

C: an introduction

Debugging

Contents

- What is debugging
- Some hints
- Common C errors
- Use a debugger

C: syntax and semantics

- Syntax
 - rules of the grammar
 - Vocabulary recognized by the language
 - ANSI standard
- Semantics
 - the meaning of what is being said

Taken from UMD CMSC 106 Introduction to C Programming

C: syntax and semantics

- Incorrect Syntax
 - The compiler gives error message at that spot and refuses to compile it.
 - The compiler gives warning message at that spot but still compiles it.
 - The compiler gives error or warning message at a spot later in the file.
- Incorrect Semantics
 - Program does nothing when run
 - Program does nothing useful when run
 - Program does the “wrong” thing when run
 - Program “crashes” or “hangs” when running

Taken from UMD CMSC 106 Introduction to C Programming

Warnings, errors, bugs

- Compile-time warnings
- Compile-time errors
 - Typographical errors
- Link-time errors
 - Missing modules or library files
- Run-time errors
 - Null pointer assignment
- Bugs
 - Unintentional functionality

Taken from C Programming Basics – Part 1 / Ritu Arora / TACC 2013

What is debugging?

- Debugging is *not* getting the compiler to compile your code without syntax errors or linking errors.
- Debugging is what you do when your code compiles, but it doesn't run the way you expected.
Basic method of all debugging:
 - Know what your program is supposed to do.
 - Detect when it doesn't.
 - Fix it.

- <http://www.cs.yale.edu/homes/aspnes/classes/223/notes.html>

Bug

Photo # NH 96566-KN First Computer "Bug", 1945

9/9

0800 Antman started
1000 " stopped - antman ✓
1300 (033) HP - MC 1.2700 9.032 447 025
033) PRO 2 2.130476415 9.037 846 995 correct
correct 2.130676415 4.615 925059 (-2)
Relays 6-2 in 033 failed speed test
in relay " 11.000 test.
Relays changed
1100 Started Cosine Tape (Sine chest)
1525 Started Multi-Adder Test.
1545 Relay #70 Panel F
(moth) in relay.
First actual case of bug being found.
Antman started.
1700 closed down.

<http://www.history.navy.mil/photos/pers-us/uspers-h/g-hoppr.htm>

KU LEUVEN

Hints

- To solve a problem, you must understand it.
Make life easier for yourself:
 - Indent your code properly.
 - Use meaningful variable names.*Follow a programming style guide*
- Develop incrementally.
Test as you go.

KU LEUVEN

Get help from the compiler

- The C compiler is not as pedantic as other compilers, but it can still help you out.
- Use the compiler flags. With gcc:
 - -g includes debugging systems
 - -Wall turns on (almost) all compiler warnings.

```
gcc -g -Wall -o foo foo.c
```
- Use the manual. On Unix the man pages are an excellent reference for all standard library functions:

```
man fgets
```

Hints

- When it doesn't work, pause, *read the output very carefully*, read your code, and think first.
 - It is way too easy to start hacking, without reviewing the evidence that's in front of your nose.
 - If there is not enough evidence to find and *understand* a problem, you need to gather more evidence.
- This is the key to debugging.

Program failure

Two ways to fail:

- Program did something different from what you expected.
- Program crashed.
 - In C, likely symptoms of a crash:
 - Segmentation fault: core dumped.
segfault_demo.c
 - Floating point exception: core dumped.
floatpointexcep_demo.c

Gathering evidence

- Program did something different from what you expected.
 - Task 1: figure out what the program did do.
 - Task 2: figure out how to make it do what you wanted it to do.
- Don't jump in to task 2 before doing task 1. You'll probably break something else, and still not fix the bug.(similar bugs may be ahead)
- Don't jump into the debugger *first* - it stops you from reading your code. A debugger is only one debugging tool.
 - Occasionally code behaves differently in a debugger.

Gathering evidence

Print the values of key variables. What are you looking for?

