



CSCE 121

Introduction to Program Design & Concepts

File I/O

Dr. Tim McGuire

Grateful acknowledgment to Dr. Philip Ritchey and Dr. Michael Moore for some of the material on which these slides are based.

File Operations

- File: a set of data stored on a computer, often on a disk drive
- Programs can read from, write to files
- Used in many applications:
 - Word processing
 - Databases
 - Spreadsheets
 - Compilers

Using Files for Data Storage

- Can use files instead of keyboard, monitor screen for program input, output
- Allows data to be retained between program runs
- Steps:
 - *Open* the file
 - *Use* the file (read from, write to, or both)
 - *Close* the file

General Process

1. Open File
2. Check that file opened successfully
3. Use file (i.e. read/write)
4. Close File

Using Files

1. Requires `fstream` header file
 - use `ifstream` data type for input files
 - use `ofstream` data type for output files
 - use `fstream` data type for both input, output files
2. Can use `>>`, `<<` to read from, write to a file
3. Can use `eof` member function to test for end of input file

Files: What is Needed

- Use `fstream` header file for file access
- File stream types:
 - `ifstream` for input from a file
 - `ofstream` for output to a file
 - `fstream` for input from or output to a file
- Define file stream objects:
 - `ifstream infile;`
 - `ofstream outfile;`

Default File Open Modes

- `ifstream`:
 - open for input only
 - file cannot be written to
 - `open` fails if file does not exist
- `ofstream`:
 - open for output only
 - file cannot be read from
 - file created if no file exists
 - file contents erased if file exists

RAII

- Resource Acquisition is Initialization (RAII)
- Preferred way of interacting with objects in C++
 - More on this later
when we create our own objects that use dynamic memory.
- Prefer to let objects get resources (e.g. open a file)
during its initialization.
- Prefer to let objects dispose of resources (e.g. close a file)
during its destruction.
 - Automatically happens for objects created in a function.

RAII with File Streams

- RAII approach

- used in Stroustrup

1. Declare/define file stream and let initialization open file stream.
2. Check if file opened successfully.
3. Use file stream. (i.e. read/write)
4. Implicitly close file. i.e. Do nothing and let the file close when the object is destroyed.

- zyBooks approach

1.
 - a. Declare/define file stream.
 - b. Explicitly open file stream.
2. Check if file opened successfully.
3. Use file stream. (i.e. read/write)
4. Explicitly close file stream.

Code differences

- zyBooks

```
ifstream infs;  
infs.open("filename.txt");  
if (infs.is_open()) {  
    // use file  
    infs.close();  
}
```

- RAII

```
ifstream infs("filename.txt");  
if (infs.is_open()) {  
    // use file  
}  
// closes automatically
```

Must #include <fstream>

Sometimes you still need old fashioned way...

- Maybe you are running a program that opens multiple files inside a loop
 - Before Loop begins, declare/define file stream.
 - Inside loop
 1. Open file
 2. Check if it opened successfully
 3. Use file
 4. Close file
- Note that here, open/close happens multiple times for the same file stream object

Letting the User Specify a Filename

- In many cases, you will want the user to specify the name of a file for the program to open.
- Starting with C++ 11, you can now pass a `string` object as an argument to a file stream object's `open` member function.
 - (Prior to this you had to pass a C-style null-terminated string)
 - *String literals are stored* in memory as null-terminated C-strings, but *string objects* are **not**.

fileopen.cpp

```
// This program lets the user enter a filename.
#include <iostream>
#include <string>
#include <fstream>
using namespace std;

int main()
{
    ifstream inputFile;
    string filename;
    int number;

    // Get the filename from the user.
    cout << "Enter the filename: ";
    cin >> filename;

    // Open the file.
    inputFile.open(filename);

    // If the file successfully opened, process it.
    if (inputFile)
```

```
{
    // Read the numbers from the file and
    // display them.
    while (inputFile >> number)
    {
        cout << number << endl;
    }

    // Close the file.
    inputFile.close();
}
else
{
    // Display an error message.
    cout << "Error opening the file.\n";
}
return 0;
}
```

Stream States

CSCE 121

Based on slides created by Carlos Soto.

Stream State

- State refers to the overall configuration of information.
- When you change a variable's value, you change the state
 - This may or may not affect a future step of the computation, but it probably does (otherwise, what is the point of the variable?)

What is the state of a stream?

- Good
 - Location in file
 - End of File
- Bad
 - Problems with the stream itself
 - Internet goes down
 - Printer loses power
 - Any kind of crash or break
 - Input is not what is expected (Logical)
 - expected type A, got (incompatible) type B

Robustness

- Recall that we want our programs to be robust.
 - i.e. they should not crash on unexpected input!
- Streams store state information in flags (i.e. bits)
- We can use stream state information to recover gracefully when there are problems with streams.

