncmp(s1, s3, 6) = ", strncmp( s1, s3, 6 ),
19         "strncmp(s1, s3, 7) = ", strncmp( s1, s3, 7 ),
20         "strncmp(s3, s1, 7) = ", strncmp( s3, s1, 7 ) );
```

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

- Example: [fig08\\_21.c](#)

```
7  const char *s1 = "Happy New Year"; /* initialize char pointer */
8  const char *s2 = "Happy New Year"; /* initialize char pointer */
9  const char *s3 = "Happy Holidays"; /* initialize char pointer */
10
11  printf("%s%s\n%s%s\n%s%s\n\n%s%2d\n%s%2d\n%s%2d\n\n",
12        "s1 = ", s1, "s2 = ", s2, "s3 = ", s3,
13        "strcmp(s1, s2) = ", strcmp( s1, s2 ),
14        "strcmp(s1, s3) = ", strcmp( s1, s3 ),
15        "strcmp(s3, s1) = ", strcmp( s3, s1 ) );
16
17  printf("%s%2d\n%s%2d\n%s%2d\n",
18        "strncmp(s1, s3, 6) = ", strncmp( s1, s3, 6 ),
19        "strncmp(s1, s3, 7) = ", strncmp( s1, s3, 7 ),
20        "strncmp(s3, s1, 7) = ", strncmp( s3, s1, 7 ) );
```



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

- Example: [fig08\\_21.c](#)

```
7  const char *s1 = "Happy New Year"; /* initialize char pointer */
8  const char *s2 = "Happy New Year"; /* initialize char pointer */
9  const char *s3 = "Happy Holidays"; /* initialize char pointer */
10
11  printf("%s%s\n%s%s\n%s%s\n\n%s%2d\n%s%2d\n%s%2d\n\n",
12        "s1 = ", s1, "s2 = ", s2, "s3 = ", s3,
13        "strcmp(s1, s2) = ", strcmp( s1, s2 ),
14        "strcmp(s1, s3) = ", strcmp( s1, s3 ),
15        "strcmp(s3, s1) = ", strcmp( s3, s1 ) );
16
17  printf("%s%2d\n%s%2d\n%s%2d\n",
18        "strncmp(s1, s3, 6) = ", strncmp( s1, s3, 6 ),
19        "strncmp(s1, s3, 7) = ", strncmp( s1, s3, 7 ),
20        "strncmp(s3, s1, 7) = ", strncmp( s3, s1, 7 ) );
```

**strcmp:** compare the first string with the second one; 0: equal; n: first one is less than second one; p: first one is greater than second one

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

- Example: [fig08\\_21.c](#)

```
7  const char *s1 = "Happy New Year"; /* initialize char pointer */
8  const char *s2 = "Happy New Year"; /* initialize char pointer */
9  const char *s3 = "Happy Holidays"; /* initialize char pointer */
10
11  printf("%s%s\n%s%s\n%s%s\n\n%s%2d\n%s%2d\n%s%2d\n\n",
12        "s1 = ", s1, "s2 = ", s2, "s3 = ", s3,
13        "strcmp(s1, s2) = ", strcmp( s1, s2 ),
14        "strcmp(s1, s3) = ", strcmp( s1, s3 ),
15        "strcmp(s3, s1) = ", strcmp( s3, s1 ) );
16
17  printf("%s%2d\n%s%2d\n%s%2d\n",
18        "strncmp(s1, s3, 6) = ", strncmp( s1, s3, 6 ),
19        "strncmp(s1, s3, 7) = ", strncmp( s1, s3, 7 ),
20        "strncmp(s3, s1, 7) = ", strncmp( s3, s1, 7 ) );
```

**strcmp:** compare the first string with the second one; 0: equal; n: first one is less than second one; p: first one is greater than second one



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

- Example: [fig08\\_21.c](#)

```
7  const char *s1 = "Happy New Year"; /* initialize char pointer */
8  const char *s2 = "Happy New Year"; /* initialize char pointer */
9  const char *s3 = "Happy Holidays"; /* initialize char pointer */
10
11  printf("%s%s\n%s%s\n%s%s\n\n%s%2d\n%s%2d\n%s%2d\n\n",
12        "s1 = ", s1, "s2 = ", s2, "s3 = ", s3,
13        "strcmp(s1, s2) = ", strcmp( s1, s2 ),
14        "strcmp(s1, s3) = ", strcmp( s1, s3 ),
15        "strcmp(s3, s1) = ", strcmp( s3, s1 ) );
16
17  printf("%s%2d\n%s%2d\n%s%2d\n",
18        "strncmp(s1, s3, 6) = ", strncmp( s1, s3, 6 ),
19        "strncmp(s1, s3, 7) = ", strncmp( s1, s3, 7 ),
20        "strncmp(s3, s1, 7) = ", strncmp( s3, s1, 7 ) );
```

**strcmp:** compare the first string with the second one; 0: equal; n: first one is less than second one; p: first one is greater than second one

**strncmp():** equivalent to **strcmp**, except that only up to a specific number of characters

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

- Example: [fig08\\_21.c](#)

```
7  const char *s1 = "Happy New Year"; /* initialize char pointer */
8  const char *s2 = "Happy New Year"; /* initialize char pointer */
9  const char *s3 = "Happy Holidays"; /* initialize char pointer */
10
11  printf("%s%s\n%s%s\n%s%s\n\n%s%2d\n%s%2d\n%s%2d\n\n",
12        "s1 = ", s1, "s2 = ", s2, "s3 = ", s3,
13        "strcmp(s1, s2) = ", strcmp( s1, s2 ),
14        "strcmp(s1, s3) = ", strcmp( s1, s3 ),
15        "strcmp(s3, s1) = ", strcmp( s3, s1 ) );
16
17  printf("%s%2d\n%s%2d\n%s%2d\n",
18        "strncmp(s1, s3, 6) = ", strncmp( s1, s3, 6 ),
19        "strncmp(s1, s3, 7) = ", strncmp( s1, s3, 7 ),
20        "strncmp(s3, s1, 7) = ", strncmp( s3, s1, 7 ) );
```

**strcmp:** compare the first string with the second one; 0: equal; n: first one is less than second one; p: first one is greater than second one

**strncmp():** equivalent to **strcmp**, except that only up to a specific number of characters

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

strcmp(s1, s2) = 0
strcmp(s1, s3) = 6
strcmp(s3, s1) = -6

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

# 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,”
- EBCDIC stands for “Extended Binary Coded Decimal Interchange Code.”

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

- ASCII Table

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	<b>NUL</b> (null)	32	20	040	&#32;	Space	64	40	100	&#64;	@	96	60	140	&#96;	`
1	1	001	<b>SOH</b> (start of heading)	33	21	041	&#33;	!	65	41	101	&#65;	A	97	61	141	&#97;	a
2	2	002	<b>STX</b> (start of text)	34	22	042	&#34;	"	66	42	102	&#66;	B	98	62	142	&#98;	b
3	3	003	<b>ETX</b> (end of text)	35	23	043	&#35;	#	67	43	103	&#67;	C	99	63	143	&#99;	c
4	4	004	<b>EOT</b> (end of transmission)	36	24	044	&#36;	\$	68	44	104	&#68;	D	100	64	144	&#100;	d
5	5	005	<b>ENQ</b> (enquiry)	37	25	045	&#37;	%	69	45	105	&#69;	E	101	65	145	&#101;	e
6	6	006	<b>ACK</b> (acknowledge)	38	26	046	&#38;	&	70	46	106	&#70;	F	102	66	146	&#102;	f
7	7	007	<b>BEL</b> (bell)	39	27	047	&#39;	'	71	47	107	&#71;	G	103	67	147	&#103;	g
8	8	010	<b>BS</b> (backspace)	40	28	050	&#40;	(	72	48	110	&#72;	H	104	68	150	&#104;	h
9	9	011	<b>TAB</b> (horizontal tab)	41	29	051	&#41;	)	73	49	111	&#73;	I	105	69	151	&#105;	i
10	A	012	<b>LF</b> (NL line feed, new line)	42	2A	052	&#42;	*	74	4A	112	&#74;	J	106	6A	152	&#106;	j
11	B	013	<b>VT</b> (vertical tab)	43	2B	053	&#43;	+	75	4B	113	&#75;	K	107	6B	153	&#107;	k
12	C	014	<b>FF</b> (NP form feed, new page)	44	2C	054	&#44;	,	76	4C	114	&#76;	L	108	6C	154	&#108;	l
13	D	015	<b>CR</b> (carriage return)	45	2D	055	&#45;	-	77	4D	115	&#77;	M	109	6D	155	&#109;	m
14	E	016	<b>SO</b> (shift out)	46	2E	056	&#46;	.	78	4E	116	&#78;	N	110	6E	156	&#110;	n
15	F	017	<b>SI</b> (shift in)	47	2F	057	&#47;	/	79	4F	117	&#79;	O	111	6F	157	&#111;	o
16	10	020	<b>DLE</b> (data link escape)	48	30	060	&#48;	0	80	50	120	&#80;	P	112	70	160	&#112;	p
17	11	021	<b>DC1</b> (device control 1)	49	31	061	&#49;	1	81	51	121	&#81;	Q	113	71	161	&#113;	q
18	12	022	<b>DC2</b> (device control 2)	50	32	062	&#50;	2	82	52	122	&#82;	R	114	72	162	&#114;	r
19	13	023	<b>DC3</b> (device control 3)	51	33	063	&#51;	3	83	53	123	&#83;	S	115	73	163	&#115;	s
20	14	024	<b>DC4</b> (device control 4)	52	34	064	&#52;	4	84	54	124	&#84;	T	116	74	164	&#116;	t
21	15	025	<b>NAK</b> (negative acknowledge)	53	35	065	&#53;	5	85	55	125	&#85;	U	117	75	165	&#117;	u
22	16	026	<b>SYN</b> (synchronous idle)	54	36	066	&#54;	6	86	56	126	&#86;	V	118	76	166	&#118;	v
23	17	027	<b>ETB</b> (end of trans. block)	55	37	067	&#55;	7	87	57	127	&#87;	W	119	77	167	&#119;	w
24	18	030	<b>CAN</b> (cancel)	56	38	070	&#56;	8	88	58	130	&#88;	X	120	78	170	&#120;	x
25	19	031	<b>EM</b> (end of medium)	57	39	071	&#57;	9	89	59	131	&#89;	Y	121	79	171	&#121;	y
26	1A	032	<b>SUB</b> (substitute)	58	3A	072	&#58;	:	90	5A	132	&#90;	Z	122	7A	172	&#122;	z
27	1B	033	<b>ESC</b> (escape)	59	3B	073	&#59;	;	91	5B	133	&#91;	[	123	7B	173	&#123;	{
28	1C	034	<b>FS</b> (file separator)	60	3C	074	&#60;	<	92	5C	134	&#92;	\	124	7C	174	&#124;	
29	1D	035	<b>GS</b> (group separator)	61	3D	075	&#61;	=	93	5D	135	&#93;	]	125	7D	175	&#125;	}
30	1E	036	<b>RS</b> (record separator)	62	3E	076	&#62;	>	94	5E	136	&#94;	^	126	7E	176	&#126;	~
31	1F	037	<b>US</b> (unit separator)	63	3F	077	&#63;	?	95	5F	137	&#95;	_	127	7F	177	&#127;	DEL

Source: [www.LookupTables.com](http://www.LookupTables.com)

# 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.



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

## 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.

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

## 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.

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

- Example: [fig08\\_23.c](#)

