

CS330

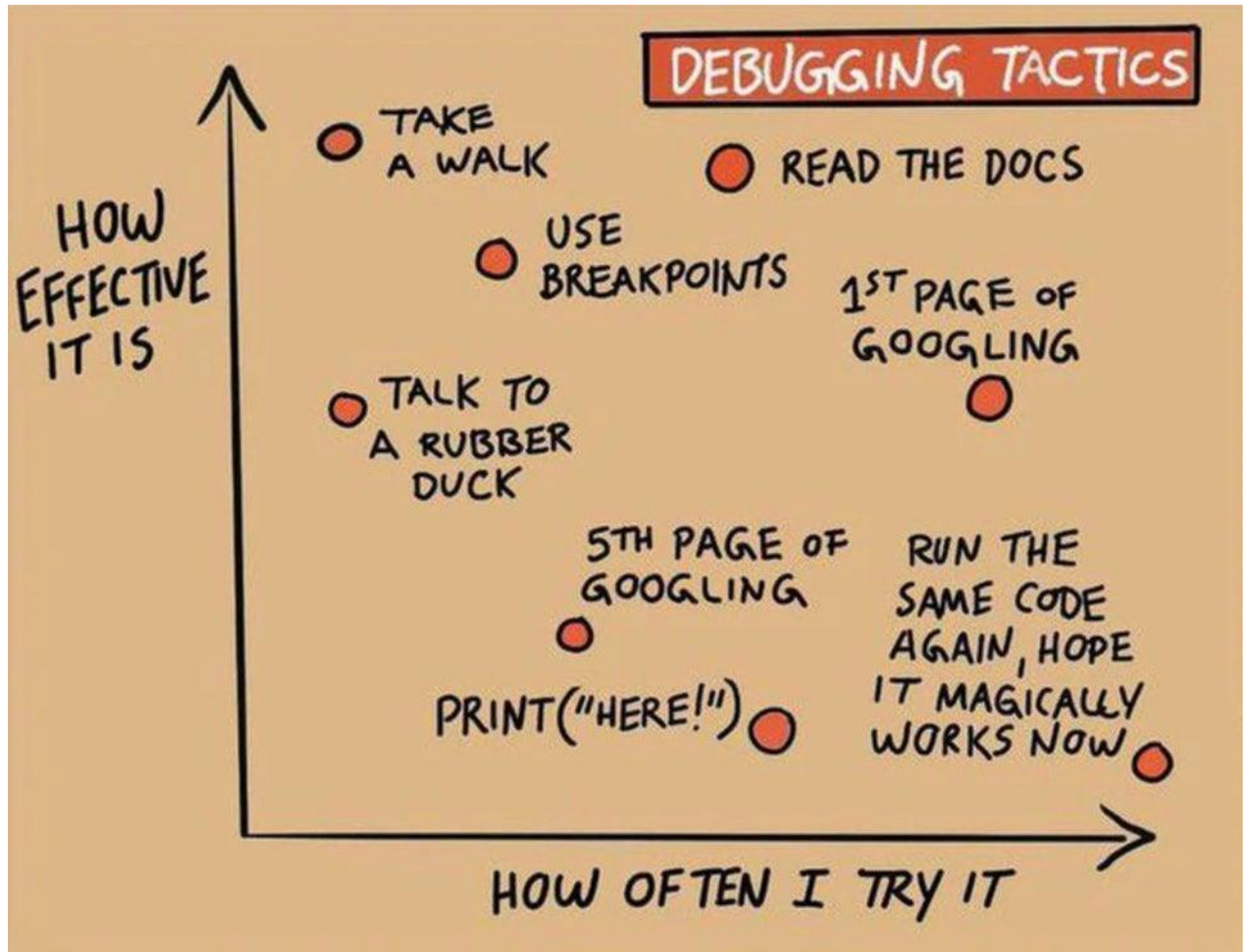
Debugging

Spring 2022

Lab 7

Today's Agenda

- Debugging



From: https://www.reddit.com/r/ProgrammerHumor/comments/t1sgzt/debugging_tactic/?utm_source=share&utm_medium=ios_app&utm_name=iossmf

Debugging Tips

- Four types of errors (John's version):

