9

C Formatted Input/Output

25

Stream Input/Output

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 will change this)**

Conversion specifier	Description
`d`	Display as a signed decimal integer.
`i`	Display as a signed decimal integer. [*Note:* The `i` and `d` specifiers are different when used with `scanf`.]
`o`	Display as an unsigned octal integer.
`u`	Display as an unsigned decimal integer.
`x` or `X`	Display as 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.

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( void )
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", 2000000000L ); /* L suffix makes literal a long */
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 */
```

d and **i** specify signed integers

h specifies a **short** number

l specifies a **long** number

o specifies an octal integer

u specifies an unsigned integer

x and **X** specify hexadecimal integers

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

25.6.1 Integral Stream Base: `dec`, `oct`, `hex` and `setbase`

- **Change a stream's integer base by inserting manipulators**
  - **`hex` manipulator**
    - **Sets the base to hexadecimal (base 16)**
  - **`oct` manipulator**
    - **Sets the base to octal (base 8)**
  - **`dec` manipulator**
    - **Resets the base to decimal**
  - **`setbase` parameterized stream manipulator**
    - **Takes one integer argument: `10`, `8` or `16`**
    - **Sets the base to decimal, octal or hexadecimal**
    - **Requires the inclusion of the `<iomanip>` header file**
  - **Stream base values are sticky**
    - **Remain until explicitly changed to another base value**

Outline

Fig25_08.cpp

(1 of 2)

```
1  // Fig. 25.8: Fig25_08.cpp
2  // Using stream manipulators hex, oct, dec and setbase.
3  #include <iostream>
4  using std::cin;
5  using std::cout;
6  using std::dec;
7  using std::endl;
8  using std::hex;
9  using std::oct;
10
11 #include <iomanip>
12 using std::setbase;
13
```

Parameterized stream manipulator **`setbase`** is in header file **`<iomanip>`**

Outline

Fig25_08.cpp

(2 of 2)

```
14 int main()
15 {
16    int number;
17
18    cout << "Enter a decimal number: ";
19    cin >> number; // input number
20
21    // use hex stream manipulator to show hexadecimal number
22    cout << number << " in hexadecimal is: " << hex
23       << number << endl;
24
25    // use oct stream manipulator to show octal number
26    cout << dec << number << " in octal is: "
27       << oct << number << endl;
28
29    // use setbase stream manipulator to show decimal number
30    cout << setbase( 10 ) << number << " in decimal is: "
31       << number << endl;
32    return 0;
33 } // end main
```

Set base to hexadecimal

Set base to octal

Reset base to decimal

```
Enter a decimal number: 20
20 in hexadecimal is: 14
20 in octal is: 24
20 in decimal is: 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 x 10^2 in scientific**
    - **150.3 is 1.503E+02 in exponential**

Conversion specifier	Description
`e` or `E`	Display a floating-point value in exponential notation.
`f`	Display floating-point values in fixed-point notation.
`g` or `G`	Display a floating-point value in either the floating-point form `f` or the exponential form `e` (or `E`), based on the magnitude of the value.
`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( void )
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 */
```

e and **E** specify exponential notation

f specifies fixed-point notation

g and **G** specify either exponential or fixed-point notation depending on the number's size

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

25.7.5 Floating-Point Numbers; Scientific and Fixed Notation (`scientific`, `fixed`)

- **Stream manipulator `scientific`**
  - **Makes floating-point numbers display in scientific format**
- **Stream manipulator `fixed`**
  - **Makes floating-point numbers display with a specific number of digits**
    - **Specified by `precision` or `setprecision`**
- **Without either `scientific` or `fixed`**
  - **Floating-point number's value determines the output format**

Outline

Fig25_18.cpp

```
4  #include <iostream>
5  using std::cout;
6  using std::endl;
7  using std::fixed;
8  using std::scientific;
10 int main()
11 {
12    double x = 0.001234567;
13    double y = 1.946e9;
14
15    // display x and y in default format
16    cout << "Displayed in default format:" << endl
17       << x << '\t' << y << endl;
18
19    // display x and y in scientific format
20    cout << "\nDisplayed in scientific format:" << endl
21       << scientific << x << '\t' << y << endl;
22
23    // display x and y in fixed format
24    cout << "\nDisplayed in fixed format:" << endl
25       << fixed << x << '\t' << y << endl;
26    return 0;
27 } // end main
```

