Programming for Engineers

Formatted I/O

ICEN 200 – Spring 2018

Prof. Dola Saha

Integer Conversion Specifiers

Conversion specifier	Description
`d`	Display as a *signed decimal integer*.
`i`	Display as a *signed decimal integer*. [*Note:* 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 *uppercase* letters `A-F` to be used in the display and `x` causes the digits `0-9` and the *lowercase* letters `a-f` to be used in the display.
`h`, `l` or `ll` (letter "ell")	Place *before* any integer conversion specifier to indicate that a `short`, `long` or `long long` integer is displayed, respectively. These are called **length modifiers**.

Example use of Integer Specifiers

```
1  // Fig. 9.2: fig09_02.c
2  // Using the integer conversion specifiers
3  #include <stdio.h>
4
5  int main(void)
6  {
7     printf("%d\n", 455);
8     printf("%i\n", 455); // i same as d in printf
9     printf("%d\n", +455); // plus sign does not print
10    printf("%d\n", -455); // minus sign prints
11    printf("%hd\n", 32000);
12    printf("%ld\n", 2000000000L); // L suffix makes literal a long int
13    printf("%o\n", 455); // octal
14    printf("%u\n", 455);
15    printf("%u\n", -455);
16    printf("%x\n", 455); // hexadecimal with lowercase letters
17    printf("%X\n", 455); // hexadecimal with uppercase letters
18 }
```

Example use of Integer Specifiers - Output

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

Floating point specifiers

Conversion specifier	Description
`e` or `E`	Display a floating-point value in *exponential notation*.
`f` or `F`	Display floating-point values in *fixed-point notation* (`F` is supported in the Microsoft Visual C++ compiler in Visual Studio 2015 and higher).
`g` or `G`	Display a floating-point value in either the *floating-point form* `f` or the exponential form `e` (or `E`), based on the magnitude of the value.
`L`	Place before any floating-point conversion specifier to indicate that a `long double` floating-point value should be displayed.

Example use of Floating Point Specifiers

```
1  // Fig. 9.4: fig09_04.c
2  // Using the floating-point conversion specifiers
3  #include <stdio.h>
4
5  int main(void)
6  {
7     printf("%e\n", 1234567.89);
8     printf("%e\n", +1234567.89); // plus does not print
9     printf("%e\n", -1234567.89); // minus prints
10    printf("%E\n", 1234567.89);
11    printf("%f\n", 1234567.89); // six digits to right of decimal point
12    printf("%g\n", 1234567.89); // prints with lowercase e
13    printf("%G\n", 1234567.89); // prints with uppercase E
14 }
```

```
1.234568e+006
1.234568e+006
-1.234568e+006
1.234568E+006
1234567.890000
1.23457e+006
1.23457E+006
```

Example use of %c and %s

```
1  // Fig. 9.5: fig09_05c
2  // Using the character and string conversion specifiers
3  #include <stdio.h>
4
5  int main(void)
6  {
7     char character = 'A'; // initialize char
8     printf("%c\n", character);
9
10    printf("%s\n", "This is a string");
11
12    char string[] = "This is a string"; // initialize char array
13    printf("%s\n", string);
14
15    const char *stringPtr = "This is also a string"; // char pointer
16    printf("%s\n", stringPtr);
17 }
```

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

Other Specifiers

Conversion specifier	Description
p	Display a pointer value in an implementation-defined manner.
%	Display the percent character.

Example of Other Specifiers

```
1  // Fig. 9.7: fig09_07.c
2  // Using the p and % conversion specifiers
3  #include <stdio.h>
4
5  int main(void)
6  {
7     int x = 12345; // initialize int x
8     int *ptr = &x; // assign address of x to ptr
9
10    printf("The value of ptr is %p\n", ptr);
11    printf("The address of x is %p\n\n", &x);
12
13    printf("Printing a %% in a format control string\n");
14 }
```

```
The value of ptr is 002EF778
The address of x is 002EF778

Printing a % in a format control string
```

