Chapter 9

Formatted Input/Output

Chapter 9 - Formatted Input/Output

Outline

9.1 Introduction

- In this chapter, we will study:
  - Formatted presentation of results in more detail
  - **scanf and printf**

- Streams (input and output)
  - **gets, puts, getchar, putchar**
  - Already seen in Chapter8

- All defined in <stdio.h>

9.2 Streams

- Streams
  - Sequences of characters organized into lines
    - Each line consists of zero or more characters and ends with newline character ‘\n’
    - ANSI C supports lines of at least 254 characters
  - Performs all input and output
  - Can often be redirected
    - Standard input – KEYBOARD
    - Standard output – SCREEN
    - Standard error – SCREEN
    - More in Chapter 11

Printf()

9.3 Formatting Output with `printf`

- `printf`
  - Precise output formatting
    - Conversion specifications: Flags, Field widths, Precisions, etc.
  - Can perform rounding, aligning columns, right/left justification, inserting literal characters, exponential format, hexadecimal format, and fixed width and precision

- Format
  - `printf(` ***format-control-string, other-arguments*** `);`
  - Format control string: describes output format
  - Other-arguments: variables correspond to each conversion specification in format-control-string
    - Each specification begins with a percent sign(%), ends with conversion specifier

9.4 Printing Integers

Conversion Specifier	Description
`d`	Display a signed decimal integer.
`i`	Display a signed decimal integer. In printf, %d and %i are the same. In scanf, %d and %i are different.
`o`	Display an unsigned octal integer.
`u`	Display an unsigned decimal integer.
`x` or `X`	Display 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`	Place before any integer conversion specifier to indicate that a `short` integer is displayed. This is called *length modifier*.
`l`	`long` integer is displayed.

Fig. 9.1 Integer conversion specifiers.

9.4 Printing Integers

- Integer
  - Whole number (no decimal point): `25`, `0`, `-9`
  - Positive, negative, or zero
  - Only minus sign prints by default
    - Use **`%+d`** format specifier for displaying the sign every time

```
int a = 20, b = -30;
printf("%+d  %+d \n", a, b);

OUTPUT:
+20  -30
```

Example : Printing integers

```c
/* Fig 9.2: fig09_02.c */
/* Using the integer conversion specifiers */
#include <stdio.h>

int main()
{
   printf( "%d\n", 455 );
   printf( "%i\n", 455 );
   // i same as d in printf

   printf( "%d\n", +455 );
   printf( "%d\n", -455 );
   printf( "%hd\n", 32000 );
   printf( "%ld\n", 2000000000 );
   printf( "%o\n", 455 );
   printf( "%u\n", 455 );
   printf( "%u\n", -455 );
   printf( "%x\n", 455 );
   printf( "%X\n", 455 );
}
```

Program Output

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

9.5 Printing Floating-Point Numbers

- Floating Point Numbers
  - Have a decimal point (`33.5`)
  - Used to print float or double values
  - `f` prints floating point
    - **Fractions are `6` digits by default**
    - At least one digit to left of point is displayed
  - e or E prints in exponential notation
  - Exponential notation is the C's scientific notation
    - `150.3` is `1.503 x 10²` in scientific
    - `150.3` is `1.503E+02` in exponential
  - `g` (or `G`) prints like in `f` or `e`, but with no trailing zeros (`1.2300` becomes `1.23`)

9.5 Printing Floating-Point Numbers

Conversion specifier	Description
`f`	Display floating-point values.
`e` or `E`	Display a floating-point value in exponential notation.
`g` or `G`	Display a floating-point value in the floating-point form `f` when value is float, OR in the exponential form `e` when value is double.
`l` or `L`	Place before any floating-point conversion specifier to indicate that a `long double` floating-point value is displayed.

Fig. 9.3 Floating-point conversion specifiers.

