File I/O

In order to read input from a file instead of from the keyboard, or write output to a file instead of to the screen, you must use the C++ facilities for file input and/or output. To begin with, use these headers:

```
#include <iostream>
#include <fstream>
```

Including the <fstream> header defines two types: std::ifstream, the input file stream type, and std::ofstream, the output file stream type. For the rest of this tutorial, assume we've said using namespace std; so that we can leave off the std::.

Output

Let's examine output first, since it's simpler. If we want to write output to a file named results.txt, we create an ofstream object attached to that file, and write to that object. Here is sample code:

```
ofstream outfile("results.txt"); // outfile is a name of our
choosing.
if ( ! outfile ) // Did the creation fail?
{
cerr << "Error: Cannot create results.txt!" << endl;
... return with failure ...
}
outfile << "This will be written to the file" << endl;
outfile << "2 + 2 = " << 2+2 << endl;
```

The first line creates an ofstream object named outfile (which we could have called whatever we wanted) and attempts to attach it to the file named results.txt. If results.txt does not already exist, it will be created; otherwise, the old contents of that file are wiped out so that what this program writes will be the new contents of the file.

If you specify a complete path name, like "C:/CS32/P3/results.txt" (note the use of forward slashes), the file will be in the folder you'd expect, C:\CS32\P3. If you do not specify a complete path name and you've launched the program from the Visual Studio environment, the file will be in the same folder as your C++ source file; if you're using Linux, the file will be in the current directory; if you're using Xcode, the file will be in the yourProject/DerivedData/yourProject/Build/Products/Debug folder, which is a good reason to give a simpler complete pathname like "/Users/yourUsername/CS32/P3/results.txt".

It's possible the attempt to create the file may fail. If so, the stream object outfile knows internally that it is in a failure state instead of a good state. You can test the state of the stream: the expression !outfile yields true if outfile is in a failure state, and false otherwise.

If the file was successfully created, you may write to it just as you would to cout or cerr.

Input

If we want to read input from a file instead of the keyboard, we create an ifstream object attached to that file and read from that object. If the file doesn't already exist, the ifstream object will be in a failure state.

```
ifstream infile("data.txt"); // infile is a name of our choosing
if ( ! infile ) // Did opening the file fail?
{
cerr << "Error: Cannot open data.txt!" << endl;
... return with failure ...
}
```

As with output files, if you specify a complete path name, like "C:/CS32/P3/data.txt" (note the use of forward slashes), the program will look for the file in the folder you'd expect, C:\CS32\P3. If you do not specify a complete path name, the program will look in the same folder it would create a file in if you didn't supply a complete path name (see the Output section above).

We can now read from the file by reading from the stream infile. There are a number of ways we can read from the stream, but only one or two will be useful for Project 3. To read an integer from the file (not useful for Project 3):

```
int k;
infile >> k;
// If you want to consume and ignore the rest of the line the
// number is found on, follow this with
infile.ignore(10000, '\n');
To read a std::string:
std::string s;
infile >> s; // read the next word into s
// or
getline(infile, s); // read a whole line into s
```

When reading input, there's a possibility that the operation will fail because there are no more characters in the file to be read (i.e., we have encountered the end of the file). This leads to a common idiom for reading and processing every line of a file, one by one:

```
string line;
while (getline(infile, line))
{
... process line
}
```

The expression in the while loop will attempt to read the next line from the file, putting the stream in a failure state if the attempt fails. It then yields true if the operation didn't fail. Therefore, we leave the loop only when we hit end of file or read a line that's too long to fit into the array. (The function you'll write for Project 3 does not have to worry about the latter situation.)

Stream Parameters

Suppose we wanted to write a function that writes some output to either a file or to the screen, at the option of the caller of the function. We can do this by having the function accept as a parameter a reference to the desired output stream:

```
void greet(ostream& outf) // outf is a name of our choosing
{
outf << "Hello" << endl;
}
```

Notice that the type named in the parameter declaration is ostream, not ofstream, and that the parameter is passed by reference. An ostream reference can refer to any kind of output stream: to cout or to any ofstream attached to a file. Here's an example:

```
int main()
{
ofstream outfile("greeting.txt");
if ( ! outfile )
{
cerr << "Error: Cannot create greeting.txt!" << endl;
return 1;
}
greet(outfile); // writes Hello to the file greetings.txt
greet(cout); // writes Hello to the screen
}
```

Similarly, if we wanted to have a function read input from either a file or the keyboard, we'd have it take a parameter that is a reference to a general input stream (an istream):

```
int countLines(istream& inf) // inf is a name of our choosing
{
int lineCount = 0;
string line;
while (getline(inf, line))
lineCount++;
return lineCount;
}
```

```
int main()
{
ifstream infile("data.txt");
if ( ! infile )
{
cerr << "Error: Cannot open data.txt!" << endl;
return 1;
}
int fileLines = countLines(infile); // reads from the file
data.txt
cout << "data.txt has " << fileLines << " lines." << endl;
cout << "Type lines, then ctrl-Z (Windows) or ctrl-D (UNIX):" <<
endl;
int keyboardLines = countLines(cin); // reads from keyboard
cout << "You typed " << keyboardLines << " lines." << endl;
}
```

To indicate end of file for keyboard input (i.e., to tell the program that there's no more that will ever be typed in for the rest of this run of the program), type control-Z on a line by itself (Windows), or control-D on a line by itself (UNIX). That character does not get delivered to the

program; instead, cin will internally record that it is in a failure state. Consider the countLines example above, when it is called with cin as the input stream that the parameter inf will refer to. When in response to the call to getline the user just types control-Z (or control-D), cin is put in a failure state, and the getline call yields false, so control leaves the loop.

Here's a summary of some names the library defines:

istream

The general input stream type.

ifstream

The input file stream type; an ifstream object will be associated with a particular file. An ifstream object is one kind of istream.

cin

The input stream object associated with the keyboard. cin is another kind of istream object.

ostream

The general output stream type.

<u>ofstream</u>

The output file stream type; an ofstream object will be associated with a particular file. An ofstream object is one kind of ostream.

<u>cout</u>

The output stream object associated with the screen. cout is another kind of ostream object.

The names ifstream and ofstream are defined by the header <fstream>. The other four are defined by the header <iostream>.