

Debugger

Winter 2022









www.phdcomics.com



Debugger

- As your code becomes longer and more complicated, it becomes more difficult to troubleshoot – especially runtime errors such as the dreaded "segfault"
- Inserting printf statements everywhere is an inefficient way to debug
- It would be easier to step carefully line-byline much like the pythontutor tool does



- A debugger is an interactive tool a programmer can use to easily debug their program
 - Start your program with certain conditions
 - Pause your program at certain points
 - Inspect and change your program mid execution
- gdb Is a popular Unix debugger



- To use gdb effectively, you should first compile your program with the -g option
 - gcc -g -o pun pun.c
- This tells the compiler to insert some additional information in the executable
- The gdb utility will use this information to give better feedback while debugging



- Then, use gdb to run your program
 - gdb ./pun
 - gdb -p <pid>

```
[wbeldman@compute Lecture-2]$ gdb ./pun
Copyright (C) 2021 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <a href="http://gnu.org/licenses/gpl.html">http://gnu.org/licenses/gpl.html</a>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./pun...
(gdb)
```





- Now you can enter commands to carefully inspect and run your program
 - h [command] Print help information (specifically about "command" if supplied).
 There are many gdb commands. Only the common ones are listed here.
 - q Quit

- More commands
 - b [filename:]function Set a breakpoint at function (in filename)
 - b [filename:]line_num Set a breakpoint at line_num (in filename)
 - bt Print a backtrace of your function calls (you can also use the command "where")



- More commands
 - r [arglist] Run the program using arglist if any
 - I List (or print) the source code at the current location
 - I [filename:]function
 - I [filename:]line_num



- More commands
 - c Continue execution after the program paused on a break point
 - n Execute the next line in your code. Do not debug function calls
 - s Execute the next line in your code. Debug function calls as well



- More commands
 - p expression Print the value of "expression".
 This could be a variable name
 - display expression Print the value of "expression" every time your code stops at a breakpoint
 - watch expression Pause and print the value of "expression" every time it changes