```
7  const char *string = "This is a test"; /* initialize char pointer */
8  char character1 = 'a'; /* initialize character1 */
9  char character2 = 'z'; /* initialize character2 */
10
11  /* if character1 was found in string */
12  if ( strchr( string, character1 ) != NULL ) {
13      printf( "'%c' was found in \"%s\".\n",
14              character1, string );
15  } else { /* if character1 was not found */
16      printf( "'%c' was not found in \"%s\".\n",
17              character1, string );
18  } /* end else */
19
20  /* if character2 was found in string */
21  if ( strchr( string, character2 ) != NULL ) {
22      printf( "'%c' was found in \"%s\".\n",
23              character2, string );
24  } else { /* if character2 was not found */
25      printf( "'%c' was not found in \"%s\".\n",
26              character2, string );
27  } /* end else */
```



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

- Example: [fig08\\_23.c](#)

```
7  const char *string = "This is a test"; /* initialize char pointer */
8  char character1 = 'a'; /* initialize character1 */
9  char character2 = 'z'; /* initialize character2 */
10
11  /* if character1 was found in string */
12  if ( strchr( string, character1 ) != NULL ) {
13      printf( "'%c' was found in \"%s\".\n",
14              character1, string );
15  } else { /* if character1 was not found */
16      printf( "'%c' was not found in \"%s\".\n",
17              character1, string );
18  } /* end else */
19
20  /* if character2 was found in string */
21  if ( strchr( string, character2 ) != NULL ) {
22      printf( "'%c' was found in \"%s\".\n",
23              character2, string );
24  } else { /* if character2 was not found */
25      printf( "'%c' was not found in \"%s\".\n",
26              character2, string );
27  } /* end else */
```

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

- Example: [fig08\\_23.c](#)

```
7  const char *string = "This is a test"; /* initialize char pointer */
8  char character1 = 'a'; /* initialize character1 */
9  char character2 = 'z'; /* initialize character2 */
10
11  /* if character1 was found in string */
12  if ( strchr( string, character1 ) != NULL ) {
13      printf( "'%c' was found in \"%s\".\n",
14              character1, string );
15  } else { /* if character1 was not found */
16      printf( "'%c' was not found in \"%s\".\n",
17              character1, string );
18  } /* end else */
19
20  /* if character2 was found in string */
21  if ( strchr( string, character2 ) != NULL ) {
22      printf( "'%c' was found in \"%s\".\n",
23              character2, string );
24  } else { /* if character2 was not found */
25      printf( "'%c' was not found in \"%s\".\n",
26              character2, string );
27  } /* end else */
```

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

- Example: [fig08\\_23.c](#)

```
7  const char *string = "This is a test"; /* initialize char pointer */
8  char character1 = 'a'; /* initialize character1 */
9  char character2 = 'z'; /* initialize character2 */
10
11  /* if character1 was found in string */
12  if ( strchr( string, character1 ) != NULL ) {
13      printf( "'%c' was found in \"%s\".\n",
14              character1, string );
15  } else { /* if character1 was not found */
16      printf( "'%c' was not found in \"%s\".\n",
17              character1, string );
18  } /* end else */
19
20  /* if character2 was found in string */
21  if ( strchr( string, character2 ) != NULL ) {
22      printf( "'%c' was found in \"%s\".\n",
23              character2, string );
24  } else { /* if character2 was not found */
25      printf( "'%c' was not found in \"%s\".\n",
26              character2, string );
27  } /* end else */
```

if the character is found, returns a pointer to the character; otherwise, return **NULL**

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

- Example: [fig08\\_23.c](#)

```
7  const char *string = "This is a test"; /* initialize char pointer */
8  char character1 = 'a'; /* initialize character1 */
9  char character2 = 'z'; /* initialize character2 */
10
11  /* if character1 was found in string */
12  if ( strchr( string, character1 ) != NULL ) {
13      printf( "'%c' was found in \"%s\".\n",
14              character1, string );
15  } else { /* if character1 was not found */
16      printf( "'%c' was not found in \"%s\".\n",
17              character1, string );
18  } /* end else */
19
20  /* if character2 was found in string */
21  if ( strchr( string, character2 ) != NULL ) {
22      printf( "'%c' was found in \"%s\".\n",
23              character2, string );
24  } else { /* if character2 was not found */
25      printf( "'%c' was not found in \"%s\".\n",
26              character2, string );
27  } /* end else */
```

if the character is found, returns a pointer to the character; otherwise, return **NULL**

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

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

- Example: [fig08\\_24.c](#)

```
8  const char *string1 = "The value is 3.14159";
9  const char *string2 = "1234567890";
10
11  printf( "%s%s\n%s%s\n\n%s\n%s%u\n",
12          "string1 = ", string1, "string2 = ", string2,
13          "The length of the initial segment of string1",
14          "containing no characters from string2 = ",
15          strcspn( string1, string2 ) );
```



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

- Example: [fig08\\_24.c](#)

```
8  const char *string1 = "The value is 3.14159";
9  const char *string2 = "1234567890";
10
11  printf( "%s%s\n%s%s\n\n%s\n%s%u\n",
12          "string1 = ", string1, "string2 = ", string2,
13          "The length of the initial segment of string1",
14          "containing no characters from string2 = ",
15          strcspn( string1, string2 ) );
```

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

- Example: [fig08\\_24.c](#)

```
8  const char *string1 = "The value is 3.14159";
9  const char *string2 = "1234567890";
10
11  printf( "%s%s\n%s%s\n\n%s\n%s%u\n",
12          "string1 = ", string1, "string2 = ", string2,
13          "The length of the initial segment of string1",
14          "containing no characters from string2 = ",
15          strcspn( string1, string2 ) );
```

return the first part  
of the string that  
does not contain any  
characters in string2

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

- Example: [fig08\\_24.c](#)

```
8  const char *string1 = "The value is 3.14159";
9  const char *string2 = "1234567890";
10
11  printf( "%s%s\n%s%s\n\n%s\n%s%u\n",
12          "string1 = ", string1, "string2 = ", string2,
13          "The length of the initial segment of string1",
14          "containing no characters from string2 = ",
15          strcspn( string1, string2 ) );
```

return the first part  
of the string that  
does not contain any  
characters in string2

```
|string1 = The value is 3.14159
|string2 = 1234567890
|
|The length of the initial segment of string1
|containing no characters from string2 = 13
```



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

- Example: [fig08\\_25.c](#)

```
7  const char *string1 = "This is a test"; /* initialize char pointer */
8  const char *string2 = "beware"; /* initialize char pointer */
9
10 printf( "%s\\\"%s\\\"\\n'%c'%s\\n\\\"%s\\\"\\n",
11         "Of the characters in ", string2,
12         *strpbrk( string1, string2 ),
13         " appears earliest in ", string1 );
```

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

- Example: [fig08\\_25.c](#)

```
7  const char *string1 = "This is a test"; /* initialize char pointer */
8  const char *string2 = "beware"; /* initialize char pointer */
9
10 printf( "%s\\\"%s\\\"\\n'%c'%s\\n\\\"%s\\\"\\n",
11         "Of the characters in ", string2,
12         *strpbrk( string1, string2 ),
13         " appears earliest in ", string1 );
```

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

- Example: [fig08\\_25.c](#)

```
7  const char *string1 = "This is a test"; /* initialize char pointer */
8  const char *string2 = "beware"; /* initialize char pointer */
9
10 printf( "%s\\\"%s\\\"\\n'%c'%s\\n\\\"%s\\\"\\n",
11         "Of the characters in ", string2,
12         *strpbrk( string1, string2 ),
13         " appears earliest in ", string1 );
```

search for the first occurrence of any character in **string2**

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

- Example: [fig08\\_25.c](#)

```
7  const char *string1 = "This is a test"; /* initialize char pointer */
8  const char *string2 = "beware"; /* initialize char pointer */
9
10 printf( "%s\\\"%s\\\"\\n'%c'%s\\n\\\"%s\\\"\\n",
11         "Of the characters in ", string2,
12         *strpbrk( string1, string2 ),
13         " appears earliest in ", string1 );
```

search for the first occurrence of any character in **string2**

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

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

- Example: [fig08\\_26.c](#)

```
8  const char *string1 = "A zoo has many animals including zebras";
9
10 int c = 'z'; /* character to search for */
11
12 printf( "%s\n%s'%c'%s\"%s\""\n",
13         "The remainder of string1 beginning with the",
14         "last occurrence of character ", c,
15         " is: ", strchr( string1, c ) );
```

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

- Example: [fig08\\_26.c](#)

```
8  const char *string1 = "A zoo has many animals including zebras";
9
10 int c = 'z'; /* character to search for */
11
12 printf( "%s\n%s'%c'%s\"%s\""\n",
13         "The remainder of string1 beginning with the",
14         "last occurrence of character ", c,
15         " is: ", strchr( string1, c ) );
```



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

- Example: [fig08\\_26.c](#)

```
8  const char *string1 = "A zoo has many animals including zebras";
9
10 int c = 'z'; /* character to search for */
11
12 printf( "%s\n%s'%c'%s\"%s\""\n",
13         "The remainder of string1 beginning with the",
14         "last occurrence of character ", c,
15         " is: ", strchr( string1, c ) );
```

search for the last occurrence of the specified character in a string

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

- Example: [fig08\\_26.c](#)

```
8  const char *string1 = "A zoo has many animals including zebras";
9
10 int c = 'z'; /* character to search for */
11
12 printf( "%s\n%s'%c'%s\"%s\""\n",
13         "The remainder of string1 beginning with the",
14         "last occurrence of character ", c,
15         " is: ", strchr( string1, c ) );
```

search for the last occurrence of the specified character in a string

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

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

- Example: [fig08\\_27.c](#)

```
8  const char *string1 = "The value is 3.14159";
9  const char *string2 = "aehi lsTuv";
10
11  printf( "%s%s\n%s%s\n\n%s\n%s%u\n",
12          "string1 = ", string1, "string2 = ", string2,
13          "The length of the initial segment of string1",
14          "containing only characters from string2 = ",
15          strspn( string1, string2 ) );
```

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

- Example: [fig08\\_27.c](#)

```
8  const char *string1 = "The value is 3.14159";
9  const char *string2 = "aehi lsTuv";
10
11  printf( "%s%s\n%s%s\n\n%s\n%s%u\n",
12          "string1 = ", string1, "string2 = ", string2,
13          "The length of the initial segment of string1",
14          "containing only characters from string2 = ",
15          strspn( string1, string2 ) );
```

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

- Example: [fig08\\_27.c](#)

```
8  const char *string1 = "The value is 3.14159";
9  const char *string2 = "aehi lsTuv";
10
11  printf( "%s%s\n%s%s\n\n%s\n%s%u\n",
12          "string1 = ", string1, "string2 = ", string2,
13          "The length of the initial segment of string1",
14          "containing only characters from string2 = ",
15          strspn( string1, string2 ) );
```

determine the length of the initial part of the **string1** that contains only characters from the **string2**

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

- Example: [fig08\\_27.c](#)

```
8  const char *string1 = "The value is 3.14159";
9  const char *string2 = "aehi lsTuv";
10
11  printf( "%s%s\n%s%s\n\n%s\n%s%u\n",
12          "string1 = ", string1, "string2 = ", string2,
13          "The length of the initial segment of string1",
14          "containing only characters from string2 = ",
15          strspn( string1, string2 ) );
```

determine the length of the initial part of the **string1** that contains only characters from the **string2**

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

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



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

- Example: [fig08\\_28.c](#)

