

# File I/O

CS 16: Solving Problems with Computers I  
Lecture #9

Ziad Matni  
Dept. of Computer Science, UCSB

# Lecture Outline

---

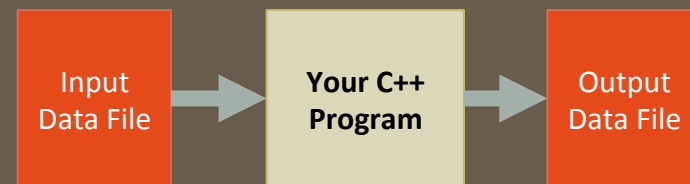
- **I/O Data Streams and File I/O**
- An introduction to Objects and Member Functions
- Handling File I/O Errors

# File I/O



- **Read (input) from a file**
  - Usually done *from beginning to the end* of file (not always)
    - No backing up to read something again (but you can start over)
    - Similar to how it's done from the keyboard
- **Write (output) to a file**
  - Usually done *from beginning to end* of file (not always)
    - No backing up to write something again (but you can start over)
    - Similar to how it's done to the screen

# Stream Variables for File I/O



You have to use “stream variables” for file I/O and they...

- Must be **declared** before you can use file I/O
- Must be **initialized** before the files can contain valid data
  - Initializing a stream means *connecting it to a file*
  - The value of the stream variable is really the filename it is connected to
- Can have their **values changed**
  - Changing a stream value means  
disconnecting from one file and then connecting to another

## Streams and Assignment

- Streams use special built-in (member) functions instead of the assignment operator to change values

- *Example:*

```
streamObjectX.open("MyBook.txt");    // connects to file  
streamObjectX.close();                // closes connection to file
```

# Declaring An Input-File Stream Variable

- Input-file streams are of type **ifstream**
- Type **ifstream** is defined in the **fstream** library
- You must use the appropriate *include* statement and *using* directives

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

- Declare an input-file stream variable with:

```
ifstream input_stream;
```

Variable **type**



Variable **name**



# Declaring An **Output-File** Stream Variable

- Output-file streams of are type **ofstream**
- Type **ofstream** is defined in the **fstream** library
- Again, you must use the *include* and *using* directives

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

- Declare an output-file stream variable using

```
ofstream output_stream;
```

Variable **type**

Variable **name**

# Connecting To A File



- Once a stream variable is declared,  
you can connect it to a file
  - Connecting a stream to a file means “opening” the file
  - Use the *open* member function of the stream object

```
input_stream.open("infile.dat");
```

**Period**

*Member function syntax*

**Double quotes**

**File name on the disk**

*Must include a true path (relative or absolute)*

# Using The Input Stream

- Once connected to a file, get input from the file using the **extraction operator (>>)**
  - Just like with **cin**

*Example:*

```
ifstream in_stream;  
in_stream.open("infile.dat");  
int one_number, another_number;  
  
in_stream >> one_number >> another_number;  
  
in_stream.close();
```

*The inputs are read from the **infile.dat** file separated by either spaces or newline characters. The input values are placed in the variables **one\_number** and **another\_number***

**DEMO!**  
*simpleRead.cpp*

# Using The Output Stream

- An output-stream works similarly using the **insertion operator** (<<)
  - Just like with **cout**

*Example:*

```
ofstream out_stream;  
out_stream.open("outfile.dat");  
  
out_stream << "one number = " << num1  
           << ", another number = " << num2;  
  
out_stream.close();
```

*The output gets written in  
the **outfile.dat** file  
(as opposed to  
the **standard output**!)*

**DEMO!**  
*simpleWrite.cpp*

# Closing a File

- After using a file, it should be closed using the `.close()` function
  - This *disconnects* the stream from the file
  - Close files to reduce the chance of a file being corrupted  
incase the program terminates abnormally
- **Example:** `in_stream.close();`
- It is important to close an output file if your program later needs to read input from the output file
- The system will automatically close files if you forget  
***as long as your program ends normally!***
  - *But I will deduct points in exams and assignments if you forget it!!*

