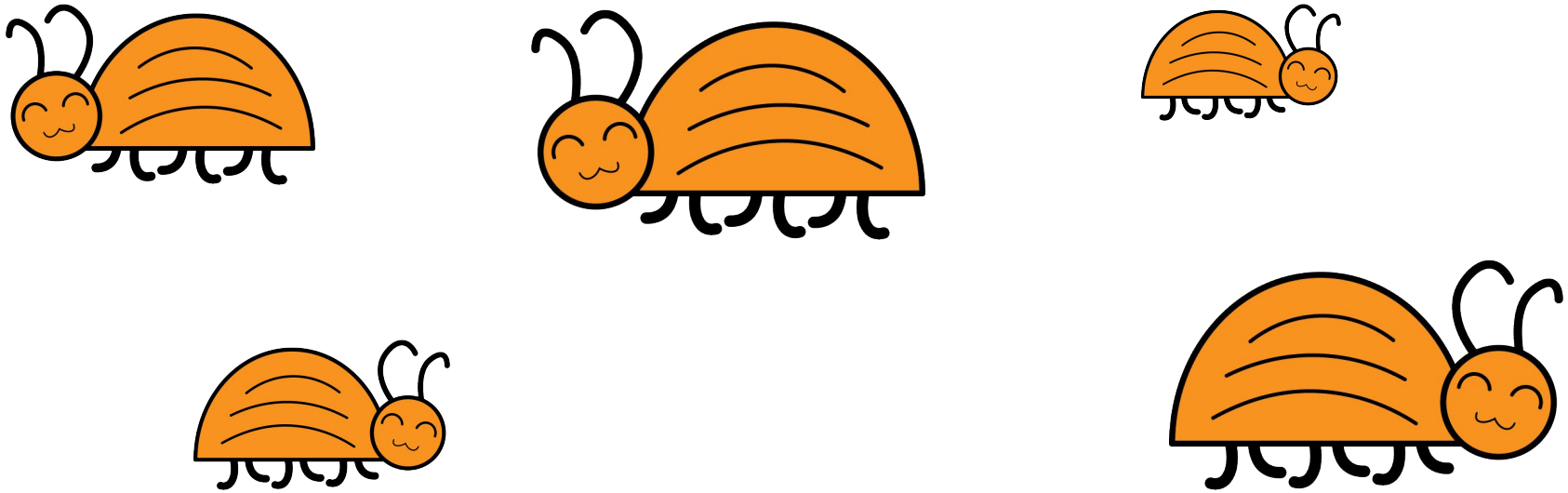
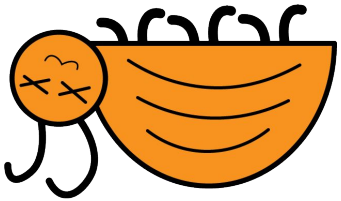
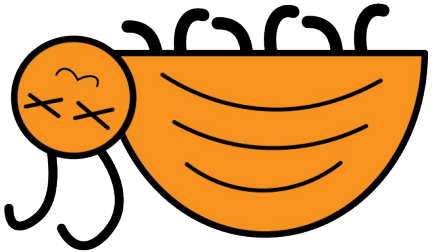
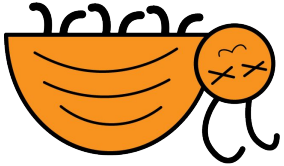
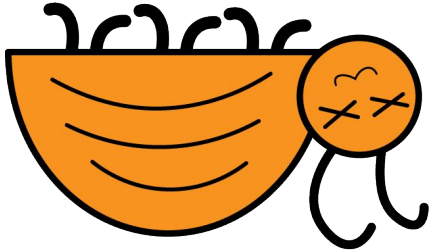
GDB



Workshop by
Akrit Shrikant and Sophia Adrangi

Slides originally by:
Kristie Lim and Jonathan Myong

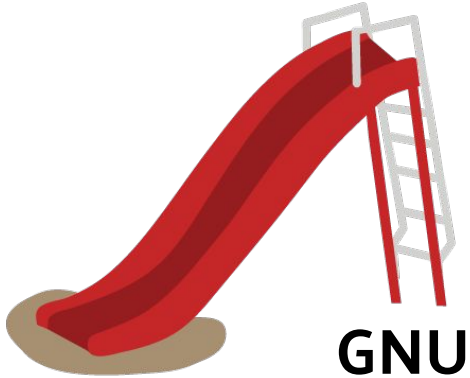
What is GDB?

