Chapter 12

Streams and File I/O

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: outStream.flush();
  - Member function *flush*, for all output streams
  - All buffered output is physically written
- Closing file automatically calls flush()

Simple File I/O

Infile:
1
2
3
4

Outfile=?

```
1)  #include <fstream>
2)  using std::ifstream;
3)  using std::ofstream;
4)  using std::endl;
5)  int main( )
6)  {
7)      ifstream inStream;
8)      ofstream outStream;

9)      inStream.open("infile.txt");
10)     outStream.open("outfile.txt");
11)     int first, second, third;
12)     inStream >> first >> second >> third;
13)     outStream   << “The sum of the first 3\n”   << “numbers in
            infile.txt\n”  << "is " << (first + second + third)  << endl;
14)     inStream.close( );
15)     outStream.close( );
16)     return 0;
17) }
```

Simple File I/O

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

```
1)  int main( )
2)  {
3)      ifstream fin;
4)      ofstream fout;

5)      fin.open("story.txt");
6)      if (fin.fail( ))
7)      {
8)          cout << "Input file opening failed.\n";
9)          exit(1);
10)     }

11)     fout.open("numstory.txt");
12)     if (fout.fail( ))
13)     {
14)         cout << "Output file opening failed.\n";
15)         exit(1);
16)     }
17)     char next;
18)     int n = 1;
19)     fin.get(next);
20)     fout << n << " ";
21)     while (! fin.eof( ))
22)     {
23)         fout << next;
24)         if (next == '\n')
25)         {
26)             n++;
27)             fout << n << ' ';
28)         }
29)         fin.get(next);
30)     }

31)     fin.close( );
32)     fout.close( );

33)     return 0;
34) } //
```

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:

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

```
outStream.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

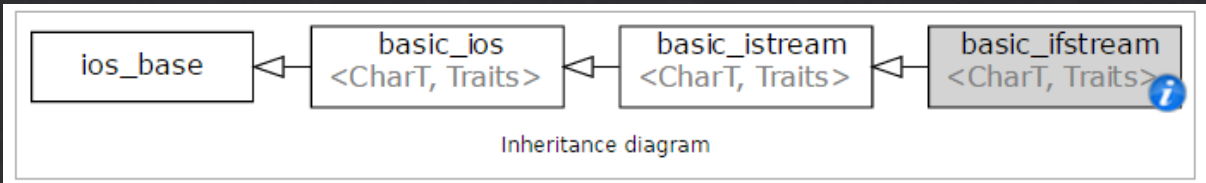
Saving Flag Settings Example

- 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



http://en.cppreference.com/w/cpp/io/basic_ifstream

- 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;
}
```

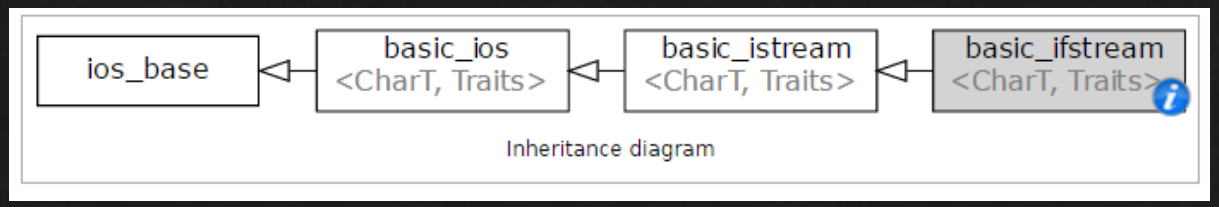
Stream Class Inheritance Example Calls

- Considering previous functions:

// ifstream fileIn;

// istream **cin**;

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

6)  #include <iostream>
7)  #include <string>
8)  #include <sstream>

9)  using namespace std;
```

http://en.cppreference.com/w/cpp/io/basic_stringstream

stringstream demo

```
1)  int main( )
2)  {
3)          stringstream ss;
4)          string scores = "Luigi 70 100 90";
5)         // Clear the stringstream
6)          ss.str("");  // gets or sets the contents
7)          ss.clear(); // clears error and eof flags