Example : Printing floats


```c
/* Fig 9.4: fig09_04.c */
/* Printing floating-point numbers with
   floating-point conversion specifiers */

#include <stdio.h>
int main()
{
   printf( "%e\n", 1234567.89 );
   printf( "%e\n", +1234567.89 );
   printf( "%e\n", -1234567.89 );
   printf( "%E\n", 1234567.89 );
   printf( "%f\n", 1234567.89 );
   printf( "%g\n", 1234567.89 );
   printf( "%G\n", 1234567.89 );
}
```

Program Output

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

9.6 Printing Strings and Characters

- `%c`
  - Prints a `char` argument

- `%s`
  - Requires a **pointer to `char`** as an argument
  - Prints a string of characters until `NULL` (`'\0'`) encountered
  - Cannot print a single **`char`** argument

- Remember
  - Use single quotes for character constants (`'z'`)
  - Use double quotes for strings `"z"`
    (which actually contains two characters, `'z'` and `'\0'`)

Example : Printing with %c and %s

```c
// Fig. 9.5: fig09_05c
// Using the character and string conversion specifiers
#include <stdio.h>

int main()
{
   char character = 'A'; // initialize char
   char string[] = "This is a string"; // initialize char array
   const char *stringPtr = "This is also a string"; // char pointer

   printf( "%c\n", character );
   printf( "%s\n", "This is a string" );
   printf( "%s\n", string );
   printf( "%s\n", stringPtr );
}
```

Program Output

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

9.8 Printing with Field Widths and Precisions

- Field width
  - Size of field in which data is printed
  - If width larger than data, default right justified
    - **If field width too small, increases to fit data**
    - Minus sign uses one character position in field
  - Integer width inserted between % and conversion specifier
  - **%4d means field width of 4 (right-justified)**

Example : Printing integers right-justified

```c
// Fig. 9.8: fig09_08.c
// Right justifying integers in a field
#include <stdio.h>

int main()
{
   printf( "%4d\n", 1 );
   printf( "%4d\n", 12 );
   printf( "%4d\n", 123 );
   printf( "%4d\n", 1234 );
   printf( "%4d\n\n", 12345 );

   printf( "%4d\n", -1 );
   printf( "%4d\n", -12 );
   printf( "%4d\n", -123 );
   printf( "%4d\n", -1234 );
   printf( "%4d\n", -12345 );
} // end main
```

Program Output

```
   1
  12
 123
1234
12345

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

9.8 Printing with Field Widths and Precisions

- Precision
  - Format
    - Use a dot (`.`) then precision number after %

      `Example: %.4f` **(Fraction part will be displayed as 4 digits)**

  - Meaning varies depending on data type
  - **Integers** (default `1`)
    - Minimum number of digits to print
      - **If data length smaller than precision, leading zeros are printed**
  - **Floating point**
    - For `e` and `f` : Number of digits to appear after decimal point
    - For `g` : Maximum number of significant digits
  - **Strings**
    - Maximum number of characters to be written from string

9.8 Printing with Field Widths and Precisions

- Field Width and Precision
  - Can both be specified such as `%width.precision`

```
printf( "%10.3f \n", 123.5269);
```

OUTPUT: `  123.527` **(Whole width is 10)**

  - Negative Width means print left justified
  - Positive Width means print right justified
  - Precision must be always positive

Example : Precision Printing

```c
// Fig. 9.9: fig09_09.c
// Printing integers, floating-point numbers and strings with precisions
#include <stdio.h>

int main()
{
   int i = 873; // initialize int i
   double f = 123.94536; // initialize double f
   char s[] = "Happy Birthday"; // initialize char array s

   puts( "Using precision for integers" );
   printf( "\t%.4d\n\t%.9d\n\n", i, i );

   puts( "Using precision for floating-point numbers" );
   printf( "\t%.3f\n\t%.3e\n\t%.3g\n\n", f, f, f );

   puts( "Using precision for strings" );
   printf( "\t%.11s\n", s );
} // end main
```

Program 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
```

"day" not displayed!

9.9 Using Flags in the `printf` Format-Control String

- Flags
  - Supplement formatting capabilities
  - Place flag immediately to the right of percent sign %
  - Several flags may be combined

9.9 Using Flags in the `printf` Format-Control String

Flag	Description
– (minus sign)	Left-justify the output within the specified field.
+ (plus sign)	• Print a plus sign preceding positive values. • Print 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.
0 (zero)	Pad a field with leading zeros.

