

# Gnu Debugger (gdb)

Debuggers are used to:

- Find semantic errors
- Locate seg faults and bus errors

# Using GDB

- When to use a debugger?
  - Sometimes you can figure out errors just by using cout (print statements)
    - Incorrect output
    - Unexpected executions
  - Debuggers permit fine-tuned control
    - An absolute must for finding subtle and more complex errors
  - Debuggers quickly provide the location of run-time errors

# Using GDB

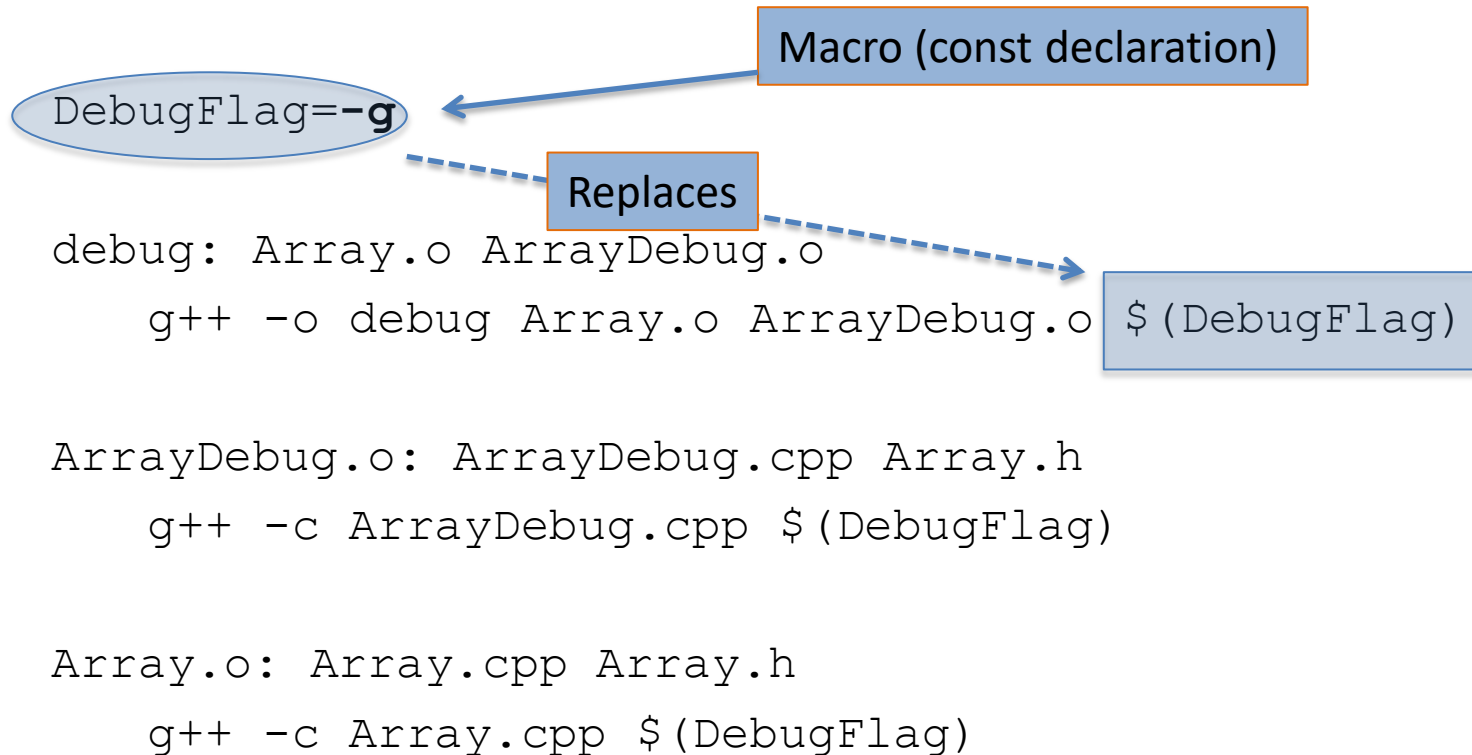
- Basic Functions of a Debugger:
  - Run Program & Enter/Exit Debug Mode
  - In Debug Mode:
    - Control Execution
    - Watch Things

The best option is usually to run gdb inside emacs

# Using GDB

- First step: Compile the program with flag for debugging
  - Flag: -g
    - Instructs the compiler to retain user's code
      - Otherwise, resulting machine code bears no resemblance to original code
    - Note use of -g in makefile (example in next slide)
      - In makefile, -g employed easily via macro