Compilers catch these

Use debugger for these

Type	Description	Example	Note
Syntax	Code is not a correct	Missing semicolon, misspelling, incorrect if statement syntax	Compiler reports Tend to make less of these with experience
Semantic	Words are spelled correctly, but sentence doesn't make sense	Assigning a String to an int variable	Compiler reports
Run-Time	Code compiles, but there is an error during code execution	/0 Segmentation fault	Code will stop executing, error message
Logic	Code compiles and runs, but the result is not correct	BankAccount prints the wrong balance	No compiler assistance, Hardest to find

- Print statements won't work well in Assembly
- If you're not sure where to start, start at the beginning, or better: try a bi-section search:
 - Insert a break point somewhere in the middle of your code
 - If variables seem ok, error is likely after the break point
 - If variables not correct, error is likely prior to break point
 - Adjust your search accordingly

Debugger Workflow (same as CS203, just in GDB)

1

Set Breakpoint(s)

Double-Click; or Right-Click (or select 'Run') , Toggle Breakpoint

2

Start Debugger

Click 'Bug' icon

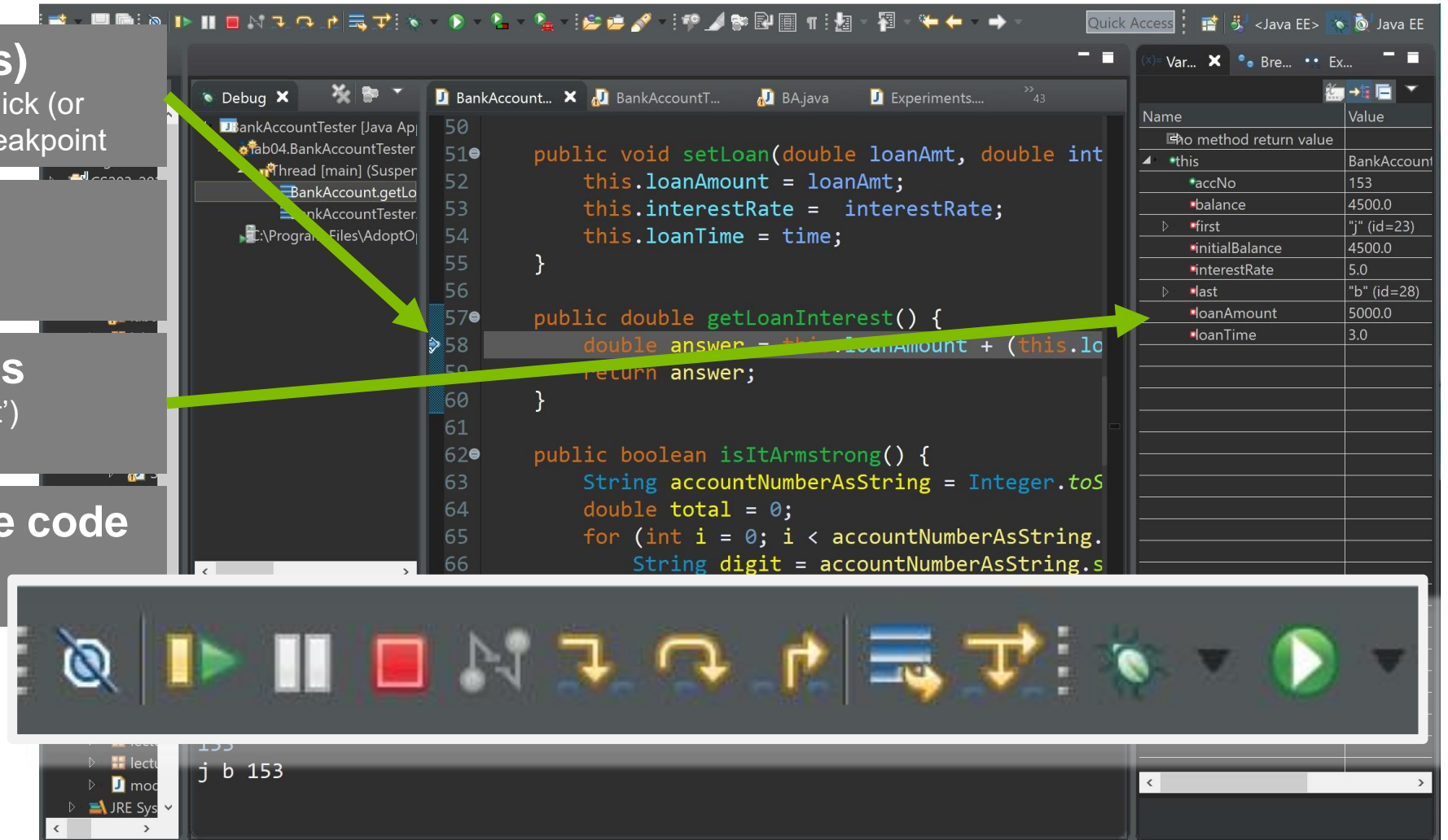
3

Inspect Variables

(also Right-Click 'Inspect')