# Member Functions

**Member function:** function associated with an object

- **.open()** is a member function of **in\_stream** in the previous examples
  - **in\_stream** is an object of class **ifstream**
- Likewise, a *different* **.open()** is a member function of **out\_stream** in the previous examples
  - Despite having the same name!
  - **out\_stream** is an object of class **ofstream**

For a list of member functions for I/O stream classes, also see:  
<http://www.cplusplus.com/reference/fstream/ifstream/>  
<http://www.cplusplus.com/reference/fstream/ofstream/>

# Classes vs. Objects

- A class is a **complex data type** that can contain variables & functions
  - Example: `ifstream`, `ofstream`, `string` are examples of C++ (built-in) classes
- When you call up a class to use it in a program,  
you *instantiate* it as an object
  - Example:  
`ifstream MyInputStream; // MyInputStream is an object of class ifstream`

## Calling a Member Function

- Calling a member function requires specifying the object containing the function
- The calling object is separated from the member function by the **dot operator**

- Example: `in_stream.open("infile.dat");`

**Calling object**

**Dot operator**

**Member function**

# Errors On Opening Files

- Opening a file can fail for several reasons
  - The file might not exist
  - The name might be typed incorrectly
  - Other reasons
- **Caution:**  
You may not see an *error message* if the call to open fails!!
  - Program execution usually continues!

## Catching Stream Errors

- Member function `fail()`, can be used to test the success of a stream operation
- `fail()` returns a Boolean type (True or False)
- `fail()` returns True (1) if the stream operation failed

# Halting Execution

- When a stream open function fails, it is generally best to stop the program then and there!
- The function `exit()`, halts a program
  - `exit(n)` returns its argument (n) to the operating system
  - `exit(n)` causes program execution to stop
  - `exit(n)` is NOT a member function! It's a function defined in `cstdlib`
- Exit requires the include and using directives

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

## Using **fail** and **exit**

- Immediately following the call to **open**,  
check that the operation was successful:

```
in_stream.open("stuff.dat");  
if( in_stream.fail() )  
{  
    cerr << "Input file opening failed.\n"; // Why cerr??  
    exit(1); // Program quits right here!  
}
```

# Appending Data to Output Files

- Output examples we've given so far *create new files*
  - If the output file that you've designated already contained data and you try to write to it again, then that data is **now lost!**
- To **append** (i.e. add) new output to the end an existing file use the constant **ios::app** defined in the **iostream** library:  
`ostream.open("important.txt", ios::app);`
  - If the file does not exist, a new file will be created
- There are other member functions that return the location in the I/O file where the next data will be
  - Helps with customizing read and writing files
  - To be used carefully! We won't go over them in CS16...

## Entering File Names for I/O Files

- Users can also enter the name of a file to be read/written
  - As an input read by `cin`
- You can use regular C++ strings for the filenames, but **ONLY** if you ensure that you are compiling with C++ version 11 (or later).
- OTHERWISE, you'll have to use C-strings
  - **WARNING!!!! PAY ATTENTION TO THIS!!!**
  - Textbook has details on how to use C-strings for filenames

# Formatting Output to Files

- Recall: Format output to the screen with:

```
cout.setf(ios::fixed);  
cout.setf(ios::showpoint);  
cout.precision(2);
```

- Similarly, format output to a file using **out\_stream** with:

```
out_stream.setf(ios::fixed);  
out_stream.setf(ios::showpoint);  
out_stream.precision(2);
```

# Let's Look at a Demo...

---

## **RWDemo.cpp**

Found in your demo folder under **demo\_lect09**

# Can I Call a Function to do File I/O?

- Yes!
- But there are strict rules about it:
  - Mainly: stream objects must be **passed by reference** into functions

## Stream Names as Arguments

- Streams can be arguments to a function
  - The function's formal parameter for the stream must be call-by-reference

- Example:

```
void make_neat(ifstream& messy_file,  
               ofstream& neat_file);
```

