

CS 106B, Lecture 4

File I/O and Debugging

Plan for Today

- File I/O (Input/Output)
- Assignment 1 Tips
- Streams + Strings

Plan for Today

- File I/O (Input/Output)
- Assignment 1 Tips
- Streams + Strings

Files

- Store data beyond the run of a program
- Easy way to gather a lot of information together (vs. user input)
- Stored in **streams** in C++
 - To read files, declare an ifstream (interface file **s**tream)
 - To write to files, declare an ofstream (output file **s**tream)

Common File I/O Pattern

- Open File
 - `#include <fstream>` // standard library pkg for files
 - `#include "filelib.h"` // contains helpful methods
 - `string promptUserForFile(stream, prompt)`
// asks user for filename and opens the file in *stream*
 - If you already have the filename:
 - `stream.open("file.txt")`
- Read/write to file (more on that soon)
- Close the file
 - `stream.close()`

Stanford Library

```
#include "filelib.h"
```

Function name	Description
<code>createDirectory(<i>name</i>)</code>	make a new directory with the given name
<code>deleteFile(<i>name</i>)</code>	erase a file from the disk
<code>fileExists(<i>name</i>)</code>	return true if the given file exists on disk
<code>getCurrentDirectory()</code>	return C++ program's directory as a string
<code>isDirectory(<i>name</i>), isFile(<i>name</i>)</code>	return true based on type of file path
<code>promptUserForFile(<i>ifstream&</i>, <i>prompt</i>)</code>	repeatedly prompt for existing file's name
<code>readEntireFile(<i>ifstream&</i>, <i>Lines&</i>)</code>	read file data into a collection of lines
<code>renameFile(<i>oldname</i>, <i>newname</i>)</code>	change a file's name

Creating and Closing

```
ifstream infile;
```

```
promptUserForFile(infile, "File?");
```

```
char ch;
```

```
while(infile.get(ch)) {  
    // do something with ch  
}
```

```
infile.close();
```

Same for every file-reading program
Creates ifstream object
Closes ifstream object

Opening File

```
ifstream infile;  
promptUserForFile(infile, "File?");
```

```
char ch;  
while(infile.get(ch)) {  
    // do something with ch  
}
```

```
infile.close();
```

Asks for the user for the filename

Opening File Alternative

```
ifstream infile;  
infile.open("File.txt");
```

```
char ch;  
while(infile.get(ch)) {  
    // do something with ch  
}
```

```
infile.close();
```

Good when **you** know the file to open

Reading Char by Char

```
ifstream infile;  
promptUserForFile(infile, "File?");
```

```
char ch;  
while(infile.get(ch)) {  
    // do something with ch  
}
```

```
infile.close();
```

Declare the variable to read data into (ch)
While loop continues **until read fails**
- Every iteration of while loop is new char

Reading Line by Line

```
ifstream infile;  
promptUserForFile(infile, "File?");
```

```
string line;  
while(getline(infile, line)) {  
    // do something with line  
}
```

```
infile.close();
```

Now reads each **line** (breaks on newline characters)
Still declare the line before the while loop
Still continues until getline fails; each while loop iteration has a different line
Notice lowercase **l** of getline

Writing Output

```
ofstream outfile;
```

```
promptUserForFile(outfile, "File?");
```

```
string word = "output";
```

```
int x = 3;
```

```
outfile << word << x;
```

```
outfile.close();
```

Similar to reading formatted input
Works a lot like cout
use << (insertion operator)
Works with (basically) any type

File Exercise

- Write a function that asks the user for a filename. Your function should print out statistics about this file including the number of lines, and the number of vowels in the file.

Strange how memories can lie dormant for so many years.
Yet those memories can be awakened and brought forth fresh and new.
Just by something you've seen, heard, or the sight of an old familiar face.

Should print out:

There are 3 lines and 60 vowels.

