James King-Holmes/Bletchley Park Trust/Photo Researchers, Inc.

Chapter Eight: Streams

Chapter Goals

- To be able to read and write files
- To convert between strings and numbers using string streams
- To process command line arguments
- To understand the concepts of sequential and random access

Topic 1

1. Reading and writing text files
2. Reading text input
3. Writing text output
4. Parsing and formatting strings
5. Command line arguments
6. Random access and binary files

Reading and Writing Files

- The C++ input/output library is based on the concept of *streams*.
- An *input stream* is a source of data.
- An *output stream* is a destination for data.
- The most common sources and destinations for data are the files on your hard disk.
  - You need to know how to read/write disk files to work with large amounts of data that are common in business, administrative, graphics, audio, and science/math programs

Streams: An Example

```
This is a stream of characters. It could be from
the  keyboard  or  from a file. Each of these is
just a character - even these:  3  -23.73 which,
when input, can be converted to: ints or doubles
or whatever type you like.
(that was a  '\n'  at the end  of the last line)
&*@&^#!%#$ (No, that was -not- a curse!!!!!!!!!!
¥1,0000,0000 (price of a cup of coffee in Tokyo)
Notice that all of this text is very plain -  No
bold or green of italics - just characters - and
whitespace (TABs, NEWLINES and, of course... the
other one you can't see: the space character:
(another '\n')
(&& another)    (more whitespace)
```

Reading and Writing Streams

The stream you just saw is a plain text file. No formatting, no colors, no video or music (or sound effects).

A program can read these sorts of plain text streams of characters from the keyboard, as has been done so far with `cin`.

Reading and Writing Disk Files

You can also read and write files stored on your hard disk:

- plain text files
- binary information (a binary file)
  - Such as images or audio recording

To read/write files, you use *variables* of the stream types:

`ifstream` for input from plain text files.

`ofstream` for output to plain text files.

`fstream` for input and output from binary files.

You must `#include <fstream>`

Opening a Stream

To read anything from a file stream,

you need to *open* the stream.

(The same for writing.)

Opening associates the variable name with the file name.

After the file is open, you refer to it ONLY by the stream variable name.

Code for Opening Streams

```
ifstream in_file;
in_file.open("input.txt"); //filename is input.dat
```

An alternative shorthand syntax combines the 2 statements:

```
ifstream in_file(“input.txt”);
```

As your program runs and tries to find this file, it WILL ONLY LOOK IN THE DIRECTORY (FOLDER) IT IS LOCATED IN!

That is a common source of errors. If the desired file is not in the executing program’s folder, the full file path must be specified.

File Path Names

File names can contain directory path information, such as:

UNIX

```
in_file.open("~/nicework/input.dat");
```

Windows

```
in_file.open("c:\\nicework\\input.dat");
```

When you specify the file name as a string literal, and the name contains backslash characters (as in Windows),

you must **supply each backslash *twice***

to avoid having unintended *escape characters* in the string.

\\ becomes a single \ when processed by the compiler.

When the File Name is in a C++ string variable

If the filename comes from the user, you will store it in a string. If you use a C-string (`char` array), the `open`() function works fine.

If you use a C++ `string`, some older library versions require you to convert it to a C-string using `c_str()` as the argument to `open`():

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

Opening a Stream, Filename is a `char[]` array

```
cout << "Please enter the file name:";
char filename[80];
cin >> filename;
ifstream in_file;
in_file.open(filename);
```

Closing a Stream

When the program ends,
all streams that you have opened
will be automatically closed.

You *can* manually close a stream with the
`close` member function:

```
in_file.close();
```

Manually closing a stream is *only* necessary
if you want to re-use the same stream variable for a different file, or want to switch from input to output on the same file.

Reading from a Stream: Use >>

You already know how to read and write using files – it is the same syntax as with `cin`, which is an istream also.

```
string name;
double value;
in_file >> name >> value;
```

Reading from a Stream, Testing for Failure

The `>>` operator returns a "not failed" condition, allowing you to combine an input statement and a test. A "failed" read yields a **`false`** and a "not failed" read yields a **`true`**.

```
if (in_file >> name >> value)
{
   // Process input
}
```

To read the entire file, use a loop:

```
while (in_file >> name >> value)
{
   // Process input
}
```

Failing to Open

The `open` method also sets a “not failed” condition.
It is a good idea to test for failure immediately:

```
in_file.open(filename.c_str());
// Check for failure after opening
if (in_file.fail())
{
   cout << "Error opening file: " << filename
  << endl;
   return -1;
}
```

Writing to a Stream: Just Like `cout <<`

To write to a file: declare an `ofstream` variable, open the file, check for failure, and then stream the data to the `ofstream` with `<<`

