# EECS 192: Mechatronics Design Lab

## Discussion 8: Debugging & Telemetry

GSI: Justin Yim

13 & 14 Mar 2019 (Week 8)

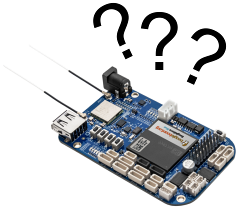
## Quick Tips

- Don't forget to order your boards' components
- Make sure individual components are reliable and working well
- Software tips:
  - Follow good coding style practices
  - Break related source code out into separate files instead of writing one monster file
    - e.g. main.c, camera.c, motor_control.c, etc.

# GNU Debugger

## Debugging

- General idea: locate bugs in your program by stopping it at particular points and looking at values
- GDB (gdb) "GNU Debugger" for C, C++, and other languages:
  - Install GDB
  - Locate where in your code to debug
  - Compile for debugging
  - Examine program with GDB
- (Documentation link)
- (Link to more detailed slides)



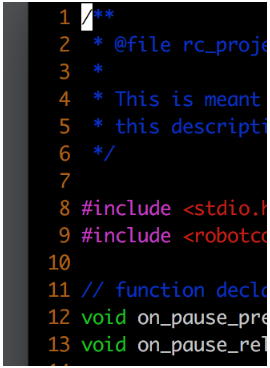
What's wrong with my code?

## Installing GDB

- GDB is not installed in our BBBL Debian distribution:
  - `sudo apt-get update`
  - `sudo apt-get install gdb`

## Line Numbers

- GDB often invovles looking at particular lines of code
- You will want to have line numbers so you can tell where GDB is working
- Cloud9 on Debian (192.168.7.2:3000) has line numbers
- Vim `set number`
- Other configurations for other text editors



```
 1 /**
 2  * @file rc_proje
 3  *
 4  * This is meant
 5  * this descripti
 6  */
 7
 8 #include <stdio.h
 9 #include <robotc
10
11 // function decl
12 void on_pause_pre
13 void on_pause_rel
```

Line numbers

## Compiling for GDB

- When compiling with gcc, use the -g option
  - `gcc -g (source) -o (output)`
- For rc library use
  `make debug`

## Commands

- To run a program in GDB:
  - `gdb (program name)`
    OR
  - `gdb`
    `(gdb) file (program name)`
- (Link to list of some useful commands)
  - help, file, run, break, watch, delete, continue, step, next, print

## Commands

See the list link for more information

- `quit` - exit gdb
- `help (topic)` - get more information about topics
- `file (program file)` - runs a program compiled for debugging

## Commands

See the list link for more information

- `run` - lets the program run as usual (until breakpoint or other event)
- breakpoints:
  - `break (line or function)` - sets a breakpoint to stop the program at a line or function call
  - `watch (variable)` - stop each time a watched variable changes
  - `continue` - continue running after stopping
  - `info breakpoints` - list info about all breakpoints
  - `delete` - clear all breakpoints

## Commands

See the list link for more information

- What do you do after you've hit a breakpoint?
  - `step` - execute the current line (stepping into a called function)
  - `next` - go to the next line (stepping over a called function)
  - `print (expression)` - display value of an expression (like a variable name)

```c
void foo() {
  printf("world");
}

int main() {
  printf("hello␣");
  foo();
  printf("!\n");
  return 0;
}
```

## GDB Example

## Logfiles

## Logfiles

- Logfiles are useful to see what went on
- For debugging "what's wrong with my ... whole car?"
- Two ways to write files while the car runs:
  - Save data to a variable (like an array or struct). Save the pre-saved data once the car stops running
    - See SkeletonBeagle/LineCamera/LineCamera.c
  - Run a low-priority low-rate loop to save *some* data
    - See telem_loop in SkeletonBeagle/rc_balance/rc_balance2.c

Beautiful telemetry data

## But really, why do I want logfiles?

- Can help catch odd bugs
- e.g.: encoder reading thread accidentally set at very low priority
    - encoder updates infrequently and sporadically
    - telemetry will show the sporadic encoder updates
- We will eventually ask you to turn in plots of control responses (you will need to log data somehow)

## Summary

- Build simple, robust components
- GDB to debug component software
- Logfiles to debug integrated systems