16.216: ECE Application Programming

Summer 2013

Lecture 11: Key Questions

August 13, 2013

1. (Review) Briefly describe the memory allocation functions and their use.

2. Explain how to use dynamic memory allocation with strings.

3. Explain how to use dynamic memory allocation with two-dimensional arrays.

4. **Example:** Write each of the following functions:

a. **`char *readLine():`** Read a line of data from the standard input, store that data in a dynamically allocated string, and return the string (as a **`char *`**)
Hint: Read the data one character at a time and repeatedly reallocate space in string

b. **`int **make2DArray(int total, int nR)`:** Given the total number of values and number of rows to be stored in a two-dimensional array, determine the appropriate number of columns, allocate the array, and return its starting address
Note: if **nR** does not divide evenly into **total**, round up. In other words, an array with 30 values and 4 rows should have 8 columns, even though 30 / 4 = 7.5

5. Explain the use of general data structures and pointer-based data structures in particular.

6. Describe the general design of a linked list.

7. Describe the structure used for each node in the list.

8. Explain the operation of the following function, which adds a node to the beginning of the list and returns a pointer to that node.

```
LLnode *addNode(LLnode *list, int v) {
  LLnode *newNode;
  // Allocate space for new node; exit if error
  newNode = (LLnode *)malloc(sizeof(LLnode));
  if (newNode == NULL) {
    fprintf(stderr,
              "Error: could not allocate new node\n");
    exit(0);
  }
  newNode->value = v;    // Copy value to new node
  newNode->next = list;  // next points to old list
  return newNode;
}
```

9. Write each of the following functions:

a. Finding item in list (Function should return pointer to node if found and return NULL otherwise)

```
LLnode *findNode(LLnode *list, int v) {











}
```

b. Removing item from list
- Must deallocate space for deleted node
- Function should return pointer to start of list after it has been modified
    - No modifications should be made if value v is not in list
    - Hint: you can use the `findNode()` function in this function, but you may not want to!
- Note: removing first element in list is special case

```
LLnode *delNode(LLnode *list, int v) {











}
```

10. Describe how to maintain a sorted linked list.

11. Write each of the following functions:
a. Adding an item to a sorted linked list
   - Use **addNode()** as a starting point
   - Instead of adding node at beginning, find appropriate place in list and then add
   - Function should return pointer to start of list after it has been modified

```
LLnode *addSortedNode(LLnode *list, int v) {
















}
```

b. Finding an item in a sorted linked list
   - Use **findNode()** as starting point—should perform same operation, but more efficiently
   - Function should return pointer to node if found
   - Return NULL otherwise

```
LLnode *findSortedNode(LLnode *list, int v) {













}
```