Chapter 9 – Formattazione Input/Output

9.2 Flussi

- Flussi
  - Sequenze di caratteri organizzate in linee
    - Ogni linea consiste di zero o più caratteri e finisce con il carattere “newline”
    - Lo standard ANSI C deve supportare linee di almeno 254 caratteri
  - Sono coinvolti in tutti gli input e output
  - Possono spesso essere reindirizzati su
    - Standard input – tastiera
    - Standard output – monitor
    - Standard error – monitor
    - Utilizzati nei processi di lettura e scrittura dei files

9.3 Formattare l'Output con `printf`

- `printf`
  - Formattazione dell'output
    - Specifiche di conversione: flags, ampiezza campi, precisione, ecc.
  - Può eseguire operazioni di arrotondamento, allineamento colonne, allineamento a destra/sinistra, inserimento di letterali, formato esponenziale, formato esadecimale, ampiezza e precisione fissati

- Formato
  - `printf(` *stringa-controllo-formato, altri-argomenti* `);`
  - *stringa-controllo-formato*: descrive il formato di output
  - *altri-argomenti*: corrisponde alle specifiche di conversione in *stringa-controllo-formato*
    - Le specifiche cominciano con un segno percentuale (`%`) e finiscono con lo specificatore di conversione

9.4 Visualizzazione di Interi

- Interi
  - Numeri interi (senza decimali): `25`, `0`, `-9`
  - Positivi, negativi, o zero
  - Di default viene stampato solo il segno meno (-) dei numeri negativi

9.4 Visualizzare interi

Specificatore di conversione	Descrizione
`d`	Visualizza un intero con segno.
`i`	Visualizza interi decimali. (*Nota:* gli specificatori `i` e `d` sono differenti se usati con `scanf`.)
`o`	Visualizza un intero ottale senza segno.
`u`	Visualizza un intero decimale senza segno.
`x` or `X`	Visualizza un intero esadecimale. Lo specificatore `X` visualizza le cifre `0-9` e le lettere `A-F` mentre `x` visualizza le cifre `0-9` e `a-f`.
`h` or `l` (letter `l`)	Messo prima di uno specificatore di conversione intero indica che un intero di tipo `short` o `long` è visualizzato. Le letter `h` ed `l` sono chiamate *modificatori di lunghezza*.

Fig. 9.1 Specificatori di conversione di interi.

Outline

fig09_02.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; /* indicates successful termination */
20
21 } /* end main */
```

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

Outline

7

Program Ouptut

8

9.5 Stampa di Numeri Floating-Point

- Numeri Floating-Point
  - Hanno un punto decimale (`33.5`)
  - Notazione esponenziale (versione per computer della notazione scientifica)
    - `150.3` è `1.503 x 10²` in notazione scientifica
    - `150.3` è `1.503E+02` in notazione esponenziale (`E` sta per esponente)
    - Utilizzo di `e` o `E`
  - `f` – visualizzazione di floating point con almeno una cifra decimale
  - `g` (o `G`) – visualizzazione in `f` o `e` senza zero non significativi (`1.2300` diventa `1.23`)
    - Utilizzare notazione esponenziale se l'esponente è minore di `-4`, o uguale o maggiore alla precisione (`6` cifre per default)

9.5 Stampa di Numeri Floating-Point

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.

Outline

fig09_04.c

```
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; /* indicates successful termination */
18
19 } /* end main */
```

Program Output

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

9.6 Visualizzazione di Stringhe e Caratteri

- `c`
  - Stampa argomenti di tipo `char`
  - Non può essere usato per stampare il primo carattere di una stringa
- `s`
  - Richiede un puntatore a `char` come argomento
  - Stampa caratteri fino al carattere `NULL` (`'\0'`)
  - Non può stampare un argomento di tipo `char`
- Nota
  - Apici singoli per caratteri (`'z'`)
  - Apici doppi per le stringhe `"z"` (contiene effettivamente due caratteri, `'z'` and `'\0'`)

Outline

fig09_05.c