```
7  const char *string1 = "abcdefabcdef"; /* string to search */
8  const char *string2 = "def"; /* string to search for */
9
10 printf( "%s%s\n%s%s\n\n%s\n%s%s\n",
11         "string1 = ", string1, "string2 = ", string2,
12         "The remainder of string1 beginning with the",
13         "first occurrence of string2 is: ",
14         strstr( string1, string2 ) );
```

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

- Example: [fig08\\_28.c](#)

```
7  const char *string1 = "abcdefabcdef"; /* string to search */
8  const char *string2 = "def"; /* string to search for */
9
10 printf( "%s%s\n%s%s\n\n%s\n%s%s\n",
11         "string1 = ", string1, "string2 = ", string2,
12         "The remainder of string1 beginning with the",
13         "first occurrence of string2 is: ",
14         strstr( string1, string2 ) );
```

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

- Example: [fig08\\_28.c](#)

```
7  const char *string1 = "abcdefabcdef"; /* string to search */
8  const char *string2 = "def"; /* string to search for */
9
10 printf( "%s%s\n%s%s\n\n%s\n%s%s\n",
11         "string1 = ", string1, "string2 = ", string2,
12         "The remainder of string1 beginning with the",
13         "first occurrence of string2 is: ",
14         strstr( string1, string2 ) );
```

search for the first  
occurrence of its  
second string in its  
first sting

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

- Example: [fig08\\_28.c](#)

```
7  const char *string1 = "abcdefabcdef"; /* string to search */
8  const char *string2 = "def"; /* string to search for */
9
10 printf( "%s%s\n%s%s\n\n%s\n%s%s\n",
11         "string1 = ", string1, "string2 = ", string2,
12         "The remainder of string1 beginning with the",
13         "first occurrence of string2 is: ",
14         strstr( string1, string2 ) );
```

search for the first  
occurrence of its  
second string in its  
first sting

```
string1 = abcdefabcdef
string2 = def

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

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

- Example: [fig08\\_29.c](#)

```
8 char string[] = "This is a sentence with 7 tokens";
9 char *tokenPtr; /* create char pointer */
10
11 printf( "%s\n%s\n\n%s\n",
12         "The string to be tokenized is:", string,
13         "The tokens are:" );
14
15 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 }
```

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

- Example: [fig08\\_29.c](#)

```
8 char string[] = "This is a sentence with 7 tokens";
9 char *tokenPtr; /* create char pointer */
10
11 printf( "%s\n%s\n\n%s\n",
12         "The string to be tokenized is:", string,
13         "The tokens are:" );
14
15 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 }
```



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

- Example: [fig08\\_29.c](#)

```
8 char string[] = "This is a sentence with 7 tokens";
9 char *tokenPtr; /* create char pointer */
10
11 printf( "%s\n%s\n\n%s\n",
12         "The string to be tokenized is:", string,
13         "The tokens are:" );
14
15 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 }
```

break string into a series of tokens by using " " (space)

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

- Example: [fig08\\_29.c](#)

```
8 char string[] = "This is a sentence with 7 tokens";
9 char *tokenPtr; /* create char pointer */
10
11 printf( "%s\n%s\n\n%s\n",
12         "The string to be tokenized is:", string,
13         "The tokens are:" );
14
15 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 }
```

break string into a series of tokens by using " " (space)

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

- Example: [fig08\\_29.c](#)

```
8 char string[] = "This is a sentence with 7 tokens";
9 char *tokenPtr; /* create char pointer */
10
11 printf( "%s\n%s\n\n%s\n",
12         "The string to be tokenized is:", string,
13         "The tokens are:" );
14
15 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 }
```

break string into a series of tokens by using " " (space)

The **NULL** argument indicates the call continue the location of the last call

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

- Example: [fig08\\_29.c](#)

```
8 char string[] = "This is a sentence with 7 tokens";
9 char *tokenPtr; /* create char pointer */
10
11 printf( "%s\n%s\n\n%s\n",
12         "The string to be tokenized is:", string,
13         "The tokens are:" );
14
15 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 }
```

break string into a series of tokens by using " " (space)

The **NULL** argument indicates the call continue the location of the last call

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

# 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.

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

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

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



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

Function prototype	Function description
<code>void *memchr( const void *s, int c, size_t n );</code>	Locates the first occurrence of <code>c</code> (converted to unsigned char) in the first <code>n</code> characters of the object pointed to by <code>s</code> . If <code>c</code> is found, a pointer to <code>c</code> in the object is returned. Otherwise, NULL is returned.
<code>void *memset( void *s, int c, size_t n );</code>	Copies <code>c</code> (converted to unsigned char) into the first <code>n</code> characters of the object pointed to by <code>s</code> . 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 Chapter 7, we call the pointer **void \*** a **generic pointer**
- Because a **void \*** pointer cannot be dereferenced, each function receives a size argument that specifies the number of characters (bytes) the function will process.

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

- **memcpy**
  - Copy 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

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

- Example: [fig08\\_31.c](#)

```
7  char s1[ 17 ]; /* create char array s1 */
8  char s2[] = "Copy this string"; /* initialize char array s2 */
9
10 memcpy( s1, s2, 17 );
11 printf( "%s\n%s\n"%s\n"\n",
12         "After s2 is copied into s1 with memcpy,",
13         "s1 contains ", s1 );
```

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

- Example: [fig08\\_31.c](#)

```
7 char s1[ 17 ]; /* create char array s1 */
8 char s2[] = "Copy this string"; /* initialize char array s2 */
9
10 memcpy( s1, s2, 17 );
11 printf( "%s\n%s\n", s1, s2 );
12     "After s2 is copied into s1 with memcpy,",
13     "s1 contains ", s1 );
```

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

- Example: [fig08\\_31.c](#)

```
7 char s1[ 17 ]; /* create char array s1 */
8 char s2[] = "Copy this string"; /* initialize char array s2 */
9
10 memcpy( s1, s2, 17 );
11 printf( "%s\n%s\n%s\n",
12         "After s2 is copied into s1 with memcpy,",
13         "s1 contains ", s1 );
```

copy the string in  
array **s2** to array **s1**



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

- Example: [fig08\\_31.c](#)

```
7 char s1[ 17 ]; /* create char array s1 */
8 char s2[] = "Copy this string"; /* initialize char array s2 */
9
10 memcpy( s1, s2, 17 );
11 printf( "%s\n%s\n%s\n",
12         "After s2 is copied into s1 with memcpy,",
13         "s1 contains ", s1 );
```

copy the string in  
array **s2** to array **s1**

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

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

- **memmove**

- Copy 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**.

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

- Example: [fig08\\_32.c](#)

```
7  char x[] = "Home Sweet Home"; /* initialize char array x */
8
9  printf( "%s%s\n", "The string in array x before memmove is: ", x );
10 printf( "%s%s\n", "The string in array x after memmove is: ",
11         memmove( x, &x[ 5 ], 10 ) );
```

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

- Example: [fig08\\_32.c](#)

```
7  char x[] = "Home Sweet Home"; /* initialize char array x */
8
9  printf( "%s%s\n", "The string in array x before memmove is: ", x );
10 printf( "%s%s\n", "The string in array x after memmove is: ",
11         memmove( x, &x[ 5 ], 10 ) );
```

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

- Example: [fig08\\_32.c](#)

```
7 char x[] = "Home Sweet Home"; /* initialize char array x */
8
9 printf( "%s%s\n", "The string in array x before memmove is: ", x );
10 printf( "%s%s\n", "The string in array x after memmove is: ",
11 memmove( x, &x[ 5 ], 10 ) );
```

copy the last 10  
bytes of array **x** into  
the first 10 bytes of  
array **x**

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

- Example: [fig08\\_32.c](#)

```
7 char x[] = "Home Sweet Home"; /* initialize char array x */
8
9 printf( "%s%s\n", "The string in array x before memmove is: ", x );
10 printf( "%s%s\n", "The string in array x after memmove is: ",
11 memmove( x, &x[ 5 ], 10 ) );
```

copy the last 10  
bytes of array **x** into  
the first 10 bytes of  
array **x**

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



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

- **memcmp**
  - Compare 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.

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

- Example: [fig08\\_33.c](#)

```
7 char s1[] = "ABCDEFGG"; /* initialize char array s1 */
8 char s2[] = "ABCDXYZ"; /* initialize char array s2 */
9
10 printf( "%s%s\n%s%s\n\n%s%2d\n%s%2d\n%s%2d\n",
11         "s1 = ", s1, "s2 = ", s2,
12         "memcmp( s1, s2, 4 ) = ", memcmp( s1, s2, 4 ),
13         "memcmp( s1, s2, 7 ) = ", memcmp( s1, s2, 7 ),
14         "memcmp( s2, s1, 7 ) = ", memcmp( s2, s1, 7 ) );
```

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

- Example: [fig08\\_33.c](#)

```
7 char s1[] = "ABCDEFGH"; /* initialize char array s1 */
8 char s2[] = "ABCDXYZ"; /* initialize char array s2 */
9
10 printf( "%s%s\n%s%s\n\n%s%2d\n%s%2d\n%s%2d\n",
11         "s1 = ", s1, "s2 = ", s2,
12         "memcmp( s1, s2, 4 ) = ", memcmp( s1, s2, 4 ),
13         "memcmp( s1, s2, 7 ) = ", memcmp( s1, s2, 7 ),
14         "memcmp( s2, s1, 7 ) = ", memcmp( s2, s1, 7 ) );
```

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

- Example: [fig08\\_33.c](#)

```
7 char s1[] = "ABCDEFGG"; /* initialize char array s1 */
8 char s2[] = "ABCDXYZ"; /* initialize char array s2 */
9
10 printf( "%s%s\n%s%s\n\n%s%2d\n%s%2d\n%s%2d\n",
11         "s1 = ", s1, "s2 = ", s2,
12         "memcmp( s1, s2, 4 ) = ", memcmp( s1, s2, 4 ),
13         "memcmp( s1, s2, 7 ) = ", memcmp( s1, s2, 7 ),
14         "memcmp( s2, s1, 7 ) = ", memcmp( s2, s1, 7 ) );
```

compare the first 4  
and 7 characters

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

- Example: [fig08\\_33.c](#)

```
7 char s1[] = "ABCDEFGH"; /* initialize char array s1 */
8 char s2[] = "ABCDXYZ"; /* initialize char array s2 */
9
10 printf( "%s%s\n%s%s\n\n%s%2d\n%s%2d\n%s%2d\n",
11         "s1 = ", s1, "s2 = ", s2,
12         "memcmp( s1, s2, 4 ) = ", memcmp( s1, s2, 4 ),
13         "memcmp( s1, s2, 7 ) = ", memcmp( s1, s2, 7 ),
14         "memcmp( s2, s1, 7 ) = ", memcmp( s2, s1, 7 ) );
```

compare the first 4  
and 7 characters

```
s1 = ABCDEFG
s2 = ABCDXYZ

memcmp( s1, s2, 4 ) = 0
memcmp( s1, s2, 7 ) = -19
memcmp( s2, s1, 7 ) = 19
```

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

- **memchr**
  - Search 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.



