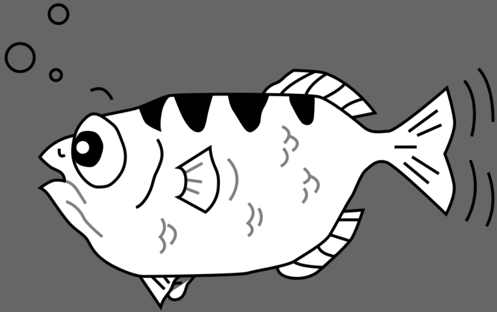
GNU Debugger (GDB)

COMP201 Lab Session
Fall 2020

KOÇ UNIVERSITY

What is GNU Debugger (GDB)?



www.gnu.org/software/gdb

"GNU Debugger" is a debugger for several languages, including C and C++. It was first written by Richard Stallman in 1986 as part of his GNU system.

- It allows you to inspect what the program is doing at a certain point during execution.
- Errors like segmentation faults may be easier to find with the help of gdb.

KOÇ UNIVERSITY

Start with GDB

When compiling you need to add a “-g” option to enable built-in debugging support :

- $ gcc [other flags] -g <source files> -o <output file>
  - e.g. : $ gcc -g main.c -o main.out

To run GDB simply run:

- $ gdb [compiled_file]
  - e.g. : $ gdb main.out

GDB prompt

- GDB has an interactive shell, much like the linux terminals. It can recall history with the arrow keys, auto-complete words (most of the time) with the TAB key, and has other nice features like help command (help [command]) that provides more details of the command.

```
GNU gdb (Ubuntu 8.1-0ubuntu3.2) 8.1.0.20180409-git
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
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-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://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 main.out...done.
(gdb)
```

Running a program in GDB

To run the program, just use

- (gdb) run

This runs the program

- If it has no serious problems (i.e. the normal program didn't get a segmentation fault, etc.), the program should run fine here too.

- If the program did have issues, then you (should) get some useful information like the line number where it crashed, and parameters to the function that caused the error:

Line Breakpoint

Running the program in case it’s buggy or crashes may not be that useful, so instead you can set a breakpoint, a line of the code your debugger stops there and you can run the code line by line while you can observe variables.

To set a breakpoint use (let’s say your source code file is named my_file.c)

- (gdb) break my_file.c:7
  or
- (gdb) b my_file.c:7

This sets a breakpoint in line 7 of the my_file.c code and when you run the code, if the program ever reaches line 7, the debugger stops there.

Function name Breakpoint

You can also tell gdb to break at a particular function. Suppose you have a function named “myfunc” then you can set a breakpoint for whenever this function is called using

- (gdb) break myfunc
  or
- (gdb) break myfunc

KOÇ UNIVERSITY

What next?

After setting suitable Breakpoints and running the program using “ (gdb) run “ command, it should stop where you tell it to. You can proceed onto the next breakpoint by typing

- (gdb) continue

Or you can execute a single-step (execute just the next line of code) by typing

- (gdb) step

  or
- (gdb) next

  or
- (gdb) n

  or
- Simply press ENTER after once you called next, the ENTER executes the last command again

KOÇ UNIVERSITY

How to check variables?

If you reached to a desired point and you want to see things like the values of variables, etc.

The “print” command prints the value of the specified variable

- (gdb) print [variable]
- e.g. : (gdb) print my_var

And “print/x” prints the value in hexadecimal

- (gdb) print/x [variable]

KOÇ UNIVERSITY

How to watch for any change?

Whereas breakpoints interrupt the program at a particular line or function, watchpoints act on variables. They pause the program whenever a watched variable's value is modified. For example, the following watch command:

- (gdb) watch [variable]
- e.g. : (gdb) watch my_var

Whenever my_var's value is modified, the program will interrupt and print out the old and new values.

KOÇ UNIVERSITY

Conditional breakpoint!

Just like regular breakpoints, except that you get to specify some criterion that must be met for the breakpoint to trigger.

- (gdb) break my_file.c:6 if i >= ARRAYSIZE

This command sets a breakpoint at line 6 of file my_file.c, which triggers only if the variable “i” is greater than or equal to the size of the array (suppose ARRAYSIZE is defined). Conditional breakpoints can most likely avoid all the unnecessary steppings and time wastings.

KOÇ UNIVERSITY

More useful commands

Other useful commands:

- **(gdb) backtrace** produces a stack trace of the function calls that lead to a seg fault (similar to Java exceptions)
- **(gdb) where** same as backtrace; you can think of this version as working even when you're still in the middle of the program
- **(gdb) finish** runs until the current function is finished
- **(gdb) delete** deletes a specified breakpoint
- **(gdb) info breakpoints** shows information about all declared breakpoints
- **(gdb) info locals** shows local variables and their values in the current score

Want a better user interface?

After entering th gdb prompt, by typing “layout next” you can enter to the display layout that shows you breakpoints and source code with the prompt together.

```
main.c
    3       #include <ctype.h>
    4       #include <string.h>
    5
B+  6
    7       char* toLower(char* s) {
  > 8         for(char *p=s; *p; p++) *p=tolower(*p);
    9         return s;
    10      }
    11      char* toUpper(char* s) {
    12        for(char *p=s; *p; p++) *p=toupper(*p);
    13        return s;
    14      }

native process 31296 In: toLower                          L8    PC: 0x5555555547e6
(gdb) b main.c:6
Breakpoint 1 at 0x7e6: file main.c, line 6.
(gdb) run
Starting program: /home/farzin/workplace/COMP201_labB/Lab_03/main.out
Palindrome test passed!
Breakpoint 1, toLower (s=0x7fffffffdb32 "radar") at main.c:8
(gdb)
```

KOÇ UNIVERSITY