GNU DeBugger

Like a debugger in XCode or Visual Studio, GDB lets you stop your program at certain points and look at variables. However, GDB is a command line tool (like ls or cd).

Fun Fact Slide

What does GNU stand for?



GNU’s Not Unix

GNU’s Not Unix

GNU’s Not Unix

GNU’s Not Unix

How will we use GDB in the lab?

You can use GDB as a debugger, but we'll be using it to analyze assembly code. We can:

- **View contents of registers**
- **View contents at different addresses such as on the stack**
- **Step through lines of assembly code**

We'll be talking about the following commands:

- **Run**
- **Breakpoints**
- **Continue**
- **Step**
- **Print**
- **Examine**

DEMO/Walkthrough

Here's where you can download our demo program:
https://tinyurl.com/avengers33

Follow along!

How to start gdb

In the folder that contains your executable, type gdb <executable_name>

```
[kristiel@lnxsrv09 ~/Desktop/33/bomb5]$ gdb bomb
GNU gdb (GDB) 8.2
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-pc-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 bomb...done.
(gdb)
```

How to quit gdb

Type quit and you'll return to your shell.

```
(gdb) quit
[kristiel@lnxsrv09 ~/Desktop/33/bomb5]$
```

Run a program

For all of the following commands, assume you type them at the gdb prompt, after (gdb)

```
run
```

If you run the program without setting any breakpoints, it will run normally without stopping.

Running With Arguments

You may want to supply arguments to your program:
(e.g. `./my_program "hello"`)

`run "hello"`

Repeatable Text

The bomb lab (and attack lab) will ask for you to supply input text.

This can get repetitive and is prone to error.

GDB allows you to supply a file as input, instead of you typing.

Each line of the file will correspond to one "prompt" by the program.

```
run < input.txt
```

Additional References: https://stackoverflow.com/q/4758175

Breakpoints

To stop the program at a particular point

```
break location
```

The argument for break can be a function name or an address. (It can also be a line number, but since you aren't given the source file, this is not very useful.)

Documentation: https://ftp.gnu.org/old-gnu/Manuals/gdb/html_node/gdb_28.html

Breakpoints

- **Break at a function name**

```
(gdb) break explode_bomb
Breakpoint 1 at 0x40165f
(gdb) run
Starting program: /w/home.01/cs/ugrad/kristiel/Desktop/33/bomb5/bomb
    .------ /------.
    |       -       |
    |               |        It's a perfect day for some mayhem.
    |               |
    |               |
 ________________________
 ===========.===========         Have some fun with my six
   / ~~~~~     ~~~~~ \           explodey phases ...
  /|     |     |     |\
  W   ---  / \  ---   W
  \.      |o o|      ./
   |                 |
   \    #########    /
    \  ## ----- ##  /        Watch your step!
     \##         ##/
      \_____v_____/
lsdjfosdjkf

Breakpoint 1, 0x000000000040165f in explode_bomb ()
```

Breakpoints

- **Break at an address (note the asterisk)**

```
(gdb) break *0x40165f
Breakpoint 1 at 0x40165f
(gdb) run
Starting program: /w/home.01/cs/ugrad/kristiel/Desktop/33/bomb5/bomb
   .------ /------.
   |       -       |
   |               |        It's a perfect day for some mayhem.
   |               |
   |               |

------------------------
===========.===========         Have some fun with my six
  / ~~~~~     ~~~~~ \           explodey phases ...
 /|     |     |     |\
 W   ---  / \  ---   W
 \.      |o o|      ./
  |                 |
  \    #########    /
   \  ## ----- ##  /        Watch your step!
    \##         ##/
     \_____v_____/
dlsjfodis

Breakpoint 1, 0x000000000040165f in explode_bomb ()
(gdb)
```

Short Form of Commands

Most commands in gdb will also have a short form, so instead of typing *`break`*, **you can just type** *`b`*.

```
(gdb) b phase_1
Breakpoint 2 at 0x400ef3
```

The previous two commands, `run` **and** `quit`**, can also be replaced by single characters.**

Disassemble

Produces the disassembled output of the specified function (like objdump)

```
disassemble <function_name>
```

The argument for disassemble is usually a function name. You can also run disassemble without an argument, but only when in a function. You can also give memory ranges.