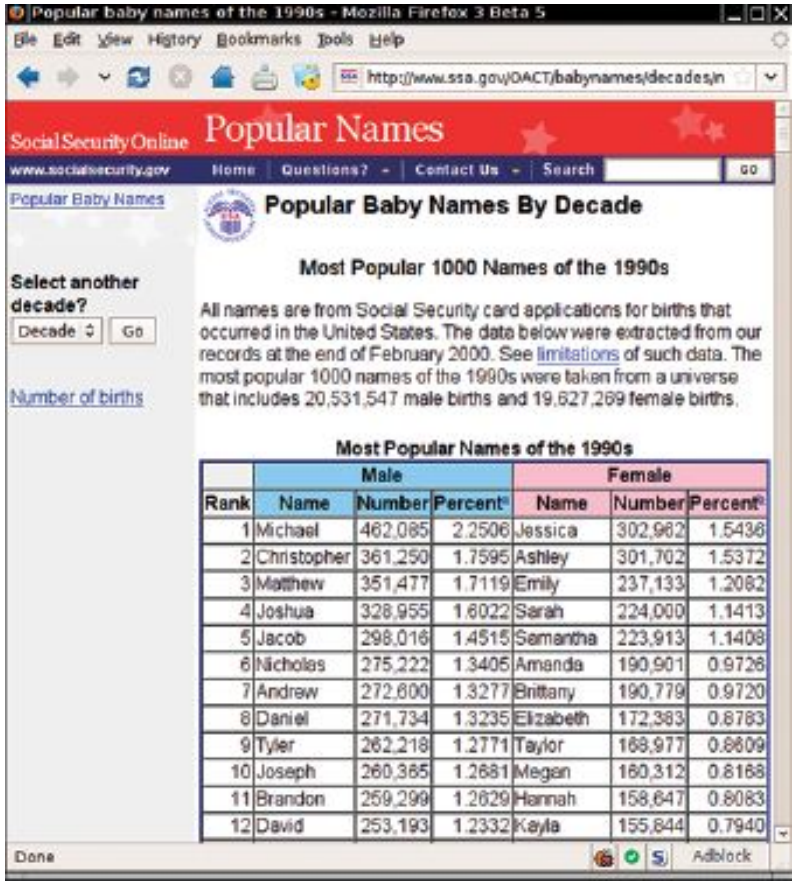
```
ofstream out_file;
out_file.open("output.txt");
if (in_file.fail()) { return -1; }
out_file << name << " " << value << endl;
```

A Programming Example: Popular Names

The Social Security Administration publishes lists of the most popular baby names @ http://www.ssa.gov/OACT/babynames/.

A query of the 1000 most popular names for the 1990s yields the screen shown here:

We saved the data as text, and pasted it into the book companion code `babynames.txt` file.



File Structure

Each line in the file contains seven entries:

- The rank (from 1 to 1,000)
- The name, frequency, and percentage of the male name of that rank
- The name, frequency, and percentage of the female name of that rank

An example line from the file:

```
10 Joseph 260365 1.2681 Megan 160312 0.8
```

Baby Names Program Planning

Sample line: `10 Joseph 260365 1.2681 Megan 160312 0.8`

We will display the top 50% of the names stored in the file.

To process each line, we first read the rank:

```
int rank;
in_file >> rank;
```

Next we read three values for that boy's name:

```
string name;
int count;
double percent;
in_file >> name >> count >> percent;
```

Then we read the 3 values for the girl data.

Let's make a helper function (`process_name`) to do the 3 reads.

Loop Control, and Reference Parameters for Functions

To stop processing after reaching 50%, we could add up the frequencies and stop when they reach 50%.

However, it is simpler to have “total” variables for boys and girls, initialized with 50.

Then we’ll subtract the frequencies as we read them and stop when the total reaches 0.

Reading or writing a file modifies the stream variable, as it counts the number of characters read or written so far.

For that reason, you must always make stream parameter variables reference parameters.

Baby Names: Code Part 1

```
// sec01/babynames.cpp
#include <iostream>
#include <fstream>
#include <string>
using namespace std;
/**
Reads name information, prints the name if total >= 0, and
adjusts the total.
@param in_file the input stream
@param total the total percentage that should still be processed
*/
void process_name(ifstream& in_file, double& total)
{
   string name;
   int count;
   double percent;
   in_file >> name >> count >> percent;
   // Check for failure after each input
   if (in_file.fail()) { return; }
   if (total > 0) { cout << name << " "; }
   total = total - percent;
}
```

Baby Names: Code Part 2

```
int main()
{
   ifstream in_file;
   in_file.open("babynames.txt");
   // Check for failure after opening
   if (in_file.fail()) { return 0; }
   double boy_total = 50;
   double girl_total = 50;
   while (boy_total > 0 || girl_total > 0)
   {
      int rank;
      in_file >> rank;
      if (in_file.fail()) { return 0; }
      cout << rank << " ";
      process_name(in_file, boy_total);
      process_name(in_file, girl_total);
      cout << endl;
   }
   return 0;
}
```

Practice It: File I/O Basics

- Write code statements for the following.
  - Answers are shown in tiny font which you may enlarge.