File Solution

```
void countStatistics() {  
    ifstream infile;  
    promptUserForFile(infile, "Please input a  
filename");  
    string line;  
    int numLines = 0;  
    int numVowels = 0;  
    while(getline(infile, line)) {  
        numLines++;  
        numVowels += countVowels(line);  
    }  
    infile.close();  
    cout << "There are " << numLines << " lines  
and " << numVowels << " vowels" << endl;  
}
```

```
int countVowels(string line) {  
    int numVowels = 0;  
    for (char ch : line) {  
        if (isVowel(ch)) {  
            numVowels++;  
        }  
    }  
    return numVowels;  
}  
  
bool isVowel(char ch) {  
    return ch == 'a' || ch == 'A' || ch == 'e' || ch == 'E' ||  
        ch == 'i' || ch == 'I' || ch == 'o' || ch == 'O' ||  
        ch == 'u' || ch == 'U';  
}
```

File Exercise

- Write a function that reads in the file "scores.txt". Each line of this file is formatted as name:score. An example would be "Tyler:78". Your function should print out the average of the scores.

```
Tyler:78  
Ashley:92  
Kate:100  
...
```

File Solution

```
void avgScores(string filename = "scores.txt") {  
    ifstream infile;  
    infile.open(filename);  
  
    string line;  
    double scores = 0;  
    double numScores = 0;  
    while(getline(infile, line)) {  
        Vector<string> parts = stringSplit(line, ":");  
        scores += stringToInteger(parts[1]);  
        numScores++;  
    }  
    infile.close();  
    cout << "The average is " << scores / numScores << endl;  
}
```


Plan for Today

- File I/O (Input/Output)
- **Assignment 1 Tips**
- Streams + Strings

Assn. 1

- Assignment 1 (Game of Life) will be released today; due **Wednesday, July 3, at 5PM**. You can work in a pair.
 - Start early!
 - Ask for help! (Piazza is great for clarifying the assignment spec)
 - Read and Re-read the spec!!!
- Qt Creator issues
 - A great page to look through if you run into QtCreator issues is here: http://web.stanford.edu/dept/cs_edu/qt-creator/qt-creator-troubleshooting.shtml
 - A lot of times you just need to re-initialize the project as described in the link. And sometimes just restart QtCreator.

Honor Code Note

- Assignment 1 (Game of Life)
 - **Honor Code Reminder:** Please review the Honor Code handout on the course website before beginning this assignment
 - **Any student who is found in violation of the Honor Code will fail the course in addition to sanctions applied by OCS**

Steps to Debugging

- Determine that you have a bug
- Isolate the bug's location
- Find the culprit code

Identifying a bug

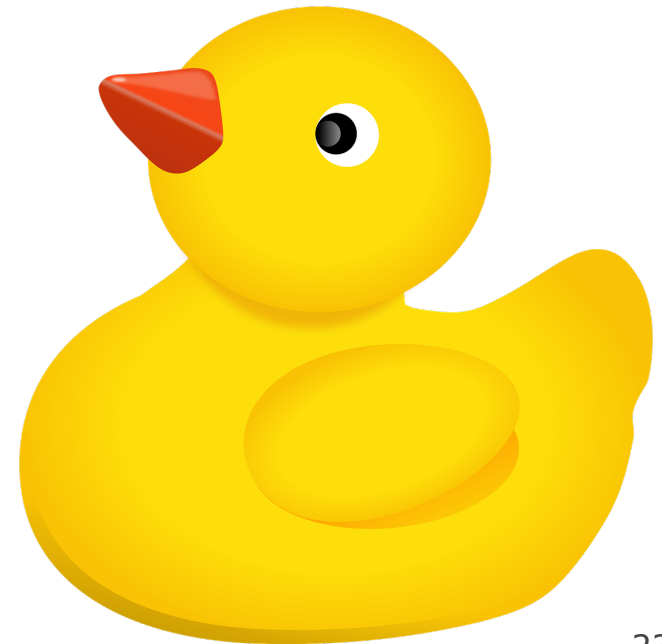
- In order to find a bug, lots and lots of testing
- What is the behavior that you think is buggy (in words)?
- Why do you think that that behavior is buggy?
 - Differs from given expected output?
 - Not what you were expecting?
- Under what circumstances does the bug appear?
- Be specific!

Isolating the Bug

- Goal: where in the code could the bug be?
- Be creative – better to think of too many places than too few
- Identify different functions that could be the culprit
 - Then run each function separately
 - Print out parameters and return values
 - Use the debugger!