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

- Example: [fig08\\_34.c](#)

```
7  const char *s = "This is a string"; /* initialize char pointer */
8
9  printf( "%s\'%c\'%s\'%s'\n",
10         "The remainder of s after character ", 'r',
11         " is found is ", memchr( s, 'r', 16 ) );
12  return 0; /* indicates successful termination */
```

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

- Example: [fig08\\_34.c](#)

```
7  const char *s = "This is a string"; /* initialize char pointer */
8
9  printf( "%s\'%c\'%s\'%s'\n",
10         "The remainder of s after character ", 'r',
11         " is found is ", memchr( s, 'r', 16 ) );
12  return 0; /* indicates successful termination */
```

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

- Example: [fig08\\_34.c](#)

```
7  const char *s = "This is a string"; /* initialize char pointer */
8
9  printf( "%s\'%c\'%s\'%s'\n",
10         "The remainder of s after character ", 'r',
11         " is found is ", memchr( s, 'r', 16 ) );
12  return 0; /* indicates successful termination */
```

search for 'r' in s  
within 16 bytes

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

- Example: [fig08\\_34.c](#)

```
7  const char *s = "This is a string"; /* initialize char pointer */
8
9  printf( "%s\'%c\'%s\'%s'\n",
10         "The remainder of s after character ", 'r',
11         " is found is ", memchr( s, 'r', 16 ) );
12  return 0; /* indicates successful termination */
```

search for 'r' in s  
within 16 bytes

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

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

- **memset**
  - Copy 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.

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

- Example: [fig08\\_35.c](#)

```
7  char string1[ 15 ] = "BBBBBBBBBBBBBBB"; /* initialize string1 */
8
9  printf( "string1 = %s\n", string1 );
10 printf( "string1 after memset = %s\n", memset( string1, 'b', 7 ) );
11 return 0; /* indicates successful termination */
```



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

- Example: [fig08\\_35.c](#)

```
7  char string1[ 15 ] = "BBBBBBBBBBBBBBB"; /* initialize string1 */
8
9  printf( "string1 = %s\n", string1 );
10 printf( "string1 after memset = %s\n", memset( string1, 'b', 7 ) );
11 return 0; /* indicates successful termination */
```

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

- Example: [fig08\\_35.c](#)

```
7 char string1[ 15 ] = "BBBBBBBBBBBBBBB"; /* initialize string1 */
8
9 printf( "string1 = %s\n", string1 );
10 printf( "string1 after memset = %s\n", memset( string1, 'b', 7 ) );
11 return 0; /* indicates successful termination */
```

copy 'b' into the first  
7 bytes of **string1**

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

- Example: [fig08\\_35.c](#)

```
7 char string1[ 15 ] = "BBBBBBBBBBBBBBB"; /* initialize string1 */
8
9 printf( "string1 = %s\n", string1 );
10 printf( "string1 after memset = %s\n", memset( string1, 'b', 7 ) );
11 return 0; /* indicates successful termination */
```

copy 'b' into the first  
7 bytes of **string1**

```
string1 = BBBBBBBBBBBBBBBB
string1 after memset = bbbbbbbBBBBBBB
```

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

Function prototype	Function description
<code>char *strerror( int errornum );</code>	Maps <code>errornum</code> 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.
<code>size_t strlen( const char *s );</code>	Determines the length of string <code>s</code> . The number of characters preceding the terminating null character is returned.

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

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

- **strerror**
  - Take an error number and creates an error message string.
  - A pointer to the string is returned

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

- Example: [fig08\\_37.c](#)

```
6 int main( void ) {  
7     printf( "%s\n", strerror( 2 ) );  
8     return 0; /* indicates successful termination */  
9 }
```

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

- Example: [fig08\\_37.c](#)

```
6 int main( void ) {  
7     printf( "%s\n", strerror( 2 ) );  
8     return 0; /* indicates successful termination */  
9 }
```



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

- Example: [fig08\\_37.c](#)

```
6 int main( void ) {  
7     printf( "%s\n", strerror( 2 ) );  
8     return 0; /* indicates successful termination */  
9 }
```

print the message of  
error number 2

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

- Example: [fig08\\_37.c](#)

```
6 int main( void ) {  
7     printf( "%s\n", strerror( 2 ) );  
8     return 0; /* indicates successful termination */  
9 }
```

print the message of  
error number 2

No such file or directory

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

- Standard error values

```
Error # 0: Unknown error: 0
Error # 1: Operation not permitted
Error # 2: No such file or directory
Error # 3: No such process
Error # 4: Interrupted system call
Error # 5: Input/output error
Error # 6: Device not configured
Error # 7: Argument list too long
Error # 8: Exec format error
Error # 9: Bad file descriptor
Error # 10: No child processes

Error # 93: Attribute not found
Error # 94: Bad message
Error # 95: EMULTIHOP (Reserved)
Error # 96: No message available on STREAM
Error # 97: ENOLINK (Reserved)
Error # 98: No STREAM resources
Error # 99: Not a STREAM
Error #100: Protocol error
Error #101: STREAM ioctl timeout
Error #102: Operation not supported on socket
```

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

- **strlen**
  - Take a string as an argument and returns the number of characters in the string—the terminating null character is not included in the length.

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

- Example: [fig08\\_38.c](#)

```
8  const char *string1 = "abcdefghijklmnopqrstuvwxyz";
9  const char *string2 = "four";
10 const char *string3 = "Boston";
11
12 printf("%s\\%s\\%s%lu\\n%s\\%s\\%s%lu\\n%s\\%s\\%s%lu\\n",
13        "The length of ", string1, " is ",
14        ( unsigned long ) strlen( string1 ),
15        "The length of ", string2, " is ",
16        ( unsigned long ) strlen( string2 ),
17        "The length of ", string3, " is ",
18        ( unsigned long ) strlen( string3 ) );
19 return 0; /* indicates successful termination */
```

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

- Example: [fig08\\_38.c](#)

```
8  const char *string1 = "abcdefghijklmnopqrstuvwxyz";
9  const char *string2 = "four";
10 const char *string3 = "Boston";
11
12 printf("%s\\%s\\%s%lu\\n%s\\%s\\%s%lu\\n%s\\%s\\%s%lu\\n",
13        "The length of ", string1, " is ",
14        ( unsigned long ) strlen( string1 ),
15        "The length of ", string2, " is ",
16        ( unsigned long ) strlen( string2 ),
17        "The length of ", string3, " is ",
18        ( unsigned long ) strlen( string3 ) );
19 return 0; /* indicates successful termination */
```

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

- Example: [fig08\\_38.c](#)

```
8  const char *string1 = "abcdefghijklmnopqrstuvwxyz";
9  const char *string2 = "four";
10 const char *string3 = "Boston";
11
12 printf("%s\\%s\\%s%lu\\n%s\\%s\\%s%lu\\n%s\\%s\\%s%lu\\n",
13        "The length of ", string1, " is ",
14        ( unsigned long ) strlen( string1 ),
15        "The length of ", string2, " is ",
16        ( unsigned long ) strlen( string2 ),
17        "The length of ", string3, " is ",
18        ( unsigned long ) strlen( string3 ) );
19 return 0; /* indicates successful termination */
```

show the lengths of  
**string1**, **string2**,  
and **string3**



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

- Example: [fig08\\_38.c](#)

```
8  const char *string1 = "abcdefghijklmnopqrstuvwxyz";
9  const char *string2 = "four";
10 const char *string3 = "Boston";
11
12 printf("%s\\%s\\%s%lu\\n%s\\%s\\%s%lu\\n%s\\%s\\%s%lu\\n",
13        "The length of ", string1, " is ",
14        ( unsigned long ) strlen( string1 ),
15        "The length of ", string2, " is ",
16        ( unsigned long ) strlen( string2 ),
17        "The length of ", string3, " is ",
18        ( unsigned long ) strlen( string3 ) );
19 return 0; /* indicates successful termination */
```

show the lengths of  
**string1**, **string2**,  
and **string3**

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

# C Formatted Input/Output

# Objectives

- In this chapter, you'll learn
  - To use **input** and **output streams**
  - To use all **print formatting** capabilities
  - To use all **input formatting** capabilities
  - To print with **field widths** and **precisions**
  - To use formatting flags in the **printf** format control string
  - To output literals and escape sequence
  - To **format** input using **scanf**

- 9.1 Introduction
- 9.2 Streams
- 9.3 Formatting Output with `printf`
- 9.4 Printing Integers
- 9.5 Printing Floating-Point Numbers
- 9.6 Printing Strings and Characters
- 9.7 Other Conversion Specifiers
- 9.8 Printing with Field Widths and Precision
- 9.9 Using Flags in the `printf` Format Control String
- 9.10 Printing Literals and Escape Sequences
- 9.11 Reading Formatted Input with `scanf`

# Introduction

- An important part of the solution to any problem is the presentation of the results.
- In this chapter, we discuss in depth the formatting features of **scanf** and **printf**.
- These functions input data from the **standard input stream** and output data to the **standard output stream**.

# Streams

- All input and output is performed with **streams**, which are sequences of bytes.
- In **input** operations, the **bytes flow** from a device (e.g., a keyboard, a disk drive, a network connection) to main memory.
- In **output** operations, **bytes flow** from main memory to a device (e.g., a display screen, a printer, a disk drive, a network connection, and so on).
- When program execution begins, three streams (**stdin**, **stdout**, **stderr**) are connected to the program automatically.

# Streams (Cont.)

- Operating systems often allow these streams to be **redirected** to other devices.
- A third stream, the **standard error stream**, is connected to the screen.



# Formatting Output with **printf**

- Every **printf** call contains a **format control string** that describes the output format.
- The format control string consists of **conversion specifiers**, **flags**, **field widths**, **precisions** and **literal characters**.
- Together with the **percent sign (%)**, these form **conversion specifications**.

# Formatting Output with **printf**

## (Cont.)

- The printf function has the form

```
printf(format-control-string,  
other-arguments);
```

- **format-control-string** describes the output format
- **other-arguments** (which are optional) correspond to each conversion specification in format-control-string.

# Printing Integers

- An integer is a whole number, such as 776, 0 or -52, that contains no decimal point.
- Integer values are displayed in one of several formats.

# Printing Integers (Cont.)

Conversion specifier	Description
d	Display as a signed decimal integer.
i	Display as a signed decimal integer. [ <i>Note:</i> The i and d specifiers are different when used with scanf.]
o	Display as an unsigned octal integer.
u	Display as an unsigned decimal integer.
x or X	Display as an unsigned hexadecimal integer. X causes the digits 0-9 and the letters A-F to be displayed and x causes the digits 0-9 and a-f to be displayed.
h or l (letter l)	Place before any integer conversion specifier to indicate that a short or long integer is displayed, respectively. Letters h and l are more precisely called <b>length modifiers</b> .

