



GLOBAL
EDITION



Chapter 12

Streams and File I/O

Absolute C++

SIXTH EDITION
Walter Savitch

ALWAYS LEARNING

PEARSON

Copyright © 2017 Pearson Education, Ltd.
All rights reserved.

Learning Objectives

- I/O Streams
 - File I/O
 - Character I/O
- Tools for Stream I/O
 - File names as input
 - Formatting output, flag settings
- Stream Hierarchies
 - Preview of inheritance
- Random Access to Files

Introduction

- Streams
 - Special objects
 - Deliver program input and output
- File I/O
 - Uses inheritance
 - Not covered until chapter 14
 - File I/O very useful, so covered here

Streams

- A flow of characters
- Input stream
 - Flow into program
 - Can come from keyboard
 - Can come from file
- Output stream
 - Flow out of program
 - Can go to screen
 - Can go to file

Streams Usage

- We've used streams already
 - cin
 - Input stream object connected to keyboard
 - cout
 - Output stream object connected to screen
- Can define other streams
 - To or from files
 - Used similarly as cin, cout

Streams Usage Like cin, cout

- Consider:
 - Given program defines stream `inStream` that comes from some file:
`int theNumber;`
`inStream >> theNumber;`
 - Reads value from stream, assigned to *theNumber*
 - Program defines stream `outStream` that goes to some file
`outStream << "theNumber is " << theNumber;`
 - Writes value to stream, which goes to file

Files

- We'll use text files
- Reading from file
 - When program takes input
- Writing to file
 - When program sends output
- Start at beginning of file to end
 - Other methods available
 - We'll discuss this simple text file access here

File Connection

- Must first connect *file* to *stream object*
- For input:
 - File → ifstream object
- For output:
 - File → ofstream object
- Classes ifstream and ofstream
 - Defined in library <fstream>
 - Named in std namespace

File I/O Libraries

- To allow both file input and output in your program:

```
#include <fstream>  
using namespace std;
```

OR

```
#include <fstream>  
using std::ifstream;  
using std::ofstream;
```

Declaring Streams

- Stream must be declared like any other class variable:
 ifstream inStream;
 ofstream outStream;
- Must then "connect" to file:
 inStream.open("infile.txt");
 - Called "opening the file"
 - Uses member function *open*
 - Can specify complete pathname

Streams Usage

- Once declared → use normally!
 `int oneNumber, anotherNumber;`
 `inStream >> oneNumber >> anotherNumber;`
- Output stream similar:
 `ofstream outStream;`
 `outStream.open("outfile.txt");`
 `outStream << "oneNumber = " << oneNumber`
 `<< " anotherNumber = "`
 `<< anotherNumber;`
 - Sends items to output file

File Names

- Programs and files
- Files have two names to our programs
 - External file name
 - Also called "physical file name"
 - Like "infile.txt"
 - Sometimes considered "real file name"
 - Used only once in program (to open)
 - Stream name
 - Also called "logical file name"
 - Program uses this name for all file activity

Closing Files

- Files should be closed
 - When program completed getting input or sending output
 - Disconnects stream from file
 - In action:
 - `inStream.close();`
 - `outStream.close();`
 - Note no arguments
- Files automatically close when program ends

File Flush

- Output often "buffered"
 - Temporarily stored before written to file
 - Written in "groups"
- Occasionally might need to force writing:
`ostream.flush();`
 - Member function *flush*, for all output streams
 - All buffered output is physically written
- Closing file automatically calls `flush()`

File Example:

Display 12.1 Simple File Input/Output (1 of 2)

Display 12.1 Simple File Input/Output

```
1  //Reads three numbers from the file infile.txt, sums the numbers,
2  //and writes the sum to the file outfile.txt.
3  #include <fstream>
4  using std::ifstream;
5  using std::ofstream;
6  using std::endl;

7  int main()
8  {
9      ifstream inStream;
10     ofstream outStream;

11     inStream.open("infile.txt");
12     outStream.open("outfile.txt");

13     int first, second, third;
14     inStream >> first >> second >> third;
15     outStream << "The sum of the first 3\n"
16                 << "numbers in infile.txt\n"
17                 << "is " << (first + second + third)
18                 << endl;
```

*A better version of this
program is given in Display 12.3.*

File Example:

Display 12.1 Simple File Input/Output (1 of 2)

```
19     inStream.close();
20     outputStream.close();

21     return 0;
22 }
```

SAMPLE DIALOGUE

*There is no output to the screen
and no input from the keyboard.*

infile.txt

(Not changed by program)

1
2
3
4

outfile.txt

(After program is run)

The sum of the first 3
numbers in infile.txt
is 6

Appending to a File

- Standard open operation begins with empty file
 - Even if file exists → contents lost
- Open for append:
 - `ofstream outStream;`
 - `outStream.open("important.txt", ios::app);`
 - If file doesn't exist → creates it
 - If file exists → appends to end
 - 2nd argument is class *ios* defined constant
 - In `<iostream>` library, `std` namespace

