EECE.2160: ECE Application Programming

Fall 2016

Lecture 30 & 31: Key Questions
November 28 & 30, 2016

Note: This handout will be used for the next two lectures—if you get the handout during Lec. 30, please bring it to Lec. 31!

1. (Review) Explain the dynamic allocation functions `malloc()`, `calloc()`, `realloc()`, and `free()`.

2. **Example:** What does the following program print?

```
void main() {
   int *arr;
   int n, i;

   n = 7;
   arr = (int *)calloc(n, sizeof(int));
   for (i = 0; i < n; i++)
      printf("%d ", arr[i]);
   printf("\n");

   n = 3;
   arr = (int *)realloc(arr, n * sizeof(int));
   for (i = 0; i < n; i++) {
      arr[i] = i * i;
      printf("%d ", arr[i]);
   }

   n = 6;
   arr = (int *)realloc(arr, n * sizeof(int));
   for (i = 0; i < n; i++) {
      arr[i] = 10 - i;
      printf("%d ", arr[i]);
   }

   free(arr);
}
```

3. What are the common pitfalls of dynamic memory allocation?

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

5. Describe the general design of a linked list.

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

7. 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;
}
```

8. 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. Write the following function used to remove a node 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) {




}
```

9. Describe how to maintain a sorted linked list.

10. Write each of the following functions:

c. 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) {




}
```

d. 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) {




}
```