# Quick notes

- Check Piazza
- Reminders:
  - Assign4 is due friday
  - Python and diff/patch
- Assignment 10 (only applies to students enrolled in Lab 1)
  - Email me your topic
  - Create presentation (10 min?) + brief summary (<200 words?)
    - Must present to class (email me if you're in a different timezone)
  -
- Questions?

# Feedback / Office Hours / Other

- Tameez Latib
  - tameezlatib@gmail.com, please add “CS35L” to the subject line
  - Office Hours: Monday **4pm-6pm** (or by appointment)
  - Feedback: https://forms.gle/6kcJ2aJtzAzFMhHQ7 (anonymous google form)
- If you guys are stressed out:
  - CAPS (https://www.counseling.ucla.edu/)
    - Free with UC ship

# C : debugging!

- Print statements aren’t ‘formal’ debugging
- Why?
  - Step through code
  - Traceback
  - Memory management
- Valgrind and GDB

# Valgrind

- Use -g flag (gcc -g file.c)
- Valgrind --leak-check=full ./file.exe
- Demo output
- https://www.valgrind.org/docs/manual/quick-start.html

```
==18912== Using Valgrind-3.8.1 and LibVEX; rerun with -h for copyright info
==18912== Command: ./a.out
==18912==
==18912==
==18912== HEAP SUMMARY:
==18912==     in use at exit: 20 bytes in 1 blocks
==18912==   total heap usage: 1 allocs, 0 frees, 20 bytes allocated
==18912==
==18912== 20 bytes in 1 blocks are definitely lost in loss record 1 of 1
==18912==    at 0x4A06A2E: malloc (vg_replace_malloc.c:270)
==18912==    by 0x400539: fcn (mem_leak.c:5)
==18912==    by 0x400557: main (mem_leak.c:10)
==18912==
==18912== LEAK SUMMARY:
==18912==    definitely lost: 20 bytes in 1 blocks
==18912==    indirectly lost: 0 bytes in 0 blocks
==18912==      possibly lost: 0 bytes in 0 blocks
==18912==    still reachable: 0 bytes in 0 blocks
==18912==         suppressed: 0 bytes in 0 blocks
==18912==
==18912== For counts of detected and suppressed errors, rerun with: -v
==18912== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 8 from 6)
```

```c
#include <stdio.h>
#include <stdlib.h>

void fcn(){
        int *ptr = (int*)malloc(5*sizeof(int));
        ptr[0] =512;
}

int main(){
        fcn();
        return 0;
}
```

# Valgrind

- Basically used for memory management stuff
- Shows us variable not initialised

```
==18974== Memcheck, a memory error detector
==18974== Copyright (C) 2002-2012, and GNU GPL'd, by Julian Seward et al.
==18974== Using Valgrind-3.8.1 and LibVEX; rerun with -h for copyright info
==18974== Command: ./a.out
==18974==
==18974== Conditional jump or move depends on uninitialised value(s)
==18974==    at 0x400534: main (uninit_val.c:4)
==18974==
==18974==
==18974== HEAP SUMMARY:
==18974==     in use at exit: 0 bytes in 0 blocks
==18974==   total heap usage: 0 allocs, 0 frees, 0 bytes allocated
==18974==
==18974== All heap blocks were freed -- no leaks are possible
==18974==
==18974== For counts of detected and suppressed errors, rerun with: -v
==18974== Use --track-origins=yes to see where uninitialised values come from
==18974== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 8 from 6)
```

```c
#include <stdio.h>
int main(){
        int x;
        if (x > 0){
                printf("x is positive")
        }
}
```

# Valgrind

- Shows us invalid read/write

```
==19089== Invalid write of size 4
==19089==    at 0x4005A0: main (invalid_read.c:7)
==19089==  Address 0x4c37050 is 4 bytes after a block of size 12 alloc'd
==19089==    at 0x4A06A2E: malloc (vg_replace_malloc.c:270)
==19089==    by 0x400589: main (invalid_read.c:5)
==19089==
==19089== Invalid read of size 4
==19089==    at 0x4005AE: main (invalid_read.c:8)
==19089==  Address 0x4c37050 is 4 bytes after a block of size 12 alloc'd
==19089==    at 0x4A06A2E: malloc (vg_replace_malloc.c:270)
==19089==    by 0x400589: main (invalid_read.c:5)
```

```c
#include <stdio.h>
#include <stdlib.h>
int main(){
        int * x;
        x = malloc(sizeof(int)*3);
        x[0] = 1;
        x[4] = 10;
        printf("x[4]: %d", x[4]);
}
```

# Valgrind

- Other examples:
  - Invalid free
    - (e.g. free twice)
  - Losing a pointer, but memory still allocated
    - (e.g. Malloc, and then declare ptr = Null)

# GDB

- Step through code line by line
- Can check variable values
- Issue commands
- Breakpoints, traceback, etc

# GDB

- Gcc -g file.c
- Gdb a.out
  - Some commands you can use:
    - Run [args] or run < file (runs the program with arguments/inputs)
    - Break [file] line_number (set a breakpoint)
      - Info b (lists breakpoints)
      - Delete / disable /enable breakpoint_num
    - Step [n]  (go to next [n] lines)
    - Next [n] (go to next [n] lines, do not go into functions)
    - Continue (until next breakpoint)
    - Print var
    - Watch var (pause when var changed) Rwatch var (pause when var read)
    - List (nearby lines)
  - Try: info local/args/frame

# Questions??