# This is CS50.

## Lab 3

## Concept Deep Dive

## Week 3 Concepts:

- Structures in C
- Sorting
- Searching
- Recursion

## STRUCTURES IN C

Thus far, we’ve seen data types which are singular and narrow in their purpose:

- **`int`**
- **`float`**
- **`string`**
- **`bool`**
- etc.

## STRUCTURES IN C

But what if we need to represent more complex data structures in memory that are not so singular?

## STRUCTURES IN C

But what if we need to represent more complex data structures in memory that are not so singular?

**We can use utilize structures in C to help create new data types composed of variables of any number of different types themselves**

## DEFINING A STRUCT

```
typedef struct
{
     string brand;
     int year;
     float mpg;
}
car;
```

## DEFINING A STRUCT

```
typedef struct
{
     string brand;
     int year;
     float mpg;
}
car;
```

Definition

Values contained in struct

Name of structure

## DEFINING A STRUCT

```
typedef struct

{

     string brand;

     int year;

     float mpg;

}

car;
```

```
car my_car;

my_car.brand = “Chevy”;

printf("Brand: %s\n", my_car.brand);
```

## Any questions for me?

[illegible]

## Sorting: BUBBLE SORT

The least efficient, but most straightforward, method of sorting an array

Performs better when array is already sorted.

A good explanation is here.

## SORTING: SELECTION SORT

Similar efficiency as bubble sort. Performs no better when given sorted array.

An example of selection sort is as follow:

A good explanation is here.

## SORTING: MERGE SORT

Most efficient among the 3.

A good explanation is here.

## SORTING: CALCULATING EFFICIENCY

Big-O Notation is the way to go.

## SEARCHING: BINARY SEARCH

Here is a good example of Binary Search that was covered in class.

## Any questions for me?

## SOLVING A PROBLEM RECURSIVELY

1. Define the base case
    a. This is the condition that will cause your code to stop calling itself
2. Identify how your function parameter will change for each recursive call
3. Write your function!

## Example 1: FACTORIAL

### Task:

We will write a recursive function factorial.c that takes an integer n and computes its factorial.

### Sample Usage:

```
$ ./factorial
What would you like to factorial?: 5
Answer: 120
```

## Example 1: FACTORIAL

```c
#include <cs50.h>
#include <stdio.h>

int factorial(int n);
int main(void)
{
    // Get input
    int n = get_int("Find factorial of: ");
    int answer = factorial(n);
    printf("%i\n", answer);
}

int factorial(int n)
{
    // CODE HERE
}
```

## What is the base case?

## Base case: $0! = 1$

## Example 1: FACTORIAL

```c
#include <cs50.h>
#include <stdio.h>

int factorial(int n);
int main(void)
{
    // Get input
    int n = get_int("Find factorial of: ");
    int answer = factorial(n);
    printf("%i\n", answer);
}

int factorial(int n)
{
    if (n == 0)
        return 1;
    // Recursive Call
}
```

## What is the recursive call?

$$
\begin{aligned}
3! &= 3 * 2! \\
&= 3 * 2 * 1! \\
&= 3 * 2 * 1 * 0! \\
&= 3 * 2 * 1 * 1
\end{aligned}
$$

## Recursive call: (n-1)!

## Example 1: FACTORIAL

```
int factorial(int n)
{
    if (n == 0)
        return 1;
    return n * factorial(n - 1);
}
```

## Any questions for me?

## Lab Time!

## Task:

Try to determine what sorts algorithms (Bubble sort, Selection Sort and Merge Sort) that sort1, sort2, sort3 have based on the examples given!

## Tips:

- To run the sorts on the text files, in the terminal, run ./[program_name] [text_file.txt]. Make sure you have made use of cd to move into the sort directory!
    - For example, to sort reversed10000.txt with sort1, run ./sort1 reversed10000.txt.

## Tips:

- You may find it helpful to time your sorts. To do so, run `time ./[sort_file] [text_file.txt]`.
  - For example, you could run `time ./sort1 reversed10000.txt` to run sort1 on 10,000 reversed numbers. At the end of your terminal's output, you can look at the real time to see how much time actually elapsed while running the program.

## Any questions for me?

## Let’s do lab!

**sort1 uses:** Bubble Sort

**How do you know?:** On a large random file, sort1 takes much longer than an already-sorted file of the same size. For already-sorted files, sort1 returns results almost instantaneously, regardless of size. This suggests that sort1 has a different upper bound runtime than lower bound runtime. This is consistent with Bubble Sort, which runs in O(n^2) and Ω(n).

**sort2 uses:** Merge Sort

**How do you know?:** On larger random files, sort2 is the fastest of the three sorts. Since we know that Merge Sort runs in O(n log n) while Bubble Sort and Selection Sort run in the slower O(n^2), sort2 is likely Merge Sort.

**sort3 uses:** Selection Sort

**How do you know?:** On a large random file, sort3 takes just as long as sort1, but performs no better when the list is already sorted. This suggests that the algorithm runs in O(n^2) and Ω(n^2), which is consistent with Selection Sort.