```
Displayed in default format:
0.00123457      1.946e+009

Displayed in scientific format:
1.234567e-003   1.946000e+009

Displayed in fixed format:
0.001235        1946000000.000000
```

25.7.6 Uppercase/Lowercase Control (`uppercase`)

- Stream manipulator `uppercase`
  - Causes hexadecimal-integer values to be output with uppercase **X** and **A**-**F**
  - Causes scientific-notation floating-point values to be output with uppercase **E**
  - These letters output as lowercase by default
  - Reset `uppercase` setting with `nouppercase`

Outline

Fig25_19.cpp

(1 of 1)

```
1  // Fig. 25.19: Fig25_19.cpp
2  // Stream manipulator uppercase.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6  using std::hex;
7  using std::showbase;
8  using std::uppercase;
9
10 int main()
11 {
12    cout << "Printing uppercase letters in scientific" << endl
13       << "notation exponents and hexadecimal values:" << endl;
14
15    // use std:uppercase to display uppercase letters; use std::hex and
16    // std::showbase to display hexadecimal value and its base
17    cout << uppercase << 4.345e10 << endl
18       << hex << showbase << 123456789 << endl;
19    return 0;
20 } // end main
```

```
Printing uppercase letters in scientific
notation exponents and hexadecimal values:
4.345E+010
0X75BCD15
```

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'`)

Common Programming Error 9.6

Using double quotes around a character constant creates a pointer to a string consisting of two characters, the second of which is the terminating null. A character constant is a single character enclosed in single quotes.

Outline

fig09_05.c

```
1  /* Fig 9.5: fig09_05c */
2  /* Printing strings and characters */
3  #include <stdio.h>
4
5  int main( void )
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 */
```

c specifies a character will be printed

s specifies a string will be printed

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

9.8 Printing with Field Widths and Precision

- **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`**

Outline

fig09_08.c

```
1  /* Fig 9.8: fig09_08.c */
2  /* Printing integers right-justified */
3  #include <stdio.h>
4
5  int main( void )
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 */
```

A field width of 4 will make C attempt to print the number in a 4-character space

Note that C considers the minus sign a character

```
   1
  12
 123
1234
12345

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

The field width does not work if the provided width is not enough !!

9.8 Printing with Field Widths and Precision

- **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`

Outline

`fig09_09.c`

```
2  /* Using precision while printing integers,
3     floating-point numbers, and strings */
4  #include <stdio.h>
5
6  int main( void )
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 */
```

Precision for integers specifies the minimum number of characters to be printed

Precision for **`f`** and **`e`** specifiers controls the number of digits after the decimal point

Precision for the **`g`** specifier controls the maximum number of significant digits printed

Precision for strings specifies the maximum number of characters to be printed

```
Using precision for integers
        0873
        000000873

Using precision for floating-point numbers
        123.945
        1.239e+002
        124

Using precision for strings
        Happy Birth
```

25.6.2 Floating-Point Precision (`precision`, `setprecision`)

- **Precision of floating-point numbers**
  - **Number of digits displayed to the right of the decimal point**
  - **`setprecision` parameterized stream manipulator**
  - **`precision` member function**
    - **When called with no arguments, returns the current precision setting**
  - **Precision settings are sticky**
    - **Remain until explicitly changed**

Outline

Fig25_09.cpp

(1 of 2)

```
1  // Fig. 25.9: Fig25_09.cpp
2  // Controlling precision of floating-point values.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6  using std::fixed;
7
8  #include <iomanip>
9  using std::setprecision;
10
11 #include <cmath>
12 using std::sqrt; // sqrt prototype
13
14 int main()
15 {
16    double root2 = sqrt( 2.0 ); // calculate square root of 2
17    int places; // precision, vary from 0-9
18
19    cout << "Square root of 2 with precisions 0-9." << endl
20       << "Precision set by ios_base member function "
21       << "precision:" << endl;
22
23    cout << fixed; // use fixed-point notation
24
25    // display square root using ios_base function precision
26    for ( places = 0; places <= 9; places++ )
27    {
28       cout.precision( places );
29       cout << root2 << endl;
30    } // end for
```

