# Agenda

## Coming up:

- Midterm 1 - Friday, Feb 21st, 5-7PM
  - Hand written
    - structs
    - classes
    - cmd line args
    - pointers
    - parameter passing: by-val, by-ref, by-ptr, by-arr
    - array doubling
    - linked lists
    - maybe stacks and/or queues
  - Conflict make-up
    - will post info soon
    - Wednesday prior
  - Special accommodations:
    - probably same time, different location
    - watch for announcements

## This week:

- Assignment 3:
  - due this Sunday, 6PM (Feb. 9)
- Moodle quiz released today

## Today:

- Continue Singly Linked List C++ implementations:
  - insert
  - destroy
  - delete

# The Header File

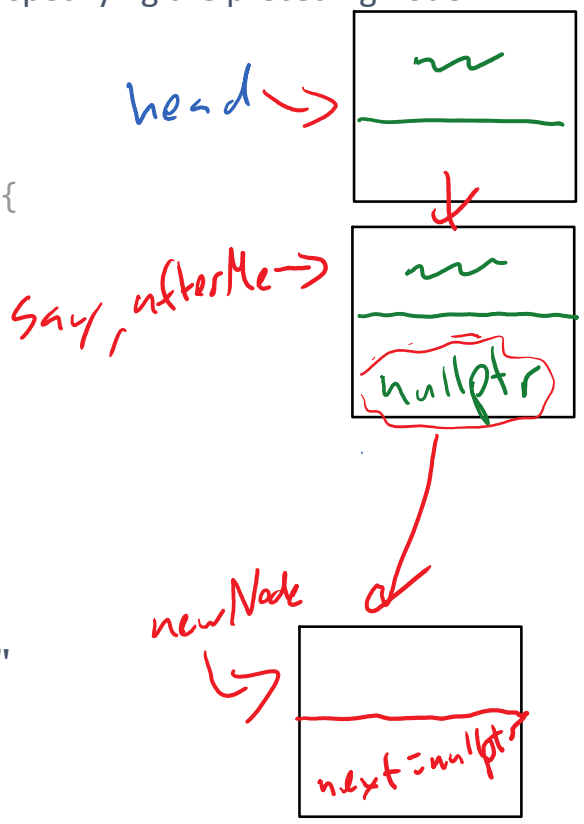
```
// SLL.hpp - interface file (header file)
struct Node{
    string key;
    Node *next;
};
class SLL{
private:
    Node* head;
public:
    SLL(); // constructor declaration
    ~SLL(); // destructor declaration
    Node* search(string sKey);
    // Precodndition: sKey parameter is a string type
    // Postcondition:  if found, returns a pointer to the node containing sKey value.
    // If not found, returns a null pointer.
    void displayList();
    // Precondition: the head node is defined.
    // Post condition: display the key values of the entire list, starting with
    // first node and ending with last node.
    void insert(Node* afterMe, string newValue);
    // Precondition: afterMe is a valid pointer to a node in the linked list.
    // newValue is a valid string.
    // Postcondition: a new node is created and newValue is stored as its key.
    // The new node is added after node containing afterMe.
    void deleteNode(Node* deleteNode);
    // Precondition: head and tail pointers are set.
    // Post condition: node where with a matching key value is deleted.
};
```

# Implementations in C++ (insert)

```
void insert(Node* afterMe, string newValue);
// Precondition: afterMe is a valid pointer to a node in the linked list.
// newValue is a valid string.
// Postcondition: a new node is created and newValue is stored as its key.
// The new node is added after node containing afterMe.
```

Must account for different scenarios:

1) Empty list
2) List is not empty. User wants to insert node at the beginning of list (thus making it the new **head.)** ← afterMe = nullptr
3) List is not empty. User wants to insert a node specifying the preceding node.




```
void SLL::insert(Node* afterMe, string newValue){
    if(head == nullptr){
        // empty list
        Node* head = new Node;
        head->key = newValue;
        head->next = nullptr;
    }
    else if(afterMe==nullptr){
    // implies the user wants to add new node at head
        Node* newNode = new Node;
        newNode->key = newValue;
        newNode->next = head; //" old head"
        head = newNode;
    }
    else{
    // list is not empty, add new node either in the middle
    // or at the end of linked list
        Node* newNode = new Node;
        newNode->key = newValue;
        newNode->next = afterMe->next
        afterMe->next = newNode
    }
}
```

# The Destructor

Just like there is a constructor that gets called automatically when an object is instantiated, the destructor gets called when its function pops off the stack.

No need to define destructors when not working with dynamic memory.

Anytime data structure that uses explicit dynamic memory allocations (new keyword), needs to have a destructor defined to deallocate.

The syntax in definition uses the ~<class name>. E.g.:

```
class SLL{
private:
      ...
public:
   SLL();   // constructor
   ~SLL(); // destructor

}
```

We can also call the destructor manually. E.G.:

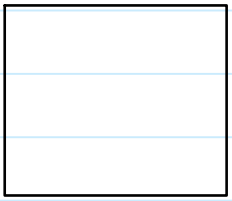
```
int main(){
   SLL S0;
   S0.~SLL();

}
```

# Implementations in C++ (destructor)

Approach:

```
SLL::~SLL(){
}
```



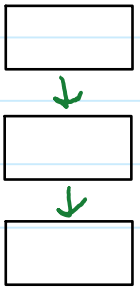
# Implementations in C++ (delete)

Tuesday, September 17, 2019 2:33 PM

Remove node from LL pointed to by ptr.

Function should never be called with null ptr.

2 cases to consider:



```
void SLL::deleteNode(Node* deleteNode){
}
```