McGill UNIVERSITY

C LAB 3: DEBUGGING CODE WITH GDB

ECSE 427/COMP 310: WINTER 2023

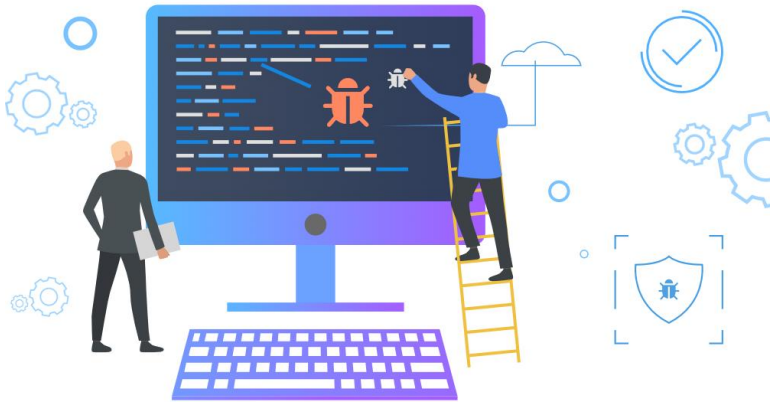
TA: MUSHFIQUE (MOHAMMAD MUSHFIQUR RAHMAN)

CONTENTS

- WHAT IS DEBUGGING?
- SIGNIFICANCE OF DEBUGGING
- RULES FOR DEBUGGING
- TECHNIQUES FOR DEBUGGING
- GDB – THE GNU DEBUGGER
- EXERCISE – LIVE DEMO
- Q&A

WHAT IS DEBUGGING?

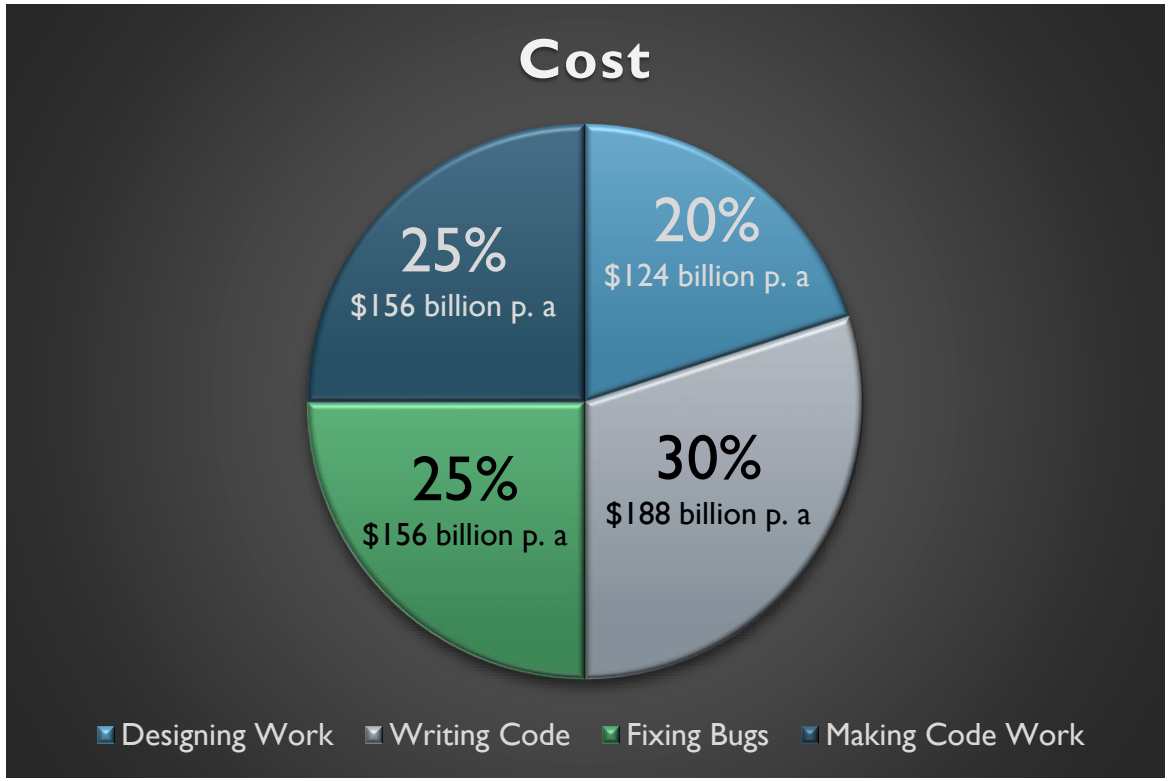
- **Debugging** – The process of finding and fixing errors/bugs in the source code of any computer program.
- *"If debugging is the process of removing software bugs, then programming must be the process of putting them in."*
  - Edsger W. Dijkstra



