# Discussion 5(11/2)

ECE 17

# Assignment 3 Typo

### - Part 2 - Implement Your LinkedList Class

We've given you a starting point for this assignment. Open your `LinkedList.hpp` class (take a look at that class now). Here's the interface for this class:

```
class LinkedList {
protected:

  //You'll add OCF Methods here...

  Node*   first();  //retrieve first node in the list.
  size_t  size();   //return # of items in the list...
  Node*   find(const int &aValue, Node *anOrigin=nullptr);
  Node*   append(const int &aValue);
  Node*   insertAfter(const int &aValue, Node *anOrigin=nullptr);
  Node*   remove(const int &aValue);

protected:
  Node    *root;
  //other members?
};
```

The functions should be “public” and not “protected”

# Assignment 2 Grades

- Released on canvas
- Any grading issues, please email me: brl072@ucsd.edu
- Assignment 1 grades will be released shortly

# Setting functions as const

Why do we do this?

```
class LinkedList {
protected:

  //You'll add OCF Methods here...

  Node*   first();  //retrieve first node in the list.
  size_t  size();   //return # of items in the list...
  Node*   find(const int &aValue, Node *anOrigin=nullptr);
  Node*   append(const int &aValue);
  Node*   insertAfter(const int &aValue, Node *anOrigin=nullptr);
  Node*   remove(const int &aValue);

protected:
```

- We do this to copy constructors, find(), append(), insertAfter(), remove(), etc

# Setting functions as const

- We do this in order to protect the object passed in, so to prevent accidentally changing it
  - Ex: in find, we only care about the value, and have no intention of modifying it
- Some cases => performance boost

```
class LinkedList {
protected:

  //You'll add OCF Methods here...

  Node*   first();  //retrieve first node in the list.
  size_t  size();   //return # of items in the list...
  Node*   find(const int &aValue, Node *anOrigin=nullptr);
  Node*   append(const int &aValue);
  Node*   insertAfter(const int &aValue, Node *anOrigin=nullptr);
  Node*   remove(const int &aValue);

protected:
```

# find(const int &aValue, Node* anOrigin)

- Start with the passed in node, check if it is a nullptr or not
  - If passed in nullptr -> we start at the root
  - If passed in not nullptr -> we start at that node
- Keeping going down the node and next node
  - Check if the current Node's value equals that of the "aValue" passed in, if it is => return that node
  - Be sure to check if the node is nullptr or not before checking ite value
    - Two ways to do this: (choose one or the other)
      - Within a loop: break if current node is a nullptr
      - Within a loop: break if next node is a nullptr
  - Return nullptr if no matching Node is found

# Node*  remove(const int &aValue);

- Two parts to this method:
  1) Find if node exists
  2) Remove node from list
- To find if node exists => use find method to see if node exists.
  - Ex: Node* toDelete = find(aValue, nullptr)
  - If node exists -> continue to remove it, else just return a nullptr
- Note: this function only removes one Node. If duplicate valued nodes exists, it is up to the caller to handle them
- Note: This function only removes Node from linkedList. Atual destruction of the Node is not handled here

# Node*  remove(const int &aValue); cont

- 3 conditions to consider: removing head, removing tail, removing something in the middle
- What we need to think about:
  - head/root change?
  - Do I need to consider the previous node's "next"
  - What do I set the "next" on Node that I am removing
    - this can be handled by the caller as well
  - What to do with nodeCount

# Deleting demo

Lets play with a train

# Any questions?