Right Justifying Integers

```
1   // Fig. 9.8: fig09_08.c
2   // Right justifying integers in a field
3   #include <stdio.h>
4
5   int main(void)
6   {
7      printf("%4d\n", 1);
8      printf("%4d\n", 12);
9      printf("%4d\n", 123);
10     printf("%4d\n", 1234);
11     printf("%4d\n\n", 12345);
12
13     printf("%4d\n", -1);
14     printf("%4d\n", -12);
15     printf("%4d\n", -123);
16     printf("%4d\n", -1234);
17     printf("%4d\n", -12345);
18  }
```

Right Justifying Integers - Output

```
    1
   12
  123
 1234
12345

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

Printing with Field Widths & Precisions

```
1  // Fig. 9.9: fig09_09.c
2  // Printing integers, floating-point numbers and strings with precisions
3  #include <stdio.h>
4
5  int main(void)
6  {
7     puts("Using precision for integers");
8     int i = 873; // initialize int i
9     printf("\t%.4d\n\t%.9d\n\n", i, i);
10
11    puts("Using precision for floating-point numbers");
12    double f = 123.94536; // initialize double f
13    printf("\t%.3f\n\t%.3e\n\t%.3g\n\n", f, f, f);
14
15    puts("Using precision for strings");
16    char s[] = "Happy Birthday"; // initialize char array s
17    printf("\t%.11s\n", s);
18 }
```

Printing with Field Widths & Precisions -Output

```
Using precision for integers
        0873
        000000873

Using precision for floating-point numbers
        123.945
        1.239e+002
        124

Using precision for strings
        Happy Birth
```

Format Control String Flags

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

Right and Left Justifying Values

```
1  // Fig. 9.11: fig09_11.c
2  // Right justifying and left justifying values
3  #include <stdio.h>
4
5  int main(void)
6  {
7     puts("12345678901234567890123456789012345678901234567890\n");
8     printf("%10s%10d%10c%10f\n\n", "hello", 7, 'a', 1.23);
9     printf("%-10s%-10d%-10c%-10f\n", "hello", 7, 'a', 1.23);
10 }
```

```
12345678901234567890123456789012345678901234567890

     hello         7         a  1.230000

hello     7         a         1.230000
```

Example of + flag

```
1  // Fig. 9.12: fig09_12.c
2  // Printing positive and negative numbers with and without the + flag
3  #include <stdio.h>
4
5  int main(void)
6  {
7     printf("%d\n%d\n", 786, -786);
8     printf("%+d\n%+d\n", 786, -786);
9  }
```

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

Using Space Flag

```
1  // Fig. 9.13: fig09_13.c
2  // Using the space flag
3  // not preceded by + or -
4  #include <stdio.h>
5
6  int main(void)
7  {
8     printf("% d\n% d\n", 547, -547);
9  }
```

```
 547
-547
```

Using # Flag

```
1  // Fig. 9.14: fig09_14.c
2  // Using the # flag with conversion specifiers
3  // o, x, X and any floating-point specifier
4  #include <stdio.h>
5
6  int main(void)
7  {
8     int c = 1427; // initialize c
9     printf("%#o\n", c);
10    printf("%#x\n", c);
11    printf("%#X\n", c);
12
13    double p = 1427.0; // initialize p
14    printf("\n%g\n", p);
15    printf("%#g\n", p);
16 }
```

```
02623
0x593
0X593

1427
1427.00
```

Using 0 Flag

```
1  // Fig. 9.15: fig09_15.c
2  // Using the 0 (zero) flag
3  #include <stdio.h>
4
5  int main(void)
6  {
7     printf("%+09d\n", 452);
8     printf("%09d\n", 452);
9  }
```

```
+00000452
000000452
```

Escape Sequence

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 (typically, flashing the window in which the program is running).
`\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.

Conversion Specifiers for scanf

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

Conversion Specifiers for scanf

Conversion specifier	Description
Floating-point numbers	
`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 `double` or `long double` value is to be input. The corresponding argument is a pointer to a `double` or `long double` variable.
Characters and strings	
`c`	Read a *character*. The corresponding argument is a pointer to a `char`; no null (`'\0'`) is added.
`s`	Read a *string*. The corresponding argument is a pointer to an array of type `char` that's large enough to hold the string and a terminating null (`'\0'`) character—which is automatically added.
Scan set	
[*scan characters*]	Scan a string for a set of characters that are stored in an array.

Conversion Specifiers for scanf

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

Reading Integer Input