SIGNIFICANCE OF DEBUGGING

- According to a study from the Judge Business School of the University of Cambridge, UK:
  - Global software developer wages is estimated to be around USD $624 billion per annum (p. a)
  - **Debugging takes 50% of the development time**
  - Hence, **debugging costs** the global software industry around **$312 billion per annum (p. a)**

Source: Evans Data Corporation (2012), Payscale (2012), RTI (2002), CVP surveys (2012)

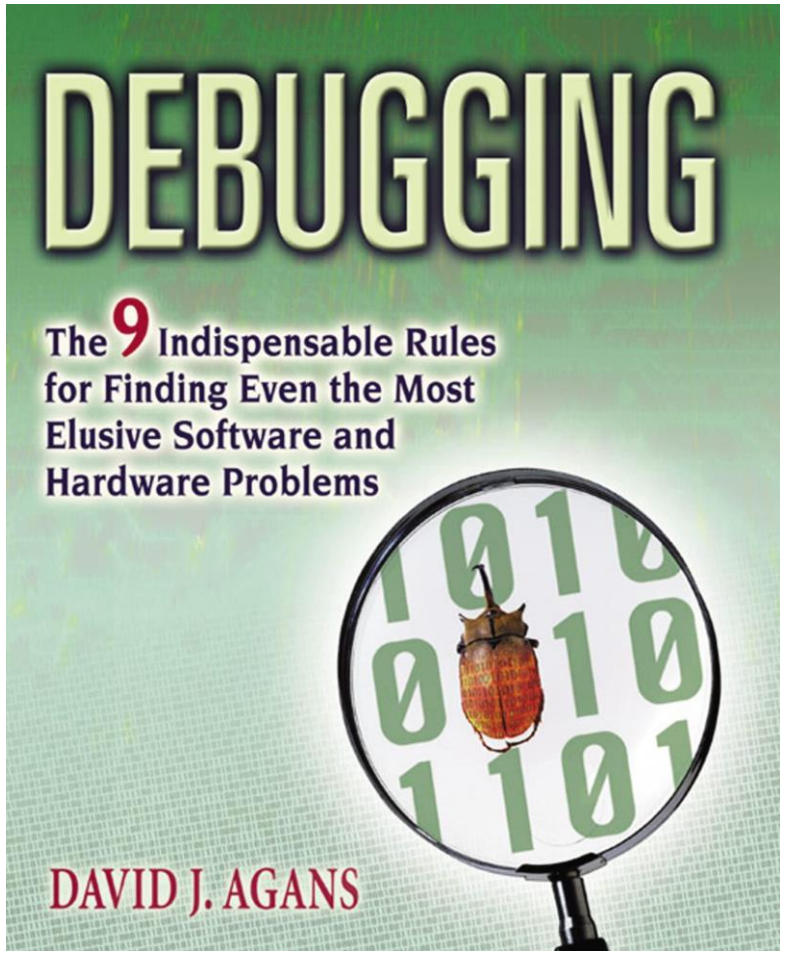


McGill UNIVERSITY

RULES FOR DEBUGGING

