Chapter 9: Formatted Input/Output
=================================

* All input and output is performed with streams - sequences of characters organized into lines.
* Each line consists of zero or more characters and ends with the newline character.
* When program execution begins, three streams are connected to the program automatically.
  1. Standard input stream: is connected to the keyboard
  2. Standard output stream: is connected to the screen
  3. Standard error stream: is connected to the screen for error messages.
* Operating system often allow these streams to be redirected to other devices.

Formatting Output and Printf
-----------------------------

* Every "printf" call contains a format control string that describes the output format.
* The format control string consists of:
  1. conversion specifiers (begins with a percent sign and ends with a conversion specifier).
  2. flags
  3. field widths
  4. precisions
  5. literal characters
* The "printf" function has the form:
  printf(format-control-string, other arguments);
* The format-control-string describes the output format, and other-arguments(optional) correspond to each conversion specification in the format-control-string.

Printing Integers
------------------

d   Display a signed decimal integer.
i   Display a signed decimal integer (different with "d" with "scanf").
o   Display an unsigned octal integer.
u   Display an unsigned decimal integer.
x or X  Display an unsigned hexadecimal integer (x for a-f and X for A-F).
h or l  Place before any integer conversion specifier to indicate that a short or long integer is displayed respectively.

* E.g.
  ```c
  #include <stdio.h>
  main()
  {
    printf("%d\n", 455);
    printf("%i\n", 455);
    printf("%d\n", +455);
    printf("%d\n", -455);
    printf("%d\n", 32000);
    printf("%ld\n", 2000000000);
    printf("%o\n", 455);
    printf("%u\n", 455);
    printf("%x\n", 455);
    printf("%X\n", 455);
  }
  ```
Printing Floating-Point Numbers
-------------------------------

- **e or E**  Display a floating-point value in exponential notation.
- **f**  Display floating-point values.
- **g or G**  Display a floating-point value in either the floating-point form "f" or the exponential form "e" (or "E").
- **L**  Place before any floating-point conversion specifier to indicate that a "long double" floating-point value is displayed.

* Values printed with the conversion specifiers "e", "E", and "f" are output with 6 digits of precision to the right of the decimal point by default.
* "f" always prints at least one digit to the left of the decimal point.
* "e" and "E" always print exactly one digit to the left of the decimal point.
* e.g.

```c
#include <stdio.h>
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);
}
```

Printing Strings and Characters
-------------------------------

- 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).
* E.g.

```c
#include <stdio.h>
main()
{
  char character = 'A';
  char string[] = "This is a string;"
  char *stringPtr = "This is also a string";

  printf("%c\n", character);
  printf("%s\n", "This is a string");
  printf("%s\n", string);
  printf("%s\n", stringPtr);
  return 0;
}
```

Other Conversion Specifiers
---------------------------

- **p**  Display a pointer value in an implementation defined manner.
- **n**  Store the number of characters already output in the current "printf" statement. A pointer of an integer is
supplied as the corresponding argument. Nothing is displayed.

% Display the percent character.

* E.g.

```c
#include <stdio.h>
main()
{
  int *ptr;
  int x = 12345, y;

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

  printf("Total characters printed on this line is: %n", &y);
  printf(" \%d\n\n", y);
  y = printf("This line has 28 characters.");
  printf("\%d characters were printed\n\n", y);

  printf("Printing a %\ in a format control string\n");
  return 0;
}
```

Printing with Field Widths and Precisions
-----------------------------------------

* If the field width is larger than the data being printed, the data will normally be right-justified within that field.
* An integer representing the field width is inserted between the percent sign (%) and the conversion specifier in the conversion specification.
* The field width is automatically increased to print values wider than the field, and the minus sign for a negative value uses one character position in the field width.
* E.g.

```c
#include <stdio.h>
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);

  return 0;
}
```

* Precision meanings for different data types:
  integer The minimum number of digits to be printed. Zero are prefixed to the printed value until the total number of digits is equivalent to the precision.
floating-point The number of digits to appear after the decimal point.
(e, E, and f) The maximum number of significant digits to be printed.
(g and G) The maximum number of characters to be written from the string.

string (s) The number of digits to appear after the decimal point.

* To use precision, place a decimal point (.) followed by an integer representing the precision between the percent sign and the conversion specifier.
* E.g.
  
  printf("%.4d\n\t%.9d\n\n", i, i);

* When a floating-point value is printed with a precision smaller than the original number of decimal places in the value, the value is rounded.
* E.g.
  
  printf("%9.3f", 123.456789);

* It is possible to specify the field width and the precision using integer expressions in the argument list following the format control string.
* E.g.
  
  printf("%*.f", 7, 2, 98.736);

Using Flags in the Printf Format Control String
-----------------------------------------------
* Function "printf" also provides flags to supplement its output formatting capabilities.

- 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" and "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. For "g" and "G" specifiers, trailing zeros are not eliminated.
0 Pad a field with leading zeros.

* To use a flag in a format control string, place the flag immediately to the right of the percent sign.
* Several flags may be combined in one conversion specification.
* E.g.

