



Debugging

CS2263 – Systems Software Development

Where Do You Think You're Going?, Dire Straits (Communiqué, 1979), <https://www.youtube.com/watch?v=dsc9j5i1A>



Learning Outcomes

At the conclusion of this lecture students should be able to:

- Decode both compile-time and run-time error messages
- Identify and remedy common compile-time errors
- Describe at least two defensive programming strategies



References

Lu, Yung-Hsiang. 2015. Intermediate C Programming. CRC Press. New York. 466pp (Chapter 3)

Kernighan, W and K Pike. 1999. The Practice of Programming. Addison-Wesley. 267pp (Chapter 5)

Steve Maguire. 1993. Writing Solid Code. 256pp



Debugging

Goals

- Produce working code by removing errors
- Be efficient in use of time.



Where Do Errors Come From?

- Programs
- Your programs
- Specifically
 - Syntax errors: unintentional typographic errors
 - Comprehension errors: not understanding how to use constructs
 - Algorithmic errors: Flawed understanding/implementation of the algorithm
 - Misunderstanding of the requirements

Removing errors from code is called debugging
Introducing errors into code is called programming



Identifying the Error

- The most difficult part
 - Compile versus run-time
- Finding and understanding the error is 90% of the work
- Know the typical errors of the language
 - In your limited experience, what are they?
 - Compile-time
 - Run-time



Typical Errors I

```
int n;  
  
//wrong  
scanf("%d", n);  
  
//right  
scanf("%d", &n);
```

```
int n;  
int iNArg;  
  
//wrong  
scanf("%d", &n);  
  
//right  
iNArg = scanf("%d", &n);
```



Typical Errors II

```
int n=1;  
double d=PI;  
printf("%d %f\n", d, n);
```

Output: -266631570 0.000000

```
int n;  
float d;  
int iNArg;  
iNArg = scanf("%d %lf\n", &n, &d);
```



Finding Run-Time Errors: Simple Method

Place `printf ()` statements to demonstrate progress through the program and identify problems

- Move them through the program until you identify the problem statement(s)
- Can you read the program inputs?
- Can you perform each calculation/manipulation correctly?



Typical Errors III

Other Common Errors

- Not initializing local variables
- Wrong array indexes ("off by one")

```
int iVal[10];  
//wrong  
iVal[10] = 6;
```
- Trying to use java style array length variable

```
//wrong  
iVal.length
```



Typical Errors IV

Many of these common errors caught by enabling all warnings of compiler (`-Wall`)

```
printferr.c: In function 'main':  
printferr.c:8: warning: int format, double arg (arg 2)  
printferr.c:8: warning: double format,  
different type arg (arg 3)
```

- Don't ignore compiler warnings!



Finding Run-Time Errors: Debugger

- Compile the program with `-g` to maintain symbol tables and sync with source (e.g. `gcc -g -o crap crap.c`)
- Load and run the program in a debugger (e.g. `ddd <name>`)
 - Establish break-points to stop the program, or
 - Follow the program flow ("step" through the program), one expression at a time.
 - Examine contents of variables.
- ALSO
 - Consider printing out your program listing on paper and look it over (reflection), line-by-line. "*The medium is the message.*"
 - Consider explain your code to someone else. Or the duck.
 - See lecture resources for Rubber Duck Debugging



Fixing Bugs

- Understand the problem before you fix it
- Understand the program, not just the problem
- Confirm error diagnosis (“if my diagnosis is correct, then ...”)
- Save original source code before making change (version control)
- Fix the problem, not the symptom (no band-aid solutions)
- Check your fix *and look for similar errors*



The Best Way to Debug

Don't put the bugs into the program in the first place

- Ensure you know the syntax for functions
- Walk through your code before you compile
 - Don't let the compiler find the bugs for you – it won't necessarily.

Incremental testing

- Write a small unit of code. Examine it. Compile it. Test it.
- Run test cases in your head ("What happens if...?")