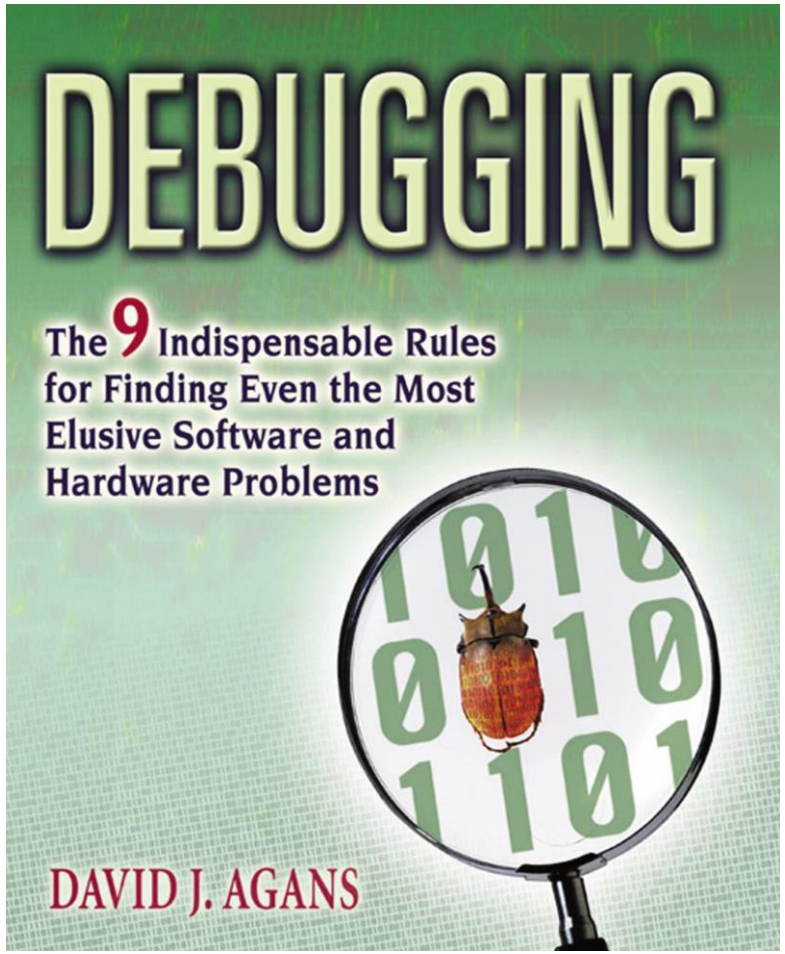
RULES FOR DEBUGGING

1. **Understand the System**
   - The most difficult part
   - Make sure you know what you are working with
   - Ex – You know how Stack & Heap work, otherwise you can't debug pointers.
2. **Make it Fail**
   - You need to be able to reproduce the error, otherwise you can't fix it
   - No way to verify if your fix works
3. **Quit Thinking and Look**
   - Contemplating your code won't fix it
   - Most tedious part, so be patient
   - Need to explore your code



