

Files and Streams I

Classes and Objects

A **class**:

- is a mechanism that allows a programmer to define their own, complex data types
- can be used to add functionality to an existing data type or to create an entirely new data type
- combines both data and functionality

An **object**:

- is a variable whose *data type* is some class
- also known as an **instance** of a class

Stream Classes

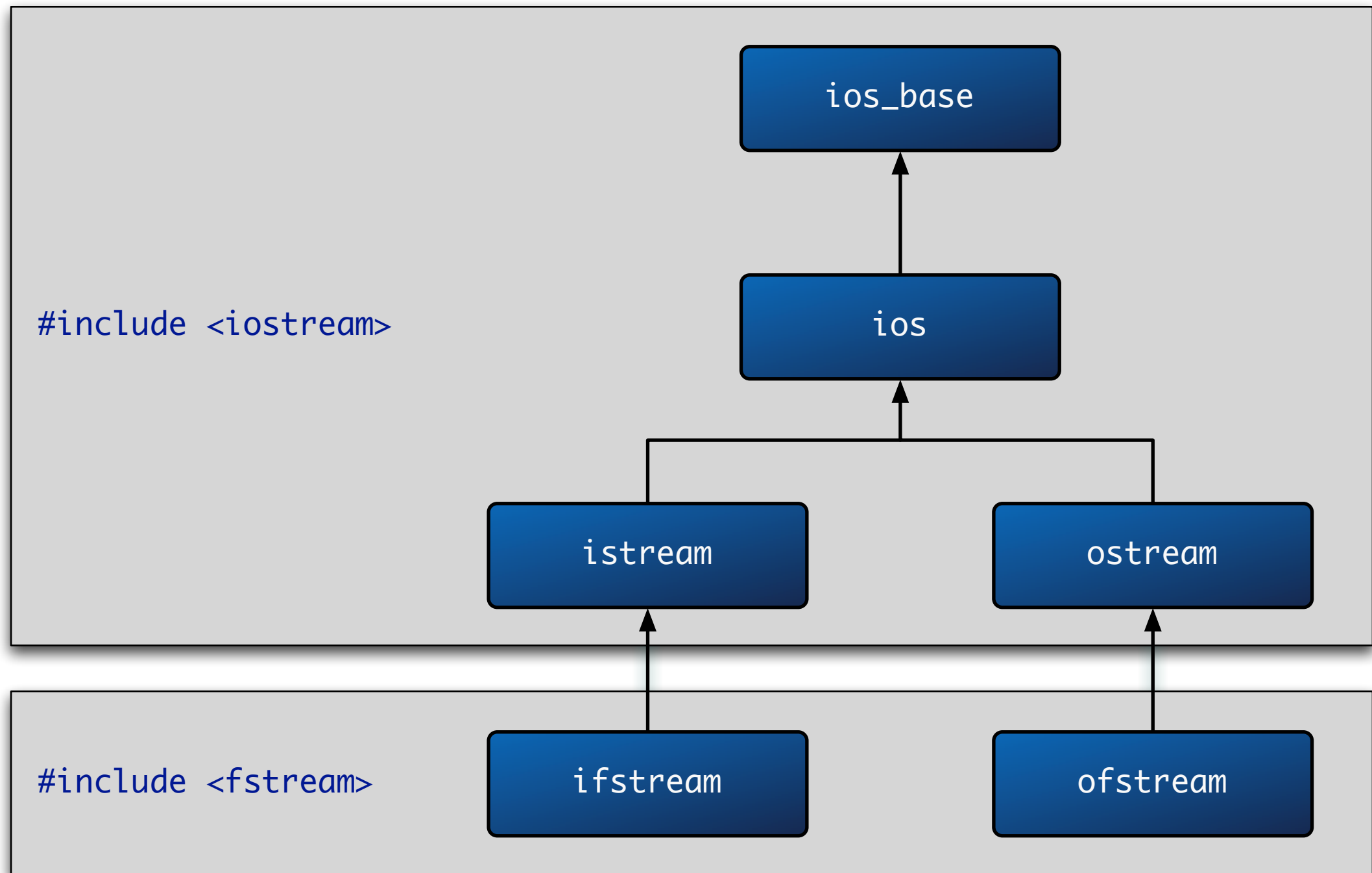
C++ uses *streams* for input and output

Streams are classes!

