

CS 162

Intro to Computer Science II

Lecture 10

File I/O

2/5/24



Oregon State
University

Odds and Ends

- Design 2, quiz 2 past due
- Lab 5 posted
- Assignment 2 rubrics posted
- Assignment 1 demo due Friday (2/9)

How to Debug Memory Leaks?

```
==29865== LEAK SUMMARY:  
==29865==      definitely lost: 20 bytes in 1 blocks  
==29865==      indirectly lost: 0 bytes in 0 blocks  
==29865==      possibly lost: 0 bytes in 0 blocks  
==29865==      still reachable: 0 bytes in 0 blocks
```

- How to fix it?

locate your memory leaks:

Compile your program with `-g` flag:

```
g++ [yourfile.cpp] -g -o [output]
```

Then run valgrind with the `--leak-check=full` flag:

```
valgrind --leak-check=full ./[output]
```

This shows which line(s) causes memory leaks:

```
==29865== 20 bytes in 1 blocks are definitely lost in loss record 1 of 1  
==29865==    at 0x4C2AC38: operator new[](unsigned long) (vg_replace_malloc.c:433)  
==29865==    by 0x40069E: main (test_cpp:7)
```

How to Debug Seg Fault?

Segmentation fault (core dumped)

- What causes a seg fault?
 - Many reasons... but likely it is caused by your program trying to access an invalid memory address.

- i.e.

```
int * p = nullptr;  
*p = 10;
```

How to Debug Seg Fault?

- How to fix it?

Step 1: locate your seg fault:

Compile your program with `-g` flag:

```
g++ [yourfile.cpp] -g -o [output]
```

Then run valgrind:

```
valgrind ./[output]
```

This shows which line causes seg fault:

```
==25001== Process terminating with default action of signal 11 (SIGSEGV)
==25001== Access not within mapped region at address 0x0
==25001==    at 0x40065D: main (test.cpp:8)
```

How to Debug Seg Fault?

- How to fix it?

Step 2: Fully analyze your program flow till the line that causes the seg fault.

You may inspect the values of your variables using `cout` statements

In lab ⁶~~5~~, we will introduce a very powerful debugging tool (GDB) that allows you to see what is “inside” the program while it is running.

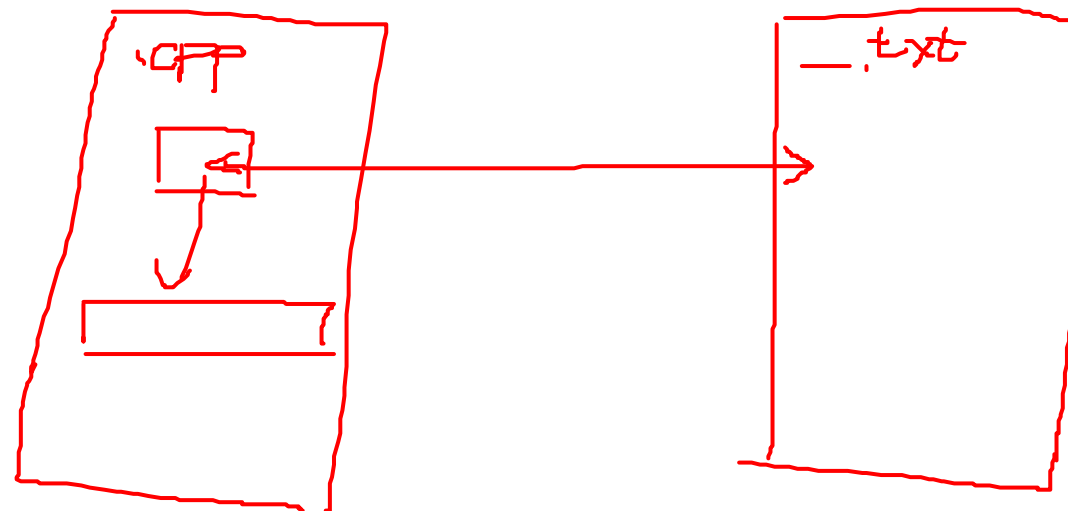
GDB and valgrind are your best friends when debugging!!!