Use member function **`precision`** to set **`cout`** to display **`places`** digits to the right of the decimal point

Outline

Fig25_09.cpp

(2 of 2)

```
31
32    cout << "\nPrecision set by stream manipulator "
33       << "setprecision:" << endl;
34
35    // set precision for each digit, then display square root
36    for ( places = 0; places <= 9; places++ )
37       cout << setprecision( places ) << root2 << endl;
38
39    return 0;
40 } // end main
```

Use parameterized stream manipulator **`setprecision`** to set **`cout`** to display **`places`** digits to the right of the decimal point

```
Square root of 2 with precisions 0-9.
Precision set by ios_base member function precision:
1
1.4
1.41
1.414
1.4142
1.41421
1.414214
1.4142136
1.41421356
1.414213562

Precision set by stream manipulator setprecision:
1
1.4
1.41
1.414
1.4142
1.41421
1.414214
1.4142136
1.41421356
1.414213562
```

25.6.3 Field Width (`width`, `setw`)

- **(for `ostream`) Number of character positions in which value is outputted**
  - **Fill characters are inserted as padding**
  - **Values wider than the field are not truncated**
- **(for `istream`) Maximum number of characters inputted**
  - **For `char` array, maximum of one fewer characters than the width will be read (to accommodate null character)**
- **Member function `width` of base class `ios_base`**
  - **Sets the field width**
  - **Returns the previous width**
    - **`width` function call with no arguments just returns the current setting**
- **Parameterized stream manipulator `setw`**
  - **Sets the field width**
- **Field width settings are not sticky**

Common Programming Error 25.1

The width setting applies only for the next insertion or extraction (i.e., the width setting is not “sticky”); afterward, the width is set implicitly to 0 (i.e., input and output will be performed with default settings). Assuming that the width setting applies to all subsequent outputs is a logic error.

Outline

Fig25_10.cpp

```
8  int main()
9  {
10    int widthValue = 4;
11    char sentence[ 10 ];
12
13    cout << "Enter a sentence:" << endl;
14    cin.width( 5 ); // input only 5 characters from sentence
15
16    // set field width, then display characters based on that width
17    while ( cin >> sentence )
18    {
19       cout.width( widthValue++ );
20       cout << sentence << endl;
21       cin.width( 5 ); // input 5 more characters from sentence
22    } // end while
23
24    return 0;
25 } // end main
```

```
Enter a sentence:
This is a test of the width member function
This
   is
      a
   test
      of
      the
      widt
          h
        memb
           er
         func
          tion
```

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 printed only if a digit follows it.) For `g` and `G` specifiers, trailing zeros are not eliminated.
`0` (zero)	Pad a field with leading zeros.

Outline

fig09_11.c

```
1  /* Fig 9.11: fig09_11.c */
2  /* Right justifying and left justifying values */
3  #include <stdio.h>
4
5  int main( void )
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
10    return 0; /* indicates successful termination */
11
12 } /* end main */
```

`-` flag left justifies characters in a field

```
     hello         7         a  1.230000

hello     7         a         1.230000
```

fig09_12.c

```
1  /* Fig 9.12: fig09_12.c */
2  /* Printing numbers with and without the + flag */
3  #include <stdio.h>
4
5  int main( void )
6  {
7     printf( "%d\n%d\n", 786, -786 );
8     printf( "%+d\n%+d\n", 786, -786 );
9
10    return 0; /* indicates successful termination */
11
12 } /* end main */
```

`+` flag forces a plus sign on positive numbers

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

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( void )
7  {
8     printf( "% d\n% d\n", 547, -547 );
9
10    return 0; /* indicates successful termination */
11
12 } /* end main */
```

Space flag forces a space on positive numbers

```
 547
-547
```

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( void )
6  {
7     printf( "%+09d\n", 452 );
8     printf( "%09d\n", 452 );
9
10    return 0; /* indicates successful termination */
11
12 } /* end main */
```

