Chapter 9
Formatted Input/Output

Associate Prof. Yuh-Shyan Chen
Dept. of Computer Science and Information Engineering
National Chung-Cheng University

Outline

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 Precisions
9.9 Using Flags in the printf Format-Control String
9.10 Printing Literals and Escape Sequences
9.11 Formatting Input with scanf

习题练习:

Try to write a recursive function to reverse a string (call by reference).

For example, char a[] = "ABCDEFGH";
   /* printf( "%s \n", a); */
   // AB CDEF GH
   reverse_string(a);
   /* printf( "%s \n", a); */
   // HG FEDCBA */

9.1 Introduction

In this chapter
○ Presentation of results
○ scanf and printf
○ Streams (input and output)
   ○ gets, puts, getchar, putchar (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
    - ANSI C must support lines of at least 254 characters
  - Performs all input and output

Cont.

- Can often be redirected
  - Standard input - keyboard
  - Standard output - screen
  - Standard error - screen
  - More Chapter 11

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

Cont.

- printf(format-control-string, other-arguments);
  - format control string: describes output format
  - other-arguments: correspond to each conversion specification in format-control-string
  - each specification begins with a percent sign, ends with conversion specifier
9.4 Printing Integers

- **Integer**
  - Whole number (no decimal point): 25, 0, -9
  - Positive, negative, or zero
  - Only minus sign prints by default (later we shall change this)

9.4 Printing Integers

<table>
<thead>
<tr>
<th>Conversion Specifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>d</code></td>
<td>Display a signed decimal integer</td>
</tr>
</tbody>
</table>
| `i`                  | Display a signed decimal integer (Note: The `d` and `i` specifiers are different when used with `scanf`.)
| `o`                  | Display an unsigned octal integer |
| `u`                  | Display an unsigned decimal integer |
| `x` or `X`           | Display an unsigned hexadecimal integer |
| `h` or `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 length modifiers.

```c
1 /* Fig 9.2: fig09_02.c */
2 /* Using the integer conversion specifiers */
3 #include <stdio.h>
4
5 int main()
6 {
7    printf( "%d\n", 455 );
8    printf( "%i\n", 455 );  /* i same as d in printf */
9    printf( "%d\n", +455 );
10   printf( "%d\n", -455 );
11   printf( "%hd\n", 32000 );
12   printf( "%ld\n", 2000000000 );
13   printf( "%o\n", 455 );
14   printf( "%u\n", 455 );
15   printf( "%u\n", -455 );
16   printf( "%x\n", 455 );
17   printf( "%X\n", 455 );
18
19   return 0;
20 }
```

9.5 Printing Floating-Point Numbers

- **Floating Point Numbers**
  - Have a decimal point (33.5)
  - **Exponential notation** (computer’s version of scientific notation)
    - `150.3` is `1.503 \times 10^2` in scientific
    - `150.3` is `1.503E+02` in exponential (`E` stands for exponent)
    - use `e` or `E`
Continue

- **f** - print floating point with at least one digit to left of decimal
- **g** (or **G**) - prints in **f** or **e(E)** with no trailing zeros (1.2300 becomes 1.23)
  - Use exponential if exponent less than −4, or greater than or equal to precision (6 digits by default)

---

### Conversion Specifier

<table>
<thead>
<tr>
<th>Conversion Specifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>e</strong> or <strong>E</strong></td>
<td>Display a floating-point value in exponential notation.</td>
</tr>
<tr>
<td><strong>f</strong></td>
<td>Display floating-point values.</td>
</tr>
<tr>
<td><strong>g</strong> or <strong>G</strong></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>

---

#### 9.6 Printing Strings and Characters

- **c**
  - Prints char argument
  - Cannot be used to print the first character of a string
- **s**
  - Requires a pointer to char as an argument
  - Prints characters until NULL ("\0") encountered
  - Cannot print a char argument
Remember

- Single quotes for character constants ('z')
- Double quotes for strings "z" (which actually contains two characters, 'z' and '\0')

### 9.7 Other Conversion Specifiers

- **P**
  - Displays pointer value (address)
- **n**
  - Stores number of characters already output by current `printf` statement
  - Takes a pointer to an integer as an argument
  - Nothing printed by a %n specification
  - Every `printf` call returns a value
    - Number of characters output
    - Negative number if error occurs

- **%**
  - Prints a percent sign
- **%%**
  - Displays a percent sign
# 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` - field width of 4

- **Precision**
  - Meaning varies depending on data type
  - Integers (default 1) - minimum number of digits to print
    - If data too small, prefixed with zeros
  - Floating point - number of digits to appear after decimal (`e` and `f`)
    - 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 (III)

- Format
  - Precision: use a dot (.) then precision number after %
    `%.3f`
  - Can be combined with field width
    `%.5.3f`

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

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

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>Left-justify the output within the specified field.</td>
</tr>
<tr>
<td>+</td>
<td>Display a plus sign preceding positive values and a minus sign preceding negative values.</td>
</tr>
<tr>
<td>space</td>
<td>Pad a field with a space before a positive value not printed with + flag.</td>
</tr>
<tr>
<td>#</td>
<td>Force a decimal point for a floating-point number printed with e, E, f, g, or G specifiers.</td>
</tr>
<tr>
<td>0</td>
<td>Pad a field with leading zeros.</td>
</tr>
</tbody>
</table>

```c
#include <stdio.h>

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

  printf( "Using precision for integers
"
          "\t%.4d
          \t%.9d
"
          "Using precision for floating-point numbers
"
          "\t%.3f
          \t%.3e
          \t%.3g
"
          "Using precision for strings
"
          "\t%.11s
", i, i, f, f, f, s);

  return 0;
}
```
9.10 Printing Literals and Escape Sequences

- **Printing Literals**
  - 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
9.11 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

---

### Conversion Specifiers

- **Integers**
  - `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.

- **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 longdouble value is to be input.

- **Characters and strings**
  - `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**
  - A set of characters enclosed in square brackets `[ ]`
  - Preceded by `%` sign
  - Scans input stream, looking only for characters in scan set
  - Whenever a match occurs, stores character in specified array
  - Stops scanning once a mismatch is found
Inverted scan sets
- Use a caret: [^aeiou]
- Causes characters not in the scan set to be stored

9.11 Formatting Input with `scanf` (IV)
- Skipping characters
  - Include character to skip in format control
  - Or, use `*` (assignment suppression character)
    - Skips any type of character without storing it

Using a scan set

```c
/* Fig 9.20: fig09_20.c */
/* Reading characters and strings */

#include <stdio.h>

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

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

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

    return 0;
}
```

Enter string: Sunday
The input was: the character "S" and the string "unday"
```c
/* Fig 9.22: fig09_22.c */
/* Using an inverted scan set */
<stdio.h>

int main()
{
    char z[9] = {'\0'};

    printf("Enter a string: ");
    scanf("[^aeiou]", z);
    printf("The input was "z"");

    return 0;
}
```

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

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

    printf("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
", month1, day1, year1);

    printf("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
", month2, day2, year2);

    return 0;
}
```

Enter a string: String
The input was "Str"

Enter a date in the form mm-dd-yyyy: 11-18-2000
month = 11  day = 18  year = 2000

Enter a date in the form mm/dd/yyyy: 11/18/2000
month = 11  day = 18  year = 2000