Alternative Syntax for File Opens

- Can specify filename at declaration
 - Passed as argument to constructor
- `ifstream inStream;`
`inStream.open("infile.txt");`

EQUIVALENT TO:

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

Checking File Open Success

- File opens could fail
 - If input file doesn't exist
 - No write permissions to output file
 - Unexpected results
- Member function fail()
 - Place call to fail() to check stream operation success

```
inStream.open("stuff.txt");  
if (inStream.fail())  
{  
    cout << "File open failed.\n";  
    exit(1);  
}
```

Character I/O with Files

- All cin and cout character I/O same for files!
- Member functions work same:
 - get, getline
 - put, putback,
 - peek, ignore

Checking End of File

- Use loop to process file until end
 - Typical approach
- Two ways to test for end of file

- Member function eof()

```
inStream.get(next);  
while (!inStream.eof())  
{  
    cout << next;  
    inStream.get(next);  
}
```

- Reads each character until file ends
 - eof() member function returns bool

End of File Check with Read

- Second method
 - read operation returns bool value!
(inStream >> next)
 - Expression returns true if read successful
 - Returns false if attempt to read beyond end of file
 - In action:

```
double next, sum = 0;
while (inStream >> next)
    sum = sum + next;
cout << "the sum is " << sum << endl;
```

Tools: File Names as Input

- Stream open operation
 - Argument to open() is string type
 - Can be literal (used so far) or variable

```
char fileName[16];  
ifstream inStream;  
cout << "Enter file name: ";  
cin >> fileName;  
inStream.open(fileName);
```
 - Provides more flexibility

Formatting Output with Stream Functions

- Recall chapter 1 "magic formula":
 `cout.setf(ios::fixed);`
 `cout.setf(ios::showpoint);`
 `cout.precision(2);`
- Outputs numbers in "money" form (12.52)
- Can use on any output stream
 - File streams have same member functions as cout object

Output Member Functions

- Consider:
 `ostream.setf(ios::fixed);`
 `ostream.setf(ios::showpoint);`
 `ostream.precision(2);`
- Member function `precision(x)`
 - Decimals written with "x" digits after decimal
- Member function `setf()`
 - Allows multitude of output flags to be set

More Output Member Functions

- Consider:
 `ostream.width(5);`
- Member function `width(x)`
 - Sets width to "x" for outputted value
 - Only affects "next" value outputted
 - Must set width before each value in order to affect all
 - Typical to have "varying" widths
 - To form "columns"

Flags

- Recall: member function `setf()`
 - Sets condition of output flags
- All output streams have `setf()` member
- Flags are constants in class `ios`
 - In library `<iostream>`, `std` namespace

setf() Examples

- Common flag constants:
 - `ostream.setf(ios::fixed);`
 - Sets fixed-point notation (decimal)
 - `ostream.setf(ios::showPoint)`
 - Always include decimal point
 - `ostream.setf(ios::right);`
 - Sets right-justification
- Set multiple flags with one call:
`ostream.setf(ios::fixed | ios::showpoint |
ios::right);`

Manipulators

- Manipulator defined:
"A function called in nontraditional way"
 - Can have arguments
 - Placed after insertion operator
 - Do same things as member functions!
 - In different way
 - Common to use both "together"
- `setw()` and `setprecision()` are in library `<iomanip>`, `std` namespace

Manipulator Example: setw()

- setw() manipulator:
cout << "Start" << setw(4) << 10
<< setw(4) << 20 << setw(6) << 30;
– Results in:
Start 10 20 30
- Note: setw() affects only NEXT
outputted value
– Must include setw() manipulator before each
outputted item to affect all

Manipulator setprecision()

- setprecision() manipulator:

```
cout.setf(ios::fixed | ios::showpoint);  
cout    << "$" << setprecision(2) << 10.3 << " "  
        << "$" << 20.5 << endl;
```

- Results in:

\$10.30 \$20.50

Saving Flag Settings

- Flag settings "stay" until changed
- Precision and setf flags can be saved and restored
 - Function `precision()` returns current setting if called with no arguments
 - Member function `flags()` provides similar capability

Saving Flag Settings Example

- `void outputStuff(ofstream& outStream)`
`{`
 `int precisionSetting = outStream.precision();`
 `long flagSettings = outStream.flags();`
 `outStream.setf(ios::fixed | ios::showpoint);`
 `outStream.precision(2);`
 `outStream.precision(precisionSetting);`
 `outStream.flags(flagSettings);`
`}`
- Function to save & restore "typical" settings
 - Call: `outputStuff(myStream);`

Restoring Default setf Settings

- Can also restore default settings:
`cout.setf(0, ios::floatfield);`
- Not necessarily the "last" setting!
- Default values are implementation-dependent
- Does not reset precision settings
 - Only setf settings

Stream Hierarchies

- Class Relationships
 - "Derived from"
 - One class obtained from another class
 - Then features are "added"
 - Example:
 - *Input file* streams class is derived from class of *all* input streams
 - It then adds open and close member functions
 - i.e.: ifstream is derived from istream

Class Inheritance "Real" Example

- Class of all convertibles is derived from class of all automobiles
 - Every convertible is an automobile
 - Convertible "adds features" to automobile

Stream Class Inheritance

- Consider:
- If D is derived class of class B →
 - All objects of type D are also of type B
 - e.g., A convertible is also an automobile
- Regarding streams:
 - An ifstream object is also an istream object
 - Should use istream objects for parameters
 - More objects can be plugged in!

