Lecture 23 (3-18-26)

Debugging (C)

Motivating Questions:

1. How do we fix **memory errors** with **valgrind**?
2. How do we dubug **errors** using **gdb**?
3. What exactly is a :
    1. pointer?
    2. array?
    3. string?

Arrays

```
int a[]; //declaring
int b[] = {7,1,4} //initializing
```

- Name of an **array** is a **label** to the **first element** (decays into a **pointer**)
- `a[0] == *a`
- `a[0] == *(a+0)`
- `a[1] == *(a+1)`
- VALID SYNTAX, BUT NEVER USE:
    - The following are equivalent:
        - `a[1]`
        - `1[a]` $\rightarrow$ never use this!
- When you pass an array into a function:
    - you actually pass the pointer(address) to the array
    - this is why you cannot find the size of an array within a function
- So you should pass the size of the array to the function OR have your array have a sentinel at the end.
- looping through an array with pointers:
    -
```
for (int *p = a; *p > 0; p++){
  printf("a[%lu] = %d\n", p - a, *p)
```

```
}
```

Strings

- Strings in C are **arrays of characters** terminated by **NUL**
  - "NUL" refers to `\0`
  - "NULL" actually refers to `NULL`

`char *s = "dog";`

- in this example you are allocating two things
- You allocate:
  - the string `dog`
  - the pointer `s`
  - `s` does not correspond to the address of `d`
    - rather `s` is allocated and has the VALUE OF THE ADDRESS of `d`
- Remember the Memory Regions (covered extensively in data structures):
  - *Top*
    - **stack** (local, parameters)
    - **heap** (user managed memory)
    - **data** (global, static, string literals)
    - **code** (instructions)
  - *Bottom*

GDB

- `bt` → gives you a backtrace
  - will tell you the exact line where your error occured (ex: segfault)
- `f 2` → takes you to frame 2
- `p buffer` → prints the value of buffer
- `p main` → prints the address of main
  - if the outputs are near each other, you can infer that `buffer` is in data segment
    - things in the data segment are READ ONLY
- `b 12` → sets breakpoint at line 12

Programming Challenge: Palindromic Permutations

```
#include <stdbool.h>
#include <
#include <
#include <

bool is_palindrome(const char *s){
     const char  *head = s;
     const char  *tail = s + strlen(s);

     while (*head == *tail){
          head++;
          tail--;
     }

     return head >= tail;
}

bool is_palindrome_permutation(const char *s){
     return false;
}

bool is_palindrome_permutation_bitset(const char *s){
     return false;
}

int main(int argc, char *argv[]){
     char buffer[BUFSIZ] = "";

     while (fgets(bufffer, BUFSIZ, stdin)){
          if (is_palindrome(buffer)){
               puts("YEAH")
          }
     }

     return 0;
}
```