#include <stdio.h>
main()
{
    printf("%10s%10d%10c%10f
\n", "hello", 7, 'a', 1.23);
    printf("%-10s%-10d%-10c%-10f
\n", "hello", 7, 'a', 1.23);
    return 0;
}

* E.g.

#include <stdio.h>
main()
{
    printf("%d\n\n", 786, -786);
    printf("%+d\n%+d\n", 786, -786);
}
main()
{
    printf("% d\n% d\n", 547 -547);
    return 0;
}

* E.g.
#include <stdio.h>
main()
{
    int c = 1427;
    float p = 1427.0
    printf("%#o\n", c);
    printf("%#x\n", c);
    printf("%#X\n", c);
    printf("%g\n", p);
    printf("%#g\n", p);
    return 0;
}

* E.g.
#include <stdio.h>
main()
{
    printf("%+09d", 452);
    printf("%09d", 452);
    return 0;
}

Printing Literals and Escape Sequences
--------------------------------------
* Various control characters must be represented by escape sequences.
* An escape sequence is represented by a backslash (\) followed by a particular escape character.
    \'          Output the single quote (') character.
    \"          Output the double quote (") character.
    \?          Output the question mark (?) character.
    \\          Output the backslash (\) character.
    \a          Cause an audible (bell) or visual alert.
    \b          Move the cursor back one position on the current line.
    \f          Move the cursor to the start of the next logical page.
    \n          Move the cursor to the beginning of the next line.
    \r          Move the cursor to the beginning of the current line.
    \t          Move the cursor to the next horizontal tab position.
    \v          Move the cursor to the next vertical tab position.

Formatting Input with Scanf
---------------------------
* Every "scanf" statement contains a format control string that describes the format of the data to be input.
* The "scanf" function is written in the following form:
    scanf(format-control-string, other-arguments);
Integers
d           Read an optionally signed decimal integer
i           Read an optionally signed decimal, octal, or hexadecimal integer.
Read an octal (unsigned) integer.
Read an unsigned decimal integer.
Place before any of the integer conversion specifiers to indicate that a "short" or "long" integer is to be input.
Place before any of the floating-point conversion specifiers to indicate that a "double" or "long double" value is to be input.
Read a character. No null ("\0") is added.
Read a string.
Scan a string for a set of characters that are stored in an array.
Read a pointer address
Store the number of characters input so far
Skip a percent sign (%) in the input

* E.g.
/* Try -70 -70 0x70 70 70 70 */
#include <stdio.h>
main()
{
    int a, b, c, d, e, f, g;
    printf("Enter seven integers: ");
    scanf("%d%i%i%i%o%u%x", &a, &b, &c, &d, &e, &f, &g);
    printf("The input displayed as decimal integers is: \n");
    printf("%d %d %d %d %d %d %d
", a, b, c, d, e, f, g);
    return 0;
}

* E.g.
/* Try 1.27987 1.27987e+03 3.38476e-06 */
#include <stdio.h>
main()
{
    float a, b, c;
    printf("Enter three floating-point numbers: \n");
    scanf("%e%f%g", &a, &b, &c);
    printf("Here are the numbers entered in plain\n");
    printf("floating-point notation: \n");
    printf("%f %f %f \n", a, b, c);
    return 0;
}

* E.g.
/* Try Sunday */
#include <stdio.h>
main()
{
    char x, y[9];
    printf("Enter a string: ");
    scanf("%c%s", &x, y);

    printf("The input was:\n");
    printf("the character "\%c" ", x);
printf("and the string \"%s\"\n", y);
    return 0;
}

* E.g

/* Try ooeecoahah */
#include <stdio.h>
main()
{
    char z[9];
    printf("Enter string: ");
    scanf("%[aeiou]", z);
    printf("The input was \"%s\"\n", z);
    return 0;
}

* E.g.

/* Try String */
#include <stdio.h>
main()
{
    char z[9];
    printf("Enter a string: ");
    scanf("%^aeiou", z);
    printf("The input was \"%s\"\n", z);
    return 0;
}

* E.g.

/* Try 123456 */
#include <stdio.h>
main()
{
    int x, y;
    printf("Enter a six digit integer: ");
    scanf("%2d%d", &x, &y);
    printf("The integers input were %d and %d\n", x, y);
    return 0;
}

* E.g.

#include <stdio.h>
main()
{
    int month1, day1, year1, month2, day2, year2;
    printf("Enter a date in the form mm-dd-yy: ");
    scanf("%d%c%d%c%d", &month1, &day1, &year1);
    printf("month = %d day = %d year = %d\n\n", month1, day1, year1);

    printf("Enter a date in the form mm/dd/yy: ");
    scanf("%d%c%d%c%d", &month2, &day2, &year2);
    printf("month = %d day = %d year = %d\n\n", month2, day2, year2);
}

* "scanf" provides the assignment suppression character "*", which enables "scanf" to read any type of data from the input and discard it without assigning it to a variable.