4

Step through the code



For more advanced topics: <https://www.baeldung.com/eclipse-debugging>

Using gdb to better understand your program

`gdb` is an extremely powerful tool for debugging programs, helping you identify any runtime or logical errors (e.g. segmentation faults, core dumps, array out-of-bounds exceptions).

To debug your programs using `gdb`, you will need to compile with the `-g` option. This will instruct the compiler to generate the executable with source level debug info.

```
gcc -g name.c -o name
```

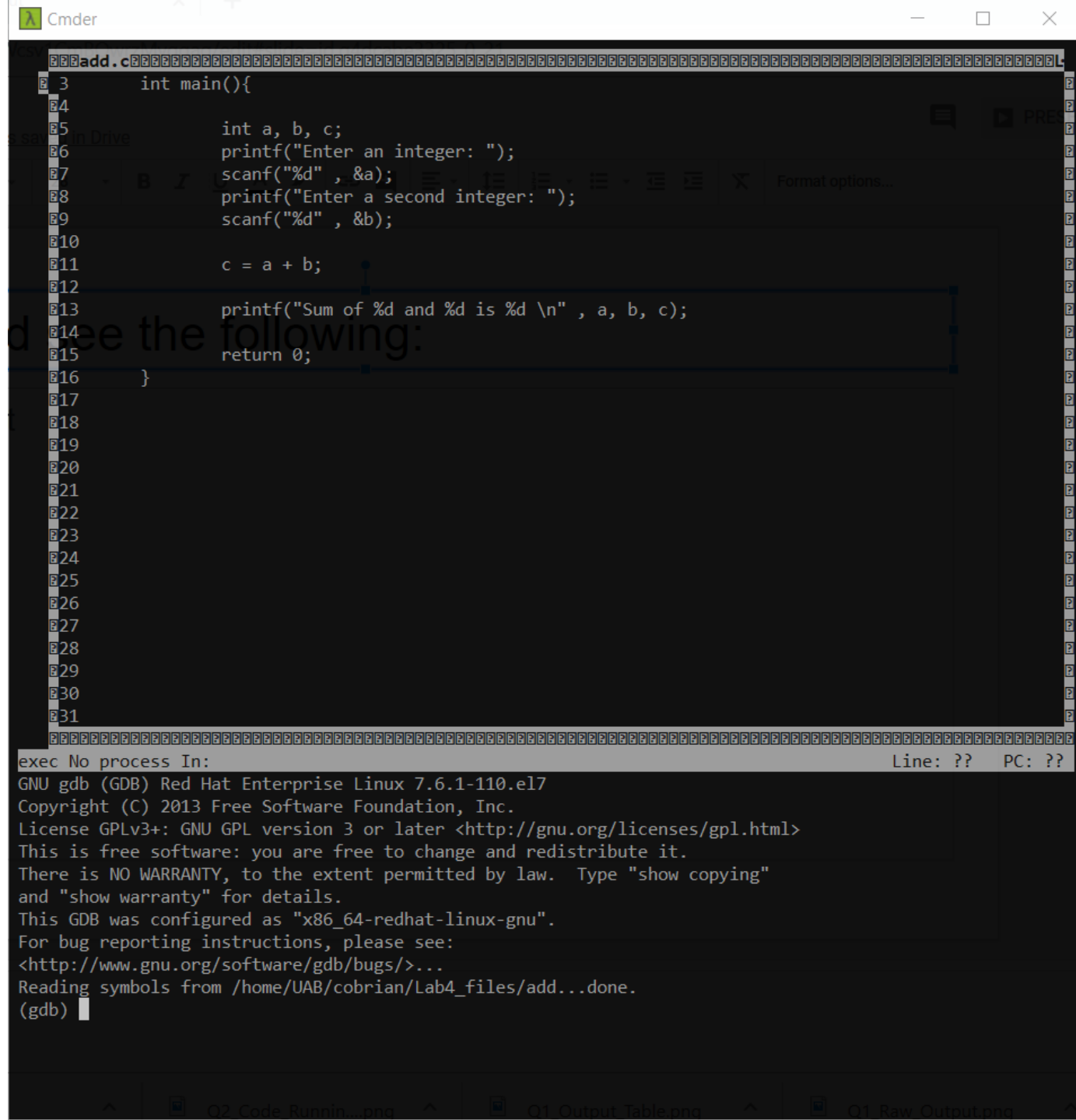
Now you can enter the `gdb` debugger with one of the following commands:

```
gdb name            gdb -tui name
```