8)          // Put the scores into the stringstream
9)          ss << scores;

10)         // Extract the name and average the scores
11)         string name = "";
12)         int total = 0, count = 0, average = 0;
13)         int score;
14)         ss >> name;             // Read the name
15)         while (ss >> score) // Read until the end of string
16)         {
17)                 count++;
18)                 total += score;
19)         }
20)
```

```
21)         if (count > 0)
22)         {
23)           average = total / count;
24)         }

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

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

32)         return 0;
33) }
```

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

```
1)  #include <iostream>
2)  #include <string>
3)  #include <sstream>
4)   int main()
5)  {
6)      std::string str = "Hello, world";
7)      std::istringstream in(str);
8)      std::string word1, word2;
9)
10)     in >> word1;
11)     in.seekg(0); // rewind
12)     in >> word2;
13)
14)     std::cout << "word1 = " << word1 << '\n'
15)               << "word2 = " << word2 << '\n';
16) }
```

Output= ?

Output:

word1 = Hello,
word2 = Hello,

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

Binary file copy

```
1)  #include<…>
2)  int main(int argc, char *argv[])
3)  {
4)      //Open an input and output stream in binary mode
5)      ifstream in("myimage.jpg",ios::binary);
6)      ofstream out("myimage1.jpg",ios::binary);

7)      if(in.is_open() && out.is_open())
8)      {
9)          while(!in.eof())
10)         {
11)             out.put(in.get());
12)         }
13)     }
14)     //Close both files
15)     in.close();
16)     out.close();
17) }
```

Any better ways?

```
1)  class Student
2)  {
3)      char name[20];
4)      int mark;
5)  public:
6)      void GetStudentData();
7)      void ShowStudentData();
8)  };

9)  void Student ::  GetStudentData()
10) {
11)     cout << "Enter Student Name:" << endl;
12)     cin >> name;
13)     cout << "Enter Student Mark:" << endl;
14)     cin >> mark;
15) }

16) void Student ::  ShowStudentData()
17) {
18)     cout << "Student Details are:" << endl;
19)     cout << "Name: " <<  name << endl
20)          << "Mark: " <<  mark << endl;
21) }
```

```
25) int main(int argc, char *argv[])
26) {
27)     char ans='y';
28)     Student sobj;
29)     ofstream out("student.dat", ios::app);  //open in append mode

30)     if(out.is_open())  {
31)         //Loop will continue until something other then y is entered
32)         while( ans == 'y')              {
33)             cout << endl << "Continue ?";
34)             cin >> ans;
35)             if(ans == 'y')              {
36)                 sobj.GetStudentData();
37)                 out.write((char*) & sobj, sizeof(sobj));
38)             }
39)         }
40)     }
41)     out.close();

42)     ifstream in("student.dat");
43)     if(in.is_open())    {
44)         while(!in.eof())                {
45)             in.read((char*) &sobj, sizeof(sobj));
46)             if(!in.eof()) {
47)                 sobj.ShowStudentData();
48)             }
49)         }
50)     }
51)     in.close();
52)     return true;
53) }
```

Reading and Writing Complex Data

