

# Week 3: File I/O and Formatting

Gaddis: 3.7, 3.8, 5.11

CS 1428  
Fall 2014

Jill Seaman

1

# 3.7 Formatting Output

- Formatting: the way a value is printed:
  - spacing
  - decimal points, fractional values, number of digits
  - scientific notation or decimal format
- `cout` has a standard way of formatting values of each data type
- use “stream manipulators” to override this
- they require `#include <iomanip>`

2

## Formatting Output: `setw`

- `setw` is a “stream manipulator”, like `endl`
- `setw(n)` specifies the minimum width for the **next** item to be output
  - `cout << setw(6) << age << endl;`
  - print in a field at least 6 spaces wide.
  - value is right justified (padded with spaces on left).
  - if the value is too big to fit in 6 spaces, it is printed in full, using more spaces.

3

## `setw`: examples

- Example with no formatting:

```
cout << 2897 << " " << 5 << " " << 837 << endl;  
cout << 34 << " " << 7 << " " << 1623 << endl;
```

```
2897 5 837  
34 7 1623
```

- Example using `setw`:

```
cout << setw(6) << 2897 << setw(6) << 5  
    << setw(6) << 837 << endl;  
cout << setw(6) << 34 << setw(6) << 7  
    << setw(6) << 1623 << endl;
```

```
2897    5    837  
34     7    1623
```

4

## Formatting Output: setprecision

- `setprecision(n)` specifies the number of significant digits to be output for floating point values.
- it remains in effect until it is changed
- the default seems to be 6, and it rounds up

```
cout << 123.45678 << endl;
cout << setprecision(4);
cout << 1.3 << endl;
cout << 123.45678 << endl;
cout << setprecision(2) << 34.21;
```

```
123.457
1.3
123.5
34
```

5

## Formatting Output: fixed

- `fixed` forces floating point values to be output in decimal format, and not scientific notation.
- when used with `setprecision`, the value of `setprecision` is used to determine the number of digits after the decimal

```
cout << 12345678901.23 << endl;
cout << fixed << setprecision(2);
cout << 12345678901.23 << endl;
cout << 123.45678 << endl;
```

```
1.23457e+10
12345678901.23
123.46
```

Note: there is no need for `showpoint` when using `setprecision` with `fixed`

6

## Formatting Output: right and left

- `left` causes all subsequent output to be left justified in its field
- `right` causes all subsequent output to be right justified in its field. This is the default.

```
double x = 146.789, y = 24.2, z = 1.783;
cout << setw(10) << x << endl;
cout << setw(10) << y << endl;
cout << setw(10) << z << endl;
```

The examples in the book are WRONG

```
146.789
 24.2
 1.783
```

```
double x = 146.789, y = 24.2, z = 1.783;
cout << left << setw(10) << x << endl;
cout << setw(10) << y << endl;
cout << setw(10) << z << endl;
```

```
146.789
24.2
1.783
```

7

## 3.8 Working with characters and string objects

- Using the `>>` operator to input strings (and chars) can cause problems:
- It skips over any leading whitespace chars (spaces, tabs, or line breaks)
- It stops reading strings when it encounters the next whitespace character!

```
string name;
cout << "Please enter your name: ";
cin >> name;
cout << "Your name is " << name << endl;
```

```
Please enter your name: Kate Smith
Your name is Kate
```

8

## Using `getline` to input strings

- To work around this problem, you can use a C++ function named `getline`.
- `getline(cin, var);` reads in an entire line, including all the spaces, and stores it in a string variable.

```
string name;
cout << "Please enter your name: ";
getline(cin, name);
cout << "Your name is " << name << endl;
```

```
Please enter your name: Kate Smith
Your name is Kate Smith
```

9

## Using `cin.get` to input chars

- To read a single character:
- Can use `>>`:
- Use `cin.get()`:

```
char ch;
cout << "Press any key to continue";
cin >> ch;
```