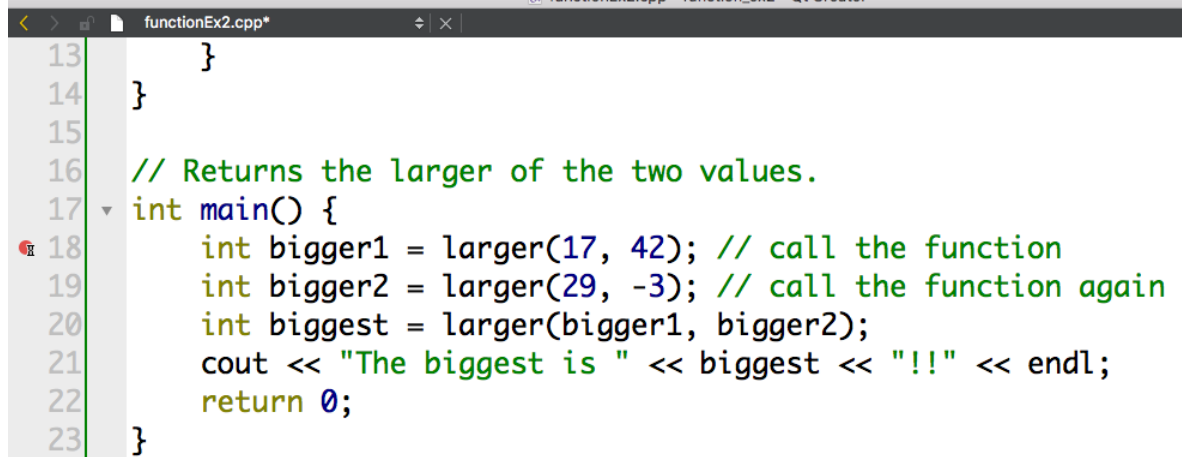
Finding the Bug

- Once you've found the function, need to find the bug
- What does each line of code do?
 - Use print statements or the debugger to verify your assumptions
 - Explain each line of code to your partner or an inanimate object
- Draw pictures – keep track of values in data structures and variable values
- If you still can't find it, get help!
 - LaIR
 - OH



