Input/output

Victor Eijkhout, Harika Gurram,
Je'aime Powell, Charley Dey

Fall 2018

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 from running cunformat in code directory io:

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;
  cout << "Width is 6:" << endl;
  cout << setw(6) << 1 << 2 << 3 << endl;
  cout << endl;
```

Output from running width in code directory io:

Padding character

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

Code:

```
#include <iomanip>
using std::setfill;
using std::setw;
//codesnippet formatpad
  /* ... */

int main() {

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

Output from running formatpad in code directory io:

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

Left alignment

Instead of right alignment you can do left:

Code:

```
#include <iomanip>
using std::left;
using std::setfill;
using std::setw;
  /* ... */
  for (int i=1; i<200000000; i*=10)
    cout << "Number: "
         << left << setfill('.') << setw(6)
 << i << endl;
```

Output from running formatleft in code directory io:

Number base

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

Code:

```
#include <iomanip>
using std::setbase;
using std::setfill;
  /* ... */
  cout << setbase(16) << setfill(' ');
  for (int i=0; i<16; i++) {
    for (int j=0; j<16; j++)
      cout << i*16+j << " " ;
    cout << endl;
  }
```

Output from running format16 in code directory io:

Exercise 1

Code:

```
#include <iomanip>
using std::right;
using std::setbase;
using std::setfill;
using std::setw;
  /* ... */
  cout << setbase(16) << setfill('0') << right ;
  for (int i=0; i<16; i++) {
    for (int j=0; j<16; j++)
      cout << setw(2) << i*16+j << " " ;
    cout << endl;
  }
```

Output from running block16 in code directory io:

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:

Code:

```
#include <iomanip>
using std::left;
using std::setfill;
using std::setw;
using std::setprecision;
  /* ... */
  x = 1.234567;
  for (int i=0; i<10; i++) {
    cout << setprecision(4) << x << endl;
    x *= 10;
  }
```

Output from running formatfloat in code directory io:

(Notice the rounding)

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:

Code:

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

Output from running fix in code directory io:

Aligned fixed point output

Combine width and precision:

Code:

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

Output from running align in code directory io:

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:

```
#include <fstream>
using std::ofstream;
  /* ... */
  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);
```