- ▶ Problem: will skip over blanks, tabs, newlines to get to the first non-whitespace char.

```
char ch;
cout << "Press any key to continue";
cin.get(ch);
```

- ▶ Will read the next character entered, even whitespace

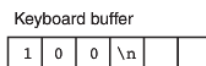
10

## Mixing `>>` with `getline` and `cin.get`

- Mixing `cin>>x` with `getline(cin,y)` or `cin.get(ch)` in the same program can cause input errors that are hard to detect

```
int number;
string name;
cout << "Enter a number: ";
cin >> number; // Read an integer
cout << "Enter a name: ";
getline(cin,name); // Read a string
cout << "Name " << name << endl;
```

```
Enter a number: 100
Enter a name: Name
```



The program did not allow me to type a name

cin stops reading here, but does not read the \n character.

`getline(cin,name)` then reads the \n and immediately stops (name is empty)

11

## Using `cin.ignore`

- `cin.ignore(20, '\n')` skips the next 20 characters, or until `'\n'` is encountered.
- Use it before a `getline` to consume the newline so it will start reading characters from the following line.

```
int number;
string name;
cout << "Enter a number: ";
cin >> number; // Read an integer
cin.ignore(20, '\n'); // skip the newline
cout << "Enter a name: ";
getline(cin,name); // Read a string
cout << "Name " << name << endl;
```

```
Enter a number: 100
Enter a name: Jane Doe
Name Jane Doe
```

12

## 5.11 Using Files for Data Storage

- Variables are stored in Main Memory/RAM
  - values are lost when program is finished executing
- To preserve the values computed by the program: save them to a file
- Files are stored in Secondary Storage
- To have your program manipulate values stored in a file, they must be input into variables first.

13

## File Stream Objects

- File stream data types:
  - ifstream
  - ofstream
- use `#include <fstream>` for these
- objects of type `ofstream` can output (write) values to a file. (like `cout`)
- objects of type `ifstream` can input (read) values from a file. (like `cin`)

14

## Steps to File I/O

- Define a file stream variable.
- Open the file
- Use the file
  - `ifstream`: read values from the file
  - `ofstream`: store (write) values to the file
- Close the file

15

## Define and open file stream objects

- To input from a file, declare an `ifstream` variable and open a file by its name.

```
ifstream fin;  
fin.open("mydatafile.txt");
```

- If the file "mydatafile.txt" does not exist, it will cause an error.

- To output to a file, declare an `ofstream` variable, and open a file by its name.

```
ofstream fout;  
fout.open("myoutputfile.txt");
```

- If the file "myoutputfile.txt" does not exist, it will be created.
- If it does exist, it will be overwritten

- The stream variable is associated with the file.

16

## Closing file stream objects

- To close a file stream when you are done reading/writing:

```
fin.close();
fout.close();
```

- Not required, but good practice.

17

## Writing to Files

- Use the stream insertion operator on the file output stream variable:

```
#include <iostream>
#include <fstream>
using namespace std;

int main() {
    ofstream fout;
    outFile.open("demofile.txt");

    int age;
    cout << "Enter your age: ";
    cin >> age;

    fout << "Age is: " << age << endl;
    fout.close();
    return 0;
}
```

Output  
demofile.txt:

```
Age is: 20
```

18

## Reading from Files

- Use the stream extraction operator on the file input stream variable to copy data into variable:

```
#include <iostream>
#include <fstream>
using namespace std;

int main() {
    string name;

    ifstream fin;
    fin.open("Names.txt");
    fin >> name;

    cout << name << endl;
    fin.close();
}
```

Names.txt: Tom  
Dick  
Harry

Screen output: Tom

19

## Reading from files

- When opened, file stream's read position points to first character in file.
- extraction operator (>>) starts at read position and skips whitespace to read data into the variable.
- The read position then points to whitespace after the value it just read.
- The next extraction (>>) starts from the new read position.
- Just like with cin.

20