Side tip: can also be called by `disas`
Side side tip: `set disassemble-next-line on`

Documentation: https://sourceware.org/gdb/current/onlinedocs/gdb/Machine-Code.html
https://visualgdb.com/gdbreference/commands/disassemble

⚠️ Bomb Lab Tip ⚠️

Right when you start gdb, always type:

```
b explode_bomb
```

before doing anything else.

That way, when you run the bomb program, you will stop execution if you reach the `explode_bomb` **function.**

⚠️ Bomb Lab Tip 2 ⚠️

You can find the function names and signatures (i.e. argument types and return types) of a program by typing:

```
info functions
```

Continue

Keep running to the next breakpoint

```
continue
```

When at a breakpoint, continues running the program until another breakpoint, or the end

Continue

- **Until another breakpoint**

```
Breakpoint 2, 0x0000000000400ef3 in phase_1 ()
(gdb) c
Continuing.

Breakpoint 1, 0x0000000000401657 in explode_bomb ()
(gdb)
```

`Continue` **is abbreviated to** `c`

Continue

- **Until the end :(**

```
Breakpoint 2, 0x0000000000401657 in explode_bomb ()
(gdb) c
Continuing.
Oh, you really stepped in it, mate!

BOOM!!!
The bomb has blown up.
Your instructor has been notified.
[Inferior 1 (process 20916) exited with code 010]
(gdb)
```

Step

Executes line by line

```
step
```

Runs one line in the source code, and stops and return

Step

```
step count
```

The *count* argument is optional, and allows for stepping through multiple lines

Step

```
(gdb) s
Single stepping until exit from function phase_1,
which has no line number information.

Breakpoint 1, 0x0000000000401657 in explode_bomb ()
```

Without line number information given, acts like the command "`continue`"

`Step` **is abbreviated to** `s`

Stepi

Executes instruction by instruction

```
stepi
```

Runs one machine instruction, and stops and return

Step

```
Breakpoint 1, 0x0000000000400ef3 in phase_1 ()
(gdb) si
0x0000000000400ef4 in phase_1 ()
(gdb) si
0x0000000000400ef7 in phase_1 ()
(gdb) si
0x0000000000400efb in phase_1 ()
(gdb)
```

`Step` **works even without line number information**

`Stepi` **is abbreviated to** `si`

Next

Also executes line by line…with one difference

```
next
```

Runs one line in the source code, but *doesn't* "step into" functions. IE, proceeds through subroutine calls.
Like step, can also be run with the [count] argument

Nexti

Executes instruction by instruction, proceeding through subroutine calls

```
nexti
```

Runs one machine instruction, and stops and return

Print

To print the value of registers (and other variables)

```
p/format what_to_print
```

The argument for break can be a register or some number. (It can also be a variable name, but since you aren't given the source file, this is not very useful.)

Format

The character after the slash that says how to format the argument

Some format characters:

- `x` **for hexadecimal**
- `d` **for decimal**
- `c` **for character**
- **nothing for default**

```
(gdb) p/x 16
$6 = 0x10
(gdb) p/d 0x10
$7 = 16
(gdb) p/c 65
$8 = 65 'A'
(gdb) p $rsp
$9 = (void *) 0x7fffffffe068
```

Full list of output formats:
https://ftp.gnu.org/old-gnu/Manuals/gdb/html_node/gdb_54.html

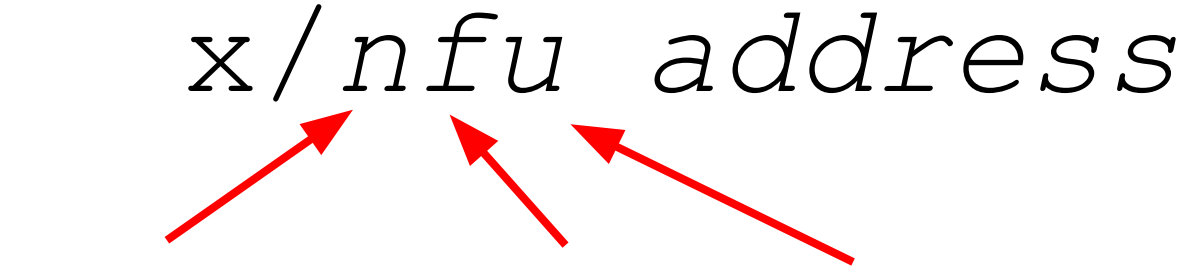
Print

Don’t forget the dollar sign when printing a register. Also note that there is no % sign.

```
p $rax
```

Examine

Take a look at what an address points to



number format unit

Examine

The *nfu* options are optional, and default values will be passed in if you don't have one or more of the characters.