```
1  /* Fig 9.5: fig09_05c */
2  /* Printing strings and characters */
3  #include <stdio.h>
4
5  int main()
6  {
7     char character = 'A'; /* initialize char */
8     char string[] = "This is a string"; /* initialize char array */
9     const char *stringPtr = "This is also a string"; /* char pointer */
10
11    printf( "%c\n", character );
12    printf( "%s\n", "This is a string" );
13    printf( "%s\n", string );
14    printf( "%s\n", stringPtr );
15
16    return 0; /* indicates successful termination */
17
18 } /* end main */
```

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

9.7 Altri Specificatori di Conversione

- p
  - Visualizza il valore di un puntatore (indirizzo)
- n
  - Memorizza il numero di caratteri già visualizzati dalla `printf` corrente
  - Prende un puntatore ad un intero come argomento
  - Una chiamata `printf` restituisce come valore
    - Il numero di caratteri in output
    - Un numero negativo in caso di errore
- %
  - Stampa un segno percentuale
  - %%

9.7 Altri Specificatori di Conversione

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
/* Fig 9.7: fig09_07.c */
/* Using the p, n, and % conversion specifiers */
#include <stdio.h>

int main()
{
   int *ptr;      /* define pointer to int */
   int x = 12345; /* initialize int x */
   int y;         /* define 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 */
```


Outline

fig09_07.c (1 of 2)


```
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
```


Outline

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

- 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 );`

Outline

fig09_08.c

```
1  /* Fig 9.8: fig09_08.c */
2  /* Printing integers right-justified */
3  #include <stdio.h>
4
5  int main()
6  {
7     printf( "%4d\n", 1 );
8     printf( "%4d\n", 12 );
9     printf( "%4d\n", 123 );
10    printf( "%4d\n", 1234 );
11    printf( "%4d\n\n", 12345 );
12
13    printf( "%4d\n", -1 );
14    printf( "%4d\n", -12 );
15    printf( "%4d\n", -123 );
16    printf( "%4d\n", -1234 );
17    printf( "%4d\n", -12345 );
18
19    return 0; /* indicates successful termination */
20
21 } /* end main */
```

```
 1
  12
 123
1234
12345

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

Outline

Program Output


```
1  /* Fig 9.9: fig09_09.c */
2  /* Using precision while printing integers,
3     floating-point numbers, and strings */
4  #include <stdio.h>
5
6  int main()
7  {
8     int i = 873;                /* initialize int i */
9     double f = 123.94536;       /* initialize double f */
10    char s[] = "Happy Birthday"; /* initialize char array s */
11
12    printf( "Using precision for integers\n" );
13    printf( "\t%.4d\n\t%.9d\n\n", i, i );
14
15    printf( "Using precision for floating-point numbers\n" );
16    printf( "\t%.3f\n\t%.3e\n\t%.3g\n\n", f, f, f );
17
18    printf( "Using precision for strings\n" );
19    printf( "\t%.11s\n", s );
20
21    return 0; /* indicates successful termination */
22
23 } /* end main */
```

Outline

fig09_09.c

Outline

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
```


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
```


Outline

fig09_11.c

Program Output


```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
```


Outline

fig09_12.c

Program Output

Outline

fig09_13.c

```
1  /* Fig 9.13: fig09_13.c */
2  /* Printing a space before signed values
3     not preceded by + or - */
4  #include <stdio.h>
5
6  int main()
7  {
8     printf( "% d\n% d\n", 547, -547 );
9
10    return 0; /* indicates successful termination */
11
12 } /* end main */
```

Program Output