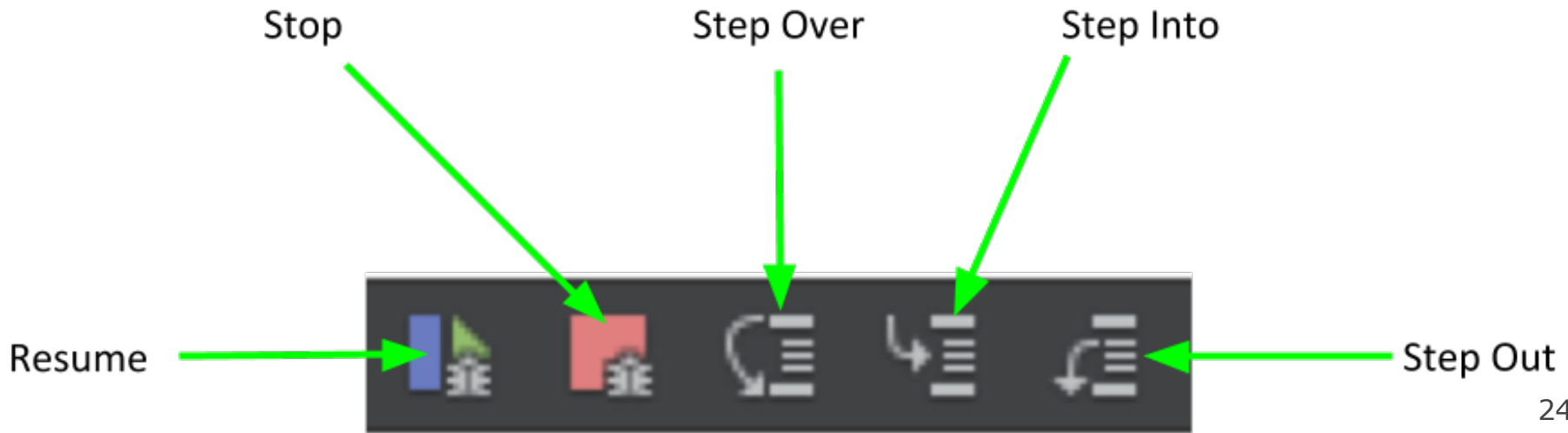
Using the Debugger

- Add a breakpoint – program will pause at that line of code



```
13     }  
14 }  
15  
16 // Returns the larger of the two values.  
17 int main() {  
18     int bigger1 = larger(17, 42); // call the function  
19     int bigger2 = larger(29, -3); // call the function again  
20     int biggest = larger(bigger1, bigger2);  
21     cout << "The biggest is " << biggest << "!!" << endl;  
22     return 0;  
23 }
```

- "Step" through code execution, line by line



Print Debugging

- Alternative to debugger – personal choice (debugger is more powerful, but doesn't represent collections well)
- Idea: print relevant information at every line
- Tips for good print debugging
 - Give good messages at each line (slightly longer, but WAY better output)
 - Print variable values WITH the variable name
 - Debug a section at a time (can be overwhelming otherwise)
 - Add if statements to conditionally print

Plan for Today

- File I/O (Input/Output)
- Assignment 1 Tips
- Streams + Strings

istringstream

```
#include <sstream>
```

- An **istringstream** lets you tokenize a string.

```
// read specific word tokens from a string
istringstream input("Jenny Smith 8675309");
string first, last;
int phone;
input >> first >> last;    // first="Jenny", last="Smith"
input >> phone;            // 8675309
```

```
// read all tokens from a string
istringstream input2("To be or not to be");
string word;
while (input2 >> word) {
    cout << word << endl;    // To\nbe\nor\nnot\n ...
}
```

ostreamstream

```
#include <sstream>
```

- An **ostreamstream** lets you write output into a string buffer.
 - Use the **str** method to extract the string that was built.

```
// produce a formatted string of output
```

```
int age = 42, iq = 95;
```

```
ostreamstream output;
```

```
output << "Zoidberg's age is " << age << endl;
```

```
output << " and his IQ is " << iq << "!" << endl;
```

```
string result = output.str();
```

```
// result = "Zoidberg's age is 42\nand his IQ is 95!\n"
```

Formatted I/O

`#include <iomanip>`

– helps produce formatted output, a la `printf`

Member name	Description
<code>setw(<i>n</i>)</code>	right-aligns next token in a field <i>n</i> chars wide
<code>setfill(<i>ch</i>)</code>	sets padding chars inserted by <code>setw</code> to the given char (default ' ')
<code>setbase(<i>b</i>)</code>	prints future numeric tokens in base- <i>b</i>
<code>left</code> , <code>right</code>	left- or right-aligns tokens if <code>setw</code> is used
<code>setprecision(<i>d</i>)</code>	prints future doubles with <i>d</i> digits after decimal
<code>fixed</code>	prints future doubles with a fixed number of digits
<code>scientific</code>	prints future doubles in scientific notation

```
for (int i = 2; i <= 2000; i *= 10) {           // 2      1.41
    cout << left << setw(4) << i                // 20     4.47
        << right << setw(8) << fixed             // 200    14.14
        << setprecision(2) << sqrt(i) << endl;  // 2000   44.72
}
```

Look Ahead

- Assignment 0 due **tonight at 5PM**
- Assignment 1 (Game of Life) will be released today; due **Wednesday, July 3, at 5PM**. You can work in a pair
- No class on July 4th
 - There is no section on July 4th either. This means section attendance for next week is optional. We will record a section on Wednesday right after class in the same room.
 - We recommend if you have a section on Wednesday to still attend, and if you have a section on Thursday to watch the taped section online.

Bug: Mix lines/tokens

```
cout << "How old are you? ";
int age;
cin >> age;

cout << "And what's your name? ";
string name;
getline(cin, name);
cout << "Wow, " << name << " is " << age << "!" << endl;
```

user input:

17\n

Stuart\n

```
// output:
// How old are you: 17
// And what's your name: Stuart
// Wow,  is 17!
```

- *Advice:* Don't mix `getline` and `>>` on the same input stream.
- *Advice:* Always use Stanford `getX` methods to read from `cin`.