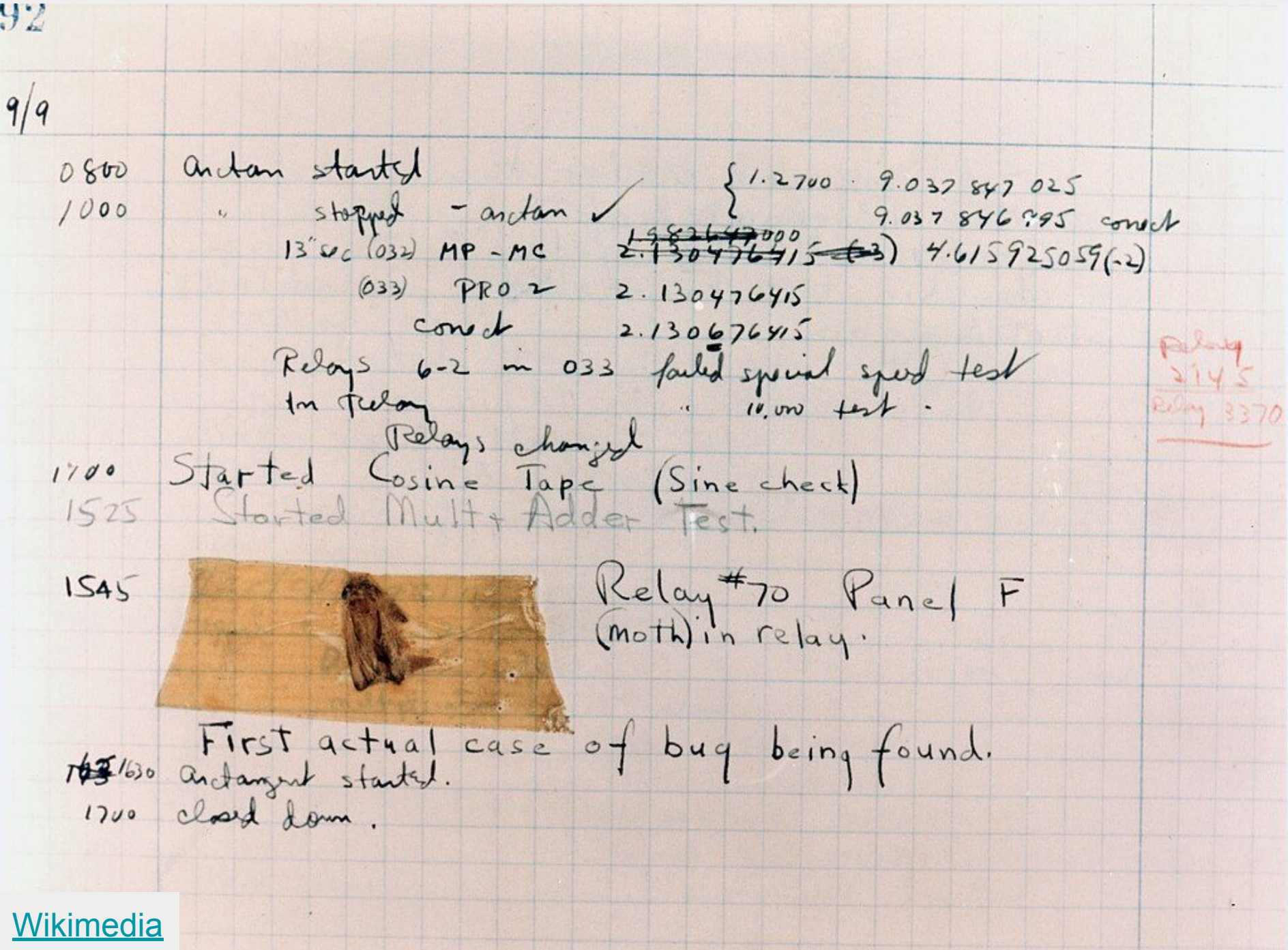
Debugging in C

Adapted from materials by Dr. Carrier



Wikimedia

Debugging

Debugging

What is debugging?

Debugging

What is debugging?

- Trying to fix a broken code

Debugging

What is debugging?

- Trying to fix a broken code

What is a debugger?

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*

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?

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 in C under Linux

We'll be using gdb (GNU Project Debugger)

Debugging in C under Linux

We'll be using gdb (GNU Project Debugger)

Steps to run gdb:

Debugging in C under Linux

We'll be using gdb (GNU Project Debugger)

Steps to run gdb:

1. Compile your code in debug mode

```
gcc -g my_code.c
```

Debugging in C under Linux

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
   ```

Note for 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

Inside gdb

```
class/c/structs$ gdb ./a.out
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 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)
```

Inside gdb

```
class/c/structs$ gdb ./a.out
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 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)
```

Inside gdb

You can run commands within gdb, basics include:

Inside gdb

You can run commands within gdb, basics include:

- run - Start executing your program from the beginning

Inside 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)

Inside 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

Inside 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

1. Start gdb with your program

Debugging a segfault

1. Start gdb with your program
2. `run`

```
Program received signal SIGSEGV, Segmentation fault.
__GI___libc_realloc (oldmem=0x64, bytes=18446744073709519336) at ./malloc/malloc.c:3426
3426    ./malloc/malloc.c: No such file or directory.
(gdb)
```

Debugging a segfault

1. Start gdb with your program
2. `run`

```
Program received signal SIGSEGV, Segmentation fault.
__GI___libc_realloc (oldmem=0x64, bytes=18446744073709519336) at ./malloc/malloc.c:3426
3426    ./malloc/malloc.c: No such file or directory.
(gdb)
```

3. Examine the callstack with `backtrace` or `bt`

```
(gdb) bt
#0  __GI___libc_realloc (oldmem=0x64, bytes=18446744073709519336) at ./malloc/malloc.c:3426
#1  0x0000555555555230 in Append (vec_ptr=0x7fffffffdc10, x=7) at vector_broken.c:21
#2  0x0000555555555310 in main () at vector_broken.c:39
```

Debugging a segfault

1. Start gdb with your program
2. `run`

```
Program received signal SIGSEGV, Segmentation fault.
__GI___libc_realloc (oldmem=0x64, bytes=18446744073709519336) at ./malloc/malloc.c:3426
3426    ./malloc/malloc.c: No such file or directory.
(gdb)
```

3. Examine the callstack with `backtrace` or `bt`

```
(gdb) bt
#0  __GI___libc_realloc (oldmem=0x64, bytes=18446744073709519336) at ./malloc/malloc.c:3426
#1  0x0000555555555230 in Append (vec_ptr=0x7fffffffdc10, x=7) at vector_broken.c:21
#2  0x0000555555555310 in main () at vector_broken.c:39
```

4. Load a frame (`f`) using its number (e.g., `f 1`)

Debugging a segfault (page 2)

Debugging a segfault (page 2)

5. You can use list to get more context

Debugging a segfault (page 2)

5. You can use list to get more context
6. Check the state of your variables

Debugging a segfault (page 2)

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`

Debugging a segfault (page 2)

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

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
```

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

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`

`continue` will resume until next breakpoint

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`

`continue` will resume until next breakpoint

`step` to execute one line of code

  `step n` to execute n lines of code

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`

`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”

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
```

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!

```
==7995== Conditional jump or move depends on uninitialised value(s)
==7995==    at 0x1091F4: Append (vector_broken.c:16)
==7995==    by 0x109349: main (vector_broken.c:43)
```