



COS 217: Introduction to Programming Systems

Debugging

The material for this lecture is drawn, in part, from
The Practice of Programming (Kernighan & Pike) Chapter 5

 PRINCETON UNIVERSITY

1

Goals of this Lecture 


Help you learn about:


- Strategies and tools for debugging your code

Why?

- Debugging large programs can be difficult
- A mature programmer knows a wide variety of debugging **strategies**
- A mature programmer knows about **tools** that facilitate debugging
 - Debuggers
 - Version control systems
 - Profilers (a future lecture)

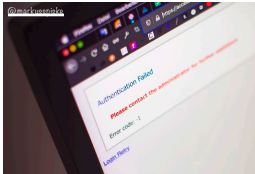
2

How to get the most out of this lecture ... 




Fully "participate" in the Bug Hunts!

3



1. UNDERSTAND ERROR MESSAGES

4


Understand Error Messages 

```
#include <stdio,h>
/* Print "hello, world" to stdout and
return 0.
int main(void)
{ printf("hello, world\n")
return 0;
}
```

What are the errors? (No fair looking at the next slide!)

Debugging at **build-time** is easier than debugging at **run-time**, if and only if you...
Understand the error messages!

5

Understand Error Messages 

```
#include <stdio,h>
/* Print "hello, world" to stdout and
return 0.
int main(void)
{ printf("hello, world\n")
return 0;
}
```

Which tool (preprocessor, compiler, or linker) reports the error(s)?

```
$ gcc217 hello.c -o hello
hello.c:1:19: fatal error: stdio,h: No such file or directory
#include <stdio,h>
                  ^
compilation terminated.
```

6

Understand Error Messages

```
#include <stdio.h>
/* Print "hello, world" to stdout and
return 0. */
int main(void)
{ printf("hello, world\n");
return 0;
}
```

What are the errors? (No fair looking at the next slide!)

7

7

Understand Error Messages

```
#include <stdio.h>
/* Print "hello, world" to stdout and
return 0. */
int main(void)
{ printf("hello, world\n");
return 0;
}
```

Which tool (preprocessor, compiler, or linker) reports the error(s)?

```
$ gcc217 hello.c -o hello
hello.c:2:1: error: unterminated comment
/* Print "hello, world" to stdout and
^
```

8

8

Understand Error Messages

```
#include <stdio.h>
/* Print "hello, world" to stdout and
return 0. */
int main(void)
{ printf("hello, world\n");
return 0;
}
```

What are the errors? (No fair looking at the next slide!)

9

9

Understand Error Messages

```
#include <stdio.h>
/* Print "hello, world" to stdout and
return 0. */
int main(void)
{ printf("hello, world\n");
return 0;
}
```

Which tool (preprocessor, compiler, or linker) reports the error(s)?

```
$ gcc217 hello.c -o hello
hello.c: In function 'main':
hello.c:6:4: error: expected ';' before 'return'
return 0;
^
hello.c:7:1: warning: control reaches end of non-void
function [-Wreturn-type]
}
```

10

10

Understand Error Messages

```
#include <stdio.h>
/* Print "hello, world" to stdout and
return 0. */
int main(void)
{ printf("hello, world\n");
return 0;
}
```

What are the errors? (No fair looking at the next slide!)

11

11

Understand Error Messages

```
#include <stdio.h>
/* Print "hello, world" to stdout and
return 0. */
int main(void)
{ printf("hello, world\n");
return 0;
}
```

Which tool (preprocessor, compiler, or linker) reports the error(s)?

```
$ gcc217 hello.c -o hello
hello.c: In function 'main':
hello.c:6:4: warning: implicit declaration of function
'printf' [-Wimplicit-function-declaration]
printf("hello, world\n");
^
/tmp/cc201XR0.o: In function 'main':
hello.c:(.text+0x10): undefined reference to 'printf'
collect2: error: ld returned 1 exit status
```

12

12

Understand Error Messages

```
#include <stdio.h>
/* Print "hello, world" to stdout and
return 0. */
int main(void)
{ printf("hello, world\n");
  return 0;
}
```

What are the errors?

