Input/output

Victor Eijkhout and Carrie Arnold and Charlie Dey

Fall 2017

Formatted output

Formatted output

- `cout` uses default formatting
- Possible: pad a number, use limited precision, format as hex/base2, etc
- Many of these output modifiers need

```
#include <iomanip>
```

Default unformatted output

Code:

```
for (int i=1; i<200000000; i*=10)
  cout << "Number: " << i << endl;
cout << endl;
```

Output:

Reserve space

You can specify the number of positions, and the output is right aligned in that space by default:

Code:

```
  cout << "Width is 6:" << endl;
  for (int i=1; i<200000000; i*=10)
    cout << "Number: "
 << setw(6) << i << endl;
  cout << endl;
```

Output:

Padding character

Normally, padding is done with spaces, but you can specify other characters:

Code: **Output:**

```
  for (int i=1; i<200000000; i*=10)
    cout << "Number: "
<< setfill('.') << setw(6) << i << endl;
  cout << endl;
```

Note: single quotes denote characters, double quotes denote strings.

Left alignment

Instead of right alignment you can do left:

Code: **Output:**

```
  for (int i=1; i<200000000; i*=10)
    cout << "Number: "
<< left << setfill('.') << setw(6) << i << endl;
```

Number base

Finally, you can print in different number bases than 10:

Code: **Output:**

```
cout << setbase(16) << setfill(' ');
for (int i=0; i<16; i++) {
  for (int j=0; j<16; j++)
    cout << i*16+j << " " ;
  cout << endl;
}
```

Exercise 1

Make the above output more nicely formatted:

```
00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f
20 21 22 23 24 25 26 27 28 29 2a 2b 2c 2d 2e 2f
etc
```

Exercise 2

Use integer output to print fixed point numbers aligned on the decimal:

```
  1.345
 23.789
456.1234
```

Use four spaces for both the integer and fractional part.

Hexadecimal

Hex output is useful for pointers (chapter **??**):

```
int i;
cout << "address of i, decimal: " << (long)&i << endl;
cout << "address if i, hex    : " << std::hex << &i << endl
```

Back to decimal:

```
cout << hex << i << dec << j;
```

Floating point formatting

Floating point precision

Use `setprecision` to set the number of digits before and after decimal point:

```
x = 1.234567;
for (int i=0; i<10; i++) {
  cout << setprecision(4) << x << endl;
  x *= 10;
}
```

Output

```
1.235
12.35
123.5
1235
1.235e+04
1.235e+05
1.235e+06
1.235e+07
1.235e+08
1.235e+09
```

(Notice the rounding)

Fixed point precision

Fixed precision applies to fractional part:

```
cout << "Fixed precision applies to fractional part:" << en
x = 1.234567;
cout << fixed;
for (int i=0; i<10; i++) {
  cout << setprecision(4) << x << endl;
  x *= 10;
}
```

Output

```
1.2346
12.3457
123.4567
1234.5670
12345.6700
123456.7000
1234567.0000
12345670.0000
123456700.0000
1234567000.0000
```

Aligned fixed point output

Combine width and precision:

```
x = 1.234567;
cout << fixed;
for (int i=0; i<10; i++) {
  cout << setw(10) << setprecision(4) << x << endl;
  x *= 10;
}
```

Output

```
    1.2346
   12.3457
  123.4567
 1234.5670
12345.6700
123456.7000
1234567.0000
12345670.0000
123456700.0000
1234567000.0000
```

Scientific notation

```
cout << "Combine width and precision:" << endl;
x = 1.234567;
cout << scientific;
for (int i=0; i<10; i++) {
  cout << setw(10) << setprecision(4) << x << endl;
  x *= 10;
}
```

Output

```
Combine width and precision:
1.2346e+00
1.2346e+01
1.2346e+02
1.2346e+03
1.2346e+04
1.2346e+05
1.2346e+06
1.2346e+07
1.2346e+08
1.2346e+09
```

File output

Text output to file

Streams are general: work the same for console out and file out.

```
#include <fstream>
```

Use:

```
ofstream file_out;
file_out.open("fio_example.out");
/* ... */
file_out << number << endl;
file_out.close();
```

Binary output

```
ofstream file_out;
file_out.open("fio_binary.out",ios::binary);
/* ... */
file_out.write( (char*)(&number),4);
```