Debuggers

- Symbolic debuggers (adds symbol table to compiled binary)
- Conditional compiling

GDB

- Insert breakpoints
- Navigate memory, stacks, etc
- Look at core files
- Attach running processes
- GUI frontend - DDD

Compiling for GDB

- `-g` option while compiling

GDB commands

- `help <command>`
- `s/n` Single step
- `run` Run binary (starts from main)
- `l` List
- `disas foo` Disassemble function foo
- `b` Insert breakpoint
  - `b <line_num>`
  - `b foo()` Insert breakpoint at first executable line in function foo()
- `continue` Continue from breakpoint to end at full speed
- `display <symbol>` Display value held by `symbol`
- `watch <symbol>` Watchpoint (dynamic breakpoint)
  - Stop whenever symbol value changes
- `return` Return to calling routine immediately
- `delete <break-point>` Remove breakpoint
  - `clear` Removes all breakpoints
- `info registers` Show data in processor GPRs

The stack

- Divided into contiguous "frames"
- Each frame has data associated with a single function
  - Created when a function is called
- A frame contains-
  - Args given to the function
  - Local vars
  - Address where it is executing
- Initial frame - `main`

- `backtrace n` lists stack frame info for first n frames
- `frame n` Lists info about the nth frame
- `info frame`
- `x <addr>` Examine stack
  - `x/10x $sp` Show 10 values in hex from stack pointer register

Debugging processes

- Program should have a fork() sys call
- Can monitor either the child or the parent at one time
- `set follow-fork-mode <mode=parent/child>` Follow parent or child after forking. Default is parent
- `$ gdb ./a.out <pid>` Attach process with ID pid to gdb (may need superuser priviliges)
  - e.g. A program stuck in an infinite loop - when attached, program stops execution wherever it was before and enters debug environment.

Examining the core

- `ulimit -c <size>` Generate a core after compiling of given size
- `$ gdb a.out core` Assign core and debug. Program stops where the core was generated/dumped