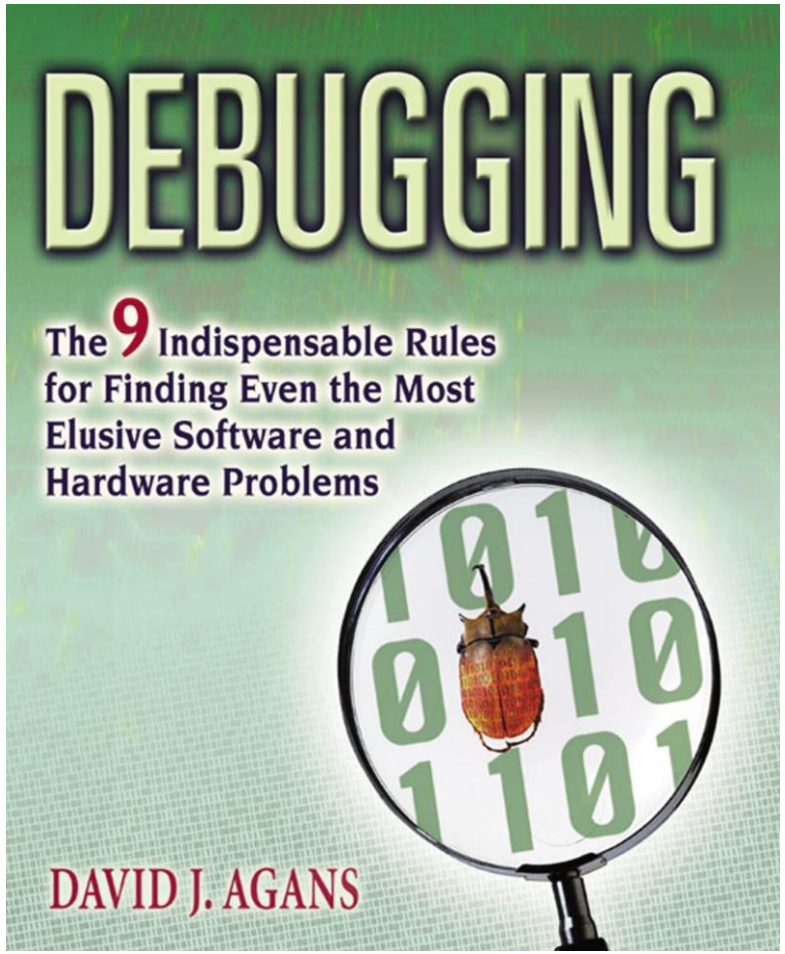
RULES FOR DEBUGGING

4. **Divide & Conquer**
   - Eliminate the parts that are bug free
   - Try to isolate the potential sources of the bug(s)
5. **Change One Thing at a Time**
   - Can't know the exact effect of each change if you change multiple variables/functions at the same time
   - Make a single change and test, then repeat
   - Sometimes it seems obvious, but that assumption can be dangerous!
6. **Keep an Audit Trail**
   - Helps when dealing with huge chunks of code
   - Make a note of what you already tried, and what you want to try
   - Eliminates repeats, especially when you debug in multiple sessions
   - Helps other collaborators as well



RULES FOR DEBUGGING

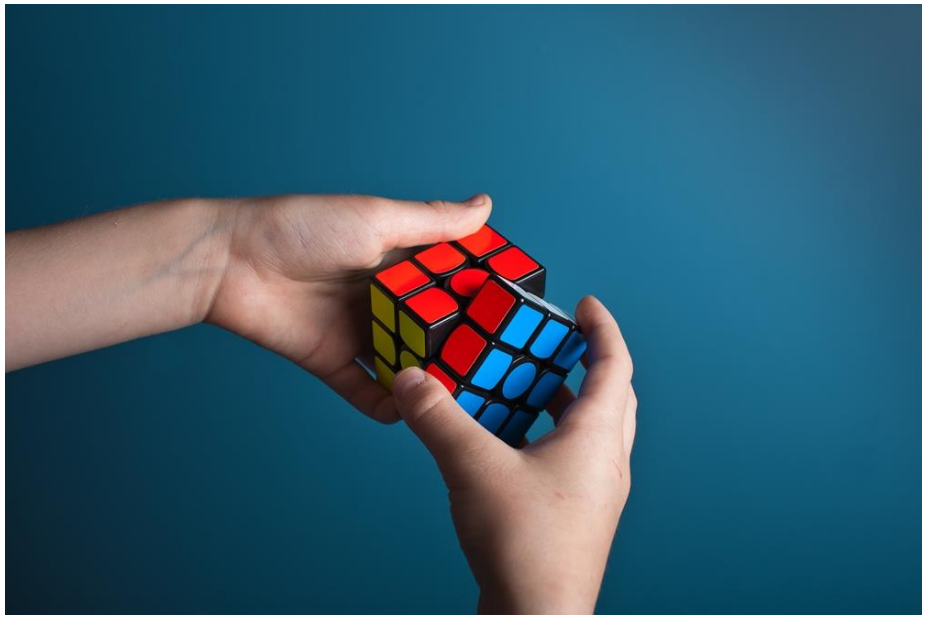
7. **Check the Plug**
   - Often the bug arises from the simplest of reasons – the system is unplugged
   - Don't take anything for granted, always check yourself
   - Ex – a pointer was never initialized, but been used everywhere
8. **Get a Fresh View**
   - Take a break!
   - Starring at the code for too long can make you hallucinate!
   - Try explaining the bug to someone, you might get the answer on your own
9. **If You Didn't Fix It, It Ain't Fixed**
   - Bugs don't go away just because it's Winter!
   - It might come back in the most crucial moment – just before submission



