File I/O

CS 16: Solving Problems with Computers I
Lecture #12

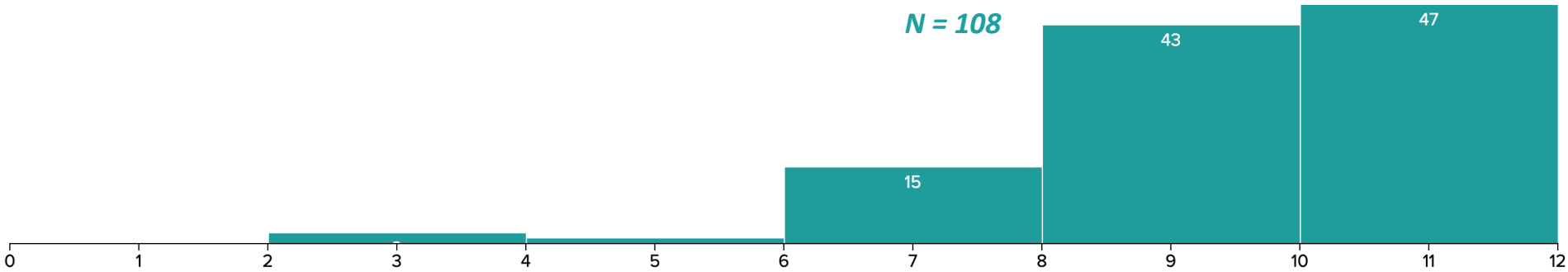
Ziad Matni
Dept. of Computer Science, UCSB

```
int main(int argc, char *argv[])
{
    if (argc > 1)
        filename = argv[1];
    ifstream setIn(filename);
    ifstream vecIn(filename);
    set<string> wordSet = getWordSet(setIn);
    vector<string> wordVec = getWordVec(vecIn);
    map<string, string> wordMap = generateMap(wordVec);

    string name = filename.substr(0, filename.size() - 4);
    string setFilename = name + "_set.txt";
    string vecFilename = name + "_vec.txt";
    string mapFilename = name + "_1_1.txt";

    // Writes set file
    ofstream setOut(setFilename);
    for (set<string>::iterator it = wordSet.begin(); it != wordSet.end(); it++)
    {
        setOut << *it << endl;
    }
    setOut.close();

    // Writes vector file
    ofstream vecOut(vecFilename);
    for (int i = 0; i < wordVec.size(); ++i)
    {
        vecOut << wordVec[i] << endl;
    }
    vecOut.close();

    // Writes to map
    ofstream mapOut(mapFilename);
    printMap(wordMap, mapOut);
    mapOut.close();

    // Generate and print random string
    string str = "";
    for (int i = 0; i < 100; i++)
    {
        cout << wordMap[str] << " ";
        str = wordMap[str];
    }
    cout << endl << endl << endl;

    // Generate more intelligent map
    map<string, vector<string>> wordVecMap;
    str = "";
    for (int i = 0; i < wordVec.size(); i++)
    {
        wordVecMap[str].push_back(wordVec[i]);
        str = wordVec[i];
    }
```

Administrative

- New lab (#6) and new homework (#6) are out!
- Homework 5 and Lab 5 were due yesterday
  - How did they go?
- Quiz 6 is on Friday

Quiz 5



- Mean: **9.13/12 (76.1%)**
- Median: **9/12**
- Not as good as other times! ☹

Q3

Consider the function definition below for `replaceWord()` and its sample call output.
The *goal* of this function is to ask the user to enter a sentence, ask for a word in the sentence, and its replacement, and finally re-construct the sentence with the 1st word replaced by the 2nd word. To be clear, the code should replace just the *first* instance of the word getting replaced.

For example, a function call would result in this dialouge:

```
Enter a sentence: I like horses, yes!
What word do you want to replace? horses
What word do you want to replace it with? Roach
I like Roach, yes!
```

```
void replaceWord(){
    string ask, word1, word2;
    cout << "Enter a sentence: ";
// This next line is correct usage - it captures whitespaces in a string input.
    getline(cin, ask);
    cout << "What word do you want to replace? ";
    cin >> word1;
    cout << "What word do you want to replace it with? ";
    cin >> word2;
    cout << ______________________ << endl;
}
```

Analysis:

A. Looks syntactically ok, but has logic problems (doesn't do a replace).
B. Looks syntactically ok, and does a replace.
C. Cannot use - as operator on strings!
D. Looks syntactically ok, but has logic problems (doesn't do a replace).

A. `ask.erase(ask.find(word1), word1.size()) + ask.insert(ask.find(word1), word2)`

B. `ask.replace(ask.find(word1), word1.size(), word2)`

C. `ask.insert(ask.find(word1), word2) - word1`

D. `ask.substr(ask.rfind(word1), word1.size())`

E. None of the above will work to do this.

Q4

Given the following program snippet, which of these statements will produce a compile error (i.e. syntax error)? Choose as many answers as needed (no partial credit given):

```
string ask = "I like this";
string word1 = "this", word2 = "that thing over there";
cout << ____________________________
```

LOOKS A LOT LIKE (some) PARTS OF Q3!

You have to look for answers that have <u>syntax</u> errors.

