Chapter 9 - Formatted Input/Output

Outline

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
    - Linjustificatione characters, ends with newline character
    - ANSI C must support 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 Chapter 11

9.3 Formatting Output with `printf`

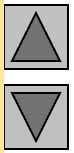
- **`printf`**
  - **`precise output formatting`**
    - Conversion specifications: flags, field widths, precisions, etc.
  - Can perform rounding, aligning columns, right/left justification, inserting literal characters, exponential format, hexadecimal format, and fixed width and precision
- Format

  `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)

Conversion Specifier	Description
`d`	Display a signed decimal integer.
`i`	Display a signed decimal integer. (*Note:* The **`i`** and **`d`** 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. **`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` or **`l`** (letter **`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*.



Outline

1. Print

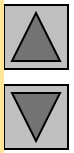
```c
/* Fig 9.2: fig09_02.c */
/* Using the integer conversion specifiers */
#include <stdio.h>

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 );

   return 0;
}
```

Program Output

```
455
455
455
-455
32000
2000000000
707
455
65081
1c7
1C7
```

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 x 10²` in scientific
    - `150.3` is `1.503E+02` in exponential (`E` stands for exponent)
    - use `e` or `E`
  - `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)



Outline

1. Print

```
1  /* Fig 9.4: fig09_04.c */
2  /* Printing floating-point numbers with
3     floating-point conversion specifiers */
4
5  #include <stdio.h>
6
7  int main()
8  {
9     printf( "%e\n", 1234567.89 );
10    printf( "%e\n", +1234567.89 );
11    printf( "%e\n", -1234567.89 );
12    printf( "%E\n", 1234567.89 );
13    printf( "%f\n", 1234567.89 );
14    printf( "%g\n", 1234567.89 );
15    printf( "%G\n", 1234567.89 );
16
17    return 0;
18 }
```

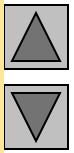
Program Output

```
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 **`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'`**)

```c
/* Fig 9.5: fig09_05c */
/* Printing strings and characters */
#include <stdio.h>

int main()
{
   char character = 'A';
   char string[] = "This is a string";
   const 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;
}
```



Outline

1. Initialize variables

2. Print

Program Output

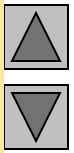
```
A
This is a string
This is a string
This is also a string
```

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
  - **`%%`**

```c
/* Fig 9.7: fig09 07.c */
/* Using the p, n, and % conversion specifiers */
#include <stdio.h>

int 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\n" );
   printf( "%d characters were printed\n\n", y );

   printf( "Printing a %% in a format control string\n" );

   return 0;
}
```

```
The value of ptr is 0065FDF0
The address of x is 0065FDF0

Total characters printed on this line is: 41

This line has 28 characters
28 characters were printed

Printing a % in a format control string
```



Outline

1. Initialize variables

2. Print

Program Output

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

9.8 Printing with Field Widths and Precisions (II)

- 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

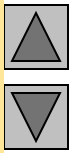
9.8 Printing with Field Widths and Precisions (III)

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

    `%.3f`
  - Can be combined with field width

    `%5.3f`
  - Can use integer expressions to determine field width and precision
    - Use `*`
    - Negative field width - left justified
    - Positive field width - right justified
    - Precision must be positive

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

Outline

```c
/* Fig 9.9: fig09_09.c */
/* Using precision while printing integers,
   floating-point numbers, and strings */
#include <stdio.h>

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

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

   return 0;
}
```

1. Initialize variables

2. Print

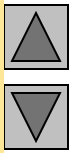
Program Output

```
Using precision for integers
        0873
        000000873

Using precision for floating-point numbers
        123.945
        1.239e+02
        124

Using precision for strings
        Happy Birth
```

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

Flag	Description
- (minus sign)	Left-justify the output within the specified field.
+ (plus sign)	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 only printed if a digit follows it.) For **g** and **G** specifiers, trailing zeros are not eliminated.
0 (zero)	Pad a field with leading zeros.



Outline

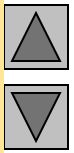
1. Print

```
1  /* Fig 9.11: fig09_11.c */
2  /* Right justifying and left justifying values */
3  #include <stdio.h>
4
5  int main()
6  {
7     printf( "%10s%10d%10c%10f\n\n", "hello", 7, 'a', 1.23 );
8     printf( "%-10s%-10d%-10c%-10f\n", "hello", 7, 'a', 1.23 );
9     return 0;
10 }
```

Program Output

```
hello         7         a  1.230000

hello     7         a         1.230000
```

```c
/* Fig 9.14: fig09_14.c */
/* Using the # flag with conversion specifiers
   o, x, X and any floating-point specifier */
#include <stdio.h>

int main()
{
   int c = 1427;
   double p = 1427.0;

   printf( "%#o\n", c );
   printf( "%#x\n", c );
   printf( "%#X\n", c );
   printf( "\n%g\n", p );
   printf( "%#g\n", p );

   return 0;
}
```



Outline

1. Initialize variables

2. Print

Program Output

```
02623
0x593
0X593

1427
1427.00
```

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

9.10 Printing Literals and Escape Sequences (II)

Escape sequence	Description
`\'`	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.

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` (II)

Conversion specifier	Description
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 **`long double`** 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	
[scan characters	Scan a string for a set of characters that are stored in an array.
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 **`scanf`**. The corresponding argument is a pointer to integer
`%`	Skip a percent sign (**`%`**) in the input.

9.11 Formatting Input with `Scanf` (III)

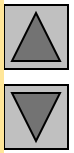
- Scan sets
  - Set of characters enclosed in square brackets `[]`
    - Preceded by `%` sign
  - Scans input stream, looking only for characters in scan set
    - Whenever a match occurs, stores character in specified array
    - Stops scanning once a mismatch is found
  - Inverted scan sets
    - Use 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

```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:\n" );
   printf( "the character \"%c\" ", x );
   printf( "and the string \"%s\"\n", y );

   return 0;
}
```

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



Outline

1. Initialize variables

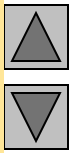
2. Input

3. Print

Program Output

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

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

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

   return 0;
}
```

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



Outline

1. Initialize variable

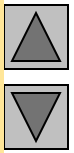
2. Input

3. Print

Program Output

```c
/* Fig 9.24: fig09 24.c */
/* Reading and discarding characters from the input stream */
#include <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\n\n",
      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\n",
      month2, day2, year2 );

   return 0;
}
```

```
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
```



Outline

1. Initialize variables

2. Input

3. Print

Program Output