- they encapsulate data relating to the status and behavior of the stream
 - the state of the stream
 - formatting flags
 - display width
 - display precision
- they contain *methods* (functionality) for manipulating their behavior
 - examples include `setf()`, `precision()`, and `width()`

`cin` and `cout` are both *instances* (or objects) of stream classes

Stream Classes



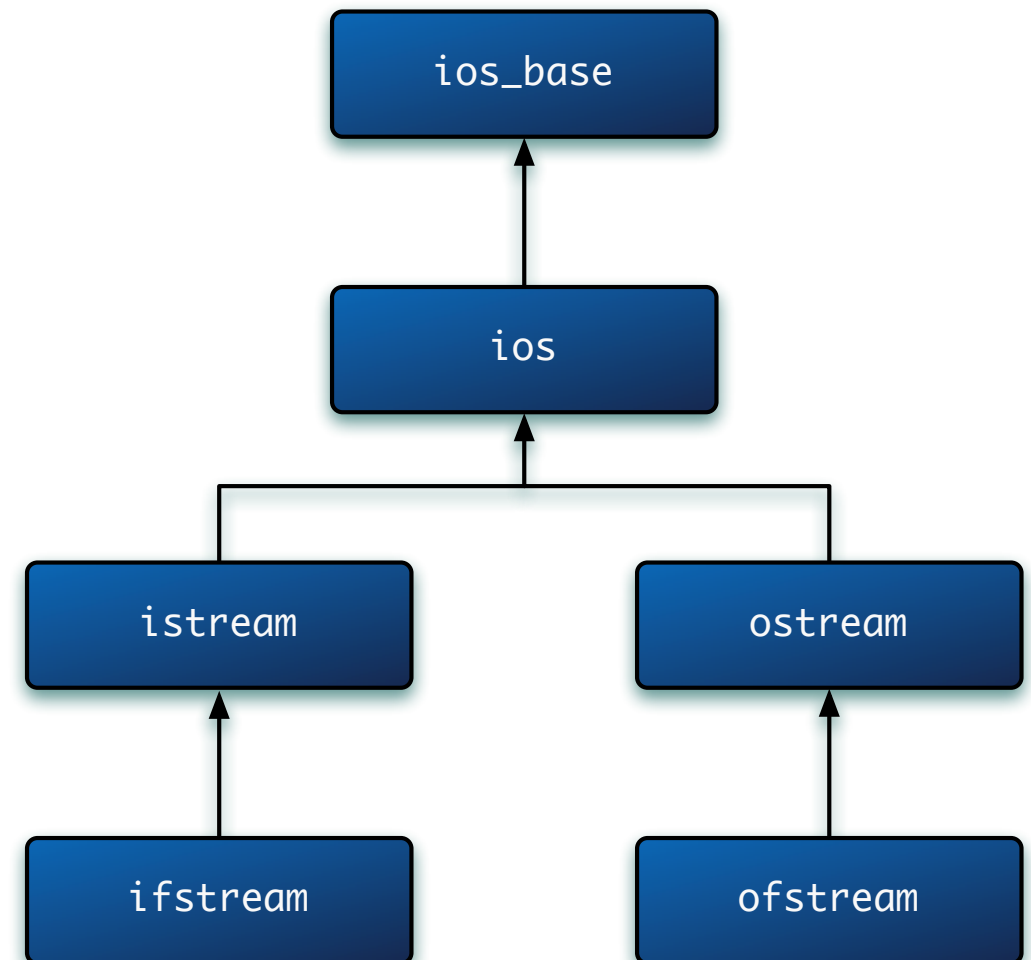
Stream Classes

Notice that:

- ifstream “inherits” from istream
- ofstream “inherits” from ostream

Which means:

- the filestream classes have the exact same functionality as the streams we’ve seen so far, but
- they operate on files instead, and
- we have to create the stream objects ourselves