0 flag fills empty spaces with zeros

```
+00000452
000000452
```

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( void )
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 */
```

flag prefixes a **0** before octal integers

flag prefixes a **0x** before hexadecimal integers

flag forces a decimal point on floating-point numbers with no fractional part

```
02623
0x593
0X593

1427
1427.00
```

25.7.1 Trailing Zeros and Decimal Points (`showpoint`)

- **Stream manipulator `showpoint`**
  - **Floating-point numbers are output with decimal point and trailing zeros**
    - **Example**
      - **`79.0` prints as `79.0000` instead of `79`**
  - **Reset `showpoint` setting with `noshowpoint`**

Outline

Fig25_13.cpp

```
4  #include <iostream>
5  using std::cout;
6  using std::endl;
7  using std::showpoint;
8
9  int main()
10 {
11    // display double values with default stream format
12    cout << "Before using showpoint" << endl
13       << "9.9900 prints as: " << 9.9900 << endl
14       << "9.9000 prints as: " << 9.9000 << endl
15       << "9.0000 prints as: " << 9.0000 << endl << endl;
16
17    // display double value after showpoint
18    cout << showpoint
19       << "After using showpoint" << endl
20       << "9.9900 prints as: " << 9.9900 << endl
21       << "9.9000 prints as: " << 9.9000 << endl
22       << "9.0000 prints as: " << 9.0000 << endl;
23    return 0;
24 } // end main
```

```
Before using showpoint
9.9900 prints as: 9.99
9.9000 prints as: 9.9
9.0000 prints as: 9

After using showpoint
9.9900 prints as: 9.99000
9.9000 prints as: 9.90000
9.0000 prints as: 9.00000
```

25.7.2 Justification (`left`, `right` and `internal`)

- **Justification in a field**
  - **Manipulator `left`**
    - **fields are left-justified**
    - **padding characters to the right**
  - **Manipulator `right`**
    - **fields are right-justified**
    - **padding characters to the left**
  - **Manipulator `internal`**
    - **signs or bases on the left**
      - **`showpos` forces the plus sign to print**
    - **magnitudes on the right**
    - **padding characters in the middle**

Outline

Fig25_14.cpp

(1 of 2)

```
1  // Fig. 25.14: Fig15_14.cpp
2  // Demonstrating left justification and right justification.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6  using std::left;
7  using std::right;
8
9  #include <iomanip>
10 using std::setw;
11
12 int main()
13 {
14    int x = 12345;
15
16    // display x right justified (default)
17    cout << "Default is right justified:" << endl
18       << setw( 10 ) << x;
19
20    // use left manipulator to display x left justified
21    cout << "\n\nUse std::left to left justify x:\n"
22       << left << setw( 10 ) << x;
23
24    // use right manipulator to display x right justified
25    cout << "\n\nUse std::right to right justify x:\n"
26       << right << setw( 10 ) << x << endl;
27    return 0;
28 } // end main
```

```
Default is right justified:
     12345

Use std::left to left justify x:
12345

Use std::right to right justify x:
     12345
```

Outline

Fig25_14.cpp

(2 of 2)


```
1  // Fig. 15.15: Fig15_15.cpp
2  // Printing an integer with internal spacing and plus sign.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6  using std::internal;
7  using std::showpos;
8
9  #include <iomanip>
10 using std::setw;
11
12 int main()
13 {
14    // display value with internal spacing and plus sign
15    cout << internal << showpos << setw( 10 ) << 123 << endl;
16    return 0;
17 } // end main
+      123
```

Outline

Fig25_15.cpp

(1 of 1)

25.7.3 Padding (`fill`, `setfill`)

- **Padding in a field**
  - **Fill characters are used to pad a field**
    - **Member function `fill`**
      - **Specifies the fill character**
        - **Spaces are used if no value is specified**
      - **Returns the prior fill character**
    - **Stream manipulator `setfill`**
      - **Specifies the fill character**

Outline

Fig25_16.cpp

(1 of 2)