TECHNIQUES FOR DEBUGGING

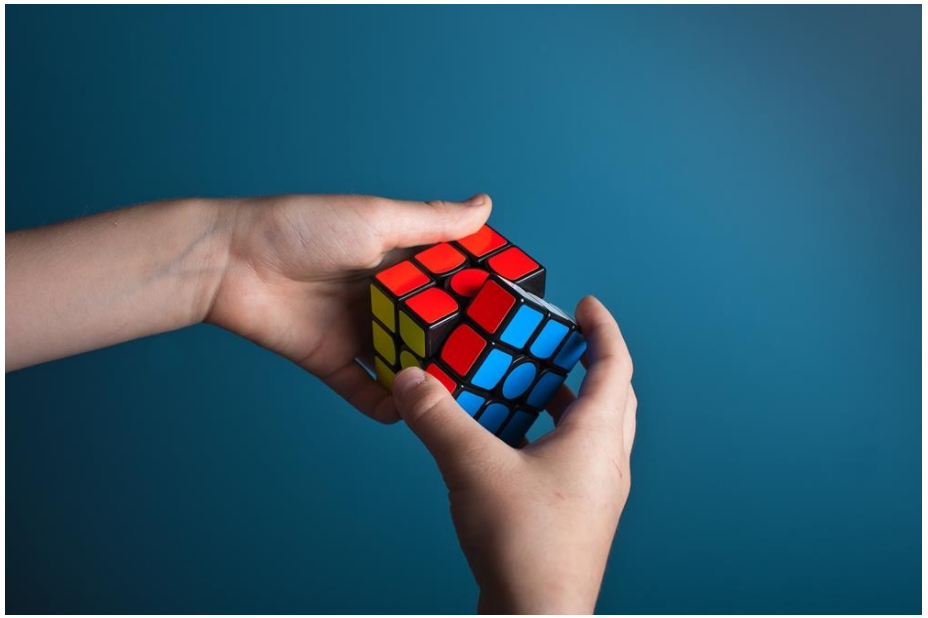
TECHNIQUES FOR DEBUGGING

1. **Print Statements**
   - Simplest way
   - Print the value of the variables in places around the suspected code, and check where the bug is
   - Doesn't need any IDE (Integrated Development Environment)
2. **Commenting things out**
   - As if you are re-coding your program
   - Start with commenting out the suspected code segment
   - Uncomment a small portion and test, then repeat
   - Doesn't need any IDE
3. **Error Handling**
   - Allows the program to run without crashing
   - Catches the error and reports it



