Escola de Engenharia
Universidade do Minho

Programação – Aula Teórica 9
Input e Output Formatado

Luís Paulo Reis
lpreis@dsi.uminho.pt
Professor Associado do Departamento de Sistemas de Informação, Escola de Engenharia, Universidade do Minho, Portugal

(Slides Baseados em Deitel e Deitel 2010 e L.P.Reis et al., 2006)

Escola de Engenharia
Universidade do Minho

Input e Output Formatado

Outline

Objectives

- **In this lesson, you will learn:**
  - To understand input and output streams
  - To be able to use all print formatting capabilities
  - To be able to use all input formatting capabilities

9.1 Introduction

- **In this lesson:**
  - Presentation of results on the display
  - Reading of values from the keyboard
  - `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
  - Can often be redirected
    - Standard input – keyboard
    - Standard output – screen
    - Standard error – screen

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

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*.

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 (later we shall change this)

```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; /* indicates successful termination */

} /* end main */
```

```
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`)
  - 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 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)

Escola de Engenharia
Universidade do Minho

9.5 Printing Floating-Point Numbers

Conversion specifier	Description
`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.

Fig. 9.3 Floating-point conversion specifiers.

```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 );

    return 0; /* indicates successful termination */

} /* end main */
```

```
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'; /* 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 );

    return 0; /* indicates successful termination */

} /* end main */
```

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

Escola de Engenharia
Universidade do Minho

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
  - %%

Escola de Engenharia
Universidade do Minho

9.7 Other Conversion Specifiers

Conversion specifier	Description
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 to an integer is supplied as the corresponding argument. Nothing is displayed.
%	Display the percent character.

Fig. 9.6 Other conversion specifiers.

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

int main()
{
   int *ptr;
   int x = 12345;
   int y;

    ptr = &x;      /* assign address of x to ptr */
    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:%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; /* indicates successful termination */

} /* end main */
```

```
The value of ptr is 0012FF78
The address of x is 0012FF78

Total characters printed on this line: 38

This line has 28 characters
28 characters were printed

Printing a % in a format control 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 – field width of 4

9.8 Printing with Field Widths and Precisions

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

      `%.3f`

9.8 Printing with Field Widths and Precisions

- **Field width and precision**
  - Can both be specified
    - `%width.precision`

      `%5.3f`
  - Negative field width – left justified
  - Positive field width – right justified
  - Precision must be positive
  - Can use integer expressions to determine field width and precision values
    - Place an asterisk (*) in place of the field width or precision
      - Matched to an `int` argument in argument list
    - Example:

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

```c
/* Fig 9.8: fig09_08.c */
/* Printing integers right-justified */
#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 );

   return 0; /* indicates successful termination */

} /* end main */
```

```
   1
  12
 123
1234
12345

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

```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"; // char array

    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; /* indicates successful termination */

 } /* end main */
```

```
Using precision for
integers
        0873
        000000873

Using precision for
floating-point numbers
        123.945
        1.239e+002
        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.

Fig. 9.10 Format control string flags.

```c
/* 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 );

    return 0; /* indicates successful termination */

} /* end main */
```

```
     hello         7         a  1.230000

hello     7         a         1.230000
```

```c
/* Fig 9.12: fig09_12.c */
/* Printing 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 );

    return 0; /* indicates successful termination */

} /* end main */
```

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

```c
/* Fig 9.13: fig09_13.c */
/* Printing a space before signed values
   not preceded by + or - */
#include <stdio.h>

int main()
{
   printf( "% d\n% d\n", 547, -547 );

    return 0; /* indicates successful termination */

} /* end main */
```

```
 547
-547
```

```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;      /* initialize c */
   double p = 1427.0; /* initialize p */

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

    return 0; /* indicates successful termination */

} /* end main */
```

```
02623
0x593
0X593

1427
1427.00
```

```c
/* Fig 9.15: fig09_15.c */
/* Printing with the 0( zero ) flag fills in leading zeros */
#include <stdio.h>

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

    return 0; /* indicates successful termination */

} /* end main */
```

```
+00000452
000000452
```

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

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.

Fig. 9.16 Escape sequences.

9.11 Formatting Input with `scanf`

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.

Fig. 9.17 Conversion specifiers for `scanf`.

9.11Formatting Input with `scanf`

Conversion specifier	Description
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.

Fig. 9.17 Conversion specifiers for scanf.

Escola de Engenharia
Universidade do Minho

9.11Formatting 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

9.11Formatting Input with `scanf`

- **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 character not in the scan set is found
  - Inverted scan sets
    - Use a caret `^`: `[^aeiou]`
    - Causes characters not in the scan set to be stored
- **Skipping characters**
  - Include character to skip in format control
  - Or, use `*` (assignment suppression character)
    - Skips any type of character without storing it

```c
/* Fig 9.18: fig09_18.c */
/* Reading integers */
#include <stdio.h>

int main()
{
   int a; /* define a */
   int b; /* define b */
   int c; /* define c */
   int d; /* define d */
   int e; /* define e */
   int f; /* define f */
   int g; /* define 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\n", a, b, c, d, e, f, g );

   return 0; /* indicates successful termination */

} /* end main */
```

```
Enter seven integers: -70 -70 070 0x70 70 70 70
The input displayed as decimal integers is:
-70 -70 56 112 56 70 112
```

```c
/* Fig 9.19: fig09_19.c */
/* Reading floating-point numbers */
#include <stdio.h>

/* function main begins program execution */
int main()
{
   double a; /* define a */
   double b; /* define b */
    double c; /* define c */

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

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

    return 0; /* indicates successful termination */

 } /* end main */
```

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

int main()
{
   char x;       /* define x */
   char y[ 9 ]; /* define array y */

    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; /* indicates successful termination */

} /* end main */
```

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

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

/* function main begins program execution */
int main()
{
   char z[ 9 ]; /* define array z */

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

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

    return 0; /* indicates successful termination */

 } /* end main */
```

```
Enter string: ooeeooahah
The input was "ooeeooa"
```

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

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

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

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

    return 0; /* indicates successful termination */

 } /* end main */
```

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

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

int main()
{
   int x; /* define x */
   int y; /* define 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; /* indicates successful termination */

 } /* end main */
```

```
Enter a six digit integer: 123456
The integers input were 12 and 3456
```

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

int main()
{
   int month1; /* define month1 */
   int day1;   /* define day1 */
   int year1;  /* define year1 */
   int month2; /* define month2 */
   int day2;   /* define day2 */
   int year2;  /* define 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; /* indicates successful termination */

} /* end main */
```

```
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
```

Escola de Engenharia
Universidade do Minho

Questões?

Programação – Aula Teórica 9

Input e Output Formatado

Luís Paulo Reis
lpreis@dsi.uminho.pt
Professor Associado do Departamento de Sistemas de Informação, Escola de Engenharia, Universidade do Minho, Portugal

(Slides Baseados em Deitel e Deitel 2010 e L.P.Reis et al., 2006)