ECE220 Computer Systems and Programming

Lab 9

1 After this week's lectures, you should be able to...

1. Explain the difference between static and dynamic memory allocation in terms of their allocation mechanisms, lifetime, location, and size.
2. Dynamically allocate and free memory in C using malloc() and free().
3. Explain the difference between an array and a linked list in terms of their memory allocation, memory structure, order of access and insertion/deletion mechanisms.
4. Implement linked list operations such as insertion, deletion, and searching.

2 After today's lab, you should be able to...

1. Correctly use malloc() and realloc() by passing the appropriate arguments.

3 Exercises

1. In lecture 11, we learned that the syntax in line 3 and 4 below are equivalent. Rewrite line 5 without using square brackets.

```
1 char word[10];
2 char solution[4][10];
3 word[2] = 'a';
4 *(word + 2) = 'a';
5 solution[3][4] = 'a';
```

((soluion+3)+4) = 'a'

2. This exercise aims to show you what NOT to do in MP9. In this MP, we will dynamically allocate a 2D array to represent a maze. Check maze.h to see its definition. Your friend Ben Bitdiddle allocated the 2D array as following in line 3. He got a segmentation fault at line 6, meaning his program was accessing invalid memory locations. You've identified that the way he's allocating the 2D array on line 3 is incorrect. Explain to Ben Bitdiddle why the segmentation fault occurred and why his allocation was incorrect. (Hint 1: use your answer from question 1. Hint 2: when malloc is called, the allocated memory is not initialized and may contain garbage.)

```
1 maze_t* maze; // initialization omitted. Assume correct.
2 int i, j, height, width; // initialization omitted. Assume correct.
3 maze->cells = (char**)malloc(height * width * sizeof(char));
4 for(i = 0; i < height; i++){
5     for(j = 0; j < width; j++){
6         maze->cells[i][j] = '*';
7     }
8 }
```

line3: maze->cells = (char**)malloc(height * sizeof(char *));
between line4~5: maze->cells[i] = (char*)malloc(width * sizeof(char));