Stream Classes

ostream

```
1 // cout is an instance of the ostream class
2 cout << "output";
3
```

istream

```
1 // cin is an instance of the istream class
2 cin >> myVar;
3
```

cout and cin are variables that are predefined in the std namespace

```
1 // provides the cout and cin objects
2 using namespace std;
3
```

- they come already linked to the console (that black window that pops up)

Stream Classes

ofstream

```
1 // create a file output stream
2 ofstream out_file("somefile.txt");
3
4 // print something to the file
5 out_file << "Hello to a file!" << endl;
6
7 // close the file once we're done
8 out_file.close();
9
```

ifstream

```
1 // create a file input stream
2 ifstream input_file("somefile.txt");
3
4 // read something from the file
5 input_file >> myVar;
6
7 // close the file once we're done
8 input_file.close();
9
```

We have to create filestream variables ourselves

- creating a filestream object “opens” a file (behind the scenes)
 - also need to check that the file successfully opened (not included above)!
- once created, the filestream works just like cout or cin
- we also have to “close” our filestream objects after we’re done using them

Filestreams

The assignment for today involves reading from a file, so we're going to cover everything you need to get it done:

- creating `ifstream`
- checking for errors
- handling errors
- reading from files
- closing files

On Wednesday, we'll finish up with:

- creating and using `ofstream`
- using `getline()`
- a more detailed discussion of output formatting

Opening a File for Reading

General syntax:

```
// remember to #include <fstream>
ifstream identifier("somefile.txt");
```

This creates an `ifstream` object

- basically just a variable whose data type is `ifstream`
- identifier is the name you'll use to refer to the stream
- you *must* use the parenthesis to specify the file to open (not the assignment operator)
- the file must exist in the same directory as your project

Checking for Failure

The `ifstream` object can be *evaluated* as a boolean value:

- `true` if the file successfully opened
- `false` if there was an error (such as if the file doesn't exist)

```
// create the ifstream object
```

```
ifstream infile("mydata.dat");
```

```
// check for failure using '!infile' as the condition
```

```
if (!infile) {
```

```
    // this code gets executed if there was an error
```

```
}
```

Handling Failures

For this class, if the file didn't open successfully, just print an error message and exit the program. Nice and simple. =)

```
// create the ifstream object
ifstream infile("mydata.dat");

// check for failure
if (!infile) {
    cout << "Unable to open the file!" << endl;
    exit(1);           // end the program
}
```

Reading from a File

You can use an `ifstream` much like you would `cin`

- both use the `>>` operator
- both try to “interpret” the input depending on the data type of the variable you’re using
- both ignore whitespace

The `ifstream` object can be *evaluated* as a boolean value:

- `true` if all input so far has been successful
- `false` if there was an input error (reaching the end of the file or an unexpected value)
 - the stream is considered to be in “failure” mode...
- we will exploit this behavior often =)

Reading from a File

I/O expressions evaluate to the stream object after ‘doing their thing’

This statement:

```
infile >> variable1 >> variable2 >> variable3;
```

gets evaluated like this:

```
infile >> variable1 >> variable2 >> variable3; // read into variable1, return infile
infile >> variable2 >> variable3;                // read into variable2, return infile
infile >> variable3;                             // read into variable3, return infile
infile;                                           // no side effect (but can be evaluated as true or false)
```

We will exploit this behavior often. =)

Reading from a File

You can easily read an entire file (start to finish) using a loop

- use the input statement as the condition of the loop!
 - “infile >> variable” attempts to read from the file and gets evaluated as “infile”
 - “infile” will be considered either true or false, depending on the outcome of the input operation
- upon reaching the end of the file (EOF) or an unexpected value, the stream object will evaluate as **false** and the loop will end

Assume infile was successfully opened and that variable has already been declared:

```
// read from the file until there is no more data
while (infile >> variable) {
    // do something with the input
}
```

Reading from a File

Assume infile was successfully opened and that var1, var2, and var3 have already been declared:

```
// reads data 3 at a time from the file until EOF
while (infile >> var1 >> var2 >> var3) {
    // do something with the input
}
```

Using a while loop like this is the most common way we'll read data from a file!