```
1)  #include <fstream.h>
2)      class Data {
3)          int    key;
4)          double value;
5)      };
6)      …
7)      Data x;
8)      Data *y = new Data[10];
9)       …
10)     fstream myFile ("data.bin", ios::in | ios::out | ios::binary);
11)     myFile.seekp (location1);
12)     myFile.write ((char*)&x, sizeof (Data));
13)     …
14)     myFile.seekg (0);
15)     myFile.read ((char*)y, sizeof (Data) * 10);
16)     …
17)     myFile.close();
18)     …
```

C Run-Time Library Reference

- The main categories of Microsoft run-time library routines are:

Argument Access	Buffer Manipulation
Byte Classification	Character Classification
Data Alignment	Data Conversion
Debug Routines	Directory Control
Error Handling (CRT)	Exception Handling Routines
File Handling	Floating-Point Support
Input and Output	Internationalization
Memory Allocation	Process and Environment Control
Robustness	Run-Time Error Checking
Searching and Sorting	String Manipulation (CRT)
System Calls	Time Management

Sample check

Sample check

Directory Control

Routine	Use
_chdir, _wchdir	Change current working directory
_chdrive	Change current drive
_getcwd, _wgetcwd	Get current working directory for default drive
_getdcwd, _wgetdcwd	Get current working directory for specified drive
_getdiskfree	Populates a **_diskfree_t** structure with information about a disk drive.
_getdrive	Get current (default) drive
_getdrives	Returns a bitmask representing the currently available disk drives.
_mkdir, _wmkdir	Make new directory
_rmdir, _wrmdir	Remove directory
_searchenv, _wsearchenv, _searchenv_s, _wsearchenv_s	Search for given file on specified paths

Take a look at

File Handling

Routine	Use
_chsize _chsize_s	Change file size
_filelength, _filelengthi64	Get file length
_fstat, _ stat64, _fstati64 64, _fstat64i32	Get file-status information on descriptor
_get_osfhandle	Return operating-system file handle associated with existing C run-time file descriptor
_isatty	Check for character device
_locking	Lock areas of file
_open_osfhandle	Associate C run-time file descriptor with existing operating-system file handle
_setmode	Set file-translation mode

```
1)  // crt_fstat.c
2)  // This program uses _fstat to report
3)  // the size of a file named F_STAT.OUT.
4)
5)  #include <io.h>
6)  #include <fcntl.h>
7)  #include <time.h>
8)  #include <sys/types.h>
9)  #include <sys/stat.h>
10) #include <stdio.h>
11) #include <stdlib.h>
12) #include <string.h>
13) #include <errno.h>
14) #include <share.h>
15)
16) int main( void )
17) {
18)   struct _stat buf;
19)   int fd, result;
20)   char buffer[] = "A line to output";
21)   char timebuf[26];
22)   errno_t err;
23)
24)   _sopen_s( &fd,
25)           "f_stat.out",
26)           _O_CREAT | _O_WRONLY | _O_TRUNC,
27)           _SH_DENYNO,
28)           _S_IREAD | _S_IWRITE );
29)   if( fd != -1 )
30)     _write( fd, buffer, strlen( buffer ) );
31)  // Get data associated with "fd":
32)   result = _fstat( fd, &buf );
33)
34)   // Check if statistics are valid:
35)   if( result != 0 )
36)   {
37)     if (errno == EBADF)
38)       printf( "Bad file descriptor.\n" );
39)     else if (errno == EINVAL)
40)       printf( "Invalid argument to _fstat.\n" );
41)   }
42)   else
43)   {
44)     printf( "File size     : %ld\n", buf.st_size );
45)     err = ctime_s(timebuf, 26, &buf.st_mtime);
46)     if (err)
47)     {
48)       printf("Invalid argument to ctime_s.");
49)       exit(1);
50)     }
51)     printf( "Time modified : %s", timebuf );
52)   }
53)   _close( fd );
54) }
```

Determine the size of a file

- Ver 1:

```
1) ifstream file( "example.txt", ios::binary | ios::ate);
2) return file.tellg();
```

- Ver 2:

```
1)std::streampos fileSize( const char* filePath )
2){
3)    std::streampos fsize = 0;
4)    std::ifstream file( filePath, std::ios::binary );
5)    fsize = file.tellg();
6)    file.seekg( 0, std::ios::end);
7)    fsize = file.tellg() - fsize;
8)    file.close();
9)    return fsize;
10)}
```

Copy a directory (folder)

- Can you write one?

Folder copy: system command

1) #include <csystem>
2) system("copy c:\users\ e:\Backup\");

Folder copy: Boost version

```
1)  #include <iostream>
2)  #include “boost/filesystem.hpp”
3)  int main(int argc, char *argv[])
4)  {
5)      boost::filesystem::path path1("/usr/local/include"); // your source path
6)      boost::filesystem::path::iterator pathI = path1.begin();
7)      while (pathI != path1.end())
8)      {
9)          std::cout << *pathI << std::endl; // here you could copy the file or
                                               // create a directory
10)         ++pathI;
11)     }
12)     return 0;
13) }
```

http://www.boost.org/doc/libs/1_42_0/libs/filesystem/doc/index.htm