# Array Debug Example's Makefile



If `-g` is removed from macro, `$(DebugFlag)` is replaced by nothing

# Starting GDB

- Run gdb inside emacs
    - Provides dual window environment
      - Top window: Command environment
      - Bottom Window: Code Being Debugged
1. Build Using *make*
  2. Start emacs
  3. ESC-x (Display at bottom: M-x)
  4. gdb <Enter> <Enter>

You will be in the debugging environment

There will be a single window at this time

# Run Program & Enter/Exit Debug Mode

- Breakpoints
  - Designate a location where execution is suspended and debug mode entered
  - Command:
    - break <argument>
  - Three possibilities for <argument>
    - line number
    - function name
    - PC address

Note: Underlined character(s) in command are shortcuts

# Run Program & Enter/Exit Debug Mode

- Break Command Arguments
  - line number
    - Use <file name>:<line number> in other files
      - Example: `b Array.cpp:121`
    - Can appear alone in application file (some versions of gdb only)
  - function name
    - Can appear alone in application file
    - Use <class name>::<function name> in other files
      - Example: `b Array::~~Array`
  - PC address
    - Preface address with \*
    - More commonly used with assembler code

**Note:** Tab completion for setting breakpoints is available



# Run Program & Enter/Exit Debug Mode

- Set up breakpoints before starting the program
- Run the program
  - Command: `r`un <cmd line argument(s)>
    - program will run until it hits a breakpoint
- Resume execution:
  - Command: `c`ontinue

You can also use `r`un to restart a currently running program if you want to go back to the beginning

# Run Program & Enter/Exit Debug Mode

- When a breakpoint is encountered:
  - Execution stops
  - The screen will split
    - New window opens showing current file with arrow (=>) to current line
      - this line hasn't actually been executed yet
  - Program is in debug mode
    - Use debugger commands
      - Control
      - Watch
- Removing Breakpoints
  - Once a breakpoint's usefulness has ended it may be removed
  - Command: ddelete <breakpoint number>
    - No argument will cause prompt to delete all breakpoints
    - Breakpoint number is by order breakpoints were established
      - given when created or when reached during execution

# Control Execution

Run one line at a time

- Commands:
  - step
  - next
- The difference between step and next is when the current statement is a function call
  - **next** executes the function
    - If function has breakpoint, it will stop there and re-enter debug mode
  - **step** enters the function to debug it
    - Stops at first line to await next command

# Control Execution

- Other commands:
  - finish
    - Resume execution until end of current function or a breakpoint is encountered
  - up <# frames>
    - Go up the number of functions indicated in the stack
    - If the argument is 1, goes to the line where the current function was called
  - down <# frames>
    - Opposite of up

# Control Execution

## Entering a function

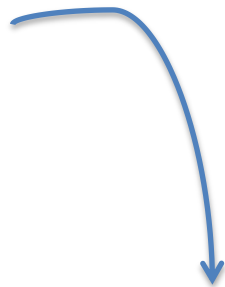
- When a function is entered, gdb displays information about this call
  - Name of function
  - Parameters, including values
- Pitfall: Entering a library function
  - e.g. The stream insertion operator
    - The window footer gives file name and line number
  - DO NOT try to debug in here
    - Use fin to exit back to where you entered

# Watching Stuff

- View variable and test functions
  - Commands:
    - print
    - display (no shortcut key)
  - **print** displays value of its argument
    - argument can be quite intricate
      - array : shows address; you can supply subscript
      - object: will try to provide value of all members
      - if item is address, \* can be used to dereference
      - argument can be function call!!
        - » function will be executed
        - » if function alters program data, alteration sticks
  - **display** is a persistent print
    - shows argument value after each command when argument is in scope

# Finding Causes of Crashes

- Run-time Errors' Location(s) are not Reported in Unix
  - Must use gdb to find the location and examine program state at time of crash
  - Usually, the state at the time of crash is preserved
    - If not, once location is determined, set breakpoint before line of crash to examine variables, etc;
  - Procedure



# Determine Location of Crash

- Steps to find location:
  1. Start debugger
  2. Run program using same input
    - No breakpoints; just let it crash
  3. Use `where` command to show run-time stack
    - displays sequence of function calls to arrive at current location
    - Each function's call in the stack is numbered
    - Find the 1<sup>st</sup> function in the list that you wrote. Note the number **X**
      - The first several functions may be library functions
  4. Issue command `up <X>`
    - Screen will split and display line where crash occurred (`=>` denotes)
    - Use `print` or `display` to examine variables for irregularities.



# Resources

- [Quick Primer by Dr. Spiegel](#)
- [Complete Manual - Delore.com](#)
- [GDB Cheat Sheet](#)
- [YoLinux Command Cheat Sheet](#)