Closing Files

After you're done using your files, you should explicitly close them:

```
// create the file stream
```

```
ifstream infile("somefile.txt");
```

```
// check for failure and then read some values...
```

```
// close the file stream
```

```
infile.close();
```

`close()` is a member function of `fstream` objects

The File Pointer

C++ keeps track of its current position in a file

- this is done using a behind-the-scenes “file pointer”
- when you first open a file, the file pointer normally begins at the very start of the file
- the file pointer gets advanced every time you read in a value
- when the file pointer reaches the end of the file, there are no more values left to read and subsequent read attempts will cause the file stream to go into “failure” mode

The File Pointer

Part of SomeProgram.cpp:

```
1 // assume infile was successfully opened
2 ifstream infile("somefile.txt");
3
4 // create a variable to store the input
5 string str;
6
7 // reading from the file advances the file pointer!
8 infile >> str;
9 infile >> str;
10 infile >> str;
11 infile >> str;
12 infile >> str;
13 infile >> str;
14 infile >> str; // no more data, so infile goes into "failure" mode
15
```

Somefile.txt:

File Pointer



File Pointer



```
1 This is a simple data file.
2
```

Data Types and Input

The type of the variable you're reading into directly affects how C++ will interpret the input value.

If the variable is an `int`, then only values like 1234, -234, and +234, can be successfully read.

Input Exercises

Given this data file:

1	1000	-60	50.0	10	1E5	30	+80
2	↑		↑				
3	FP		FP				
4	20	400	Hi!	Argh!	>.<	What's	Up, Doc?
5							

1000 -60 50

Where will the filestream encounter an error? What values get read?

```
1 // assume that infile opened successfully
2 ifstream infile("data.txt");
3
4 // declare a variable to store the data
5 int input;
6
7 while (infile >> input) {
8     cout << input << endl;
9 }
10
11 cout << "\nThe file stream is in failure state." << endl;
12 infile.close();
13
```

Input Exercises

Given this data file:

1	1000	-60	50.0	10	1E5	30	+80
2							
3	FP						
4	20	400	Hi!	Argh!	>.<	What's Up, Doc?	
5							

1000.0 -60.0 50.0 10.0 100000.0 30.0 80.0 20.0 400.0

Where will the filestream encounter an error? What values get read?

```
1 // assume that infile opened successfully
2 ifstream infile("data.txt");
3
4 // declare a variable to store the data
5 double input;
6
7 while (infile >> input) {
8     cout << input << endl;
9 }
10
11 cout << "\nThe file stream is in failure state." << endl;
12 infile.close();
13
```

Input Exercises

Given this data file:

```
1 1000    -60    50.0    10    1E5    30    +80
2
3 FP
4 20      400    Hi!     Argh!   >.<    What's Up, Doc?
5
```

'1' '0' '0' '0' '-' '6' '0' '5' '0' '.' '0' '1' '0' '1' 'E' '5' '3' '0' '+' '8' '0' '2' '0' '4' '0'
'0' 'H' 'i' '!' 'A' 'r' 'g' 'h' '!' '>' '.' '<' 'W' 'h' 'a' 't' '\\' 's' 'U' 'p' ',' 'D' 'o' 'c' '?'

Where will the filestream encounter an error? What values get read?

```
1 // assume that infile opened successfully
2 ifstream infile("data.txt");
3
4 // declare a variable to store the data
5 char input;
6
7 while (infile >> input) {
8     cout << input << endl;
9 }
10
11 cout << "\nThe file stream is in failure state." << endl;
12 infile.close();
13
```

Input Exercises

Given this data file:

```
1 1000    -60    50.0    10    1E5    30    +80
2
3 FP
4 20      400    Hi!     Argh!   >.<    What's Up, Doc?
5
```