A. `ask.erase(ask.find(word1), word1.size()) + ask.insert(ask.find(word1), word2);`

B. `ask.replace(ask.find(word1), word1.size(), word2);`

C. `ask.insert(ask.find(word1), word2) - word1;`

D. `ask.substr(ask.rfind(word1), word1.size());`

E. `ask.append(word1) + ask.find(word1);`

Q5

There's more than 1 way to do this correctly, although there are common elements to all solutions.

Here is one solution:

```
// pre-Condition: function is given an input_string and a sub_string.
//                Assume the sub_string is found at least ONCE
//                in the input_string
// post-Condition: function prints out, on separate lines, the
//                 string indices of the start of each occurrence of
//                 the sub-string in the given string.
// Example:
// input_string = "With a banjo on my knee and ban the bomb-ban!"
// sub_string = "ban"
// The function prints this:
// 7
// 28
// 41
void findAllStrings(string input_string, string sub_string) {
    int index_of_found = 0, next_position = 0;

    while (index_of_found != -1) {

// MISSING CODE HERE

}
```

```
    next_position = input_string.find(sub_string, index_of_found);
    if ( next_position != -1) {
        cout << next_position << endl;
    }
    index_of_found = next_position + 1;
} // close while loop
```

Lecture Outline

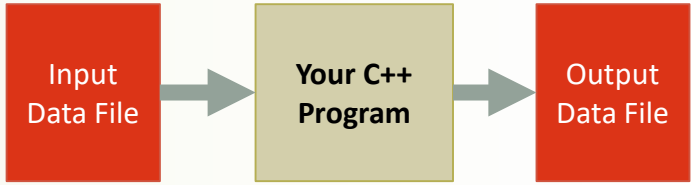
- **I/O Data Streams and File I/O**
- An introduction to Objects and Member Functions
- Handling File I/O Errors

Input / Output via Files

(i.e. not keyboard, not display)

- Instead of the std. input, a program can *read* **inputs from a file**
- Instead of the std. output, a program can *write* **outputs to a file**

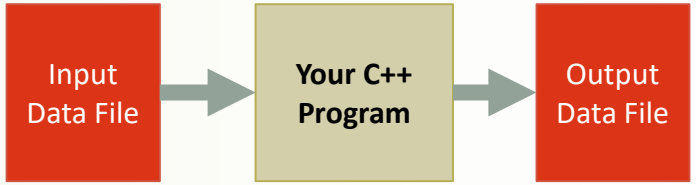
File I/O



- **Read (input) from a file**
  - Usually done *from beginning to the end* of file
    - 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
    - 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 (add this)

```
#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;
```

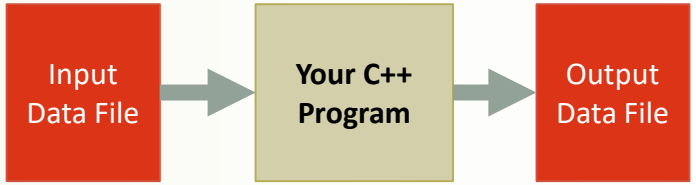
- Declare an output-file stream variable using

```
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 of correct syntax:*** `in_stream.close();`

- It's important to **close** an output file:
  - Esp. if your program needs to (later on) read input from the same output file
- The system will automatically close files if you forget ***as long as your program ends normally!***
  - *i.e. without runtime errors*
  - *But I will deduct points in exams and assignments if you forget it!! (because it's good practice)*

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
string Message;         // Message is an object of class string
```

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 (or instance)** 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 (or instance) 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/

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");`

Dot operator

Member function

Calling object

Errors On Opening Files

- Opening a file can fail for several reasons
  - The file might not exist
  - The name might be typed incorrectly
  - You did not put the TRUE path of the file (very common mistake)
  - 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 (i.e. 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!
  - We used a similar technique with command-line argument progs

- 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!
}
```

DEMO!

RWDemo.cpp

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

- METHOD 1: The Boolean expression **(in_stream.eof( ))**
  - Utilizes the member function **eof()** … *eof = end-of-file*
  - Expression is ***True*** <u>if you have reached the end of file</u>
  - Expression is otherwise ***False***

- METHOD 2: The Boolean expression **(in_stream >> next)**
  - Does 2 things:
    * Reads a value from the object **in_stream** and stores it in variable **next**
    * **Then** 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*

- E.g.: 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 += next;
   count++;
}
average = sum / count;
```

Works with both ints and doubles!

End of File Example

*using **while (!ifstrem.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 method** **when input is treated as text/strings/chars**

  and use 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!*

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

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: c1 = '**A**' c2 = '**B**' c3 = '**\n**'

- On the other hand: `cin >> c1 >> c2 >> c3;`

  would place **'C'** in `c3` because "**>>**" operator **<u>skips newline characters</u>**

The End of The Line using **get**

- To read and echo an **entire line** of input by *collecting all characters* <u>*before*</u> *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

Difference Between **get** and **getline** (a summary)

- Allow you to use input streams that include white-spaces
  - ***Unlike cin***, which separates inputs by white-spaces
  - Recall: white-space = space, tab, newline characters

.get

```
char c_fin, c_cin;
ifstream inf;

inf.get(c_fin);
cin.get(c_cin);
```

getline

```
string fstring, cin_string;
ifstream inf;

getline(inf, fstring);
getline(cin, cin_string);
```

YOUR TO-DOs

- ❑ Start **Lab 6** today
- ❑ Do **Homework 6**
- ❑ Do **Quiz 6** this week (Fri.)

</LECTURE>