# Printing Integers (Cont.)

- Example: [fig09\\_02.c](#)

```
5 int main( void ) {
6     printf( "%d\n", 455 );
7     printf( "%i\n", 455 ); /* i same as d in printf */
8     printf( "%d\n", +455 );
9     printf( "%d\n", -455 );
10    printf( "%hd\n", 32000 );
11    printf( "%ld\n", 2000000000L ); /* L suffix makes literal a long */
12    printf( "%o\n", 455 );
13    printf( "%u\n", 455 );
14    printf( "%u\n", -455 );
15    printf( "%x\n", 455 );
16    printf( "%X\n", 455 );
17    return 0; /* indicates successful termination */
}
```

# Printing Integers (Cont.)

- Example: [fig09\\_02.c](#)

```
5 int main( void ) {
6     printf( "%d\n", 455 );
7     printf( "%i\n", 455 ); /* i same as d in printf */
8     printf( "%d\n", +455 );
9     printf( "%d\n", -455 );
10    printf( "%hd\n", 32000 );
11    printf( "%ld\n", 2000000000L ); /* L suffix makes literal a long */
12    printf( "%o\n", 455 );
13    printf( "%u\n", 455 );
14    printf( "%u\n", -455 );
15    printf( "%x\n", 455 );
16    printf( "%X\n", 455 );
17    return 0; /* indicates successful termination */
}
```

# Printing Integers (Cont.)

- Example: [fig09\\_02.c](#)

```
5 int main( void ) {
6     printf( "%d\n", 455 );
7     printf( "%i\n", 455 ); /* i same as d in printf */
8     printf( "%d\n", +455 );
9     printf( "%d\n", -455 );
10    printf( "%hd\n", 32000 );
11    printf( "%ld\n", 2000000000L ); /* L suffix makes literal a long */
12    printf( "%o\n", 455 );
13    printf( "%u\n", 455 );
14    printf( "%u\n", -455 );
15    printf( "%x\n", 455 );
16    printf( "%X\n", 455 );
17    return 0; /* indicates successful termination */
}
```

%u: converted to the  
unsigned value  
4294966841



# Printing Integers (Cont.)

- Example: [fig09\\_02.c](#)

```
5 int main( void ) {
6     printf( "%d\n", 455 );
7     printf( "%i\n", 455 ); /* i same as d in printf */
8     printf( "%d\n", +455 );
9     printf( "%d\n", -455 );
10    printf( "%hd\n", 32000 );
11    printf( "%ld\n", 2000000000L ); /* L suffix makes literal a long */
12    printf( "%o\n", 455 );
13    printf( "%u\n", 455 );
14    printf( "%u\n", -455 );
15    printf( "%x\n", 455 );
16    printf( "%X\n", 455 );
17    return 0; /* indicates successful termination */
```

%u: converted to the  
unsigned value  
4294966841

```
455
455
455
-455
32000
2000000000
707
455
4294966841
1c7
1C7
```

# Printing Floating-Point Numbers

- A floating-point value contains a decimal point as in 33.5, 0.0 or -657.983.
- Floating-point values are displayed in one of several formats.
- The **conversion specifiers `e` and `E`** display floating-point values in **exponential notation** — the computer equivalent of **scientific notation** used in mathematics.

# Printing Floating-Point Numbers (Cont.)

Conversion specifier	Description
<code>e</code> or <code>E</code>	Display a floating-point value in exponential notation.
<code>f</code>	Display floating-point values in fixed-point notation. [ <i>Note:</i> In C99, you can also use <code>F</code> .]
<code>g</code> or <code>G</code>	Display a floating-point value in either the floating-point form <code>f</code> or the exponential form <code>e</code> (or <code>E</code> ), based on the magnitude of the value.
<code>L</code>	Place before any floating-point conversion specifier to indicate that a long double floating-point value is displayed.

# Printing Floating-Point Numbers (Cont.)



## **Error-Prevention Tip 9.1**

*When outputting data, be sure that the user is aware of situations in which data may be imprecise due to formatting (e.g., rounding errors from specifying precisions).*

# Printing Floating-Point Numbers (Cont.)

- Example: [fig09\\_04.c](#)

```
8  printf( "%e\n", 1234567.89 );
9  printf( "%e\n", +1234567.89 );
10 printf( "%e\n", -1234567.89 );
11 printf( "%E\n", 1234567.89 );
12 printf( "%f\n", 1234567.89 );
13 printf( "%g\n", 1234567.89 );
14 printf( "%G\n", 1234567.89 );
15 return 0; /* indicates successful termination */
```

# Printing Floating-Point Numbers (Cont.)

- Example: [fig09\\_04.c](#)

```
8 printf( "%e\n", 1234567.89 );
9 printf( "%e\n", +1234567.89 );
10 printf( "%e\n", -1234567.89 );
11 printf( "%E\n", 1234567.89 );
12 printf( "%f\n", 1234567.89 );
13 printf( "%g\n", 1234567.89 );
14 printf( "%G\n", 1234567.89 );
15 return 0; /* indicates successful termination */
```

```
1.234568e+06
1.234568e+06
-1.234568e+06
1.234568E+06
1234567.890000
1.23457e+06
1.23457E+06
```

# Printing Strings and Characters

- The **c** and **s** conversion specifiers are used to print individual **characters** and **strings**, respectively.
- **Conversion specifier c** requires a **char** argument.
- **Conversion specifier s** requires a **pointer to char** as an argument.
- Conversion specifier **s** causes characters to be printed until a terminating null (**' \0 '**) character is encountered.



# Printing Strings and Characters (Cont.)



## Common Programming Error 9.3

*Using %c to print a string is an error. The conversion specifier %c expects a char argument. A string is a pointer to char (i.e., a char \*).*



## Common Programming Error 9.4

*Using %s to print a char argument often causes a fatal execution-time error called an access violation. The conversion specifier %s expects an argument of type pointer to char.*

# Printing Strings and Characters (Cont.)



## Common Programming Error 9.5

*Using single quotes around character strings is a syntax error. Character strings must be enclosed in double quotes.*



## Common Programming Error 9.6

*Using double quotes around a character constant creates a pointer to a string consisting of two characters, the second of which is the terminating null.*

# Printing Strings and Characters (Cont.)

- Example: [fig09\\_05.c](#)

```
6 char character = 'A'; /* initialize char */
7 char string[] = "This is a string"; /* initialize char array */
8 const char *stringPtr = "This is also a string"; /* char pointer */
9
10 printf( "%c\n", character );
11 printf( "%s\n", "This is a string" );
12 printf( "%s\n", string );
13 printf( "%s\n", stringPtr );
14 return 0; /* indicates successful termination */
```

# Printing Strings and Characters (Cont.)

- Example: [fig09\\_05.c](#)

```
6 char character = 'A'; /* initialize char */
7 char string[] = "This is a string"; /* initialize char array */
8 const char *stringPtr = "This is also a string"; /* char pointer */
9
10 printf( "%c\n", character );
11 printf( "%s\n", "This is a string" );
12 printf( "%s\n", string );
13 printf( "%s\n", stringPtr );
14 return 0; /* indicates successful termination */
```

```
A
This is a string
This is a string
This is also a string
```

# Other Conversion Specifiers

Conversion specifier	Description
p	Display a pointer value in an implementation-defined manner.
n	Store the number of characters already output in the current <code>printf</code> statement. A pointer to an integer is supplied as the corresponding argument. Nothing is displayed.
%	Display the percent character.

# Other Conversion Specifiers (Cont.)

- Example: [fig09\\_07.c](#)

```
6  int *ptr; /* define pointer to int */
7  int x = 12345; /* initialize int x */
8  int y; /* define int y */
9
10 ptr = &x; /* assign address of x to ptr */
11 printf( "The value of ptr is %p\n", ptr );
12 printf( "The address of x is %p\n\n", &x );
13
14 printf( "Total characters printed on this line:%n", &y );
15 printf( " %d\n\n", y );
16
17 y = printf( "This line has 28 characters\n" );
18 printf( "%d characters were printed\n\n", y );
19
20 printf( "Printing a %% in a format control string\n" );
21 return 0; /* indicates successful termination */
```



# Other Conversion Specifiers (Cont.)

- Example: [fig09\\_07.c](#)

```
6  int *ptr; /* define pointer to int */
7  int x = 12345; /* initialize int x */
8  int y; /* define int y */
9
10 ptr = &x; /* assign address of x to ptr */
11 printf( "The value of ptr is %p\n", ptr );
12 printf( "The address of x is %p\n\n", &x );
13
14 printf( "Total characters printed on this line:%n", &y );
15 printf( " %d\n\n", y );
16
17 y = printf( "This line has 28 characters\n" );
18 printf( "%d characters were printed\n\n", y );
19
20 printf( "Printing a %% in a format control string\n" );
21 return 0; /* indicates successful termination */
```



# Other Conversion Specifiers (Cont.)

- Example: [fig09\\_07.c](#)

```
6  int *ptr; /* define pointer to int */
7  int x = 12345; /* initialize int x */
8  int y; /* define int y */
9
10 ptr = &x; /* assign address of x to ptr */
11 printf( "The value of ptr is %p\n", ptr );
12 printf( "The address of x is %p\n\n", &x );
13
14 printf( "Total characters printed on this line:%n", &y );
15 printf( " %d\n\n", y );
16
17 y = printf( "This line has 28 characters\n" );
18 printf( "%d characters were printed\n\n", y );
19
20 printf( "Printing a %% in a format control string\n" );
21 return 0; /* indicates successful termination */
```

print the address of **x**

# Other Conversion Specifiers (Cont.)

- Example: [fig09\\_07.c](#)

```
6  int *ptr; /* define pointer to int */
7  int x = 12345; /* initialize int x */
8  int y; /* define int y */
9
10 ptr = &x; /* assign address of x to ptr */
11 printf( "The value of ptr is %p\n", ptr );
12 printf( "The address of x is %p\n\n", &x );
13
14 printf( "Total characters printed on this line:%n", &y );
15 printf( " %d\n\n", y );
16
17 y = printf( "This line has 28 characters\n" );
18 printf( "%d characters were printed\n\n", y );
19
20 printf( "Printing a %% in a format control string\n" );
21 return 0; /* indicates successful termination */
```

print the address of **x**

# Other Conversion Specifiers (Cont.)

- Example: [fig09\\_07.c](#)

```
6  int *ptr; /* define pointer to int */
7  int x = 12345; /* initialize int x */
8  int y; /* define int y */
9
10 ptr = &x; /* assign address of x to ptr */
11 printf( "The value of ptr is %p\n", ptr );
12 printf( "The address of x is %p\n\n", &x );
13
14 printf( "Total characters printed on this line:%n", &y );
15 printf( " %d\n\n", y );
16
17 y = printf( "This line has 28 characters\n" );
18 printf( "%d characters were printed\n\n", y );
19
20 printf( "Printing a %% in a format control string\n" );
21 return 0; /* indicates successful termination */
```

print the address of **x**

store the number of  
characters into **y**

# Other Conversion Specifiers (Cont.)

- Example: [fig09\\_07.c](#)

```
6  int *ptr; /* define pointer to int */
7  int x = 12345; /* initialize int x */
8  int y; /* define int y */
9
10 ptr = &x; /* assign address of x to ptr */
11 printf( "The value of ptr is %p\n", ptr );
12 printf( "The address of x is %p\n\n", &x );
13
14 printf( "Total characters printed on this line:%n", &y );
15 printf( " %d\n\n", y );
16
17 y = printf( "This line has 28 characters\n" );
18 printf( "%d characters were printed\n\n", y );
19
20 printf( "Printing a %% in a format control string\n" );
21 return 0; /* indicates successful termination */
```

print the address of **x**

store the number of  
characters into **y**

# Other Conversion Specifiers (Cont.)

- Example: [fig09\\_07.c](#)

```
6  int *ptr; /* define pointer to int */
7  int x = 12345; /* initialize int x */
8  int y; /* define int y */
9
10 ptr = &x; /* assign address of x to ptr */
11 printf( "The value of ptr is %p\n", ptr );
12 printf( "The address of x is %p\n\n", &x );
13
14 printf( "Total characters printed on this line:%n", &y );
15 printf( " %d\n\n", y );
16
17 y = printf( "This line has 28 characters\n" );
18 printf( "%d characters were printed\n\n", y );
19
20 printf( "Printing a %% in a format control string\n" );
21 return 0; /* indicates successful termination */
```

print the address of **x**

store the number of  
characters into **y**

print the % character



# Other Conversion Specifiers (Cont.)

- Example: [fig09\\_07.c](#)

```
6  int *ptr; /* define pointer to int */
7  int x = 12345; /* initialize int x */
8  int y; /* define int y */
9
10 ptr = &x; /* assign address of x to ptr */
11 printf( "The value of ptr is %p\n", ptr );
12 printf( "The address of x is %p\n\n", &x );
13
14 printf( "Total characters printed on this line:%n", &y );
15 printf( " %d\n\n", y );
16
17 y = printf( "This line has 28 characters\n" );
18 printf( "%d characters were printed\n\n", y );
19
20 printf( "Printing a %% in a format control string\n" );
21 return 0; /* indicates successful termination */
```

print the address of **x**

store the number of  
characters into **y**

print the % character

```
The value of ptr is 0x7fff5fbfeddc
The address of x is 0x7fff5fbfeddc

Total characters printed on this line: 38

This line has 28 characters
28 characters were printed

Printing a % in a format control string
```

# Printing with Field Widths and Precision

- The exact size of a field in which data is printed is specified by a **field width**.
- An integer representing the field width is inserted between the **percent sign (%)** and the conversion specifier (e.g., **%4d**).
- Field widths can be used with all conversion specifiers.