The **TUI** version is recommended for new users and shows the source code and `gdb` command line in one window

You should see something similar to the picture, with **-tui**

The program here shows you how to take two numbers from user input and compute their sum!



The image shows a C code editor window with a dark theme. The code is a simple program to add two integers. Below the code editor is a GDB terminal window showing the output of the program.

```
add.c
1  int main(){
2
3      int a, b, c;
4      printf("Enter an integer: ");
5      scanf("%d", &a);
6      printf("Enter a second integer: ");
7      scanf("%d", &b);
8
9      c = a + b;
10
11     printf("Sum of %d and %d is %d \n", a, b, c);
12     return 0;
13 }
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
```

exec No process In: Line: ?? PC: ??

GNU gdb (GDB) Red Hat Enterprise Linux 7.6.1-110.el7
Copyright (C) 2013 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-redhat-linux-gnu".
For bug reporting instructions, please see:
<<http://www.gnu.org/software/gdb/bugs/>>...
Reading symbols from /home/UAB/cobrian/Lab4_files/add...done.
(gdb)

GDB Basics

- **To set a breakpoint:**
 - (b)reak <line number>
 - **or to break at the start of a function:** (b)reak <name of function>
 - **To remove a breakpoint:** clear <line number>
- run [any command line arguments]
- **Inspect variables**
 - **To display once:** (p)rint <variable name>
 - **To display at each step:** display <variable name>
 - **To print all local variables:** info locals
- **Step through the code**
 - (n)ext (don't dive into functions, step-over)
 - (s)tep (dive into functions, step-into)
 - (c)ontinue (continue running until the end or next breakpoint)
- RETURN (or ENTER) will repeat the last command entered, TAB will attempt to autocomplete
- (q)uit
- **Might be helpful to compile C without Optimizations**
 - gcc -g hello.c -o hello -O0

Resources:

Beej's Quick Guide: <https://beej.us/guide/bggdb/>
<https://www.gnu.org/software/gdb/documentation/>
Manual: <http://sourceware.org/gdb/current/onlinedocs/gdb.pdf>

Other useful GDB info

- **Adjusting –tui (text user interface)**
 - Just resize window to ‘repaint’
 - info win: lists the current window names. Source code (src), Command (cmd), Registers (reg)
 - layout next, layout prev
 - layout reg: to show registers
 - tui reg general: also displays registers
 - Can cycle through different register groups via: tui reg <next | prev | general | all>
 - focus <name>: changes which TUI window is active for scrolling
 - winheight <name> + 5: increases the size of the named window by 5 lines
 - winheight <name> 25: set the size of the height of the named window
 - refresh: refresh the screen, similar to ‘clear’ in the shell/terminal (also CNTRL-L)
- **Breakpoints**
 - start: same as ‘break main’ followed by ‘run’

<https://sourceware.org/gdb/onlinedocs/gdb/TUI.html#TUI>
<https://sourceware.org/gdb/onlinedocs/gdb/TUI-Commands.html>

Other GDB useful info - variables

- **Print stack trace (one frame per line):** (bt) backtrace [options] [count]
 - full prints local variables
- x/nfu addr
 - *n is the repeat count (# of bytes to print)*
 - *f is display format (like print) 'x', 'd', 'u', 'o', 't', 'a', 'c', 'f', 's'), and in addition 'i' (for machine instructions)*
 - *u is unit size: b bytes, h halfwords (2 bytes), w words (four bytes), g giant words (8 bytes)*
 - *addr is starting address in hex, or \$register, or variable name*