Mid-semester Review

Three stages of compilation

1. Preprocessor
    - Processes all # directives (includes, etc)
2. Compilation
    - Takes .c files and compiles them into object files (.o)
3. Linker
    - Takes object files (.o) and links them to produce a final executable

How to read error messages

1. Use of undeclared identifier
    - Means you used an identifier (variable name) without declaring it

- For instance, if we use the variable `inches_per_foot` before declaring it:

```
1  my_program.c: In function main:
2  my_program.c:30: error: inches_per_foot undeclared (first use in this
       function)
```

- This tells us in my_program.c, inside of the function main, on line 30 (indicated by `:30`), we used a variable we didn't declare

2. Use of a unknown/undefined function
    - Means we didn't link to a function we used correctly. This may mean a particular library we are using may be incorrectly installed, or we didn't compile all of our code.
    - No matter the cause, we need to figure out why the **linker** can't find the function's compiled definition

- For instance:

```
1  Undefined symbols for architecture x86_64:
2    "_sqrt", referenced from:
3        _main in main.o
4  ld: symbol(s) not found for architecture x86_64
5  clang: error: linker command failed with exit code 1 (use -v to see
       invocation)
```

- This means the `_sqrt` symbol could not be found. What is a symbol? Symbols are a part of object files.
- Anytime you see `ld` it means the linker failed.

3. Implicit declarations
   - Typically means you forgot in include a library
   - Also could mean you are using a function before the compiler is aware of the function. I.e. you forgot to create a function prototype, defined the function after main, but used the function in main (meaning main doesn't know what function you are talking about)

- For instance:

```
1  warning: implicit declaration of function 'printf'
```

- Means we forgot to include `stdio.h` (`#include <stdio.h>`)
- Suppose we define the function my_func after main but use it in main without putting a prototype before main:

```
1  warning: implicit declaration of function 'my_func'
```

- Fix this by adding a prototype of the function before main or moving the definition to before main

5. Missing semicolon
   - Means we forgot to put a semicolon to end a statement

```
1  Expected ';' after expression
```

- Fix this by adding a semicolon in the correct place
- **Any of these error/warning messages may be accompanied with other errors/warnings**

Exercises

1. Write a C program to print the contents of an array of C-strings (note this requires printing a multidimensional array

```c
1  #include <stdio.h>
2  #include <string.h>
3
4  //1. Write a C program to print the contents of an array of C-strings (
       note this requires printing a multidimensional array)
5
6  void print_strings(char ** word_arr, size_t n_words){
7    for(int i = 0; i < n_words; i++){
8      printf("%s ", word_arr[i]);
9    }
```

```
10    printf("\n");
11  }
12
13  int main(){
14    char *a[] = {
15      "cs",
16      "50",
17      "is",
18      "awesome"
19    };
20    print_strings(a, 4);
21  }
```

2. Write a C program to count the number of occurrences of a user-specified value in a 2-dimensional integer array

```
1  #include <stdio.h>
2  #include <string.h>
3
4  // Write a C program to count the number of occurrences of a user-
       specified value in a 2-dimensional integer array
5
6  int count_occurances(int int_arr[][5], size_t num_cols, size_t num_rows
       , int target_value){
7    int count = 0;
8
9    // loop over rows
10   for(int i = 0; i < num_rows; i++){
11     // loop over columns
12     for(int j = 0; j < num_cols; j++){
13       // compare value at this position to target value
14       if(target_value == int_arr[i][j]){
15         count++;
16       }
17     }
18   }
19
20   return count;
21  }
22
23  int main(){
24    int a[5][5] = {
25      {1, 2, 3, 4, 5},
```

```
26        {2, 2, 3, 4, 54},
27        {6, 2, 7, 4, 5},
28        {1, 2, 3, 4, 36},
29        {10, 99, 3, 4, 5},
30     };
31     int count = count_occurances(a, 5, 5, 1);
32     printf("1's count: %d\n", count);
33     count = count_occurances(a, 5, 5, 7);
34     printf("7's count: %d\n", count);
35  }
```

3. Write a C program to perform binary search. A binary search search algorithm finds the position of a target value within a sorted array. Here's the algorithm:

```
1  Sorted array: L = [1, 3, 4, 6, 8, 9, 11]
2  Target value: X = 4
3  Compare X to 6. X is smaller. Repeat with L = [1, 3, 4].
4  Compare X to 3. X is larger. Repeat with L = [4].
5  Compare X to 4. X equals 4, so the position is returned.
```

```
1  #include <stdio.h>
2  #include <string.h>
3
4  /*
5   3. Write a C program to perform binary search. A binary search search
        algorithm finds the position of a target value within a sorted
        array. Here's the algorithm:
6
7   Sorted array: L = [1, 3, 4, 6, 8, 9, 11]
8   Target value: X = 4
9   Compare X to 6. X is smaller. Repeat with L = [1, 3, 4].
10  Compare X to 3. X is larger. Repeat with L = [4].
11  Compare X to 4. X equals 4, so the position is returned.
12
13  */
14
15 // low = lowest index to search in arr
16 // high = highest index to search in arr
17 int binary_search(int arr[], int low, int high, int target){
18
19    while (low <= high){
20       int mid = (high + low) / 2;
21
```

```
22        printf("Running loop. low: %d, mid: %d, high: %d\n", low, mid, high
            );
23
24        // if the target is at the mid position
25        if(arr[mid] == target){
26          return mid;
27        }
28
29        // if the target is less than the value at mid
30        if(target < arr[mid])
31        {
32          // move to compare left side of arr
33          high = mid - 1;
34        }
35        else
36        {
37          // move the compare right side of arr
38          low = mid + 1;
39        }
40      }
41      // return -1 to indicate the target value is not found
42      return -1;
43  }
44
45  int main(){
46      int a[] = {1, 4, 6, 8, 9, 11, 13};
47      size_t a_len = sizeof(a) / sizeof(int);
48      int target = 13;
49      int pos_rec = binary_search_recursive(a, 0, a_len-1, target);
50      printf("%d found at position %d\n", target, pos);
51  }
```

4. Write a C program to perform binary search recursively

```
1  #include <stdio.h>
2  #include <string.h>
3
4  // 4. Write a C program to perform binary search recursively
5
6  int binary_search_recursive(int arr[], int low, int high, int target){
7      // base case 1. Value is not in arr. low and high have crossed
8      if(low > high){
9          return -1;
```

```c
10      }
11      int mid = (high + low) / 2;
12      // base case 2. target value is stored at mid
13      if(arr[mid] == target){
14        return mid;
15      }
16      // otherwise recurse.
17      // if target is less than value at mid, recurse left
18      if(target < arr[mid]){
19        return binary_search_recursive(arr, low, mid-1, target);
20      }
21      // if target is greater than value at mid, recurse right
22      else
23      {
24        return binary_search_recursive(arr, mid+1, high, target);
25      }
26
27  }
28
29  int main(){
30      int a[] = {1, 4, 6, 8, 9, 11, 13};
31      size_t a_len = sizeof(a) / sizeof(int);
32      int target = 13;
33      int pos_rec = binary_search_recursive(a, 0, a_len-1, target);
34      printf("%d found at position %d using recursion\n", target, pos_rec);
35  }
```