Some particularly useful formatting characters for x:

- `s` **string**
- `i` **instruction**

Examine

- **Example:**
  - **View 10 instructions from the current instruction**

```
x/10i $rip
```

If you don't put a number, it will default to 1.

gdb will default to the last format you used if no format is specified

***$pc* (for program counter) is the same as *$rip* if you want to type one less character.**

Examine

- **Example:**
  - **View 10 instructions from the current instruction**

```
(gdb) x/10i $rip
=> 0x40146b <welcome_message>:  cmp    $0x1,%edi
   0x40146e <welcome_message+3>:        je     0x401471 <welcome_message+6>
   0x401470 <welcome_message+5>:        retq
   0x401471 <welcome_message+6>:        sub    $0x8,%rsp
   0x401475 <welcome_message+10>:       mov    $0x402ba0,%edi
   0x40147a <welcome_message+15>:       callq  0x400b60 <puts@plt>
   0x40147f <welcome_message+20>:       mov    $0x402be8,%edi
   0x401484 <welcome_message+25>:       callq  0x400b60 <puts@plt>
   0x401489 <welcome_message+30>:       mov    $0x402c30,%edi
   0x40148e <welcome_message+35>:       callq  0x400b60 <puts@plt>
```

Examine

- **Example:**
  - **View 20 bytes in hexadecimal at a particular address (b stands for bytes, it is often the default)**

```
x/20xb 0x402c30
```

```
(gdb) x/20xb 0x402c30
0x402c30:       0x20    0x20    0x20    0x7c    0x20    0x20    0x20    0x20
0x402c38:       0x20    0x20    0x20    0x20    0x20    0x20    0x20    0x20
0x402c40:       0x20    0x20    0x20    0x7c
```

Examine

- **Example:**
  - **View string stored at particular address**
  - **From running the previous command, I know that `0x402c30` is moved to `%rdi` (the parameter register) before calling `puts` (function used to print), so let's examine the memory at this address as a string!**

```
x/s 0x402c30
```

```
   0x401489 <welcome_message+30>:       mov    $0x402c30,%edi
   0x40148e <welcome_message+35>:       callq  0x400b60 <puts@plt>
(gdb) x/s 0x402c30
0x402c30:       "   |", ' ' <repeats 15 times>, "|        It's a perfect day for some mayhem.   "
```

Examine

- Note that this is the exact same address, just formatted differently.
- Each byte represents a character. (0x20 is the ASCII character for space and 0x7c is the vertical bar). You could manually figure out the string by looking up the characters in an ASCII table, but GDB does this for you!

```
   0x401489 <welcome_message+30>:       mov    $0x402c30,%edi
   0x40148e <welcome_message+35>:       callq  0x400b60 <puts@plt>
(gdb) x/s 0x402c30
0x402c30:       "   |", ' ' <repeats 15 times>, "|        It's a perfect day for some mayhem.   "
```

```
(gdb) x/20xb 0x402c30
0x402c30:       0x20    0x20    0x20    0x7c    0x20    0x20    0x20    0x20
0x402c38:       0x20    0x20    0x20    0x20    0x20    0x20    0x20    0x20
0x402c40:       0x20    0x20    0x20    0x7c
```

And more...

You might find other gdb commands useful too, and we encourage you to explore. Here are even more commands:

- disassemble
  - Show a disassembled function
  - https://ftp.gnu.org/old-gnu/Manuals/gdb/html_node/gdb_49.html
- display
  - Useful defaults for p and x, more display options
  - https://ftp.gnu.org/old-gnu/Manuals/gdb/html_node/gdb_56.html
- backtrace
  - View which functions you're in
  - https://ftp.gnu.org/old-gnu/Manuals/gdb/html_node/gdb_42.html
- info registers
  - Print values of all registers
  - https://stackoverflow.com/questions/5429137/how-to-print-register-values-in-gdb

GDB cheat sheet:
http://csapp.cs.cmu.edu/2e/docs/gdbnotes-x86-64.pdf

Slides:

https://docs.google.com/presentation/d/1mUR75UbBu6btTzC04bI8zY0FE9almjGUcFVZq3fwOLw

Contact:

devyanbiswas@outlook.com
k.leong@outlook.com