```
1  // Fig. 25.16: Fig25_16.cpp
2  // Using member function fill and stream manipulator setfill to change
3  // the padding character for fields larger than the printed value.
4  #include <iostream>
5  using std::cout;
6  using std::dec;
7  using std::endl;
8  using std::hex;
9  using std::internal;
10 using std::left;
11 using std::right;
12 using std::showbase;
13
14 #include <iomanip>
15 using std::setfill;
16 using std::setw;
17
18 int main()
19 {
20    int x = 10000;
21
22    // display x
23    cout << x << " printed as int right and left justified\n"
24       << "and as hex with internal justification.\n"
25       << "Using the default pad character (space):" << endl;
26
27    // display x with base
28    cout << showbase << setw( 10 ) << x << endl;
29
```

Outline

Fig25_16.cpp

(2 of 2)

```
30    // display x with left justification
31    cout << left << setw( 10 ) << x << endl;
32
33    // display x as hex with internal justification
34    cout << internal << setw( 10 ) << hex << x << endl << endl;
35
36    cout << "Using various padding characters:" << endl;
37
38    // display x using padded characters (right justification)
39    cout << right;
40    cout.fill( '*' );
41    cout << setw( 10 ) << dec << x << endl;
42
43    // display x using padded characters (left justification)
44    cout << left << setw( 10 ) << setfill( '%' ) << x << endl;
45
46    // display x using padded characters (internal justification)
47    cout << internal << setw( 10 ) << setfill( '^' ) << hex
48       << x << endl;
49    return 0;
50 } // end main
```

```
10000 printed as int right and left justified
and as hex with internal justification.
Using the default pad character (space):
     10000
10000
0x    2710

Using various padding characters:
*****10000
10000%%%%%
0x^^^^2710
```

25.7.4 Integral Stream Base (`dec`, `oct`, `hex`, `showbase`)

- **Integral base with stream insertion**
  - **Manipulators `dec`, `hex` and `oct`**
- **Integral base with stream extraction**
  - **Integers prefixed with `0` (zero) → octal values**
  - **Integers prefixed with `0x` or `0X` → hexadecimal values**
  - **All other integers → treated as decimal values**
- **Stream manipulator `showbase`**
  - **Forces integral values to be outputted with their bases**
    - **Decimal numbers are output by default**
    - **Leading `0` for octal numbers**
    - **Leading `0x` or `0X` for hexadecimal numbers**
  - **Reset the `showbase` setting with `noshowbase`**

Outline

Fig25_17.cpp

(1 of 1)

```
1  // Fig. 25.17: Fig25_17.cpp
2  // Using stream manipulator showbase.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6  using std::hex;
7  using std::oct;
8  using std::showbase;
9
10 int main()
11 {
12    int x = 100;
13
14    // use showbase to show number base
15    cout << "Printing integers preceded by their base:" << endl
16       << showbase;
17
18    cout << x << endl; // print decimal value
19    cout << oct << x << endl; // print octal value
20    cout << hex << x << endl; // print hexadecimal value
21    return 0;
22 } // end main
```

```
Printing integers preceded by their base:
100
0144
0x64
```

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
`\'` (single quote)	Output the single quote (`'`) character.
`\"` (double quote)	Output the double quote (`"`) character.
`\?` (question mark)	Output the question mark (`?`) character.
`\\` (backslash)	Output the backslash (`\`) character.
`\a` (alert or bell)	Cause an audible (bell) or visual alert.
`\b` (backspace)	Move the cursor back one position on the current line.
`\f` (new page or form feed)	Move the cursor to the start of the next logical page.
`\n` (newline)	Move the cursor to the beginning of the next line.
`\r` (carriage return)	Move the cursor to the beginning of the current line.
`\t` (horizontal tab)	Move the cursor to the next horizontal tab position.
`\v` (vertical tab)	Move the cursor to the next vertical tab position.

Fig. 9.16 | Escape sequences.

9.11 Formatting Input with `scanf`

- `scanf`
  - **Input can be formatted much like output can**
  - **`scanf` conversion specifiers are slightly different from those used with `printf`**

Good Programming Practice 9.2

When inputting data, prompt the user for one data item or a few data items at a time. Avoid asking the user to enter many data items in response to a single prompt.

Good Programming Practice 9.3

Always consider what the user and your program will do when (not if) incorrect data is entered—for example, a value for an integer that is nonsensical in a program's context, or a string with missing punctuation or spaces.

Conversion specifier	Description
Integers	
`d`	Read an optionally signed decimal integer. The corresponding argument is a pointer to an `int`.
`i`	Read an optionally signed decimal, octal or hexadecimal integer. The corresponding argument is a pointer to an `int`.
`o`	Read an octal integer. The corresponding argument is a pointer to an unsigned `int`.
`u`	Read an unsigned decimal integer. The corresponding argument is a pointer to an unsigned `int`.
`x` or `X`	Read a hexadecimal integer. The corresponding argument is a pointer to an unsigned `int`.
`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`**. (Part 1 of 3.)

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. The corresponding argument is a pointer to a `double` or `long double` variable.
Characters and strings	
`c`	Read a character. The corresponding argument is a pointer to a `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.

Fig. 9.17 | Conversion specifiers for **`scanf`**. (Part 2 of 3.)

Conversion specifier	Description
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 call to `scanf`. The corresponding argument is a pointer to an `int`.
%	Skip a percent sign (%) in the input.

Fig. 9.17 | Conversion specifiers for **`scanf`**. (Part 3 of 3.)

Outline

fig09_18.c