Define an input file stream variable named in.	`ifstream in;`	An ifstream object is an input file stream variable.
Associate the input file stream with a file named in.txt.	`in.open("in.txt");`	The open function associates the input stream with the disk file in.txt and opens it for reading.
Define a variable word; Use word to read the first word from the file.	`string word; in >> word;`	Simply use the same input operations with which you are already familiar. The >> operator reads the next word.
Declare an output file stream variable named out.	`ofstream out;`	The output file stream is an ofstream.
Use the output stream variable to create a new file named out.txt.	`out.open("out.txt");`	Use the open function to create the desired text file.
Write a line consisting of the word Hello to out.txt.	`out << "Hello" << endl;`	To write to a file, use the << operator and the endlmanipulator tha you have already used with cout.

Practice It: File I/O and a Function

What is the effect of the following statements? (answers shown in tiny font below)

```
ifstream in_file;
in_file.open("");
string word = "\n";
in_file >> word;
```

The statements have no effect.
Because there cannot be a file whose name is the empty string, in_file is set to the "failed" state, and no operations have any effect.

What is wrong with this function?

```
double sum(ifstream in)
{
   double total = 0;
   double input;
   while (cin >> input)
   {
    total = total + input;
   }
return total;
}
```

1. `in` should have been declared as a reference parameter.
2. The input statement should have been `in >> input`.

Practice It: Negative Image Code

1. Modify the imagemod.cpp program to turn the green values of each pixel to red, the blue values to green, and the red values to blue for a psychedelic effect.

- Hint: you only need to change one function!

2. If a BMP file stores a 100 × 100 pixel image, with the image data starting at offset 64, what is the total file size?
    - 10,000 bytes
    - 30,000 bytes
    - 30,064 bytes
    - 40,000 bytes

Answer: 30,064 bytes. We need 3 × 100 bytes for each scan line. There is no padding because this number is divisible by 4. The total size = 3 × 100 × 100 + 64 = 30,064 bytes.

Chapter Summary, Part 1

Develop programs that read and write files.

- To read or write files, use variables of type `fstream`, `ifstream`, or `ofstream`.
- When opening a file stream, supply the name of the file stored on disk.

```
ifstream infile;
infile.open("filename.txt");
```

- Read from a file stream with the same operations that you use with `cin`.

```
int num;
infile >> num;
```

- Write to a file stream with the same operations that you use with `cout`.

```
ofstream out("filename.txt");
out << num;
```

- Always use a reference parameter for a stream function argument, because streams are modified as they are read or written

```
void process_name(ifstream& in_file, double& total)
```

Chapter Summary, Part 2

Be able to process text in files.

• When reading a `string` with the >> operator, the white space between words is consumed.

• You can `get` individual characters from a stream and `unget` the last one.

```
in_file.get(ch);
if (isdigit(ch))
{
   in_file.unget(); // Put the digit back
   data >> n; // Read integer starting with ch
}
```

• You can read a line of input with `getline()` and then process it further. There are 2 flavors of the function:

```
string line;
ifstream in_file("myfile.txt");
getline(in_file, line);

char cstring[100];
in_file.getline(cstring, 99);
```

Chapter Summary, Part 3

Write programs that neatly format their output.

- `#include <iomanip>`
- Use the `setw` manipulator to set the width of the next output.

```
out << setw(40) << left << country << setw(15) << right << density
<< endl;
```

- Use the `fixed` and `setprecision` manipulators to format floating-point numbers with a fixed number of digits after the decimal point.

```
out_file << fixed << x << endl << setprecision(2) << x;
```

Convert between strings and numbers.

- Use an `istringstream` to convert the numbers inside a `string` to integers or floating-point numbers.

```
istringstream strm;
strm.str("January 24, 1973");
string month, comma;
int day, year;
strm >> month >> day >> comma >> year;
```

- Use an `ostringstream` to convert numeric values to `string`s.

Chapter Summary, Part 4

Process the command line arguments of a C++ program.

• Programs that start from the command line can receive the name of the program and the command line arguments in the `main` function.

```
int main(int argc, char* argv[])
```

Develop programs that read and write binary files.

• Open the file with:

```
fstream strm;
strm.open("img.gif", ios::in | ios::out | ios::binary);
```

• You can access any position in a random access file by moving the file pointer prior to a read or write operation.

```
strm.seekg(position); //read = get
char c = strm.get();
strm.seekp(position); // write = put
position = strm.tellg();
position = strm.tellp();
```