Today's Topics:

- File I/O

File I/O

- File input output
- Allows us to read and write data to files for long term storage
- General algorithm
 1. Create file object
 2. Open the file
 3. Perform action on the file (read/write/etc.)
 4. Close the file



File Stream Objects

```
#include <fstream> //input output file stream class
using namespace std;
int main() {
    fstream f; //create a file stream object
    ifstream fin; //create an input-only file stream
    ofstream fout; //create an output-only file stream
    return 0;
}
```

Open the file

```
int main() {  
    fstream f; //create the object  
    f.open ("file.txt", ios::app); //open(const char* filename, mode)  
    return 0;  
}
```

- Modes (default is input & output for fstream)

- ✓ ios::in → input: file open for reading
- ✓ ios::out → output: file open for writing
- ios::binary → binary: operations are performed in binary mode
- ios::ate → at end: output position starts at the end of the file
- ✓ ios::app → append: all output operations happen at the end of the file, appending to the existing contents
- ios::trunc → truncate: existing file contents are discarded


Open the file

```
int main() {  
    fstream f; //create the object  
    f.open ("file.txt", ios::app); //open(const char* filename, mode)  
    return 0;  
}
```

- Modes can be combined using [the bitwise OR operator](#)
 - `f.open ("file.txt", ios::out | ios::app);`
- Not all combination of modes are valid
 - E.g. append and truncate

Warning about opening files

- If there is already a file open in the stream it will not open another file
 - Check if the stream has a file open using `is_open()` or with `fail()`

```
f.open ("some_file.txt");  
if (f.is_open()) {  
     //perform operations  
}  
else{  
    cout << "Error opening file" << endl;  
}
```

! f.fail()

Perform Action on the File

- Reading (Precondition: the file is not empty)

```
int num = 0;
```

```
ifstream f;
```

```
f.open ("numbers.txt");
```

```
f >> num;
```

```
//can read the entire file by doing a while (!f.eof()){}
```

```
//(eof == end of file)
```

```
//read a single character with get(), read a line with getline()
```

FYI:

- Writing (Caution: know where the cursor is in the file)

```
ofstream f;
```

```
f.open("an_awesome_story.txt");
```

```
f << "Once upon a time..." << endl;
```

Close the file

- Don't forget to do this when you are done with the file

```
my_file_obj.close();
```

File Input – Using “space” as delimiter

[string] _ [int]

1

2

```
ifstream fin;  
fin.open ("book.txt");  
if (!fin.is_open())  
    return 1;  
while (!fin.eof()){  
    string tmp_string;  
    int tmp_int;  
    // read non-blank characters;  
    fin >> tmp_string >> tmp_int;  
    cout << "Text: " << tmp_string << endl;  
    cout << "Int: " << tmp_int << endl;  
}  
fin.close();
```

3

4

File Input Strategies

- What if the input file does not delineate text with spaces?
 - E.g. “student_name,grade,gpa”
- `getline(cin, dest_string);`
 - Reads an entire line at once
 - Previously used this when accepting user input from the console
- `getline(cin, dest_string, ',');`
 - Keeps reading text until reaching the specified char
 - Discards the specified char
 - Can be used to handle an alternate delimiter (e.g. comma)

The Newline Character

- Most user-readable files use newlines
 - Makes the text much easier to read
- Often used to indicate “new entry”
 - Make sure that your code handles these correctly
- Hint: Use [std::istream::ignore\(\)](#)
 - Discards one or more characters from the input stream
 - Useful for discarding newline characters
 - Common usage: `cin.ignore()` → throw away the next char

File Output

- You control the delimiters, newlines, etc.
- Easier to handle

```
string output_file = "book_stats.txt";  
ofstream fout;  
fout.open (output_file.c_str(), ios::app);  
if (!fout.is_open()) {  
    cout << "Error, unable to open the file!" << endl;  
    return 1;  
}  
fout << "Hello world!" << endl;  
fout.close();
```

File I/O Demo