# Printing with Field Widths and Precision (Cont.)

- Example: [fig09\\_08.c](#)

```
5 int main( void ) {
6     printf( "%4d\n", 1 );
7     printf( "%4d\n", 12 );
8     printf( "%4d\n", 123 );
9     printf( "%4d\n", 1234 );
10    printf( "%4d\n\n", 12345 ); /* data too large */
11
12    printf( "%4d\n", -1 );
13    printf( "%4d\n", -12 );
14    printf( "%4d\n", -123 );
15    printf( "%4d\n", -1234 ); /* data too large */
16    printf( "%4d\n", -12345 ); /* data too large */
17    return 0; /* indicates successful termination */
18 }
```

# Printing with Field Widths and Precision (Cont.)

- Example: [fig09\\_08.c](#)

```
5 int main( void ) {
6     printf( "%4d\n", 1 );
7     printf( "%4d\n", 12 );
8     printf( "%4d\n", 123 );
9     printf( "%4d\n", 1234 );
10    printf( "%4d\n\n", 12345 ); /* data too large */
11
12    printf( "%4d\n", -1 );
13    printf( "%4d\n", -12 );
14    printf( "%4d\n", -123 );
15    printf( "%4d\n", -1234 ); /* data too large */
16    printf( "%4d\n", -12345 ); /* data too large */
17    return 0; /* indicates successful termination */
18 }
```

```
  1
 12
123
1234
12345

-1
-12
-123
-1234
-12345
```

# Printing with Field Widths and Precision (Cont.)

- Function **printf** also enables you to specify the precision with which data is printed.
- To use precision, place a decimal point (.), followed by an integer representing the precision between the percent sign and the conversion specifier.

# Printing with Field Widths and Precision (Cont.)

- Example: [fig09\\_09.c](#)

```
7  int i = 873; /* initialize int i */
8  double f = 123.94536; /* initialize double f */
9  char s[] = "Happy Birthday"; /* initialize char array s */
10
11  printf( "Using precision for integers\n" );
12  printf( "\t%.4d\n\t%.9d\n\n", i, i );
13
14  printf( "Using precision for floating-point numbers\n" );
15  printf( "\t%.3f\n\t%.3e\n\t%.3g\n\n", f, f, f );
16
17  printf( "Using precision for strings\n" );
18  printf( "\t%.11s\n", s );
19  return 0; /* indicates successful termination */
```

# Printing with Field Widths and Precision (Cont.)

- Example: [fig09\\_09.c](#)

```
7  int i = 873; /* initialize int i */
8  double f = 123.94536; /* initialize double f */
9  char s[] = "Happy Birthday"; /* initialize char array s */
10
11  printf( "Using precision for integers\n" );
12  printf( "\t%.4d\n\t%.9d\n\n", i, i );
13
14  printf( "Using precision for floating-point numbers\n" );
15  printf( "\t%.3f\n\t%.3e\n\t%.3g\n\n", f, f, f );
16
17  printf( "Using precision for strings\n" );
18  printf( "\t%.11s\n", s );
19  return 0; /* indicates successful termination */
```

Using precision for integers

0873

000000873

Using precision for floating-point numbers

123.945

1.239e+02

124

Using precision for strings

Happy Birth

# Printing with Field Widths and Precision (Cont.)

- To use this feature, insert an asterisk (\*) in place of the field width or precision (or both).
- The statement

```
printf( "%*.*f", 7, 2, 98.736 );
```

uses 7 for the field width, 2 for the precision and outputs the value 98.74 right justified.

# Using Flags in the **printf** Formation Control String

- Function **printf** also provides flags to supplement its output formatting capabilities.
- Several flags may be combined in one conversion specifier.



# Using Flags in the `printf` Formation Control String (Cont.)

- Function `printf` also provides flags to supplement its output formatting capabilities.
- Five flags are available for use in format control strings

Flag	Description
- (minus sign)	Left justify the output within the specified field.
+ (plus sign)	Display a plus sign preceding positive values and a minus sign preceding negative values.
<i>space</i>	Print a space before a positive value not printed with the + flag.
#	Prefix 0 to the output value when used with the octal conversion specifier <code>o</code> .  Prefix 0x or 0X to the output value when used with the hexadecimal conversion specifiers <code>x</code> or <code>X</code> .  Force a decimal point for a floating-point number printed with <code>e</code> , <code>E</code> , <code>f</code> , <code>g</code> or <code>G</code> that does not contain a fractional part. (Normally the decimal point is printed only if a digit follows it.) For <code>g</code> and <code>G</code> specifiers, trailing zeros are not eliminated.
0 (zero)	Pad a field with leading zeros.

# Using Flags in the `printf` Formation Control String (Cont.)

- Example: [fig09\\_11.c](#)

```
5 int main( void ) {  
6     printf( "%10s%10d%10c%10f\n\n", "hello", 7, 'a', 1.23 );  
7     printf( "%-10s%-10d%-10c%-10f\n", "hello", 7, 'a', 1.23 );  
8     return 0; /* indicates successful termination */  
9 }
```

# Using Flags in the `printf` Formation Control String (Cont.)

- Example: [fig09\\_11.c](#)

```
5 int main( void ) {  
6     printf( "%10s%10d%10c%10f\n\n", "hello", 7, 'a', 1.23 );  
7     printf( "%-10s%-10d%-10c%-10f\n", "hello", 7, 'a', 1.23 );  
8     return 0; /* indicates successful termination */  
9 }
```

# Using Flags in the `printf` Formation Control String (Cont.)

- Example: [fig09\\_11.c](#)

```
5 int main( void ) {  
6     printf( "%10s%10d%10c%10f\n\n", "hello", 7, 'a', 1.23 );  
7     printf( "%-10s%-10d%-10c%-10f\n", "hello", 7, 'a', 1.23 );  
8     return 0; /* indicates successful termination */  
9 }
```

right justification

# Using Flags in the `printf` Formation Control String (Cont.)

- Example: [fig09\\_11.c](#)

```
5 int main( void ) {  
6     printf( "%10s%10d%10c%10f\n\n", "hello", 7, 'a', 1.23 );  
7     printf( "%-10s%-10d%-10c%-10f\n", "hello", 7, 'a', 1.23 );  
8     return 0; /* indicates successful termination */  
9 }
```

right justification

# Using Flags in the `printf` Formation Control String (Cont.)

- Example: [fig09\\_11.c](#)

```
5 int main( void ) {  
6     printf( "%10s%10d%10c%10f\n\n", "hello", 7, 'a', 1.23 );  
7     printf( "%-10s%-10d%-10c%-10f\n", "hello", 7, 'a', 1.23 );  
8     return 0; /* indicates successful termination */  
9 }
```

right justification

left justification

# Using Flags in the `printf` Formation Control String (Cont.)

- Example: [fig09\\_11.c](#)

```
5 int main( void ) {  
6     printf( "%10s%10d%10c%10f\n\n", "hello", 7, 'a', 1.23 );  
7     printf( "%-10s%-10d%-10c%-10f\n", "hello", 7, 'a', 1.23 );  
8     return 0; /* indicates successful termination */  
9 }
```

right justification

left justification

```
      hello          7          a  1.230000  
hello      7          a          1.230000
```



# Using Flags in the `printf` Formation Control String (Cont.)

- Example: [fig09\\_12.c](#)

```
5 int main( void ) {  
6     printf( "%d\n%d\n", 786, -786 );  
7     printf( "%+d\n%+d\n", 786, -786 );  
8     return 0; /* indicates successful termination */  
9 } /* end main */
```

# Using Flags in the `printf` Formation Control String (Cont.)

- Example: [fig09\\_12.c](#)

```
5 int main( void ) {  
6     printf( "%d\n%d\n", 786, -786 );  
7     printf( "%+d\n%+d\n", 786, -786 );  
8     return 0; /* indicates successful termination */  
9 } /* end main */
```

# Using Flags in the `printf` Formation Control String (Cont.)

- Example: [fig09\\_12.c](#)

```
5 int main( void ) {  
6     printf( "%d\n%d\n", 786, -786 );  
7     printf( "%+d\n%+d\n", 786, -786 );  
8     return 0; /* indicates successful termination */  
9 } /* end main */
```

print a positive number and a negative number

# Using Flags in the `printf` Formation Control String (Cont.)

- Example: [fig09\\_12.c](#)

```
5 int main( void ) {  
6     printf( "%d\n%d\n", 786, -786 );  
7     printf( "%+d\n%+d\n", 786, -786 );  
8     return 0; /* indicates successful termination */  
9 } /* end main */
```

print a positive number and a negative number

```
786  
-786  
+786  
-786
```

# Using Flags in the `printf` Formation Control String (Cont.)

- Example: [fig09\\_13.c](#)

```
6 int main( void ) {  
7     printf( "% d\n% d\n", 547, -547 );  
8     return 0; /* indicates successful termination */  
9 } /* end main */
```

# Using Flags in the `printf` Formation Control String (Cont.)

- Example: [fig09\\_13.c](#)

```
6 int main( void ) {  
7     printf( "% d\n% d\n", 547, -547 );  
8     return 0; /* indicates successful termination */  
9 } /* end main */
```

# Using Flags in the `printf` Formation Control String (Cont.)

- Example: [fig09\\_13.c](#)

```
6 int main( void ) {  
7     printf( "% d\n% d\n", 547, -547 );  
8     return 0; /* indicates successful termination */  
9 } /* end main */
```

display a space prefix to the numbers



# Using Flags in the `printf` Formation Control String (Cont.)

- Example: [fig09\\_13.c](#)

```
6 int main( void ) {  
7     printf( "% d\n% d\n", 547, -547 );  
8     return 0; /* indicates successful termination */  
9 } /* end main */
```

display a space prefix to the numbers

```
547  
-547
```

# Using Flags in the `printf` Formation Control String (Cont.)

- Example: [fig09\\_14.c](#)

```
6 int main( void ) {
7     int c = 1427; /* initialize c */
8     double p = 1427.0; /* initialize p */
9
10    printf( "%#o\n", c );
11    printf( "%#x\n", c );
12    printf( "%#X\n", c );
13    printf( "\n%g\n", p );
14    printf( "%#g\n", p );
15    return 0; /* indicates successful termination */
16 }
```

# Using Flags in the `printf` Formation Control String (Cont.)

- Example: [fig09\\_14.c](#)

```
6 int main( void ) {
7     int c = 1427; /* initialize c */
8     double p = 1427.0; /* initialize p */
9
10    printf( "%#o\n", c );
11    printf( "%#x\n", c );
12    printf( "%#X\n", c );
13    printf( "\n%g\n", p );
14    printf( "%#g\n", p );
15    return 0; /* indicates successful termination */
16 }
```