TECHNIQUES FOR DEBUGGING

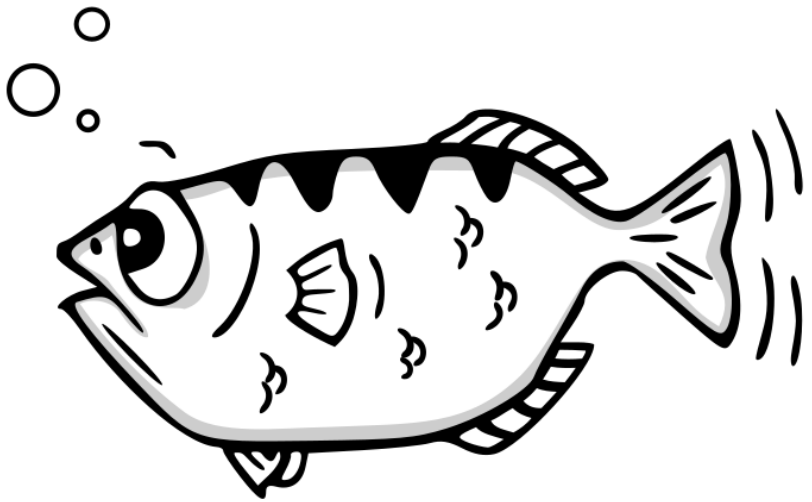
4. **Debugging Tools (Debuggers)**
   - The **most sophisticated and elegant** way of dealing with bugs
   - Usually comes along with the IDEs
   - Allows to set "**breakpoints"** in the code
   - Can **execute the code line-by-line**
   - Check **the real-time values of the variables**
   - Can resolve even the hardest bugs
5. **Tests**
   - Separate code to test the main code
   - Variants: Unit tests, Integration tests, Functional tests, etc.
   - *You don't have to worry about it for this course, since we provide them (usually the task of a Test/QA engineer)*



GDB – THE GNU DEBUGGER

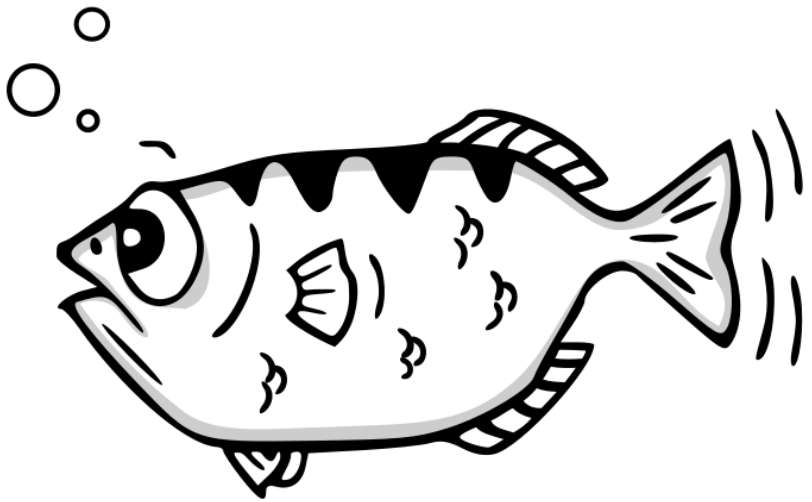
GDB – THE GNU DEBUGGER

- Written by Richard Stallman in 1986 as part of his GNU system
- Still widely used
- Has always been "Free of cost"
- Can debug C, C++, and other languages
- **Types:**
  - **Original/Classic/Command-line version (Our Focus)**
    - Has an interactive shell
    - Recalls history with arrow keys
    - Auto-completion using Tab key
  - **Integrated with IDEs**
    - Ex – GDB in Visual Studio Code
    - Visually more interactive



GDB – THE GNU DEBUGGER

- **Key Features (for both types)**
  - Pause and continue execution of the code
  - Set "breakpoints"
  - Execute code step-by-step
  - Observe values of variables
  - Make real-time changes



GDB – THE GNU DEBUGGER

- **Step 1: Compile with a flag (–g)**
  - Use the "–g" flag while compiling the code
  - Enables the built-in debugging support
  - Preserves the identifiers and symbols for debugging

```
gcc [other flags] -g <source files> -o <output file>
```

GDB – THE GNU DEBUGGER

- **Step 2: Start up GDB**
  - **Option I:** Use "gdb <program>" to turn the debugger on with the specified file
    - The program file is an ***Object File***

```
gdb <program>
```

  - **Option II:** Use "gdb" without any file specified
    - Need to load the program separately using the "file" command

```
gdb
```

```
(gdb) file <program>
```

GDB – THE GNU DEBUGGER

- **Step 3: Display the code**
  - Use "layout next" to open a window that shows the code being executed

```
(gdb) layout next
```

