Lab 12 - Recursion

Pointers, Double Pointers, Recursion, Pointer Arithmetic Notation

Review - Pointers

- It is a memory address.
  - Ideally, it will point to the beginning of a variable. (technically, we can point to the middle of an int or any type.)
  - We use it in functions to modify values of variables elsewhere.
    - This will become much more relevant when working with malloc (dynamic allocation)
  - If we are working with small types (int, char, etc.), avoid pointers when working locally.
  - If we are working with large types or arrays, we want to use malloc (to be discussed).

Review – Passing in Pointers to Functions

```
// Returns nothing
// Accepts address of int
void acceptIntPtr(int * ptr) {
    // We can modify and read ptr in addition to
    // the location it points to.

    // For now, we do not want to modify the
    // address.
    *ptr = 100; // Set wherever it was stored to 100
    int x = *ptr; // Read the value ptr points to
}
```

Review – Passing in Pointers to Functions

```
int myIntVariable = 200;
// To call acceptIntPtr, use the & operator.
acceptIntPtr(&myIntVariable);
```

Passing Array to Functions

```
char maze[8][8] = {
        {' ', 'X', ' ', ' ', ' ', ' ', ' ', ' '},
        {' ', 'X', ' ', 'X', 'X', 'X', 'X', ' '},
        {' ', 'X', ' ', 'X', ' ', ' ', ' ', ' '},
        {' ', 'X', ' ', 'X', ' ', 'X', 'X', 'X'},
        {' ', 'X', ' ', 'X', ' ', ' ', ' ', ' '},
        {' ', 'X', ' ', 'X', 'X', ' ', ' ', ' '},
        {' ', 'X', ' ', 'X', ' ', ' ', 'X', ' '},
        {' ', ' ', ' ', 'X', ' ', ' ', ' ', ' '},
 };
 Coord ** result = solveMaze(maze);     // Coord ** solveMaze(char maze[8][8])
```

solveMaze's prototype

^~~~ Passing here. We simply pass in the name.

Do not use &maze!

Linked Lists (No need to remember yet)

- A list that is connected to a next node, last node, or both.
- Depending on the implementation, it can be
  - Singly (we only see next)
  - Doubly (We see next and last)
  - Circular (We see next and last; very last item goes back to first; very first goes to last)

```
struct linkedList {
    int data;
    struct linkedList * next;
    struct linkedList * last;
};
```

Double Pointers (No need to remember yet)

- We use namely when using linked lists. We want to store the head pointer in a variable.
- We then need to pass the double pointer into a function which creates the list.
- As such, we take the address of the address.
- PARAMETER TYPES OF 2D ARRAYS MAY RESEMBLE A DOUBLE POINTER, BUT IT IS NOT NECESSARILY A DOUBLE POINTER.
- Make sure that we do not use the & operator when passing arrays unless we need to
- Example to follow next time

Recursion

- A function that calls itself.
- One or more paths of non-dead code which calls itself.
- If a function calls itself through all paths, then it will crash since we run out of RAM.
- Faster on some systems
- An alternative to loops

Bad example:

```
void foo(void) {
   foo();
}
```

Pointer Arithmetic Notation (Example)

```
Coord ** result = (Coord **)malloc(sizeof(Coord * ) * 1000); // 1000 ele array
… Initialize array of pointers …
solveMazeHelper(maze, 0, 1, *result); // See starter code for context
// EX IN void solveMazeHelper(char maze[8][8], int x, int y, Coord * result)
if (x < 5) {
    solveMazeHelper(maze, x + 1, y, result + 1);
}                                                ^~~~~~~~ result is a pointer,
```

we then use + 1 which will go to the next element in the list.

IMPORTANT NOTE: This code is not safe since it does not make sure it is within bounds.

Lab 12 Example Code

- Questions on the slides?