Stream State Flags

Flag	Meaning
goodbit	Everything is OK
eofbit	Reached end of a file
badbit	Something is wrong with stream
failbit	Stream used incorrectly

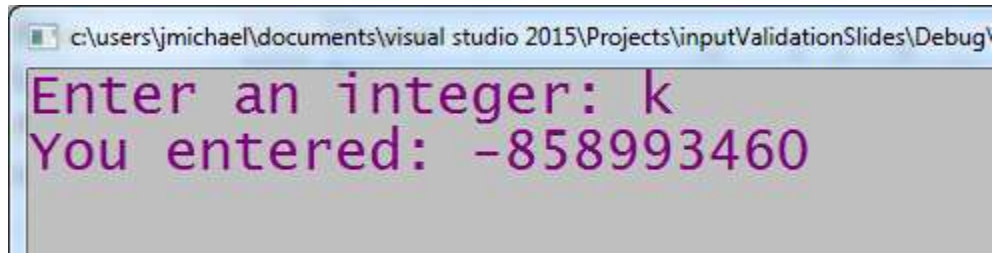
http://www.cplusplus.com/reference/ios/ios_base/iostate/

Stream State Functions

Function	eofbit	failbit	badbit
good()	none of the bits set i.e. zero		
eof()	✓		
bad()			✓
fail()		✓	✓

Using Stream State to Validate Input

```
cout << "Enter an integer: ";  
int val;  
cin >> val;  
  
cout << "You entered: " << val << endl;
```



```
c:\users\j\michael\documents\visual studio 2015\Projects\inputValidationSlides\Debug  
Enter an integer: k  
You entered: -858993460
```

Failbit is set.

So, we can catch this and ask until we get a number.

Using Stream State to Validate Input

```
cout << "Enter an integer: ";
int val;
cin >> val;

while(!cin.good()) { // any of the stream state bits is set
    cin.clear(); // set all stream state bits to zero, buffer NOT cleared
    cin.ignore(numeric_limits<streamsize>::max(), '\n');
    // clear the buffer of everything, i.e. make clean slate for input

    cout << "Enter a valid integer: ";
    cin >> val;
}

cout << "You entered: " << val << endl;
```

Checking for errors in our input

```
cin.clear();
```

- Clear stream states
- Does NOT affect buffer (so whatever caused problem is still in the buffer)

```
cin.ignore(numeric_limits<streamsize>::max(), '\n');
```

- First parameter – max number of characters to ignore (i.e. remove from buffer)
- Second parameter – character to stop ignoring characters if found before max number of characters removed from buffer
- If ignore() is called with no arguments, cin will only skip the very next character
- numeric_limits – different streams have different sizes, using this ensures that if the buffer is completely full, it will remove **all** of it.
- ‘\n’ – character representing end of line.
- #include <limits> to use numeric_limits

More on File Objects

`fstream` Object

- `fstream` object can be used for either input or output
- Must specify mode on the `open` statement
- Sample modes:
 - `ios::in` – input
 - `ios::out` – output
- Can be combined on `open` call:

```
dFile.open("class.txt", ios::in | ios::out);
```
- An alternative to using the **`open`** member function is to use the file stream object definition to open the file:

```
fstream dFile("class.txt", ios::in | ios::out);
```


File Access Flags

File Access Flag	Meaning
<code>ios::app</code>	Append mode. If the file already exists, its contents are preserved and all output is written to the end of the file. By default, this flag causes the file to be created if it does not exist.
<code>ios::ate</code>	If the file already exists, the program goes directly to the end of it. Output may be written anywhere in the file.
<code>ios::binary</code>	Binary mode. When a file is opened in binary mode, data is written to or read from it in pure binary format. (The default mode is text.)
<code>ios::in</code>	Input mode. Data will be read from the file. If the file does not exist, it will not be created and the open function will fail.
<code>ios::out</code>	Output mode. Data will be written to the file. By default, the file's contents will be deleted if it already exists.
<code>ios::trunc</code>	If the file already exists, its contents will be deleted (truncated). This is the default mode used by <code>ios::out</code> .

Using Files - Example

```
// copy 10 numbers between files
// open the files
fstream infile("input.txt", ios::in);
fstream outfile("output.txt", ios::out);
int num;
for (int i = 1; i <= 10; i++)
{
    infile >> num;    // use the files
    outfile << num;
}
infile.close();      // close the files
outfile.close();
```

Default File Open Modes

- `ifstream`:
 - open for input only
 - file cannot be written to
 - `open` fails if file does not exist
- `ofstream`:
 - open for output only
 - file cannot be read from
 - file created if no file exists
 - file contents erased if file exists

More File Open Details

- Can use filename, flags in definition:

```
ifstream gradeList("grades.txt");
```

- File stream object set to 0 (`false`) if open failed:

```
if (!gradeList) ...
```

- Can also check `fail` member function to detect file open error:

```
if (gradeList.fail()) ...
```