Chapter 9: Formatted I/O
Formatting Output with printf

Precisely formatted output is accomplished using the output function `printf`. The `printf` function has following form

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

- **Format-control-string**: Using specifications to describe output format. Each specification begins with a percent sign (%), ends with conversion specifier and is enclosed in quotation marks.
- **arguments**: correspond to each conversion specification in format-control-string.
Example:

```c
> int i = 5
> float f = 1.234f
> double d = 123.4567
> printf("i = %d
", i)
  i = 5
> printf("f = %f
", f)
  f = 1.234000
> printf("d = %lf
", d)
  d = 123.456700
> printf("i = %d, f = %f, d = %lf
", i, f, d)
  i = 5, f = 1.234000, d = 123.456700
```

- Conversion specifications: flags, field widths, precisions, etc.
- The function `printf` can perform rounding, aligning columns, right/left justification, inserting literal characters, fixed width, etc.
## Printing Integers

Table below lists the integer conversion specifiers and gives short description for each conversion specifier.

### Table  Conversion specifiers for integer

<table>
<thead>
<tr>
<th>Conversion Specifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>Display a binary number (for Ch only)</td>
</tr>
<tr>
<td>c</td>
<td>Display a character.</td>
</tr>
<tr>
<td>i</td>
<td>Display a signed decimal integer (exactly the same as d for printf(), diff for scanf(), it can also handle octal and hexadecimal numbers)</td>
</tr>
<tr>
<td>b</td>
<td>Display an unsigned binary integer.</td>
</tr>
<tr>
<td>o</td>
<td>Display an unsigned octal integer.</td>
</tr>
<tr>
<td>x or X</td>
<td>Display an unsigned hexadecimal integer.</td>
</tr>
<tr>
<td>h or l or ll</td>
<td>Place before any integer conversion identifier to indicate <strong>short</strong>, <strong>long</strong>, or <strong>long long</strong> integer.</td>
</tr>
<tr>
<td>u</td>
<td>Display an unsigned decimal integer. Such as %u, %hu, %lu, %llu.</td>
</tr>
</tbody>
</table>
Example 1:

```c
/* File: intio.c */
#include <stdio.h>

int main() {
    short i1 = 0x20;
    int i2 = 0x20000000;
    long long i3 = 0x20000000;
    printf("%c\n", 98);
    printf("%d\n", 98);
    printf("%d\n", -98);
    printf("%i\n", 98);
    printf("%o\n", 98);
    printf("%x\n", 98);
    printf("%X\n", 98);
    printf("%hd\n", i1);
    printf("%ld\n", i2);
    printf("%d\n", i2);
    printf("%lld\n", i3);
    return 0;
}
```

Output:

```
98
98
-98
98
0142
0x62
32
536870912
536870912
536870912
```
## Escape Characters

<table>
<thead>
<tr>
<th>Escape Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\a</td>
<td>(alert) Produces an audible or visible alert. The active position shall not be changed.</td>
</tr>
<tr>
<td>\b</td>
<td>(backspace) Moves the active position to the previous position on the current line.</td>
</tr>
<tr>
<td>\f</td>
<td>(form feed) Moves the active position to the initial position at the start of the next logical page.</td>
</tr>
<tr>
<td>\n</td>
<td>(new line) Moves the active position to the initial position of the next line.</td>
</tr>
<tr>
<td>\r</td>
<td>(carriage return) Moves the active position to the initial position of the current line.</td>
</tr>
<tr>
<td>\t</td>
<td>(horizontal tab) Moves the active position to the next horizontal tabulation position on the current line.</td>
</tr>
<tr>
<td>\v</td>
<td>(vertical tab) Moves the active position to the initial position of the next vertical tabulation position.</td>
</tr>
</tbody>
</table>
| "\"        | (backslash) Produces a backslash character \\.
| \'          | (single quote) Produces a single quote character ‘. |
| \"          | (double quote) Produces a double quote character “. |
| \?          | (question mark) Produces a question mark character ?.