```
 547
-547
```


Outline

fig09_14.c

```
1  /* Fig 9.14: fig09_14.c */
2  /* Using the # flag with conversion specifiers
3     o, x, X and any floating-point specifier */
4  #include <stdio.h>
5
6  int main()
7  {
8     int c = 1427;      /* initialize c */
9     double p = 1427.0; /* initialize p */
10
11    printf( "%#o\n", c );
12    printf( "%#x\n", c );
13    printf( "%#X\n", c );
14    printf( "\n%g\n", p );
15    printf( "%#g\n", p );
16
17    return 0; /* indicates successful termination */
18
19 } /* end main */
```

Program Output

```
02623
0x593
0X593

1427
1427.00
```

Outline

fig09_15.c

```
1  /* Fig 9.15: fig09_15.c */
2  /* Printing with the 0( zero ) flag fills in leading zeros */
3  #include <stdio.h>
4
5  int main()
6  {
7     printf( "%+09d\n", 452 );
8     printf( "%09d\n", 452 );
9
10    return 0; /* indicates successful termination */
11
12 } /* end main */
```

Program Output

```
+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

9.10 Printing Literals and Escape Sequences

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`.

9.11 Formatting 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.

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

- 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

Outline

fig09_18.c

```
1  /* Fig 9.18: fig09_18.c */
2  /* Reading integers */
3  #include <stdio.h>
4
5  int main()
6  {
7     int a; /* define a */
8     int b; /* define b */
9     int c; /* define c */
10    int d; /* define d */
11    int e; /* define e */
12    int f; /* define f */
13    int g; /* define g */
14
15    printf( "Enter seven integers: " );
16    scanf( "%d%i%i%i%o%u%x", &a, &b, &c, &d, &e, &f, &g );
17
18    printf( "The input displayed as decimal integers is:\n" );
19    printf( "%d %d %d %d %d %d %d\n", a, b, c, d, e, f, g );
20
21    return 0; /* indicates successful termination */
22
23 } /* end main */
```

Program Output

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

Outline

fig09_20.c

```
1  /* Fig 9.20: fig09_20.c */
2  /* Reading characters and strings */
3  #include <stdio.h>
4
5  int main()
6  {
7     char x;      /* define x */
8     char y[ 9 ]; /* define array y */
9
10    printf( "Enter a string: " );
11    scanf( "%c%s", &x, y );
12
13    printf( "The input was:\n" );
14    printf( "the character \"%c\" ", x );
15    printf( "and the string \"%s\"\n", y );
16
17    return 0; /* indicates successful termination */
18
19 } /* end main */
```

Program Output

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


Outline

fig09_23.c

```
1  /* Fig 9.23: fig09_23.c */
2  /* inputting data with a field width */
3  #include <stdio.h>
4
5  int main()
6  {
7     int x; /* define x */
8     int y; /* define y */
9
10    printf( "Enter a six digit integer: " );
11    scanf( "%2d%d", &x, &y );
12
13    printf( "The integers input were %d and %d\n", x, y );
14
15    return 0; /* indicates successful termination */
16
17 } /* end main */
```

Program Output

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

Outline

fig09_24.c

```
1  /* Fig 9.24: fig09_24.c */
2  /* Reading and discarding characters from the input stream */
3  #include <stdio.h>
4
5  int main()
6  {
7     int month1; /* define month1 */
8     int day1;   /* define day1 */
9     int year1;  /* define year1 */
10    int month2; /* define month2 */
11    int day2;   /* define day2 */
12    int year2;  /* define year2 */
13
14    printf( "Enter a date in the form mm-dd-yyyy: " );
15    scanf( "%d%*c%d%*c%d", &month1, &day1, &year1 );
16
17    printf( "month = %d  day = %d  year = %d\n\n", month1, day1, year1 );
18
19    printf( "Enter a date in the form mm/dd/yyyy: " );
20    scanf( "%d%*c%d%*c%d", &month2, &day2, &year2 );
21
22    printf( "month = %d  day = %d  year = %d\n", month2, day2, year2 );
23
24    return 0; /* indicates successful termination */
25
26 } /* end main */
```


Outline

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
```