```
1  // Fig. 9.18: fig09_18.c
2  // Reading input with integer conversion specifiers
3  #include <stdio.h>
4
5  int main(void)
6  {
7     int a;
8     int b;
9     int c;
10    int d;
11    int e;
12    int f;
13    int g;
14
15    puts("Enter seven integers: ");
16    scanf("%d%i%i%i%o%u%x", &a, &b, &c, &d, &e, &f, &g);
17
18    puts("\nThe input displayed as decimal integers is:");
19    printf("%d %d %d %d %d %d %d\n", a, b, c, d, e, f, g);
20 }
```

Reading Integer Input

```
Enter seven integers:
-70 -70 070 0x70 70 70 70

The input displayed as decimal integers is:
-70 -70 56 112 56 70 112
```

Reading Floating Point Input

```
1  // Fig. 9.19: fig09_19.c
2  // Reading input with floating-point conversion specifiers
3  #include <stdio.h>
4
5  // function main begins program execution
6  int main(void)
7  {
8     double a;
9     double b;
10    double c;
11
12    puts("Enter three floating-point numbers:");
13    scanf("%le%lf%lg", &a, &b, &c);
14
15    printf("\nHere are the numbers entered in plain:");
16    puts("floating-point notation:\n");
17    printf("%f\n%f\n%f\n", a, b, c);
18 }
```

Reading Floating Point Input

```
Enter three floating-point numbers:
1.27987 1.27987e+03 3.38476e-06

Here are the numbers entered in plain floating-point notation:
1.279870
1279.870000
0.000003
```

Reading Characters and Strings

```
1  // Fig. 9.20: fig09_20.c
2  // Reading characters and strings
3  #include <stdio.h>
4
5  int main(void)
6  {
7     char x;
8     char y[9];
9
10    printf("%s", "Enter a string: ");
11    scanf("%c%8s", &x, y);
12
13    puts("The input was:\n");
14    printf("the character \"%c\" and the string \"%s\"\n", x, y);
15 }
```

```
Enter a string: Sunday
The input was:
the character "S" and the string "unday"
```

Using a Scan Set

```
1  // Fig. 9.21: fig09_21.c
2  // Using a scan set
3  #include <stdio.h>
4
5  // function main begins program execution
6  int main(void)
7  {
8     char z[9]; // define array z
9
10    printf("%s", "Enter string: ");
11    scanf("%8[aeiou]", z); // search for set of characters
12
13    printf("The input was \"%s\"\n", z);
14 }
```

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

Using Inverted Scan Set

```c
// Fig. 9.22: fig09_22.c
// Using an inverted scan set
#include <stdio.h>

int main(void)
{
   char z[9];

   printf("%s", "Enter a string: ");
   scanf("%8[^aeiou]", z); // inverted scan set

   printf("The input was \"%s\"\n", z);
}
```

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

Input data with field width

```
1   // Fig. 9.23: fig09_23.c
2   // inputting data with a field width
3   #include <stdio.h>
4
5   int main(void)
6   {
7      int x;
8      int y;
9
10     printf("%s", "Enter a six digit integer: ");
11     scanf("%2d%d", &x, &y);
12
13     printf("The integers input were %d and %d\n", x, y);
14  }
```

```
Enter a six digit integer: 123456
The integers input were 12 and 3456
```

Reading and Discarding Characters

```
1  // Fig. 9.24: fig09_24.c
2  // Reading and discarding characters from the input stream
3  #include <stdio.h>
4
5  int main(void)
6  {
7     int month = 0;
8     int day = 0;
9     int year = 0;
10    printf("%s", "Enter a date in the form mm-dd-yyyy: ");
11    scanf("%d%*c%d%*c%d", &month, &day, &year);
12    printf("month = %d  day = %d  year = %d\n\n", month, day, year);
13
14    printf("%s", "Enter a date in the form mm/dd/yyyy: ");
15    scanf("%d%*c%d%*c%d", &month, &day, &year);
16    printf("month = %d  day = %d  year = %d\n", month, day, year);
17 }
```

```
Enter a date in the form mm-dd-yyyy: 11-18-2012
month = 11  day = 18  year = 2012

Enter a date in the form mm/dd/yyyy: 11/18/2012
month = 11  day = 18  year = 2012
```