# Using Flags in the `printf` Formation Control String (Cont.)

- Example: [fig09\\_14.c](#)

```
6 int main( void ) {
7     int c = 1427; /* initialize c */
8     double p = 1427.0; /* initialize p */
9
10    printf( "%#o\n", c );
11    printf( "%#x\n", c );
12    printf( "%#X\n", c );
13    printf( "\n%g\n", p );
14    printf( "%#g\n", p );
15    return 0; /* indicates successful termination */
16 }
```

the octal value and  
hexadecimal value

# Using Flags in the `printf` Formation Control String (Cont.)

- Example: [fig09\\_14.c](#)

```
6 int main( void ) {
7     int c = 1427; /* initialize c */
8     double p = 1427.0; /* initialize p */
9
10    printf( "%#o\n", c );
11    printf( "%#x\n", c );
12    printf( "%#X\n", c );
13    printf( "\n%g\n", p );
14    printf( "%#g\n", p );
15    return 0; /* indicates successful termination */
16 }
```

the octal value and  
hexadecimal value

# Using Flags in the `printf` Formation Control String (Cont.)

- Example: [fig09\\_14.c](#)

```
6 int main( void ) {  
7     int c = 1427; /* initialize c */  
8     double p = 1427.0; /* initialize p */  
9  
10    printf( "%#o\n", c );  
11    printf( "%#x\n", c );  
12    printf( "%#X\n", c );  
13    printf( "\n%g\n", p );  
14    printf( "%#g\n", p );  
15    return 0; /* indicates successful termination */  
16 }
```

the octal value and  
hexadecimal value

force the decimal point on a  
value printed with **g**

# Using Flags in the `printf` Formation Control String (Cont.)

- Example: [fig09\\_14.c](#)

```
6 int main( void ) {  
7     int c = 1427; /* initialize c */  
8     double p = 1427.0; /* initialize p */  
9  
10    printf( "%#o\n", c );  
11    printf( "%#x\n", c );  
12    printf( "%#X\n", c );  
13    printf( "\n%g\n", p );  
14    printf( "%#g\n", p );  
15    return 0; /* indicates successful termination */  
16 }
```

the octal value and  
hexadecimal value

force the decimal point on a  
value printed with **g**

```
02623  
0x593  
0X593  
  
1427  
1427.00
```



# Using Flags in the `printf` Formation Control String (Cont.)

- Example: [fig09\\_15.c](#)

```
5 int main( void ) {  
6     printf( "%+09d\n", 452 );  
7     printf( "%09d\n", 452 );  
8     return 0; /* indicates successful termination */  
9 } /* end main */
```

# Using Flags in the `printf` Formation Control String (Cont.)

- Example: [fig09\\_15.c](#)

```
5 int main( void ) {  
6     printf( "%+09d\n", 452 );  
7     printf( "%09d\n", 452 );  
8     return 0; /* indicates successful termination */  
9 }
```

# Using Flags in the `printf` Formation Control String (Cont.)

- Example: [fig09\\_15.c](#)

```
5 int main( void ) {  
6     printf( "%+09d\n", 452 );  
7     printf( "%09d\n", 452 );  
8     return 0; /* indicates successful termination */  
9 }
```

combine the + flag and  
the 0 flag

# Using Flags in the `printf` Formation Control String (Cont.)

- Example: [fig09\\_15.c](#)

```
5 int main( void ) {  
6     printf( "%+09d\n", 452 );  
7     printf( "%09d\n", 452 );  
8     return 0; /* indicates successful termination */  
9 }
```

combine the + flag and  
the 0 flag

```
+00000452  
000000452
```

# Printing Literals and Escape Sequences

- Most literal characters to be printed in a **printf** statement can simply be included in the format control string.
- However, there are several “problem” characters, such as the quotation mark (") that delimits the format control string itself.

# Printing Literals and Escape Sequences (Cont.)

Escape sequence	Description
\' (single quote)	Output the single quote ( ' ) character.
\" (double quote)	Output the double quote ( " ) character.
\? (question mark)	Output the question mark ( ? ) character.
\\ (backslash)	Output the backslash ( \ ) character.
\a (alert or bell)	Cause an audible (bell) or visual alert.
\b (backspace)	Move the cursor back one position on the current line.
\f (new page or form feed)	Move the cursor to the start of the next logical page.
\n (newline)	Move the cursor to the beginning of the next line.
\r (carriage return)	Move the cursor to the beginning of the current line.
\t (horizontal tab)	Move the cursor to the next horizontal tab position.
\v (vertical tab)	Move the cursor to the next vertical tab position.

# Reading Formatted Input with **scanf**

- Precise input formatting can be accomplished with **scanf**.
- Every **scanf** statement contains a format control string that describes the format of the data to be input.
- Function **scanf** is written in the following form:

```
scanf(format-control-string, other-arguments);
```

**format-control-string** describes the formats of the input, and **other-arguments** are pointers to variables in which the input will be stored.



# Reading Formatted Input with **scanf**

## (Cont.)

- Conversion Specifiers for **scanf**

Conversion specifier	Description
<i>Integers</i>	
d	Read an optionally signed decimal integer. The corresponding argument is a pointer to an <code>int</code> .
i	Read an optionally signed decimal, octal or hexadecimal integer. The corresponding argument is a pointer to an <code>int</code> .
o	Read an octal integer. The corresponding argument is a pointer to an unsigned <code>int</code> .
u	Read an unsigned decimal integer. The corresponding argument is a pointer to an unsigned <code>int</code> .
x or X	Read a hexadecimal integer. The corresponding argument is a pointer to an unsigned <code>int</code> .
h or l	Place before any of the integer conversion specifiers to indicate that a short or long integer is to be input.

# Reading Formatted Input with **scanf**

## (Cont.)

Conversion specifier	Description
<i>Floating-point numbers</i>	
e, E, f, g or G	Read a floating-point value. The corresponding argument is a pointer to a floating-point variable.
l or L	Place before any of the floating-point conversion specifiers to indicate that a <code>double</code> or <code>long double</code> value is to be input. The corresponding argument is a pointer to a <code>double</code> or <code>long double</code> variable.
<i>Characters and strings</i>	
c	Read a character. The corresponding argument is a pointer to a <code>char</code> ; no null ( <code>'\0'</code> ) is added.
s	Read a string. The corresponding argument is a pointer to an array of type <code>char</code> that is large enough to hold the string and a terminating null ( <code>'\0'</code> ) character—which is automatically added.
<i>Scan set</i>	
[ <i>scan characters</i> ]	Scan a string for a set of characters that are stored in an array.

# Reading Formatted Input with **scanf**

## (Cont.)

Conversion specifier	Description
<i>Miscellaneous</i>	
<b>p</b>	Read an address of the same form produced when an address is output with <b>%p</b> in a <b>printf</b> statement.
<b>n</b>	Store the number of characters input so far in this call to <b>scanf</b> . The corresponding argument is a pointer to an <b>int</b> .
<b>%</b>	Skip a percent sign (%) in the input.

# Reading Formatted Input with `scanf`

## (Cont.)

- Example: [fig09\\_18.c](#)

```
5 int main( void ) {
6     int a;
7     int b;
8     int c;
9     int d;
10    int e;
11    int f;
12    int g;
13
14    printf( "Enter seven integers: " );
15    scanf( "%d%i%i%i%o%u%x", &a, &b, &c, &d, &e, &f, &g );
16
17    printf( "The input displayed as decimal integers is:\n" );
18    printf( "%d %d %d %d %d %d %d\n", a, b, c, d, e, f, g );
19    return 0; /* indicates successful termination */
}
```

```
| Enter seven integers: -50 -50 050 0x50 50 50 50
| The input displayed as decimal integers is:
| -50 -50 40 80 40 50 80
```

# Reading Formatted Input with `scanf`

## (Cont.)

- Example: [fig09\\_18.c](#)

```
5 int main( void ) {  
6     int a;  
7     int b;  
8     int c;  
9     int d;  
10    int e;  
11    int f;  
12    int g;  
13  
14    printf( "Enter seven integers: " );  
15    scanf( "%d%i%i%i%o%u%x", &a, &b, &c, &d, &e, &f, &g );  
16  
17    printf( "The input displayed as decimal integers is:\n" );  
18    printf( "%d %d %d %d %d %d %d\n", a, b, c, d, e, f, g );  
19    return 0; /* indicates successful termination */  
}
```

```
| Enter seven integers: -50 -50 050 0x50 50 50 50  
| The input displayed as decimal integers is:  
| -50 -50 40 80 40 50 80
```

# Reading Formatted Input with `scanf`

## (Cont.)

- Example: [fig09\\_18.c](#)

```
5 int main( void ) {
6     int a;
7     int b;
8     int c;
9     int d;
10    int e;
11    int f;
12    int g;
13
14    printf( "Enter seven integers: " );
15    scanf( "%d%i%i%i%o%u%x", &a, &b, &c, &d, &e, &f, &g );
16
17    printf( "The input displayed as decimal integers is:\n" );
18    printf( "%d %d %d %d %d %d %d\n", a, b, c, d, e, f, g );
19    return 0; /* indicates successful termination */
}
```

%i for inputting decimal, octal and hexadecimal integers

```
| Enter seven integers: -50 -50 050 0x50 50 50 50
| The input displayed as decimal integers is:
| -50 -50 40 80 40 50 80
```



# Reading Formatted Input with **scanf**

## (Cont.)

- Example: [fig09\\_19.c](#)

```
6 int main( void ) {
7     double a;
8     double b;
9     double c;
10
11     printf( "Enter three floating-point numbers: \n" );
12     scanf( "%le%lf%lg", &a, &b, &c );
13
14     printf( "Here are the numbers entered in plain\n" );
15     printf( "floating-point notation:\n" );
16     printf( "%f\n%f\n%f\n", a, b, c );
17     return 0; /* indicates successful termination */
18 }
```



# Reading Formatted Input with **scanf**

## (Cont.)

- Example: [fig09\\_19.c](#)

```
6 int main( void ) {  
7     double a;  
8     double b;  
9     double c;  
10  
11     printf( "Enter three floating-point numbers: \n" );  
12     scanf( "%le%lf%lg", &a, &b, &c );  
13  
14     printf( "Here are the numbers entered in plain\n" );  
15     printf( "floating-point notation:\n" );  
16     printf( "%f\n%f\n%f\n", a, b, c );  
17     return 0; /* indicates successful termination */  
18 }
```

# Reading Formatted Input with `scanf`

## (Cont.)

- Example: [fig09\\_19.c](#)

```
6 int main( void ) {
7     double a;
8     double b;
9     double c;
10
11     printf( "Enter three floating-point numbers: \n" );
12     scanf( "%le%lf%lg", &a, &b, &c );
13
14     printf( "Here are the numbers entered in plain\n" );
15     printf( "floating-point notation:\n" );
16     printf( "%f\n%f\n%f\n", a, b, c );
17     return 0; /* indicates successful termination */
18 }
```

enable three different  
floating-point input  
numbers

# Reading Formatted Input with `scanf`

## (Cont.)

- Example: [fig09\\_19.c](#)

```
6 int main( void ) {  
7     double a;  
8     double b;  
9     double c;  
10  
11     printf( "Enter three floating-point numbers: \n" );  
12     scanf( "%le%lf%lg", &a, &b, &c );  
13  
14     printf( "Here are the numbers entered in plain\n" );  
15     printf( "floating-point notation:\n" );  
16     printf( "%f\n%f\n%f\n", a, b, c );  
17     return 0; /* indicates successful termination */  
18 }
```

enable three different  
floating-point input  
numbers

```
Enter three floating-point numbers:  
1.2345 1.2345e+03 1.2345e-06  
Here are the numbers entered in plain  
floating-point notation:  
1.234500  
1234.500000  
0.000001
```