- Did you initialise them?
 - C won't usually notice if you didn't.
- Were type conversions done correctly?
- Did you pass the parameters in the wrong order?
 - Print out the parameters as they were received in the function.
- Did you pass the right number of parameters?
 - C won't always notice.
- Did some memory get overwritten?
 - A variable *spontaneously* changes value!

Debugger

- What is?

A software tool that is used to detect the source of program or script errors, by performing step-by-step execution of application code and viewing the content of code variables.
- A debugger loads a program (compiled executable, or interpreted source code) and allows the user to trace through the execution.
- See what's happening in your program by stepping through its instructions and looking at the changes in your variables.

Debugger

- basic operations:
 - single-step, or execute just the next line of code
 - set a breakpoint at some place in your code,
 - execute all code until it encounters a breakpoint, and
 - print a variable's current value.

Debugger

How to debug your c program?

- Print statement (requires no extra tools)
- extra option at the compiling stage: **-g**
`gcc -g -o ex-debug ex-debug.c`
- GDB debugger: common command-line debugger
- *Start gdb with `tui` option (text user interface) – more intuitif*
`gdb -tui`

GDB: Some useful commands

- `break linenumber` – *create breakpoint at specified line*
- `break file:linenumber` – *create breakpoint at line in file*
- `run` – run program
- `c` – continue execution
- `next` – execute next line
- `step` – execute next line or step into function
- `quit` – quit gdb
- `print expression` – *print current value of the specified expression*
- `help command` – *in-program help*
- `info:` *get info ex. info break (overview of breakpoints)*
- `delete number`: *delete breakpoint, using the number from the list*
- `clear linenumber`: *remove the break at linenumber*
- *File: gdb_example_1.c*

Postmortem investigation

- If you have a core dump file, you can examine this file with a debugger
- The executable has to be compiled with the `-g` option

```
gdb <executable name> core
```

Summary

A debugging tool cannot determine **what** your bug is, but it is a great value in determining **where** it is.

Hands-on

- Compile the code and check what is going on in gdb
- sum_2_random.c
- gdb_example_1.c

Common C errors

- Missing break in a switch statement

- Using = instead of ==

```
if (a = 0) { ..... }
```

- Spurious semicolon:

```
while (x < 10);  
x++;
```

- Missing parameters:

```
printf("The value of a is %d\n");
```

- Wrong parameter type in printf, scanf, etc:

```
double num;  
printf("The value of n is %d\n", num);
```

Common C errors

- Array indexing:

```
int a[10]; has indices from 0 to 9
```

- Comparing Strings:

```
char s1 = "test"  
char s2 = "test"  
if (s1 == s2)  
printf("Equal\n");
```

- You must use strcmp() or strncmp() to compare strings.

Common C errors

- Integer division:

```
double half = 1/2;
```

- This sets half to 0 not 0.5!
- 1 and 2 are *integer* constants.

- At least one needs to be floating point:

```
double half = 1.0/2;
```

- Or cast one to floating point:

```
int a = 1, b = 2;
```

```
double half = ((double)a)/b
```

Common C errors

- Missing headers or prototypes:

```
double x = sqrt(2);
```

- sqrt is a standard function defined in the maths library.
- It won't work properly if you forget to include math.h which defines the function prototype:

```
double sqrt(double)
```

- C assumes a function returns an int if it doesn't know better.

- Also link with -lm:

```
gcc -o foo -lm foo.c
```

Common C errors

- Spurious pointers:

```
char *buffer;  
fgets(buffer, 80, stdin);
```
- With pointers, always ask yourself :
 - “*Which memory is this pointer pointing to?*”.
 - If it’s not pointing anywhere, then assign it the value NULL
- If your code allows a pointer to be NULL, you *must* check for this before following the pointer.

Errors: conclusion

Lots of ways to shoot yourself in the foot.

- Good style helps prevent dumb errors because your code is more readable.
- Debugging is an art, acquired by practice.
- If you’re really stuck, *take a break*.
 - Debugging is hard if you’re tired.
 - Your subconscious often needs space to debug your code by itself.