Ch 3:Expressions and Interactivity
Part 2

CS 1428
Fall 2011

Jill Seaman

Lecture 6

Formatting Output

- Formatting: the way a value is printed:
  - spacing
  - decimal points, fractional values
  - scientific notation
- cout has a standard way of formatting values of each data type
- cout has “stream manipulators” to override the default formatting.
- [use #include <iomanip> for these]

Unformatted Output

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

2897 5 837
34 7 1623
```

- To line up the output, we can specify the (minimum) width for each number

Formatting Output: setw

- setw is a "stream manipulator", like endl
- specifies the minimum width for the next item to be output

```
cout << "(" << setw(6) << 209 << ")";

(   209)
```

- The value is right justified and padded with spaces.

Formatting Output: 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
```

- If the value is too big to fit it's printed in full:

```
cout << "(" << setw(2) << 23456 << ")";

(23456)
```

Formatting Output: setprecision

- setprecision 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

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

123.457
1.3
123.5
34
```

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

1.23457e+10
12345678901.230000
123.46
```

Input: strings

- A string is a sequence of characters
- A string is stored sequentially in memory, withthe null character ('\0') at the end
- A string can be stored in a variable whose type is a “character array”
- An array is a sequence of variables with a single name
- The elements in the array can be accessed by number (first element, second element, etc.)

Input: strings

- an example definition of an array variable:

```
char lastName[15];
```

- the array holds 15 characters, but the last one is '\0', so really only 14.
- Input/Output with character arrays (don't type spaces in the input string):

```
cout << “Enter your last name: “;
cin >> lastName;
cout << “Your last name is: “ << lastName;

Enter your last name: Maxwell
Your last name is Maxwell
```

Formatted Input: setw

- specifies the maximum width for the next item to be input
- used to prevent putting too many characters into an array.

```
char word[5];
cout << “Enter a word: “;
cin >> setw(5) >> word;
cout << “You entered “ << word << endl;

Enter a word: tapioca
You entered tapi
```

Reading a Line of input

- cin.getline(<array>,<size>)
- getline reads <size> - 1 characters from the screen into the char array <array> (and adds '\0' at the end)
- getline reads spaces, doesn't need setw

```
char sentence[60];
cout << “Enter a sentence: “;
cin.getline(sentence, 60);
cout << “You entered “ << sentence << endl;

Enter a sentence: Life is a box of chocolates.
You entered Life is a box of chocolates.
```

Reading a Character

- << skips whitespace, so this code cannot read a space or newline from the screen:

```
char letter;                                Enter a character   j
cout << “Enter a character“;                [j]
cin >> letter;
cout << “[“ << letter << “]”;
```

- cin.get(v) will read the next character typed into v

```
char letter;                                Enter a character   j
cout << “Enter a character“;                [ ]
cin.get(letter);
cout << “[“ << letter “]”;
```