# Reading Formatted Input with **scanf**

## (Cont.)

- Example: [fig09\\_20.c](#)

```
5 int main( void ) {
6     char x;
7     char y[ 9 ];
8
9     printf( "Enter a string: " );
10    scanf( "%c%s", &x, y );
11
12    printf( "The input was:\n" );
13    printf( "the character \"%c\" ", x );
14    printf( "and the string \"%s\"\n", y );
15    return 0; /* indicates successful termination */
16 }
```

# Reading Formatted Input with **scanf**

## (Cont.)

- Example: [fig09\\_20.c](#)

```
5 int main( void ) {
6     char x;
7     char y[ 9 ];
8
9     printf( "Enter a string: " );
10    scanf( "%c%s", &x, y );
11
12    printf( "The input was:\n" );
13    printf( "the character \"%c\" ", x );
14    printf( "and the string \"%s\"\n", y );
15    return 0; /* indicates successful termination */
16 }
```

# Reading Formatted Input with **scanf**

## (Cont.)

- Example: [fig09\\_20.c](#)

```
5 int main( void ) {  
6     char x;  
7     char y[ 9 ];  
8  
9     printf( "Enter a string: " );  
10    scanf( "%c%s", &x, y );  
11  
12    printf( "The input was:\n" );  
13    printf( "the character \"%c\" ", x );  
14    printf( "and the string \"%s\"\n", y );  
15    return 0; /* indicates successful termination */  
16 }
```

**%c**: store a char  
**%s**: store a string

# Reading Formatted Input with `scanf`

## (Cont.)

- Example: [fig09\\_20.c](#)

```
5 int main( void ) {  
6     char x;  
7     char y[ 9 ];  
8  
9     printf( "Enter a string: " );  
10    scanf( "%c%s", &x, y );  
11  
12    printf( "The input was:\n" );  
13    printf( "the character \"%c\" ", x );  
14    printf( "and the string \"%s\"\n", y );  
15    return 0; /* indicates successful termination */  
16 }
```

`%c`: store a char  
`%s`: store a string

```
Enter a string: Hello  
The input was:  
the character "H" and the string "ello"
```



# Reading Formatted Input with **scanf**

## (Cont.)

- A sequence of characters can be input using a **scan set**.
- A scan set is a set of characters enclosed in square brackets, `[ ]`, and preceded by a percent sign in the format control string.

# Reading Formatted Input with **scanf**

## (Cont.)

- Example: [fig09\\_21.c](#)

```
6 int main( void ) {  
7     char z[ 9 ];  
8  
9     printf( "Enter string: " );  
10    scanf( "%[aeiou]", z ); /* search for set of characters */  
11  
12    printf( "The input was \"%s\"\n", z );  
13    return 0; /* indicates successful termination */  
14 }
```

# Reading Formatted Input with **scanf**

## (Cont.)

- Example: [fig09\\_21.c](#)

```
6 int main( void ) {  
7     char z[ 9 ];  
8  
9     printf( "Enter string: " );  
10    scanf( "%[aeiou]", z ); /* search for set of characters */  
11  
12    printf( "The input was \"%s\"\n", z );  
13    return 0; /* indicates successful termination */  
14 }
```

# Reading Formatted Input with **scanf**

## (Cont.)

- Example: [fig09\\_21.c](#)

```
6 int main( void ) {  
7     char z[ 9 ];  
8  
9     printf( "Enter string: " );  
10    scanf( "%[aeiou]", z ); /* search for set of characters */  
11  
12    printf( "The input was \"%s\"\n", z );  
13    return 0; /* indicates successful termination */  
14 }
```

the scan set [aeiou] to  
scan the input stream for  
vowels

# Reading Formatted Input with **scanf**

## (Cont.)

- Example: [fig09\\_21.c](#)

```
6 int main( void ) {  
7     char z[ 9 ];  
8  
9     printf( "Enter string: " );  
10    scanf( "%[aeiou]", z ); /* search for set of characters */  
11  
12    printf( "The input was \"%s\\n\"", z );  
13    return 0; /* indicates successful termination */  
14 }
```

the scan set `[aeiou]` to scan the input stream for vowels

```
Enter string: ooeeooahah  
The input was "ooeeooa"
```

# Reading Formatted Input with **scanf**

## (Cont.)

- The scan set can also be used to scan for characters not contained in the scan set by using an **inverted scan set**.
- To create an inverted scan set, place a **caret (^)** in the square brackets before the scan characters.
- When a character contained in the inverted scan set is encountered, input terminates.

# Reading Formatted Input with **scanf**

## (Cont.)

- Example: [fig09\\_22.c](#)

```
5 int main( void ) {  
6     char z[ 9 ];  
7  
8     printf( "Enter a string: " );  
9     scanf( "%[^aeiou]", z ); /* inverted scan set */  
10  
11     printf( "The input was \"%s\"\n", z );  
12     return 0; /* indicates successful termination */  
13 } /* end main */
```



# Reading Formatted Input with **scanf**

## (Cont.)

- Example: [fig09\\_22.c](#)

```
5 int main( void ) {  
6     char z[ 9 ];  
7  
8     printf( "Enter a string: " );  
9     scanf( "%[^aeiou]", z ); /* inverted scan set */  
10  
11     printf( "The input was \"%s\"\n", z );  
12     return 0; /* indicates successful termination */  
13 } /* end main */
```

# Reading Formatted Input with **scanf**

## (Cont.)

- Example: [fig09\\_22.c](#)

```
5 int main( void ) {  
6     char z[ 9 ];  
7  
8     printf( "Enter a string: " );  
9     scanf( "%[^aeiou]", z ); /* inverted scan set */  
10  
11     printf( "The input was \"%s\"\n", z );  
12     return 0; /* indicates successful termination */  
13 }
```

search for “non-vowels”

# Reading Formatted Input with **scanf**

## (Cont.)

- Example: [fig09\\_22.c](#)

```
5 int main( void ) {  
6     char z[ 9 ];  
7  
8     printf( "Enter a string: " );  
9     scanf( "%[^aeiou]", z ); /* inverted scan set */  
10  
11     printf( "The input was \"%s\\n\"", z );  
12     return 0; /* indicates successful termination */  
13 }
```

search for “non-vowels”

```
Enter a string: String  
The input was "Str"
```

# Reading Formatted Input with `scanf`

## (Cont.)

- Example: [fig09\\_23.c](#)

```
5 int main( void ) {  
6     int x;  
7     int y;  
8  
9     printf( "Enter a six digit integer: " );  
10    scanf( "%2d%d", &x, &y );  
11  
12    printf( "The integers input were %d and %d\n", x, y );  
13    return 0; /* indicates successful termination */  
14 }
```

# Reading Formatted Input with `scanf`

## (Cont.)

- Example: [fig09\\_23.c](#)

```
5 int main( void ) {  
6     int x;  
7     int y;  
8  
9     printf( "Enter a six digit integer: " );  
10    scanf( "%2d%d", &x, &y );  
11  
12    printf( "The integers input were %d and %d\n", x, y );  
13    return 0; /* indicates successful termination */  
14 }
```

# Reading Formatted Input with **scanf**

## (Cont.)

- Example: [fig09\\_23.c](#)

```
5 int main( void ) {  
6     int x;  
7     int y;  
8  
9     printf( "Enter a six digit integer: " );  
10    scanf( "%2d%d", &x, &y );  
11  
12    printf( "The integers input were %d and %d\n", x, y );  
13    return 0; /* indicates successful termination */  
14 }
```

x: a two-digit integer  
y: the remaining digits

# Reading Formatted Input with `scanf`

## (Cont.)

- Example: [fig09\\_23.c](#)

```
5 int main( void ) {  
6     int x;  
7     int y;  
8  
9     printf( "Enter a six digit integer: " );  
10    scanf( "%2d%d", &x, &y );  
11  
12    printf( "The integers input were %d and %d\n", x, y );  
13    return 0; /* indicates successful termination */  
14 }
```

x: a two-digit integer  
y: the remaining digits

```
Enter a six digit integer: 12345  
The integers input were 12 and 345
```

# Reading Formatted Input with **scanf**

## (Cont.)

- Often it's necessary to skip certain characters in the input stream.
- For example, a date could be entered as
  - 11-10-1999
- For example, to skip the dashes in the input, use the statement

```
scanf ( "%d-%d-%d" , &month , &day , &year ) ;
```



# Reading Formatted Input with **scanf**

## (Cont.)

- Example: [fig09\\_24.c](#)

```
13 printf( "Enter a date in the form mm-dd-yyyy: " );
14 scanf( "%d%c%d%c%d", &month1, &day1, &year1 );
15
16 printf( "month = %d  day = %d  year = %d\n\n", month1, day1, year1 );
17
18 printf( "Enter a date in the form mm/dd/yyyy: " );
19 scanf( "%d%c%d%c%d", &month2, &day2, &year2 );
20
21 printf( "month = %d  day = %d  year = %d\n", month2, day2, year2 );
22 return 0; /* indicates successful termination */
```

# Reading Formatted Input with **scanf**

## (Cont.)

- Example: [fig09\\_24.c](#)

```
13 printf( "Enter a date in the form mm-dd-yyyy: " );
14 scanf( "%d%c%d%c%d", &month1, &day1, &year1 );
15
16 printf( "month = %d  day = %d  year = %d\n\n", month1, day1, year1 );
17
18 printf( "Enter a date in the form mm/dd/yyyy: " );
19 scanf( "%d%c%d%c%d", &month2, &day2, &year2 );
20
21 printf( "month = %d  day = %d  year = %d\n", month2, day2, year2 );
22 return 0; /* indicates successful termination */
```

# Reading Formatted Input with **scanf**

## (Cont.)

- Example: [fig09\\_24.c](#)

```
13 printf( "Enter a date in the form mm-dd-yyyy: " );
14 scanf( "%d%c%d%c%d", &month1, &day1, &year1 );
15
16 printf( "month = %d  day = %d  year = %d\n\n", month1, day1, year1 );
17
18 printf( "Enter a date in the form mm/dd/yyyy: " );
19 scanf( "%d%c%d%c%d", &month2, &day2, &year2 );
20
21 printf( "month = %d  day = %d  year = %d\n", month2, day2, year2 );
22 return 0; /* indicates successful termination */
```

# Reading Formatted Input with **scanf**

## (Cont.)

- Example: [fig09\\_24.c](#)

```
13 printf( "Enter a date in the form mm-dd-yyyy: " );
14 scanf( "%d%c%d%c%d", &month1, &day1, &year1 );
15
16 printf( "month = %d  day = %d  year = %d\n\n", month1, day1, year1 );
17
18 printf( "Enter a date in the form mm/dd/yyyy: " );
19 scanf( "%d%c%d%c%d", &month2, &day2, &year2 );
20
21 printf( "month = %d  day = %d  year = %d\n", month2, day2, year2 );
22 return 0; /* indicates successful termination */
```

only the month, day, and  
year are stored

# Reading Formatted Input with **scanf**

## (Cont.)

- Example: [fig09\\_24.c](#)

```
13 printf( "Enter a date in the form mm-dd-yyyy: " );
14 scanf( "%d%c%d%c%d", &month1, &day1, &year1 );
15
16 printf( "month = %d  day = %d  year = %d\n\n", month1, day1, year1 );
17
18 printf( "Enter a date in the form mm/dd/yyyy: " );
19 scanf( "%d%c%d%c%d", &month2, &day2, &year2 );
20
21 printf( "month = %d  day = %d  year = %d\n", month2, day2, year2 );
22 return 0; /* indicates successful termination */
```

only the month, day, and  
year are stored

```
| Enter a date in the form mm-dd-yyyy: 10-11-2011
| month = 10  day = 11  year = 2011
|
| Enter a date in the form mm/dd/yyyy: 10/11/2011
| month = 10  day = 11  year = 2011
```