13

Understand Error Messages

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
  enum StateType
  { STATE_REGULAR,
    STATE_INWORD
  }
  printf("just hanging around\n");
  return EXIT_SUCCESS;
}
```

What are the errors? (No fair looking at the next slide!)

14

Understand Error Messages

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
  enum StateType
  { STATE_REGULAR,
    STATE_INWORD
  };
  printf("just hanging around\n");
  return EXIT_SUCCESS;
}
```

What does this error message even mean?

```
$ gcc217 states.c -o states
states.c:9:11: error: expected declaration specifiers or '...'
before string constant
```

15

Understand Error Messages

Caveats concerning error messages


- Line # in error message may be approximate
- Error message may seem nonsensical
- Compiler may not report the real error

Tips for eliminating error messages

- Clarity facilitates debugging
 - Make sure code is indented properly
- Look for missing "punctuation"
 - ; at ends of structure and enumerated type definitions
 - ; at ends of function declarations
 - ; at ends of do-while loops
- Work incrementally
 - Start at first error message
 - Fix, rebuild, repeat

16

2. THINK BEFORE WRITING




17

Think Before Writing

Inappropriate changes could make matters worse...

Think before changing your code

- Explain the code to:
 - Yourself
 - Someone else
 - A rubber duck / Teddy bear / stuffed tiger?
- Do experiments
 - But make sure they're disciplined



18

3. LOOK FOR COMMON BUGS



19

19

Look for Common Bugs

Some of our "favorites":

```

switch (i)
{
  case 0:
    break;
  case 1:
    ...
  case 2:
    ...
}

if (i = 5)
...

if (5 < i < 10)
...

int i;
scanf("%d", i);

char c;
c = getchar();

while (c = getchar() != EOF)
...

if (i & j)
...
    
```

What are the errors?

20

20

Look for Common Bugs

Some of our "favorites":

```

for (i = 0; i < 10; i++)
{
  for (j = 0; j < 10; i++)
  {
    ...
  }
}

for (i = 0; i < 10; i++)
{
  for (j = 10; j >= 0; j++)
  {
    ...
  }
}
    
```

What are the errors?

21

21

Look for Common Bugs

Some of our "favorites":

```

{
  int i;
  i = 5;
  if (something)
  {
    int i; ←
    i = 6;
  }
  printf("%d\n", i);
}
    
```

What value is written if this statement is present? Absent?

22

22

4. DIVIDE & CONQUER





23

23

Divide and Conquer

Divide and conquer to debug a program:

- Incrementally find smallest **input file** that illustrates the bug
- Approach 1: **Remove** input
 - Start with file
 - Incrementally remove lines until bug disappears
 - Examine most-recently-removed lines
- Approach 2: **Add** input
 - Start with small subset of file
 - Incrementally add lines until bug appears
 - Examine most-recently-added lines

24

24

Divide and Conquer

Divide and conquer: To debug a **module**...

- Incrementally find smallest **client subset** that illustrates the bug
- Approach 1: **Remove** code
 - Start with test client
 - Incrementally inactivate lines of code until bug disappears
 - Examine most-recently-removed lines
- Approach 2: **Add** code
 - Start with minimal client
 - Incrementally add lines of test client until bug appears
 - Examine most-recently-added lines

25

5. FOCUS ON NEW CHANGES



26

Focus on Recent Changes

Focus on recent changes

- Corollary: Debug now, not later

<p>Attractive but Difficult:</p> <ol style="list-style-type: none"> Compose entire program Test entire program Debug entire program 	<p>Monotonous but Easier:</p> <ol style="list-style-type: none"> Compose a little Test a little Debug a little Compose a little Test a little Debug a little ...
---	--

27

Focus on Recent Changes

Focus on recent change (cont.)

- Corollary: Maintain old versions

<p>Low overhead but Difficult recovery:</p> <ol style="list-style-type: none"> Change code Note new bug Try to remember what changed since last version 	<p>Higher overhead but Easier recovery:</p> <ol style="list-style-type: none"> Backup current version Change code Note new bug Compare code with last version to determine what changed
---	--

28

Maintaining Old Versions

Use a **Revision Control System**

(Since you have to set it up anyway to get the files, you might as well use it!)


Allows programmer to:

- Check-in** source code files from **working copy** to **repository**
- Commit** revisions from **working copy** to **repository**
 - saves all old versions
- Update** source code files from **repository** to **working copy**
 - Can retrieve old versions

- Appropriate for one-developer projects
- Extremely useful, almost *necessary* for multideveloper projects!