  - Print out the source code – use "list"

```
(gdb) list
```

  - From a specified line – use "list <line-no>"

```
(gdb) list <line-no>
```

GDB – THE GNU DEBUGGER

- **Step 4: Run the program**
  - Use the "run" or "r" command to run the program
  - Add any argument that is required by the program accordingly

```
(gdb) run <argument 1> <argument 2> ...
```

  - **Case I: *There are no bugs* –** the program will run normally
  - **Case II: *There's a bug that causes the program to crash* –** then it will print out some useful information such as line number, relevant parameter values, etc.
  - **Case III (most common): *There's a bug, but the program doesn't crash* –** the program will run normally!
    - Need to stop somewhere, and check the execution line-by-line

GDB – THE GNU DEBUGGER

- **Step 5: Set Breakpoints**
  - Use "break" or "b" to set breakpoints
  - **Option I:** Set a breakpoint in a specific line – use "<filename>:<line>" after "break"

```
(gdb) break <filename>:<line>
```

  - **Option II:** Set a breakpoint in a specific function/method – use "<function-name>" after "break"

```
(gdb) break <function-name>
```

GDB – THE GNU DEBUGGER

- **Conditional Breakpoints –** use "if <condition>" at the end while setting breakpoints

```
(gdb) break <filename>:<line> if <condition>
```

- Show existing breakpoints – use "info"

```
(gdb) info break
```

- Delete breakpoints – use "delete"

```
(gdb) delete <break num>
```

GDB – THE GNU DEBUGGER

- **Step 6: Navigate through the code**
  - Go to next line – use "next" or "n"

```
(gdb) next
```

  - Go to next breakpoint – use "continue" or "c"

```
(gdb) continue
```

  - Go inside the function/method – use "step" or "s"

```
(gdb) step
```

  - Return from current function – use "finish"

```
(gdb) finish
```

GDB – THE GNU DEBUGGER

- **Step 7: Observe/Change the variables**
  - Print current value (can dereference pointers) – use "print <variable-name>"

```
(gdb) print <variable-name>
```

  - Print value at every step – use "display <variable-name>"

```
(gdb) display <variable-name>
```

```
(gdb) info display
```

```
(gdb) undisplay <number>
```

GDB – THE GNU DEBUGGER

- **Set Watchpoints:** Break on variable change – use "watch <variable-name>"

```
(gdb) watch <variable-name>
```

```
(gdb) info watch
```

```
(gdb) unwatch <number>
```

- Print variable type – use "whatis <variable-name>"

```
(gdb) whatis <variable-name>
```

- Change variable values – use "set var <variable-name>=<value>"

```
(gdb) set var <variable>=<value>
```

GDB – THE GNU DEBUGGER

- Other useful commands:
  - **help** – provides a description of the command
  - **backtrace** – produces a stack trace of the function calls that lead to a seg fault (should remind you of Java exceptions)
  - **where** – same as backtrace; you can think of this version as working even when you're still in the middle of the program
- GDB Cheat Sheet: https://darkdust.net/files/GDB%20Cheat%20Sheet.pdf

EXERCISE – LIVE DEMO

CONCLUSION

- Codes shown in live demo are available on myCourses
- References:
  - https://undo.io/resources/reverse-debugging-whitepaper/
  - https://www.jbs.cam.ac.uk/insight/2013/financial-content-cambridge-university-study-states-software-bugs-cost-economy-312-billion-per-year/
  - https://www.amazon.ca/Debugging-Indispensable-Software-Hardware-Problems/dp/0814474578
  - https://www.tygertec.com/9-rules-debugging/
  - https://www.codecademy.com/resources/blog/how-to-debug-your-code/
  - https://www.sourceware.org/gdb/
  - https://github.com/cbourke/ComputerScienceI/tree/master/hacks/hack-debugging
  - https://www.geeksforgeeks.org/debugging-tips-to-get-better-at-it/
  - https://credentials.deakin.edu.au/problem-solving-techniques-steps-and-methods/
- Additional Resources:
  - Remote SSH using VS Code: https://code.visualstudio.com/blogs/2019/07/25/remote-ssh

THANK YOU!

Q&A