Fig. 9.10 Format control string flags.

Example : Right and left justifying

```
// Fig 9.11: fig09_11.c
// Right justifying and left justifying values
#include <stdio.h>

int main()
{
   printf( "%10s%10d%10c%10f\n\n", "hello", 7, 'a', 1.23 );
   printf( "%-10s%-10d%-10c%-10f\n", "hello", 7, 'a', 1.23 );
}
```

Program Output

```
     hello         7         a  1.230000

hello     7         a         1.230000
```

Example : Printing with the sign (+) flag

```c
// Fig. 9.12: fig09_12.c
/* Printing positive and negative numbers
with and without the + flag */

#include <stdio.h>

int main()
{
   printf( "%d\n%d\n", 786, -786 );
   printf( "%+d\n%+d\n", 786, -786 );
}
```

Program Output

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

Example : Printing leading zeros

```
// Fig. 9.15: fig09_15.c
// Using the 0 (zero) flag
#include <stdio.h>

int main()
{
   printf( "%+09d\n", 452 );
   printf( "%09d\n", 452 );
}
```

Program Output

```
+00000452
000000452
```

9.10 Printing Escape Sequences

- Escape characters
  - Most characters can be printed
  - Certain "problem" characters, such as the quotation mark "
  - Must be represented by escape sequences
    - Represented by a backslash \ followed by an escape character

9.10 Printing Escape Sequences

Escape sequence	Description
`\'`	Output the single quote (') character.
`\"`	Output the double quote (") character.
`\\`	Output the backslash (\) character.
`\a`	Alert: Cause an audible bell or visual alert.
`\b`	Backspace: Move the cursor back one position on the current line.
`\f`	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.

Fig. 9.16 Escape sequences.

Scanf()

9.11 Formatting Input with `scanf`

- `scanf`
  - Input formatting
  - Capabilities
    - Input all types of data
    - Input specific characters
    - Skip specific characters

- Format
  - **`scanf`(*format-control-string*, *other-arguments*);**
  - Format-control-string
    - Describes formats of inputs
  - Other-arguments
    - Pointers to variables where input will be stored
  - Can include field widths to read a specific number of characters from the stream

9.11 Formatting Input with scanf (Integers)

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

Fig. 9.17 Conversion specifiers for scanf.

(i is superset of d, o, x)

Example: Difference Between %d and %i in Scanf

```c
#include <stdio.h>

int main() {
  int  a, b;
  printf("Enter two integers : ");
  scanf("%d  %i", &a, &b);

  printf("a = %d  b = %i\n", a, b);
}
```

Program Output

```
Enter two integers : 500   0x1f4
a = 500  b = 500
```

Example : Inputting with a field with

```c
// Fig. 9.23: fig09_23.c
// inputting data with a field width
#include <stdio.h>

int main()
{
   int x;
   int y;

   printf( "%s", "Enter a six digit integer: " );
   scanf( "%2d%d", &x, &y );

   printf( "The integers input were %d and %d\n", x, y );
}
```

Program Output

```
Enter a six digit integer: 123456

The integers input were 12 and 3456
```

Example : Reading integers

```
// Fig. 9.18: fig09_18.c
// Reading input with integer conversion specifiers
#include <stdio.h>

int main() {
   int a,b,c,d,e,f,g;

   puts( "Enter seven integers: " );
   scanf( "%d%i%i%i%o%u%x", &a, &b, &c, &d, &e, &f, &g );

   puts( "\nThe input displayed as decimal integers is:" );
   printf( "%d %d %d %d %d %d %d\n", a, b, c, d, e, f, g );
}
```

Program Output

```
Enter seven integers: -70  -70  o70  0x70  70  70  70

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

Example: Conversion of Octal to Decimal and Hexadecimal to Decimal

$$(70)_8 = (56)_{10}$$

$$(70)_{16} = (112)_{10}$$

9.11 Formatting Input with scanf (Floating-point numbers)

Conversion specifier	Description
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.

Fig. 9.17 Conversion specifiers for scanf.

Example : Reading floats