30

6. ADD (MORE) INTERNAL TESTS




31

Add More Internal Tests

- Internal tests help **find** bugs (see "Testing" lecture)
- Internal test also can help **eliminate** bugs
 - Validating parameters & checking invariants can eliminate some functions from the bug hunt

32

7. DISPLAY TO OUTPUT



33

Display Output

Write values of important variables at critical spots

- Possibly poor:


```
printf("%d", keyvariable);
```

stdout is buffered; program may crash before output appears
- Maybe better:


```
printf("%d\n", keyvariable);
```

Printing '\n' flushes the stdout buffer, but not if stdout is redirected to a file
- Better still:


```
printf("%d", keyvariable);  
fflush(stdout);
```

Call fflush() to flush stdout buffer explicitly

34

Display Output

- Maybe even better:


```
fprintf(stderr, "%d", keyvariable);
```

Write debugging output to stderr; debugging output can be separated from normal output via redirection

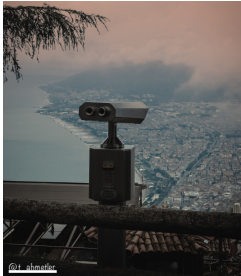
Bonus: stderr is unbuffered
- Maybe even better still:


```
FILE *fp = fopen("logfile", "w");  
fprintf(fp, "%d", keyvariable);  
fclose(fp);
```

Write to a log file

35

8. USE A DEBUGGER



36

The GDB Debugger

GNU Debugger

- Part of the GNU development environment
- Integrated with Emacs editor
- Allows user to:
 - Run program
 - Set breakpoints
 - Step through code one line at a time
 - Examine values of variables during run
 - Etc.


For details see precept materials

37

COS 217: Introduction to Programming Systems

Debugging Dynamic Memory Bugs

38



38




9. COMMON CULPRITS

39 (This overlaps with 3. "Look for Common Bugs" but is more constrained.)

39

Look for Common DMM Bugs

Some of our "favorites":

```
int *p;
... /* code not involving p */
*p = somevalue;
```

```
char *p;
... fgets(p, 1024, stdin);
```

```
int *p;
p = (int*)malloc(sizeof(int));
*p = 5;
... free(p);
... *p = 6;
```

What are the errors?

40

40

Look for Common DMM Bugs

Some of our "favorites":



```
int *p;
... p = (int*)malloc(sizeof(int));
... *p = 5;
p = (int*)malloc(sizeof(int));
```

```
int *p;
... p = (int*)malloc(sizeof(int));
... *p = 5;
... free(p);
... free(p);
```

What are the errors?

41

41

10. DIAGNOSE SEGFAULTS WITH GDB

42

42

Diagnose Seg Faults Using GDB

Segmentation fault => make it happen in gdb

- Then issue the gdb where command
- Output will lead you to the line that caused the fault
 - But that line may not be where the error resides!

43

43

11. MANUALLY INSPECT MALLOCS



44

Manually Inspect Malloc Calls

Manually inspect each call of `malloc()`

- Make sure it allocates enough memory

Do the same for `calloc()` and `realloc()`

45

44

45

Manually Inspect Malloc Calls

Some of our "favorites":

```
char *s1 = "hello, world";
char *s2;
s2 = (char*)malloc(strlen(s1));
strcpy(s2, s1);
```

```
char *s1 = "hello, world";
char *s2;
s2 = (char*)malloc(sizeof(s1));
strcpy(s2, s1);
```


```
long double *p;
p = (long double*)malloc(sizeof(long double*));
```

```
long double *p;
p = (long double*)malloc(sizeof(p));
```

What are the errors?

46

12. HARD-CODE MALLOC AMOUNTS



47

46

47

Hard-Code Malloc Calls

Temporarily change each call of `malloc()` to request a large number of bytes

- Say, 10000 bytes
- If the error disappears, then at least one of your calls is requesting too few bytes

Then incrementally restore each call of `malloc()` to its previous form

- When the error reappears, you might have found the culprit

Do the same for `calloc()` and `realloc()`

48

13. COMMENT OUT CALLS TO FREE

free

49

48

49

Comment-Out Free Calls

Temporarily comment-out every call of `free()`

- If the error disappears, then program is
 - Freeing memory too soon, or
 - Freeing memory that already has been freed, or
 - Freeing memory that should not be freed,
 - Etc.

Then incrementally “comment-in” each call of `free()`

- When the error reappears, you might have found the culprit

50

50

Meminfo Valgrind

14. USE A MEMORY PROFILER TOOL

51

51



52

52