Immediately! bools read in 0 and 1

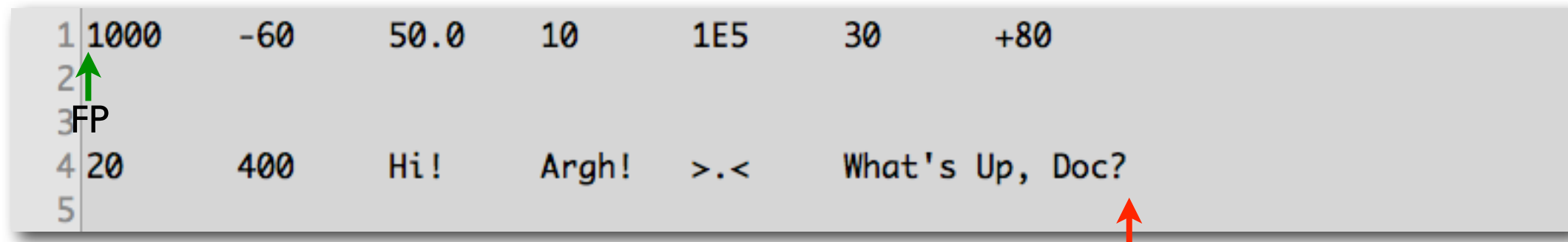
Where will the filestream encounter an error? What values get read?

```
1 // assume that infile opened successfully
2 ifstream infile("data.txt");
3
4 // declare a variable to store the data
5 bool input;
6
7 while (infile >> input) {
8     cout << input << endl;
9 }
10
11 cout << "\nThe file stream is in failure state." << endl;
12 infile.close();
13
```

Input Exercises

Given this data file:

```
1 1000    -60    50.0    10    1E5    30    +80
2
3 FP
4 20      400    Hi!     Argh!   >.<    What's Up, Doc?
5
```



"1000" "-60" "50.0" "10" "1E5" "30" "+80" "20" "400" "Hi!" "Argh!" ">.<"
"What's" "Up," "Doc?"

Where will the filestream encounter an error? What values get read?

```
1 // assume that infile opened successfully
2 ifstream infile("data.txt");
3
4 // declare a variable to store the data
5 string input;
6
7 while (infile >> input) {
8     cout << input << endl;
9 }
10
11 cout << "\nThe file stream is in failure state." << endl;
12 infile.close();
13
```


Input Exercises

Given this data file:

```
1 1000    -60    50.0    10    1E5    30    +80
2
3
4 20      400    Hi!     Argh!   >.<     What's Up, Doc?
5
```

Where will the filestream encounter an error? What values get read?

```
1 // Assume that infile opened successfully
2 ifstream infile("data.txt");
3
4 // declare a variable to store the data
5 int input1;    double input2;    string input3;
6
7 while (infile >> input1 >> input2 >> input3) {
8     cout << input1 << " " << input2 << " " << input3 << endl;
9 }
10
11 cout << "\nThe file stream is in failure state." << endl;
12 infile.close();
13
```

Input Exercises

Given this data file:

```
1 1000    -60    50.0    10    1E5    30    +80
2
3
4 20      400    Hi!     Argh!   >.<     What's Up, Doc?
5
```

Where will the filestream encounter an error? What values get read?

```
1 // Assume that infile opened successfully
2 ifstream infile("data.txt");
3
4 // declare a variable to store the data
5 int input1;    char input2;    int input3;
6
7 while (infile >> input1 >> input2 >> input3) {
8     cout << input1 << " " << input2 << " " << input3 << endl;
9 }
10
11 cout << "\nThe file stream is in failure state." << endl;
12 infile.close();
13
```

Input Exercises

6. Suppose a user types the following sequence of characters at the keyboard (there are several whitespace characters, and the final character is the digit 4):

a	4.23-2.500	-3,e-100	4
---	------------	----------	---

And the keyboard is read by a program running the code sequence to the right.

```
32 char a, b, c;  
33 double m, n, p;  
34 int x, y, z;  
35  
36 cin >> a;  
37 cin >> m;  
38 cin >> x;  
39 cin >> b;  
40 cin >> n;  
41 cin >> y;  
42 cin >> c;  
43 cin >> p;  
44 cin >> z;
```

- (a) Complete the table below: provide the value of all variables that are successfully read by `cin` before it encounters a failure state. **Leave any variables that are not valued by `cin` blank.** Note that the variable order in the table is identical to the order of input operations in the code snippet.

Variable	a	m	x	b	n	y
Value						

Variable	c	p	z			
Value						

- (b) In a sentence, explain why `cin` entered a failure state.