CSCI 103: Introduction to Programming

Lab 5
GDB and valgrind

September 22nd, 2023
What is GDB?

• **gdb** = GNU Project Debugger
• Text-based debugger that you can run from the command line and is on most every Mac/Linux operating system
  - So if you aren't on Codio ...OR... **if the Codio debugger isn't working, you should just use gdb.**
  - In fact, the Codio debugger runs it behind the scenes
• After compilation, start GDB at the terminal prompt by typing:
  
  `gdb executable`
  
  - executable is the program name (e.g. `gdb ./shapes`)
• You will learn much more about GDB in CS104
Common Causes of Program Terminations

- Segfaults (reading or writing outside memory allocated)
- Aborts
- Infinite recursion (stack overflow)
- Unhandled exceptions / error cases
Starting GDB

Start at the terminal prompt: `gdb executable`
GDB Prompt

Once started it will show some information and take you to a prompt: (gdb)

- Remember to compile with the -g flag
- If it successfully opens your program you'll see "Reading symbols from <executable>...done."
- If you see "Reading symbols from shapes...(no debugging symbols found)", you forgot to compile with -g. Stop and recompile
Basic GDB Commands

- **run or r**: Executes the program from start to end
- **backtrace or bt**: Lists all the function calls (that leads to the crash) in the stack frame
- **up [n]**: Move n frames up the stack; n defaults to 1
- **down [n]**: Move n frames down the stack; n defaults to 1
- **print or p**: Used to display a variable/expression value
- **cont**: Continue execution until the next breakpoint or termination of the program
- **break or b**: Sets a breakpoint on a particular line
- **next or n**: Executes the current line of code fully (even if it contains a function call), stopping at the next line
- **step or s**: Executes the current line of code, stopping at the beginning of a function called by the current line
- **finish or f**: Completes execution of the current function and stops upon return to the caller
- **clear**: Clear all breakpoints
- **quit or q**: Quit gdb
GDB Commands (1)

Text based commands for the buttons you use in Codio

- (gdb) `break` or `b <filename:line_number>`
  - (e.g. `b shapes.cpp:37`) - Sets a breakpoint at line 37 of shapes.cpp

- (gdb) `run` or `r`
  - Runs the program until it hits a breakpoint

- (gdb) `print` or `p <variable>`
  - Prints the value of a variable at the current point in the code
Most Common GDB Use 1 - Segfaults

- When your program segfaults, your FIRST STEP ALWAYS should be to run it in the debugger.
- Step 1: Load gdb (gdb ./shapes)
- Step 2: At the prompt, run the program using run and let the program crash
- Step 3: When it crashes it will show you the source file and line number.
  - That line is where the error manifested, but the cause may be that line or some earlier line
  - However, it may have crashed deep in some library function. Use bt or backtrace to show the function call stack and look for the first function (and line number) that you actually wrote and investigate there
- Step 4: Use print to print variables relevant to the line where it crashed
Most Common GDB Use 1 - Segfaults (cont)

```
(gdb) r
Starting program: /home/cdio/workspace/shapes
To draw a rectangle, enter: 0 top left height width
To draw an ellipse, enter: 1 cy cx height width
To save your drawing as 'output.bmp' and quit, enter: 2
0 -5 -5 100 500
```

Draw a rectangle out of bounds causing a fault

```
Program received signal SIGSEGV, Segmentation fault.
0x00000555555554d1 in draw_rectangle (top=-5, left=-5, height=100, width=500) at shapes-temp.cpp:20
0x0000055555555551c in main () at shapes-temp.cpp:65
```

```
(gdb) print r
$1 = -5
(gdb) print left
$2 = -5
```

GDB Commands (2)

Text based commands for the buttons you use in Codio

• (gdb) `step`
  – If current line is function, it steps into the function pausing at the first line

• (gdb) `next`
  – Executes the next line, even if it is a function

• (gdb) `finish`
  – Finishes the current function and pauses back in the caller

• (gdb) `cont`
  – Continue to execute the program at full speed
What is Valgrind?

• Helps track down memory allocation and usage errors
  – Misuse, forgot to deallocate, using garbage values, accessing arrays out of bounds

• $ valgrind --tool=memcheck --leak-check=yes <executable> <command line args>

• $ valgrind --tool=memcheck --leak-check=yes ./scramble wordbank.txt 57

• You should use valgrind in every lab, hw, and project from now on (and in CS104!)
Valgrind Outputs

No error

==8146==
==8146== HEAP SUMMARY:
==8146== in use at exit: 0 bytes in 0 blocks
==8146== total heap usage: 10 allocs, 10 frees, 8,856 bytes allocated
==8146== All heap blocks were freed -- no leaks are possible
==8146== For counts of detected and suppressed errors, rerun with: -v
==8146== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 2 from 2)

Error

==8178==
==8178== HEAP SUMMARY:
==8178== in use at exit: 40 bytes in 6 blocks
==8178== total heap usage: 10 allocs, 4 frees, 8,853 bytes allocated
==8178==
==8178== 40 bytes in 6 blocks are definitely lost in loss record 1 of 1
==8178== at 0x4C2AC27: operator new[] (unsigned long) (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==8178== by 0x40109A: main (scramble.cpp:48)
==8178==
==8178== LEAK SUMMARY:
==8178== definitely lost: 40 bytes in 6 blocks
==8178== indirectly lost: 0 bytes in 0 blocks
==8178== possibly lost: 0 bytes in 0 blocks
==8178== still reachable: 0 bytes in 0 blocks
==8178== suppressed: 0 bytes in 0 blocks
==8178==
==8178== For counts of detected and suppressed errors, rerun with: -v
==8178== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 2 from 2)
Common Valgrind Errors

Invalid read/write: Array or pointer access/assignment to out-of-bounds location.

Conditional jump depends on uninitialized variables: Use of uninitialized variable

Lost (leaked) memory: Did not free/delete a memory allocation

Double free: Freed/deleted an allocation two or more times

Mismatched delete: new [] / delete ..OR.. new / delete []
Questions to Ask Yourself While Debugging

- What line is the problem on?
  - If your program crashes, first run GDB to find the line number.
  - If gdb, doesn't help find the problem then run valgrind as well (as it may show some additional info that is helpful)
- Might the program behavior be caused by memory issues...always try valgrind
- When does this bug occur?
  - Are there specific circumstances? Edge case? Only breaks if it goes into a specific if statement? Step through the code to see where it executes
Your Task

Start the Lab 5 assignment on Codio (via Blackboard).

- Work through the guided demo following the procedure to find and fix errors.
- At the indicated point, raise your hand to show the TA the specified results to get credit.