CPSC 225, Spring 2003
A Quick Introduction to C++ Files and Strings

This is an introduction to two topics that are covered later in our textbook: strings and files. It will cover just the basics of these topics, which should be accessible to someone who has studied Java and has encountered the *cin* and *cout* objects in C++. In most applications, the basics are all you really need to know.

Strings

C++ actually has two types of strings: an old-style C-string which is simply an array of characters and a new C++ string which is a member of a class named *string*. (Note the lower-case "s" in the name of the class.) Neither type of string is as full-featured as Java strings; for example, you can't add a number onto the end of a string using the "+" operator. String literals such as "Hello World" are actually C strings, but in many cases they act as C++ strings. For example, you can assign a string literal to a variable of type *string*. For now, we can mostly ignore the difference and work with C++ string variables exclusively.

The C++ *string* class is defined in a library file named "string." If you want to use variables or parameters of type *string* in a program, you should add the line "`#include <string>`" at the beginning of the program. This also defines some functions for working with strings and makes it possible to use the I/O operators, << and >>, to work with *string* variables.

When you use the >> to read a value into a *string* variable, only one "word" will be read, where a word is defined as a sequence of non-blank characters. Suppose *str* is a variable of type *string* and you say "`cin >>str`" in your program. If the user types "` Hello, World`" (with a space before the H), the computer will skip the leading space, read "Hello," into *str* and stop. If you want to read an entire line into *str*, you can use:

```
getline(cin,str);
```

This will read everything from *cin* up to and including the next end-of-line character and store the result (without the end-of-line) in *str*.

Files

In C++, I/O is done with "streams." An input stream such as *cin* is a source of data that can be read into variables. An output stream such as *cout* is a place where data can be sent for display, storage, or communication. A file is a collection of data saved on a disk drive. Data in a file differs from data in variables because a file can exist before the program is run and can persist after the program ends. To read data from a file or to save data in a file, a program just has to use the right type of stream. An object of type *ifstream* is an "input file stream" that can be used to read data from a file into variables. An object of tye *ofstream* is an "output file stream" that can be used as a place to send data. The classes *ifstream* and *ofstream* are both defined in the library file named "fstream", so if you want to use files, you should "`#include <fstream>`" at the beginning of your program.

Every file that is saved on a disk drive has a name, which is used to identify the file. If you want to use an *ifstream* object to read from an existing file, you must somehow associate the stream (which is an object in a program) with the file (which is a collection of information on a disk). This is done by "opening" the file. An *ifstream* variable has an *open* function which can be used to open a file. The name of the file is a parameter to the *open* function. Once this is done, you can read from the *ifstream* object in exactly the same way you would read from *cin*. As you read data from the file, you "move along the stream." Each read operation uses up some of the data in the file. For example:

```
ifstream in;    // Create an input file stream.
in.open("data.txt");  // Use it to read from a file named data.txt.
in >> x;    // Read the first item from the file into an integer variable x.
in >> str;  // Read the next item from the file into a string variable str.
```

Note that the file's name is only used **once** in the entire program, when the file is opened. After that, the file is accessed through the *ifstream* variable. The *ifstream* variable name can be any legal variable name (*in*, *inputFile*, *data*, ...) and does not have to have anything to do with the name of the file.

Suppose you try to open an input file that doesn't exist. This is an error which will make the stream invalid. You can test for this by using the stream variable in a boolean context:

```
in.open("data.txt");
if ( ! in ) {
   cout << "Error: Can't open the file named data.txt.\n";
   exit(1);
}
```

The stream variable will also become "false" if you try to read past the end of the file. Note that when this happens, it means that the previous input operation did not succeed and did not produce any valid data to be processed. For example, a program that processes all the lines in a file might have the following form:

```
string str;
getline(in,str);  // Get the frist line from the file, if any.
while ( in ) {  // Continue if the line was sucessfully read.
    processLine(str);  // Process the line.
    getline(in,str);   // Try to get another line.
}
```

Output file streams of type *ofstream* are created and opened in a similar way. Once an *ofstream* object exists, you can use it in the same way as *cout*. When you open an output file using, for example, `out.open("result.txt")`, the output file is created and is empty until you output some data to it. An error can occur when you try to open the file, for example if you try to create a new file on a read-only disk such as a CD-ROM.

If a file of the same name already exists when you open an output stream, **the old file is erased** as soon as you open the new file. This might or might not be the right thing to do. You can avoid this by adding a second parameter to the *open* function. If you want to avoid replacing an existing file, use `ios::noreplace` as the second parameter. That is, the following commands will fail if a file named "result.txt" already exists:

```
ofstream out;
out.open("result.txt",ios::noreplace);
```

Perhaps you would like to add new data onto the end of an existing file without erasing what is already there. In that case, use `ios::app` as the second parameter. (The "app" stands for "append.")

Note on file names

A simple name such as "data.txt" refers to a file in the same directory where the program is running. A file also has a full path name that identifies it uniquely in the entire file system. For example: "/home/jsmith/cs225/survey/data.txt". By using a full path name as a file name, you can make a program that will use the same file, no matter what directory the file is run from.

Variables as file names

The file name in an *open* function is a C-style string. Because of the peculiarities of C++, you can't use a variable of type *string* as the parameter. However, if *filename* is a variable of type *string*, you can obtain the corresponding C-style string by calling the function *filename.c_str()*, and you can use this function call as a parameter to open. This makes it possible, for example, to get the name of a file from the user:

```
string filename;
ifstream datafile;
cout << "Please enter the input file name: ";
cin >> filename;
datafile.open( filename.c_str() );
```

David Eck