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:

```
my_program.c: In function main:
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:

```
Undefined symbols for architecture x86_64:
  "_sqrt", referenced from:
      _main in main.o
ld: symbol(s) not found for architecture x86_64
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:

```
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:

```
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

```
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
#include <stdio.h>
#include <string.h>

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

void print_strings(char ** word_arr, size_t n_words){
  for(int i = 0; i < n_words; i++){
    printf("%s ", word_arr[i]);
  }
  printf("\n");
}

int main(){
  char *a[] = {
    "cs",
    "50",
    "is",
    "awesome"
  };
  print_strings(a, 4);
}
```

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

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

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

int count_occurances(int int_arr[][5], size_t num_cols, size_t num_rows, int target_value){
  int count = 0;

  // loop over rows
  for(int i = 0; i < num_rows; i++){
    // loop over columns
    for(int j = 0; j < num_cols; j++){
      // compare value at this position to target value
      if(target_value == int_arr[i][j]){
        count++;
      }
    }
  }

  return count;
}

int main(){
  int a[5][5] = {
    {1, 2, 3, 4, 5},
    {2, 2, 3, 4, 54},
    {6, 2, 7, 4, 5},
    {1, 2, 3, 4, 36},
    {10, 99, 3, 4, 5},
  };
  int count = count_occurances(a, 5, 5, 1);
  printf("1's count: %d\n", count);
  count = count_occurances(a, 5, 5, 7);
  printf("7's count: %d\n", count);
}
```

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:

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

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

/*
 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:

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

 */

// low = lowest index to search in arr
// high = highest index to search in arr
int binary_search(int arr[], int low, int high, int target){

  while (low <= high){
    int mid = (high + low) / 2;

    printf("Running loop. low: %d, mid: %d, high: %d\n", low, mid, high);

    // if the target is at the mid position
    if(arr[mid] == target){
      return mid;
    }

    // if the target is less than the value at mid
    if(target < arr[mid])
    {
      // move to compare left side of arr
      high = mid - 1;
    }
    else
    {
      // move the compare right side of arr
      low = mid + 1;
    }
  }
  // return -1 to indicate the target value is not found
  return -1;
}

int main(){
  int a[] = {1, 4, 6, 8, 9, 11, 13};
  size_t a_len = sizeof(a) / sizeof(int);
  int target = 13;
  int pos_rec = binary_search_recursive(a, 0, a_len-1, target);
  printf("%d found at position %d\n", target, pos);
}
```

4. Write a C program to perform binary search recursively

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

// 4. Write a C program to perform binary search recursively

int binary_search_recursive(int arr[], int low, int high, int target){
  // base case 1. Value is not in arr. low and high have crossed
  if(low > high){
    return -1;
  }
  int mid = (high + low) / 2;
  // base case 2. target value is stored at mid
  if(arr[mid] == target){
    return mid;
  }
  // otherwise recurse.
  // if target is less than value at mid, recurse left
  if(target < arr[mid]){
    return binary_search_recursive(arr, low, mid-1, target);
  }
  // if target is greater than value at mid, recurse right
  else
  {
    return binary_search_recursive(arr, mid+1, high, target);
  }

}

int main(){
  int a[] = {1, 4, 6, 8, 9, 11, 13};
  size_t a_len = sizeof(a) / sizeof(int);
  int target = 13;
  int pos_rec = binary_search_recursive(a, 0, a_len-1, target);
  printf("%d found at position %d using recursion\n", target, pos_rec);
}
```