Additional Input Functions

- The extraction operator (`>>`) is one way to read characters from an input stream like `cin`
  - It provides a powerful way to get individual pieces of data (integers, reals, chars, strings) separated by spaces
  - However, user input doesn't always look like that
- The `iostream` library defines other ways to do input
  - The function `getline`()
  - The function `cin.ignore`()

Functions and Operators

- Operator syntax

```
operand1 operator operand2
```

  - Operator is a special symbol
  - All operators are binary (two operands)
  - An operator performs some task and *evaluates* to a value
    - Sometimes you care about the value (e.g. addition)

```
5 + 2
```

    - Sometimes you care about the side-effect (e.g. printing)

```
cout << "Hello"
```

- Function syntax

```
function( parameter2, parameter2 … )
```

  - Function name is an identifier
  - Any number of *parameters* allowed (including none)
  - Also performs some task (set of instructions) and evaluates to a value

```
add( 5, 2 )
insertion( cout, "Hello" )
```

Some Example Functions

- These functions are in the `cmath` library
  - `#include <cmath>` to use them

- To compute the square root of a number:
  - 1 input (float), 1 output (float)

```
answer = sqrt( 16.0 );
```

- To compute the power function (x^y)
  - 2 inputs (float, int), 1 output (float)

```
cout << pow( 2.0, 3 );
```

Back to Input

- Because the extraction operator reads data separated by whitespace, it cannot read a string with whitespace in it
  - Given the code:

```
string s;
cin >> s;
```

  - If the user types “University of Texas”

Back to Input

- Because the extraction operator reads data separated by whitespace, it cannot read a string with whitespace in it
  - Given the code:

```
string s;
cin >> s;
```

  - If the user types “University of Texas”
  - `s` will contain the string “University”

`getline` Function

- To read strings with spaces in them, we use the function `getline()`
  - Takes two arguments (just like extraction):
    - The input stream to read from
    - The (string) variable to store in

    ```
    getline( istreamVar, strVar );
    ```
  - Reads all characters until the end of the line
    - Stores the resulting string in the string variable
  - Evaluates to the stream that was read from
    - To support chaining, but the task (reading) is the main point

`getline` Function

- `getline()` can also take three arguments
  - The input stream to read from
  - The (string) variable to store in
  - A *delimiting* character

```
getline( istreamVar, strVar, delim );
```

- This version reads until it reaches the specified delimiting character
  - If the delimiter is '\n', it reads to the end of the line

```
getline( istreamVar, strVar, '\n' );
```

`cin.ignore` Function

- The function `cin.ignore`
  - The "." is class syntax, which we'll discuss later in the course
    - Just memorize for now
  - Takes two arguments:
    - The number of characters to ignore
    - A delimiting character
  - Reads and discards the specified number of characters
    - Unless it reaches the delimiter first
  - Evaluates to the stream that was read from
    - To support chaining, but the task (reading) is the main point

Input Failure

- Things can go wrong during execution
- If input data does not match corresponding variables, program may run into problems
- Trying to read a letter into an `int` or `double` variable will result in an input failure
- If an error occurs when reading data
  - Input stream enters the fail state

The `clear` Function

- Once in a fail state, all further I/O statements using that stream are ignored
- The program continues to execute with whatever values are stored in variables
  - This causes incorrect results
- The `clear` function restores input stream to a working state

```
cin.clear();
```

  - However, it does not remove the characters that caused the error from the input stream

Exercise

```
int x, y;
string line;
```

For the input:

```
13 28 D
14 E 98
A B 56
```

What are the values of `x`, `y` and `line` after:

```
f.  getline( cin, line, '8' );
    cin.ignore( 50, '\n' );
    cin >> x;
    cin.ignore( 50, 'E' );
    cin >> y;
```