CS 261 Recitation 2: Dynamic Array and Linked List

In order to get credit for the recitation, you need to be checked off by the end of recitation. For non-zero recitations, you can earn a maximum of 3 points for recitation work completed outside of recitation time, but you must finish this recitation before the next recitation. For extenuating circumstance, contact your recitation TAs and Instructor.

Recitation 2 Grade Breakdown:

- Part 1: Add two numbers using linked list — 4 pts (group work)
- Part 2: Sort Dynamic Arrays, Function Pointers — 6 pts (individual work)

Download and unzip the start code for this recitation:

https://classes.engr.oregonstate.edu/eecs/summer2022/cs261-001/recitations/rec2.zip

(4 pts) Part 1: Add two numbers using linked list

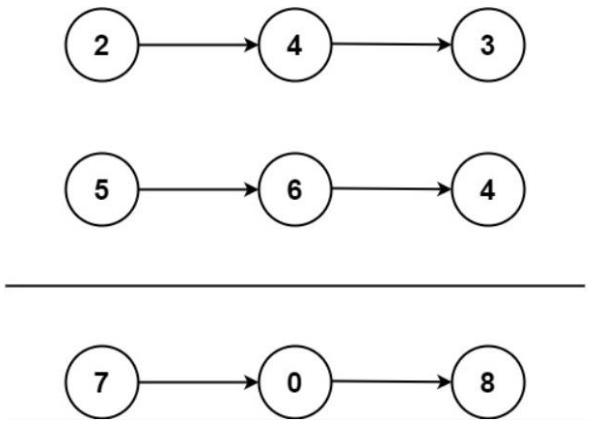
In part 1 of this week's recitation, you will be working **in a group** to design and implement a classic linked list interview question. After completing this part, you should be able to:

1. iterate through elements in a singly linked list
2. perform basic list operations
3. understand how singly linked list works

Problem Statement: You are given two **non-empty** linked lists representing two non-negative integers. The digits are stored in **reverse order**, and each of their nodes contains a single digit. Add the two numbers and return the sum as a linked list.

You may assume the two numbers do not contain any leading zero, except the number 0 itself.

For example:




```
Input: l1 = [2,4,3], l2 = [5,6,4]
Output: [7,0,8]
Explanation: 342 + 465 = 807.
```

1. Within a group of 3-4, understand the problem and take a look at the node structure provided in the `ll.c` (or `ll.h`) and the function you need to implement:
   `struct node* add_two_num (struct node* l1, struct node* l2);`
2. Discuss in your group and design the solution to `add_two_num()`.
3. Implement your design as a group.
4. Test your implementation by running
   `make test_list`
   `./test_list`
5. Once you successfully pass all test cases, share your solution with other groups.

*Note: In order to get the maximum practice of linked list and its operation, please DO NOT use functions provided in `test_list.c` as your helper functions.

(6 pts) Part 2: Sort Dynamic Arrays, Function Pointers

In part 2 of this week's recitation, you'll work on `dynarray.c`. After completing this part, you should be able to:

1. create dynamic arrays of different data types
2. sort elements in a given array
3. understand how function pointer works

This is an individual exercise: you'll do it on your own, not in a group. Here's what you must do:

(1 pt) Step 1: Follow comments instructions in the `main()` to create a dynamic array of 10 integers, assign them with random values (0-100, inclusive), sort, and print them.

You will implement everything up till Step 2 in the `main()` as well as the `sort(int *arr, int n)` function.

(2 pts) Step 2: Change the prototype of `sort()` function from

```
sort(int *arr, int n);
```

To

```
void sort(void** arr, int n, int(*cmp)(void* a, void* b));
```

Now this function is able to sort an array of a generic type (`void*`). The new `sort()` function also needs to take a function pointer as its 3rd argument to determine how to sort its elements.

Modify the `sort()` function using function pointer as well as the code you implemented in Step 1, i.e., the function call.

(2 pts) Step 3: Follow comments instructions in the `main()` to create a dynamic array of 10 students, assign them with random IDs (0-100, inclusive) and scores (0-100,

inclusive), sort (by calling the modified `sort()` in Step 2), and print them. Note: no two students should have the same IDs.

(1 pt) Step 4: Free the memory allocated in Step1-3. You will need to implement the `free_arr()` function.

To test your implementation, run

```
make dynarray
valgrind –leak-check=full dynarray
```

Make sure your program doesn't have any memory leaks!!!

Make sure you get checked off by showing them the output of your program and your group work before the end of your recitation section.

For backup purposes, please submit your work for this recitation (including all documents/text files for group work, and programs) to TEACH.