GDB and Valgrind

John Shepherd 2018
Phil Lopreiato, Neel Shah 2014

What's GDB?

- **G**NU **D**e**b**ugger
- Written by Richard Stallman in 1986
- Currently maintained by Free Software Foundation

Memory Leaks

- C allows the programmer all the power
- Memory management is left up to you
  - malloc (realloc, calloc, etc.) gives you memory for use
  - It is left up to you to tell the computer you are no longer using this memory (free), unlike other languages such as Java
- Num_of_mallocs = num_of_frees
- Data malloc'd should likely be free'd at the end of its scope (more specifically, its lifespan)

How to Catch Memory Leaks

- Manually - count mallocs and count frees, if equal then you likely have no problem
- Or allow Valgrind to do the work for you
  - **`valgrind ./linkedlist`**
- Look up the types of memory leaks that it outputs for this piece of code

How do I use gdb?

- Helps you debug programs when they aren't working properly
  - Incorrect values
  - ******Segfaults******
    - **Print statements <u>aren't</u> good for these!!!**
  - It's all broken
  - When you've run out of tears

Step 1

- Compile your code with debugging symbols enabled
  - Add -g flag to gcc command
  - Example:

```
gcc -g -o linkedlist linkedlist.c
```

Step 2.A. Run the Code

- ./linkedlist
- Let’s insert a node with value 5
- …
- …
- …
- …
- uhoh

Step 2.B.

- GDB to the rescue!
- Run the program with GDB
  - `gdb linkedlist`

The GDB Prompt

- Run ‘help’ for a list of commands

```
(gdb) help
List of classes of commands:

aliases -- Aliases of other commands
breakpoints -- Making program stop at certain points
data -- Examining data
files -- Specifying and examining files
internals -- Maintenance commands
obscure -- Obscure features
running -- Running the program
stack -- Examining the stack
status -- Status inquiries
support -- Support facilities
tracepoints -- Tracing of program execution without stopping the program
user-defined -- User-defined commands

Type "help" followed by a class name for a list of commands in that class.
Type "help all" for the list of all commands.
Type "help" followed by command name for full documentation.
Type "apropos word" to search for commands related to "word".
Command name abbreviations are allowed if unambiguous.
(gdb)
```

Step 3

- Run the program through GDB
  - `run`

```
(gdb) run
Starting program: /home/seas/johnshepherd/temp/linkedlist

List Operations
===============
1.Insert
2.Display
3.Size
4.Delete
5.Exit
Enter your choice :
```

Investigating Segfaults

- Use the ‘backtrace’ command to see the function calls prior to the segfault

- Use the ‘list’ command to see the code surrounding the error

```
(gdb) list
82              {
83                      add(num);
84              }
85              else
86              {
87                      while(temp->next!=NULL)
88                      {
89                              if(temp->data<num)
90                                      c++;
91                              temp=temp->next;
(gdb)
```

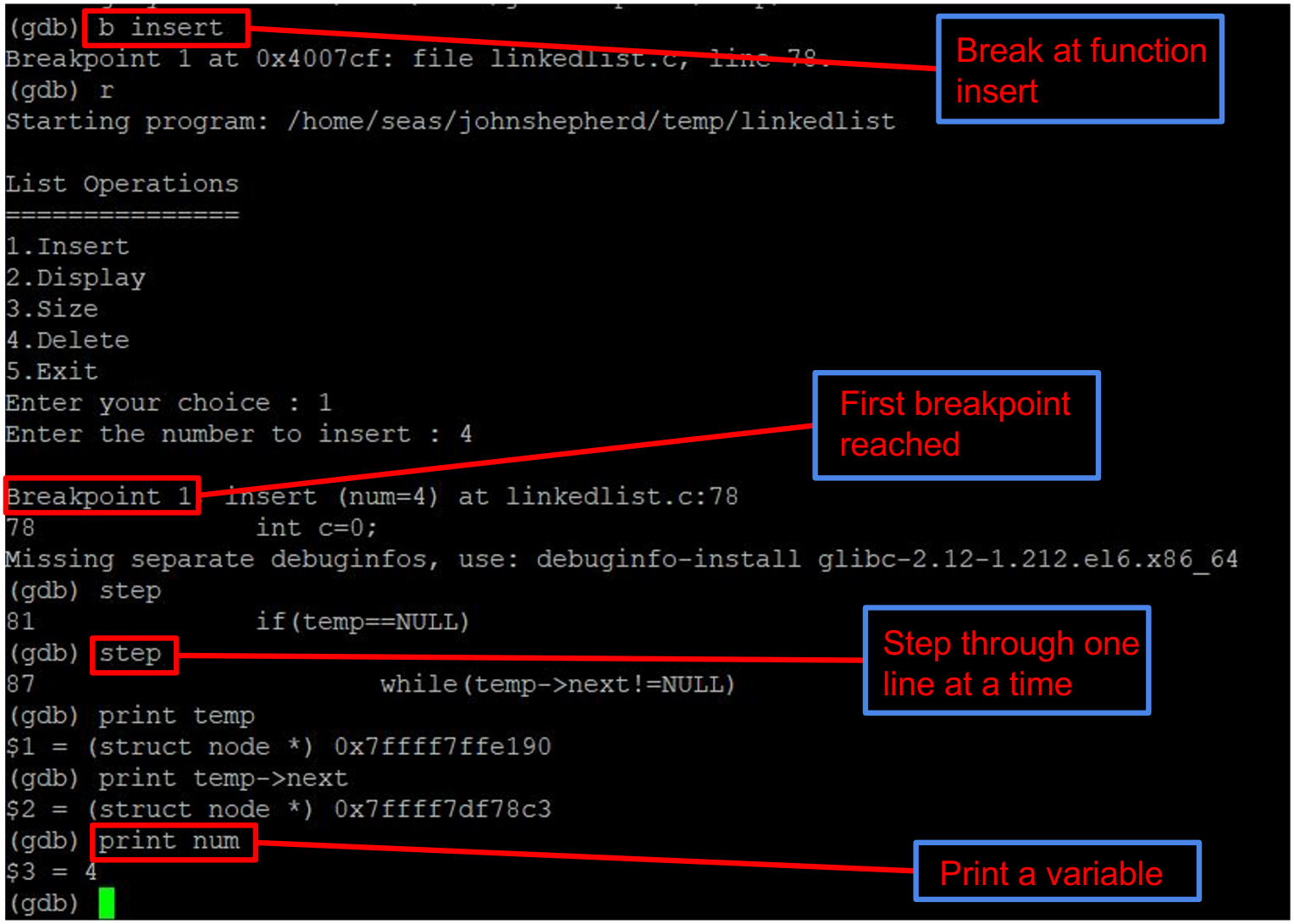
```
Program received signal SIGSEGV, Segmentation fault.
0x000000000040080b in insert (num=4) at linkedlist.c:87
87                      while(temp->next!=NULL)
Missing separate debuginfos, use: debuginfo-install glibc-2.12-1.212.el6.x86_64
(gdb) backtrace
#0  0x000000000040080b in insert (num=4) at linkedlist.c:87
#1  0x0000000000400a23 in main () at linkedlist.c:176
(gdb)
```