```
1  /* Fig 9.18: fig09_18.c */
2  /* Reading integers */
3  #include <stdio.h>
4
5  int main( void )
6  {
7     int a;
8     int b;
9     int c;
10    int d;
11    int e;
12    int f;
13    int 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 */
```

d specifies a decimal integer will be input

i specifies an integer will be input

o specifies an octal integer will be input

u specifies an unsigned decimal integer will be input

x specifies a hexadecimal integer will be input

```
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_19.c

```
1  /* Fig 9.19: fig09_19.c */
2  /* Reading floating-point numbers */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main( void )
7  {
8     double a;
9     double b;
10    double c;
11
12    printf( "Enter three floating-point numbers: \n" );
13    scanf( "%le%lf%lg", &a, &b, &c );
14
15    printf( "Here are the numbers entered in plain\n" );
16    printf( "floating-point notation:\n" );
17    printf( "%f\n%f\n%f\n", a, b, c );
18
19    return 0; /* indicates successful termination */
20
21 } /* end main */
```

e, **f**, and **g** specify a floating-point number will be input

l specifies a **double** or **long double** will be input

```
Enter three floating-point numbers:
1.27987 1.27987e+03 3.38476e-06
Here are the numbers entered in plain
floating-point notation:
1.279870
1279.870000
0.000003
```

Outline

fig09_20.c

```
1  /* Fig 9.20: fig09_20.c */
2  /* Reading characters and strings */
3  #include <stdio.h>
4
5  int main( void )
6  {
7     char x;
8     char y[ 9 ];
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 */
```

c specifies a character will be input

s specifies a string will be input

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

Outline

fig09_21.c

```
1  /* Fig 9.21: fig09_21.c */
2  /* Using a scan set */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main( void )
7  {
8     char z[ 9 ]; /* define array z */
9
10    printf( "Enter string: " );
11    scanf( "%[aeiou]", z ); /* search for set of characters */
12
13    printf( "The input was \"%s\"\n", z );
14
15    return 0; /* indicates successful termination */
16
17 } /* end main */
```

[] specifies only the initial segment of a string that contains the characters in brackets will be read

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

Outline

fig09_22.c

```
1  /* Fig 9.22: fig09_22.c */
2  /* Using an inverted scan set */
3  #include <stdio.h>
4
5  int main( void )
6  {
7     char z[ 9 ];
8
9     printf( "Enter a string: " );
10    scanf( "%[^aeiou]", z ); /* inverted scan set */
11
12    printf( "The input was \"%s\"\n", z );
13
14    return 0; /* indicates successful termination */
15
16 } /* end main */
```

[] and **^** specify only the initial segment of a string that does **not** contain the characters in brackets will be read

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

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( void )
6  {
7     int x;
8     int 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 */
```

A field width of 2 tells C to only read the first 2 characters of that input

```
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( void )
6  {
7     int month1;
8     int day1;
9     int year1;
10    int month2;
11    int day2;
12    int 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 */
```

***** is a wildcard—**`scanf`** will disregard anything between the two inputs on either side of it

```
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
```