Stream Class Inheritance Example

```
void twoSumVersion1(ifstream& sourceFile)//ifstream with an 'f'
{
    int n1, n2;
    sourceFile >> n1 >> n2;
    cout << n1 << " + " << n2 << " = " << (n1 + n2) << endl;
}
```

and

```
void twoSumVersion2(istream& sourceFile)//istream without an 'f'
{
    int n1, n2;
    sourceFile >> n1 >> n2;
    cout << n1 << " + " << n2 << " = " << (n1 + n2) << endl;
}
```

Stream Class Inheritance

Example Calls

- Considering previous functions:
- `twoSumVersion1(fileIn);` `// Legal!`
- `twoSumVersion1(cin);` `// ILLEGAL!`
 - Because `cin` is not of type `ifstream`!
- `twoSumVersion2(fileIn);` `// Legal!`
- `twoSumVersion2(cin);` `// Legal!`
 - More versatile
 - `istream` parameter accepts both objects

stringstream

- The stringstream class is another example of inheritance
 - Derived from the istream class
 - Allows you to perform stream operations to or from a string, similar to how you perform stream operations from cin or from a file
 - Shares or *inherits* the same methods
- Useful for converting strings to other data types and vice versa

Using stringstream

- To use

```
#include <sstream>
using std::stringstream;
```

- Create an object of type stringstream

```
stringstream ss;
```

- To clear and initialize to blank

```
ss.clear( );
ss.str( " " );
```

- To create a string from other variables

```
ss << c << " " << num;           // c is a char, num is an int
```

Using stringstream

- To extract variables from a string

```
ss << "x 10";
```

```
ss >> c >> num;
```

```
// c is set to 'x' and num is set to 10
```

- This class is sometimes useful when reading a string from some source and extracting fields from the string

stringstream Demo (1 of 3)

//Demonstration of the stringstream class. This program takes
//a string with a name followed by scores. It uses a
//stringstream to extract the name as a string, the scores
//as integers, then calculates the average score. The name
//and average are placed into a new string.

```
#include <iostream>
#include <string>
#include <sstream>
```

```
using namespace std;
```

```
int main( )
{
    stringstream ss;
    string scores = "Luigi 70 100 90";
```

stringstream demo (2 of 3)

```
// Clear the stringstream
ss.str("");
ss.clear();

// Put the scores into the stringstream
ss << scores;

// Extract the name and average the scores
string name = "";
int total = 0, count = 0, average = 0;
int score;
ss >> name;           // Read the name
while (ss >> score) // Read until the end of the string
{
    count++;
    total += score;
}
```

stringstream demo (3 of 3)

```
if (count > 0)
{
    average = total / count;
}

// Clear the stringstream
ss.clear();
ss.str("");
// Put in the name and average
ss << "Name: " << name << " Average: " << average;

// Output as a string
cout << ss.str() << endl;

return 0;
}
```

Random Access to Files

- Sequential Access
 - Most commonly used
- Random Access
 - Rapid access to records
 - Perhaps very large database
 - Access "randomly" to any part of file
 - Use `fstream` objects
 - input and output

Random Access Tools

- Opens same as istream or ostream
 - Adds second argument
 - `fstream rwStream;`
`rwStream.open("stuff", ios::in | ios::out);`
 - Opens with read and write capability
- Move about in file
 - `rwStream.seekp(1000);`
 - Positions put-pointer at 1000th byte
 - `rwStream.seekg(1000);`
 - Positions get-pointer at 1000th byte

Random Access Sizes

- To move about → must know sizes
 - sizeof() operator determines number of bytes required for an object:
sizeof(s) //Where s is string s = "Hello"
sizeof(10)
sizeof(double)
sizeof(myObject)
 - Position put-pointer at 100th record of objects:

rwStream.seekp(100*sizeof(myObject) – 1);

Summary 1

- Streams connect to files with open operation
- Member function fail() checks successes
- Stream member functions format output
 - e.g., width, setf, precision
 - Same usage for cout (screen) or files
- Stream types can be formal parameters
 - But must be call-by-reference

Summary 2

- istream (no "f") parameters accept cin or ifstream objects as arguments
- ostream (no "f") parameters accept cout or ofstream objects as arguments
- Member function eof
 - Used to test for end of input file
- Streams use inheritance to share common methods and variables in an “is-a” relationship between classes