```c
// Fig. 9.19: fig09_19.c
// Reading input with floating-point conversion specifiers
#include <stdio.h>

int main()
{
   double a,b,c;

   puts( "Enter three floating-point numbers:" );
   scanf( "%le%lf%lg", &a, &b, &c );

   puts( "\nHere are the numbers entered in plain" );
   puts( "floating-point notation:" );
   printf( "%f\n%f\n%f\n", a, b, c );
}
```

Program Output

```
Enter three floating-point numbers:  2.5    3.5    4.5

Here are the numbers entered in plain floating-point notation:

2.500000
3.500000
4.500000
```

9.11 Formatting Input with `scanf` (`Char` and `Strings`)

Conversion specifier	Description
`c`	Read a character. The corresponding argument is a pointer to `char`, no null `('\0')` is added.
`s`	Read a string. The corresponding argument is a pointer to an array of type `char` that is large enough to hold the string and a terminating null `('\0')` character—which is automatically added.
[scan set characters]	Scan a string for a set of characters that are stored in an array.

Fig. 9.17 Conversion specifiers for scanf.

Example : Reading chars and strings

```c
// Fig. 9.20: fig09_20.c
// Reading characters and strings
#include <stdio.h>

int main()
{
   char x;
   char y[ 9 ];

   printf( "%s", "Enter a string: " );
   scanf( "%c%8s", &x, y );

   puts( "The input was:" );
   printf( "the character \"%c\" and the string \"%s\"\n", x, y );
}
```

Program Output

```
Enter a string: Sunday

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

9.11 Formatting Input with `scanf`

- Scan Sets (Filters):
  - Set of characters enclosed in square brackets `[]`
    - Preceded by % sign
    - Example set: `[aeiou]`
    - Scans input stream, looking only for characters in scan set
    - Whenever a match occurs, stores character in specified array
    - Stops scanning once a character not in the scan set is found

9.11 Formatting Input with `scanf`

- **Inverted Scan Sets:**
  - Use a caret ^: `[^aeiou]`
  - Causes characters **not** in the scan set to be stored
  - Stops scanning once a character in the scan set is found

- **Skipping Characters:**
  - Include character to skip in format control
  - Or, use * (assignment suppression character)
    - Skips specified type of character without storing it

Example : Scanning with a scan set

```
// Fig. 9.21: fig09_21.c
// Using a scan set
#include <stdio.h>

int main()
{
   char z[ 9 ]; // define array z

   printf( "%s", "Enter string: " );
   scanf( "%8[aeiou]", z ); // search for set of characters

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

Program Output

```
Enter string: airport
The input was "ai"

Enter string: Airport
The input was ""
```

NOTICE: In first example, scanf stops at letter “r”.
In second example, scanf stops at letter “A”.

Example : Scanning with an inverted scan set

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

int main()
{
   char z[ 9 ];

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

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

Program Output

```
Enter a string: Screen

The input was "Scr"
```

NOTICE: scanf stops at first letter “e”.

Example : Reading and discarding chars

```
// Fig. 9.24: fig09_24.c
// Reading and discarding characters from the input stream
#include <stdio.h>

int main()
{
   int month1;
   int day1;
   int year1;
   int month2;
   int day2;
   int year2;

   printf( "%s", "Enter a date in the form mm-dd-yyyy: " );
   scanf( "%d%*c%d%*c%d", &month1, &day1, &year1 );

   printf( "month = %d  day = %d  year = %d\n\n", month1, day1, year1 );

   printf( "%s", "Enter a date in the form mm/dd/yyyy: " );
   scanf( "%d%*c%d%*c%d", &month2, &day2, &year2 );

   printf( "month = %d  day = %d  year = %d\n", month2, day2, year2 );
} //end main
```

Program Output

```
Enter a date in the form mm-dd-yyyy: 11-18-2003
month = 11  day = 18  year = 2003


Enter a date in the form mm/dd/yyyy: 11/18/2003
month = 11  day = 18  year = 2003
```

NOTICE:

The inputs “-” and “/” were discarded by scanf because of the “%*c” formatting.