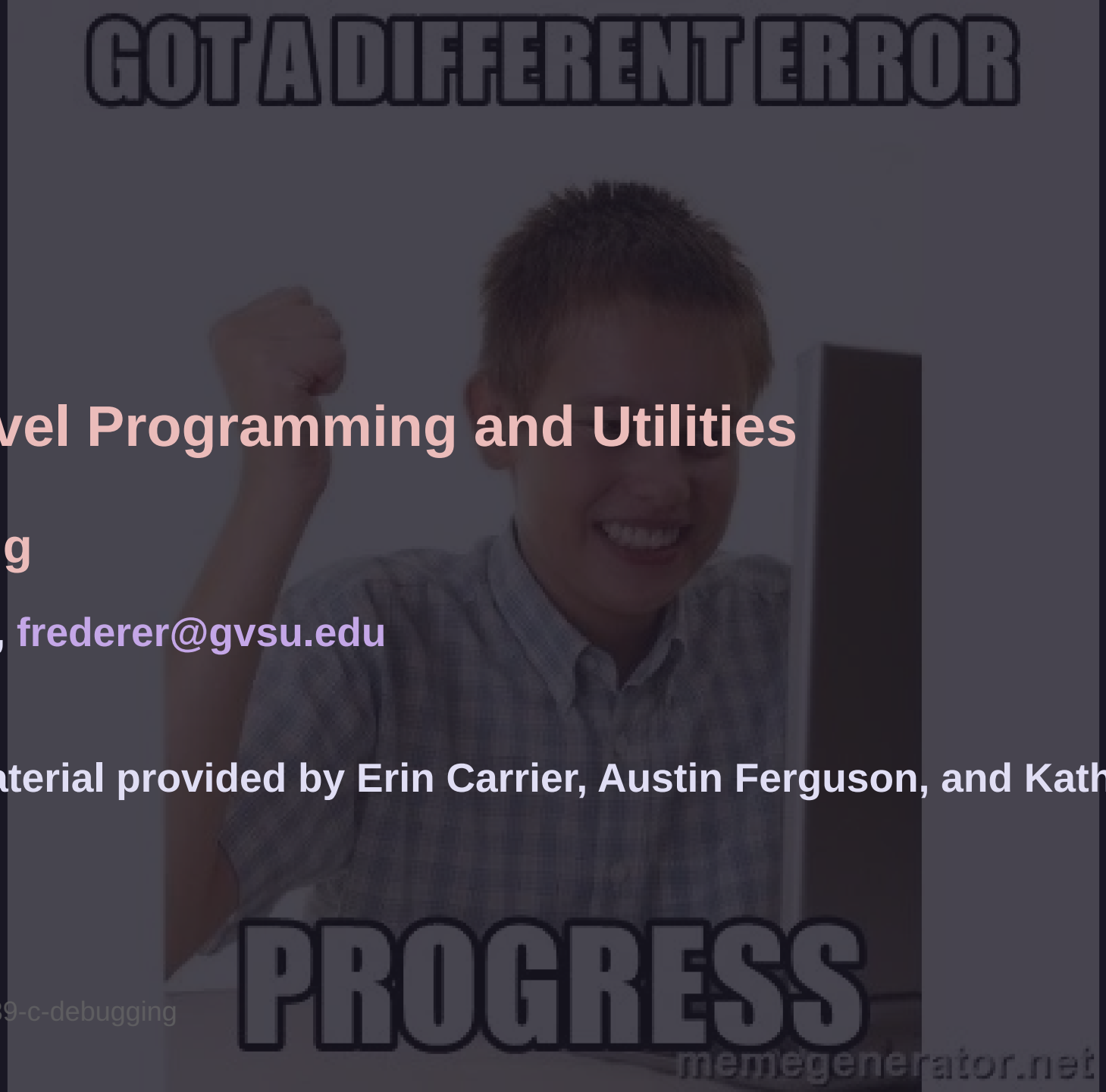

CIS241

System-Level Programming and Utilities

C - Debugging

Erik Fredericks, frederer@gvsu.edu

Fall 2025

> **Based on material provided by Erin Carrier, Austin Ferguson, and Katherine Bowers**

Debugging

What is debugging?

- Trying to fix a broken code

What is a debugger?

- A program that lets you see what’s going on inside a program as it runs

Why use a debugger?

- Helps fix segfaults and logic errors
- Provides more insights than print statements

Debugging

We'll be using `gdb` (GNU Project Debugger)

Steps to run `gdb`:

1. Compile your code in debug mode
   - `gcc -g my_code.c`
2. Launch gdb with your executable
   - `gdb ./a.out`

Mac users...

You will likely not have gdb

- You can use `lldb` instead.

Use will be very similar, but some commands will have different names.

Reference: https://lldb.llvm.org/use/map.html

Using `gdb`

```
erik@worktop:~/CIS241/c$ gdb ./a.out
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04.2) 12.1
Copyright (C) 2022 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-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://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 ./a.out...
(gdb)
```

Using `gdb`

You can run commands within `gdb`, basics include:

- `run` - Start executing your program from the beginning
- `run arg1 arg2` for command line args
- `run < input_file > output_file` (redirect)
- `quit` - exit gdb
- `kill` - stop program execution

Debugging a `segfault`

`♪~ ᕕ(ᐛ)ᕗ`

1. Start `gdb` to run program
2. `run`
3. Examine the callstack with `backtrace` or `bt`
4. Load a frame (`f`) using its number (e.g., `f 1`)

```
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./a.out...
(gdb) run
Starting program: /home/erik/CIS241/c/a.out
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Program received signal SIGSEGV, Segmentation fault.
0x000055555555513d in main () at segfault.c:6
6           *ptr = 42; // Dereferencing a null pointer
(gdb)
```

Debugging a `segfault`

5. **You can use `list` to get more context**
6. **Check the state of your variables**
   - **Print a variable with `print` (or `p`) (e.g., `p size`)**
   - **Print n items of an array with `*array@n`**
7. Try to deduce the problem! You can always examine other frames

Breakpoints

You can also use breakpoints to pause your code

To set a breakpoint:

- `break function_name`
- `break line_number`
- `break line_number if condition`

To list all breakpoints: `info break`

Breakpoints

`continue` will resume until next breakpoint

`step` to execute one line of code

- `step n` to execute n lines of code

`enable/disable breakpoint_number`

`delete breakpoint_number` (`delete` for all)

Other tools

Valgrind: "a suite of tools for debugging and profiling programs"

- Valgrind is very powerful, but very complex.

Recommendation: `memcheck`

- `valgrind --tool=memcheck ./a.out`

This will check for memory leaks and other hard-to-spot memory issues!