## Detecting the End of a File

- Input files used by a program may vary in length
  - Programs may not be able to **correctly assume** the number of items or lines in the file
  - You may not know either!
- C++ provides 2 methods that can tell you if you have reached the end of a file that you are reading

## Detecting the End of a File

- The Boolean expression **(in\_stream.eof( ))**
  - Utilizes the member function **eof()** ... or end-of-file
  - **True** if you have reached the end of file
  - **False** if you have not reached the end of file
- The Boolean expression **(in\_stream >> next)**
  - Does 2 things:
    - \* Reads a value from **in\_stream** and stores it in variable **next**
    - \* Returns a Boolean value
  - **True** if a value *can* be read and stored in next
  - **False** if *there is not a value to be read* (i.e. b/c of the end of the file)

## End of File Example

*using while (ifstream >> next) method*

- To calculate the average of the numbers in a file that contains numbers of type double:

```
ifstream in_stream;
in_stream.open("inputfile.txt")

double next, sum(0), average;
int count = 0;

while(in_stream >> next)
{
    sum = sum + next;
    count++;
}
average = sum / count;
```

## End of File Example

using *while ( !ifstream.eof() )* method

- To read each character in a file,  
and then write it to the screen:

*More  
on .get() later*

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

## Which of the 2 Should I Use?!

In general:

- Use **eof** **when input is treated as text**  
and using a member function `.get` to read input
- Use the **extraction operator (>>)** method  
**when input is numerical data**

## Member Function `get(char)`

- Member function of every input stream
  - i.e. it works for `cin` *and* for `ifstream`
- Reads ***one character*** from an input stream
- Stores the character read in a variable of **type `char`**, which is the single argument the function takes
- Does **not** use the extraction operator (`>>`)
- Does **not** skip whitespaces, like blanks, tabs, new lines
  - *Because these are characters too!*

See demo file:  
**`changeCtoCPP.cpp`**

## Using get

- These lines use **get** to read a character and store it in the variable **next\_symbol**

```
char next_symbol;  
cin.get(next_symbol);
```

- Any character will be read with these statements
  - Blank spaces too!
  - '\n' too! (The newline character)
  - '\t' too! (The tab character)

## get Syntax

See demo file:  
[get\\_example.cpp](#)

```
input_stream_object.get(char_variable);
```

- Examples:

```
char next_symbol;  
cin.get(next_symbol);
```

```
ifstream in_stream;  
in_stream.open("infile.txt");  
in_stream.get(next_symbol);
```


## More About get

- Given this code: 

```
char c1, c2, c3;  
cin.get(c1);  
cin.get(c2);  
cin.get(c3);
```

 and this input:  

AB  
CD

 *Note the  
newline after B*
- Results: in `c1 = 'A'` `c2 = 'B'` `c3 = '\n'`
- On the other hand: `cin >> c1 >> c2 >> c3;`  
would place 'C' in `c3` because ">>" operator skips newline characters

## The End of The Line using get

- To read and echo an entire line of input by collecting all characters before the newline character
- Look for '\n' at the end of the input line:

```
cout << "Enter a line of input and I will echo it.\n";  
char symbol;  
do  
{  
    cin.get(symbol);  
    cout << symbol;  
} while (symbol != '\n');
```

- All characters, including '\n' will be output

# getline function

See demo file:  
`getline_example.cpp`

- For standard inputs, `cin` is fine: **but it ignores space, tabs, and newlines**
- Sometimes, you want to get the ***entire line of data!***
- Best to use the function `getline` for that purpose.
- You have to include the `<iostream>` library (which you likely already do!)
- Popular Usage:

```
getline(ifstream_object, string);  
getline(cin, string);
```

# YOUR TO-DOS

---

- ❑ Finish Lab5 by Monday
- ❑ Do HW9 by next Tuesday
- ❑ Visit Prof's and TAs' office hours if you need help!
- ❑ Remember it takes fewer muscles to smile than to frown!

**</LECTURE>**