## Any questions for me?

## Problem Set Tips

## Problem Set: Plurality

### Prompt walkthrough, watch Bryan’s Video

## Overview:

- Complete the `vote` function.
    - `vote` takes a single argument, a string called name, representing the name of the candidate who was voted for.
    - If name matches one of the names of the candidates in the election, then update that candidate's vote total to account for the new vote. The `vote` function in this case should return true to indicate a successful ballot.
    - If name does not match the name of any of the candidates in the election, no vote totals should change, and the `vote` function should return false to indicate an invalid ballot.
    - You may assume that no two candidates will have the same name.

## Pseudocode: `vote`

```
// loop through candidate count

    // if tested name is the same as candidate’s name

        // increase votes
        // return true

// return false (Not a valid candidate)
```

**Tips: use `strcmp` to compare name and candidate name!**

## Overview:

- Complete the `print` function.
    - The function should print out the name of the candidate who received the most votes in the election, and then print a newline.
    - It is possible that the election could end in a tie if multiple candidates each have the maximum number of votes. In that case, you should output the names of each of the winning candidates, each on a separate line.

## Pseudocode: `print`

```
// initialize maximum = 0
// loop through candidates count to find maximum vote
    // if candidates’ ____ > max
    // set max = candidates’s ____
// loop through candidates count to find the winners
    // if candidates’ votes == max
    // print winners
```

## Pseudocode: `print`

```
// initialize maximum = 0
// loop through candidates count to find maximum vote
    // if candidates’ votes > max
    // set max = candidates’s votes
// loop through candidates count to find the winners
    // if candidates’ votes == max
    // print winners
```

## Problem Set: Runoff

### Prompt walkthrough, watch Bryan’s Video

## Six functions to code:

```
- vote
- tabulate
- print_winner
- find_min
- is_tie
- eliminate
```

## Overview: `vote`

1) Complete the `vote` function.
    - The function takes arguments `voter`, `rank`, and `name`. If name is a match for the name of a valid candidate, then you should update the global preferences array to indicate that the voter voter has that candidate as their rank preference (where 0 is the first preference, 1 is the second preference, etc.).
    - If the preference is successfully recorded, the function should return true; the function should return false otherwise (if, for instance, name is not the name of one of the candidates).
    - You may assume that no two candidates will have the same name.

## Tips:

- Recall that `candidate_count` stores the number of candidates in the election.
- Recall that you can use `strcmp` to compare two strings.
- Recall that `preferences[i][j]` stores the index of the candidate who is the jth ranked preference for the ith voter.

## Pseudocode

```
// loop through candidates’ count

    // if name is equal to the candidate’s name

        // update preference
        // return True

// return False (candidates not in the list)
```

## Overview: `tabulate`

2) Complete the `tabulate` function.

- The function should update the number of `votes` each candidate has at this stage in the runoff.
- Recall that at each stage in the runoff, every voter effectively votes for their top-preferred candidate who has not already been eliminated.

## Tips:

- Recall that `candidate_count` stores the number of candidates in the election.
- Recall that you can use `strcmp` to compare two strings.
- Recall that `preferences[i][j]` stores the index of the candidate who is the jth ranked preference for the ith voter.

## Pseudocode

```
// loop through voter count

    // loop through candidates count

        // find the preference in the preferences 2×2 array

          // If candidate not eliminated

                // update the vote of the preferred’ candidate
                // break
```

## Overview: `print_winner`

3) Complete the `print_winner` function.

- If any candidate has more than half of the vote, their name should be printed and the function should return true.
- If nobody has won the election yet, the function should return false.

## Tips:

- Recall that `voter_count` stores the number of voters in the election. Given that, how would you express the number of votes needed to win the election?

## Pseudocode

```
// loop through candidate count

    // Check if any candidate has more than half of the vote

        // print that candidate's name as winner
        // return True

// no winner yet (return False)
```

## Overview: `find_min`

4) Complete the `find_min` function.

- The function should return the minimum vote total for any candidate who is still in the election.

## Tips:

- You’ll likely want to loop through the candidates to find the one who is both still in the election and has the fewest number of votes. What information should you keep track of as you loop through the candidates?
- Very similar to finding max in plurality!

## Pseudocode

```
// initialize min to maximum integer
// loop through candidate count
    // if requirement met
        // reset min
// return min
```

## Overview: `is_tie`

5) Complete the `is_tie` function.

- The function takes an argument `min`, which will be the minimum number of votes that anyone in the election currently has.
- The function should return true if every candidate remaining in the election has the same number of votes, and should return false otherwise.

## Tips:

- Recall that a tie happens if every candidate still in the election has the same number of votes. Note, too, that the `is_tie` function takes an argument `min`, which is the smallest number of votes any candidate currently has. How might you use that information to determine if the election is a tie (or, conversely, not a tie)?

## Pseudocode

```
// loop through candidate count
    // if tie requirement met / not met
        // return True / False
```

# Overview: `eliminate`

6) Complete the `eliminate` function.

- The function takes an argument `min`, which will be the minimum number of votes that anyone in the election currently has.
- The function should eliminate the candidate (or candidates) who have `min` number of votes.

## Pseudocode

```
// loop through candidate count
    // if eliminate requirement met
        // set true for candidates’ eliminate
```

## More tips

- Use ! to check for true and false statements
- Use  ++ to increment counts / votes / anything
- INT_MAX (set the integer to the maximum number in integer in C)
- Use && to check for “And” statements
- Use == to check for equivalence in quantities!

## Tutorials, OHs

## Feedback form:

tinyurl.com/zad-feedback

## Thank you!

# See you next week!