Breakpoints

- Breakpoints are a way of telling C to stop the program at a certain point
- You can examine memory once the program is stopped at a point
  - Check contents of variables at a point in your code
- Great for debugging!

Breakpoint Commands

- Set breakpoint
  - **`break <line num>`**
- Set breakpoint on function
  - **`break <func name>`**
- List of breakpoints
  - **`info breakpoints`**
- Remove breakpoints
  - **`clear <breakpoint num>`** ← retrieved from info
- Skip breakpoints
  - **`ignore <breakpoint num>`** ← retrieved from info


(gdb) b insert
Breakpoint 1 at 0x4007cf: file linkedlist.c, line 78.
(gdb) r
Starting program: /home/seas/johnshepherd/temp/linkedlist
List Operations
==========
1.Insert
2.Display
3.Size
4.Delete
5.Exit
Enter your choice : 1
Enter the number to insert : 4
Breakpoint 1, insert (num=4) at linkedlist.c:78
78 int c=0;
Missing separate debuginfos, use: debuginfo-install glibc-2.12-1.212.el6.x86_64
(gdb) step
81 if(temp==NULL)
(gdb) step
87 while(temp->next!=NULL)
(gdb) print temp
$1 = (struct node *) 0x7ffff7ffe190
(gdb) print temp->next
$2 = (struct node *) 0x7ffff7df78c3
(gdb) print num
$3 = 4
(gdb)
Break at function insert
First breakpoint reached
Step through one line at a time
Print a variable

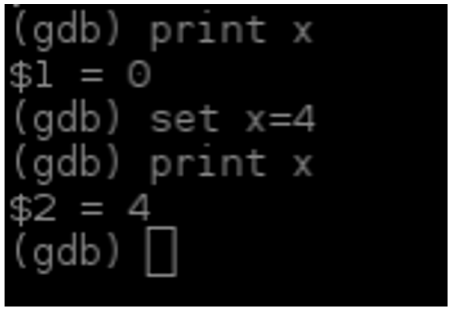
Step through code line-by-line

- When execution is stopped at a breakpoint, use ‘step’ to execute ***just*** the next line

Inspecting Your Code

- gdb allows you to view and modify all kinds of data used by the program such as
  - variables
  - stack frames
  - memory

View/Modify Variables



- Display variable contents
  - print <var name>
- Modify variable contents
  - set <var name> = <data>
- Watchpoints - stop and notify on variable change
  - watch <var name>
- Disable watchpoint
  - get id from ‘info breakpoints’
  - disable <id>

Exercise: Fix linkedlist.c

- Using gdb, valgrind, and your knowledge of C, modify the linkedlist.c program so that it runs without any errors, bugs, or memory leaks
  - Common commands:
    - `run`
    - `b <line>`
    - `list`
    - `step`
    - `continue`
    - `Quit`
    - `Up`
    - `Down`
    - `Print <var_name>`

- There are:
  - 5 bugs
  - 1 memory leak issue