Created by Harry H. Cheng, © 2009 McGraw-Hill, Inc. All rights reserved.
Example:

```c
> printf("abcd\tefd");  // horizontal tab
abcd    efd
> printf("abcd"efd");  // double quote
abcd"efd
> printf("%c", '\');  // single quote
\n> printf("?\?!");    // question mark
??!
> 
```

What gets printed if we omit the backslash? Hint: Trigraphs! (p.46)
Printing Floating-Point Numbers

The floating-point conversion specifiers are listed in Table

<table>
<thead>
<tr>
<th>Conversion Specifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>f</td>
<td>Display floating-point values.</td>
</tr>
<tr>
<td>lf</td>
<td>Was introduced in C99. It prints doubles (long float)</td>
</tr>
<tr>
<td>e or E</td>
<td>Display a floating-point value in an exponential form.</td>
</tr>
<tr>
<td>g or G</td>
<td>Display a floating-point value in either the floating-point form f or the exponential form e (or E).</td>
</tr>
</tbody>
</table>

Table Conversion specifiers for floating-point numbers
Example:

/* File: doubleio.c */
#include <stdio.h>

int main() {
    float f = 10.123
    double d = 12.345;

    printf("%f\n", f);
    printf("%f\n", d);
    printf("%lf\n", d);
    printf("%e\n", 12.345);
    printf("%e\n", -12.345);
    printf("%E\n", 12.345);
    printf("%g\n", 12.345);
    printf("%G\n", 12.345);
    printf("%g\n", 12.0);
    printf("%G\n", 12.0);
    printf("%g\n", 12345678.9);
    printf("%G\n", 12345678.9);

    return 0;
}

Output:

10.123000
12.345000
12.345000
1.234500e+001
-1.234500e+001
1.234500E+001
12.345
12.345
12
12
1.23457e+007
1.23457E+007
Printing Strings, Characters and Pointers

Conversion specifier **c** causes one character to be printed.
Conversion specifier **s** causes several characters to be printed until a terminating null (‘\0’) character is encountered.
Conversion specifier **p** displays a pointer value in an implementation-defined manner.
Example:

```c
/* File: strio.c */
#include <stdio.h>

int main() {
    char string[] = "String 2 is printed by an array of characters."
    char *strPtr = "String 3 is printed by a pointer of char."
    int i = 98;
    int *ptr;

    printf("%s\n", "String 1 is printed directly.");
    printf("%s\n", string);
    printf("%s\n", strPtr);

    ptr = &i;
    printf("The value of i is %d.\n", i);
    printf("The address of i is %p.\n", ptr);

    return 0;
}
```
Output:

String 1 is printed directly.
String 2 is printed by an array of characters.
String 3 is printed by a pointer of char.
The value of i is 98.
The address of i is 0063FDC0.
Field Widths and Precisions

• Field width
  – Field width is the size of field in which data is printed.
  – The field width takes the form of an asterisk or a decimal integer.
  – If the converted value has fewer characters than the field width, it is padded with space on the left (right-justified) or on the right (left-justified) to the field width. If field width is too small, increases it to fit data.
  – Integer width inserted between % and conversion specifier.
  – %4d indicates field width of 4.
**Example:**

```c
/* File: field.c */
#include <stdio.h>

int main() {
    int i = 12345;
    printf("%d\n", i);
    printf("%8d\n", i);
    printf("%4d\n", i);

    return 0;
}
```

**Output:**

```
12345
  12345
  12345
  12345
```
• Precision

  – Meaning varies depending on data type
    • For Integers, gives the minimum number of digits to print
    • For floating-point numbers, specifies the number of digits to appear after the decimal-point character (e, E and f).
    • For g and G conversions, specifies the maximum number of significant digits.
    • For string conversions, specifies the maximum number of characters to be written from string.
  – The precision takes the form of a period (.) followed either by an asterisk or by an optional decimal integer.
  – Precision can be combined with field width, such as %8.5f.
/* File: precision.c */
#include <stdio.h>

int main() {
    int i = 5678;
    double d = 123.45678;
    char *str = "Hello, students."
;

    printf("Precision for integers." 
);
    printf("%.3d
", i);
    printf("%.6d
", i);

    printf("\nPrecision for double." 
);
    printf("%5.3f
", d);
    printf("%7.3f
", d);
    printf("%9.3f
", d);
    printf("%.3e
", d);

    printf("\nPrecision for g and G t." 
);
    printf("%.4g
", d);

    printf("\nPrecision for strings." 
);
    printf("%s
", str);
    printf("%25s
", str);

    return 0;
}
Read and take notes:

Using asterisks in the width specifier (p.279)
## Flags

Function **printf** also provides flags to supplement its output formatting capabilities. The flag characters and their meanings are:

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>- (minus sign)</td>
<td>The result of the conversion is left-justified within the field.</td>
</tr>
<tr>
<td>+ (plus sign)</td>
<td>The result of a signed conversion always begins with a plus or minus sign. Display a plus sign preceding positive values and a minus sign preceding negative values.</td>
</tr>
<tr>
<td>space</td>
<td>Print a space before a positive value not printed with the + flag.</td>
</tr>
<tr>
<td>#</td>
<td>The result is converted to an “alternative form”. For o conversion, prefix 0 to the output value; For 0x (or 0X) conversion, prefix 0x (or 0X) to a nonzero result. For e, E, f, g, or G conversions, the result of converting a floating-point number always contains a decimal-point character, even if no digits follow it.</td>
</tr>
<tr>
<td>0 (zero)</td>
<td>Leading zeros are used to pad the field width rather than perform space padding.</td>
</tr>
</tbody>
</table>
Example:

```c
/* File: flag.c */
#include <stdio.h>

int main() {
    double d = 123.456;
    int i = 98;

    printf("\nUsing minus sign flag.\n");
    printf("%15f\n", d);
    printf("%-15f\n", d);

    printf("\nUsing plus sign flag.\n");
    printf("%f\n%f\n", d, -d);
    printf("%+f\n%+f\n", d, -d);

    printf("\nUsing # flag.\n");
    printf("%#o\n", i);
    printf("%#x\n", i);
    printf("%#X\n", i);
    printf("%n\%g\n", 12.0);
    printf("%n\%g\n", 12.0);

    printf("\nUsing 0 flag.\n");
    printf("%05d\n", i);
    return 0;
}
```

Output:

Using minus sign flag.
123.456000
123.456000

Using plus sign flag.
123.456000
-123.456000
+123.456000
-123.456000

Using # flag.
0142
0x62
0X62

12
12.0000

Using 0 flag.
00098

Trailing zeroes not removed!
9.2 Formatting Input with scanf

Precise formatting input is accomplished using the input function `scanf`. The `scanf` function has following form

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

- **Format-control-string**: Using specifications to describe input format. Each specification begins with a percent sign (%), ends with conversion specifier and is enclosed in quotation marks. The format-control-string is similar to format-control-string discussed in `printf` function.
- **arguments**: pointers to variables in which the input value will be stored.
Format specifier “%lf” for double

Format specifier “%hd” or “%hi” for short

Format specifier “%d” for decimal integer only

Format specifier “%i” for decimal, octal, hexadecimal

```c
> double d
> scanf(“%lf”, &d)
10
> d
10.0000
> scanf(“%f”, &d)
10
> d
0.0000
> short s
> scanf(“%hd”, &s)
18
> s
18
> scanf(“%d”, &s)
18
> s
0
```
Example 1:

```c
> int i
> float f
> double d
> scanf("%d", &i)
10
> i
10
> scanf("%f", &f)
10
> f
10.00
> scanf("%lf", &d)
10
> d
10.0000
> 
```
/* File: scanfstr.c */
#include <stdio.h>

int main() {
    char c, str[50];

    printf("Please enter a character: ");
    scanf("%c", &c);
    printf("The input character was: %c\n", c);

    printf("Please enter a string: ");
    scanf("%s", str);
    printf("The input string was: %s\n", str);

    return 0;
}

Execution:

Please enter a character: g
The input character was: g
Please enter a string: Hello!
The input string was: Hello!
Useful in practice: Assignment suppression character ‘*’

```c
/* File: scanfchar.c */
#include <stdio.h>

int main() {
    char c, c2;

    printf("Please enter a character: ");
    scanf("%c%c", &c, &c2);
    printf("The input character is: %c\n", c);
    printf("Please enter a character: ");
    scanf("%c", &c);
    printf("The input character is: %c\n", c);
    printf("The ASCII value of the second character is: %d\n", c2);

    return 0;
}
```

> scanfchar.c
Please enter a character: a
The input character is: a
Please enter a character: b
The input character is: b
The ASCII value of the second character is: 10
SKIP 9.3 Function `getnum()`

(Not standard!)
To do for next time
(not homework, do not turn in for grading)

End of chapter problems:

• Pencil-and-paper: 1, 3, 4
• Programming: 2, 5