Announcements

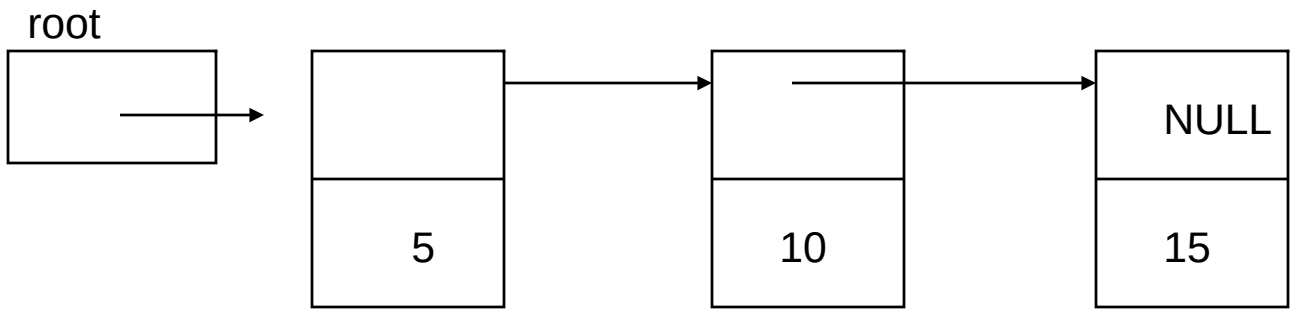
- Midterm #1
  - Re-grade requests due by 1:45 PM
- Program #2
  - Due on Tuesday
- Reading
  - Chapter 12, 10.2 (Today)
  - Notes (Tuesday)

Linked Data Structures

- Like classes with refs to same class in Java
- Have pointers to a struct of the same time
- Example Declaration:

```
typedef struct NODE {
        struct NODE *next;
        int value;
} Node;
```

- Next is a pointer to another Node



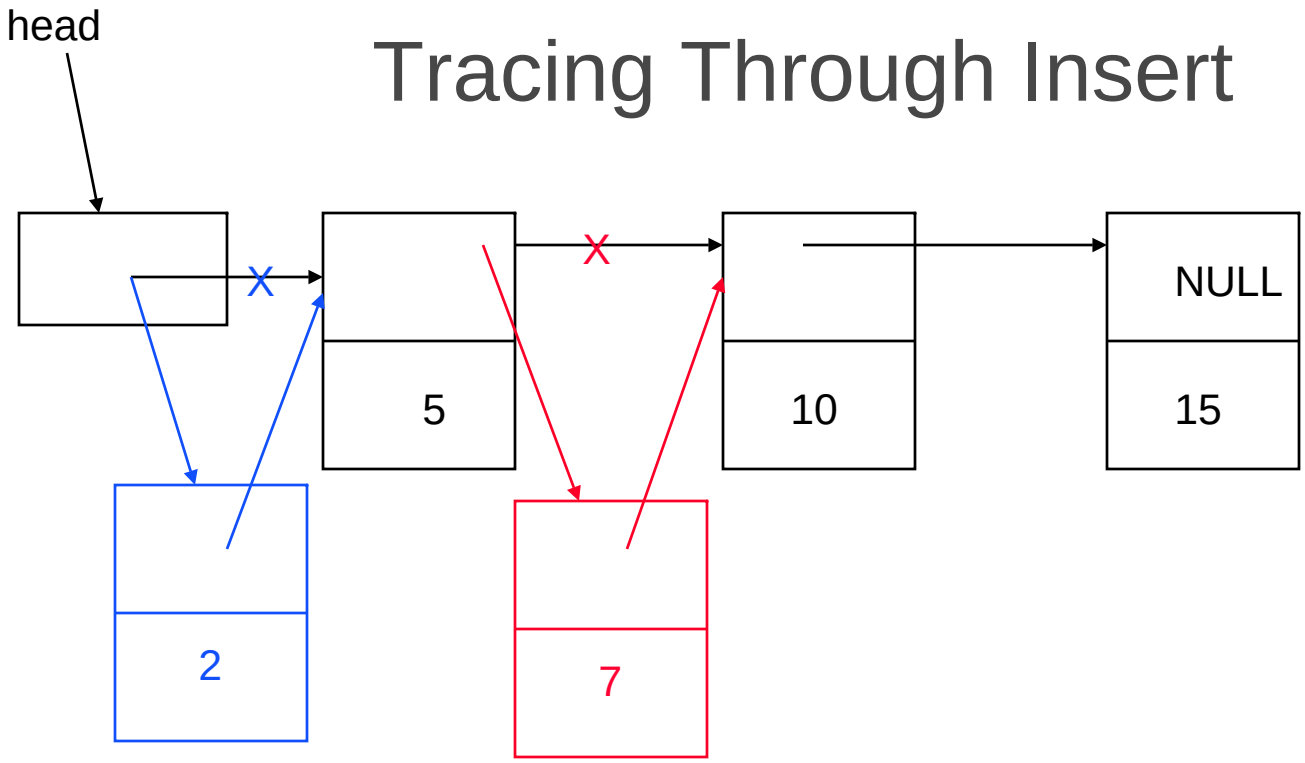
Lookup in a Linked Structure

```
int Lookup(Node *current, int value) {
        while (current && current->value != new_value)  {
            current = current->next;
        }
        if (current) {
            return 1;
        } else {
            return -1;
        }
}
```

Inserting Into a Linked List

```
int insert(Node **head, int new_value) {
        Node *prev = NULL, *newItem;
        Node *current = *head;
        while (current && current->value < new_value)  {
                prev = current;
                current = current->next;
        }
        newItem = (Node *) malloc(sizeof(Node));
        if (!newItem) return -1;
        newItem->value = new_value;
        newItem->next = current;
        if (!prev) {
                *head = newItem;
        } else {
                prev->next = newItem;
        }
}
```

Tracing Through Insert



- Insert 7
- Insert 2

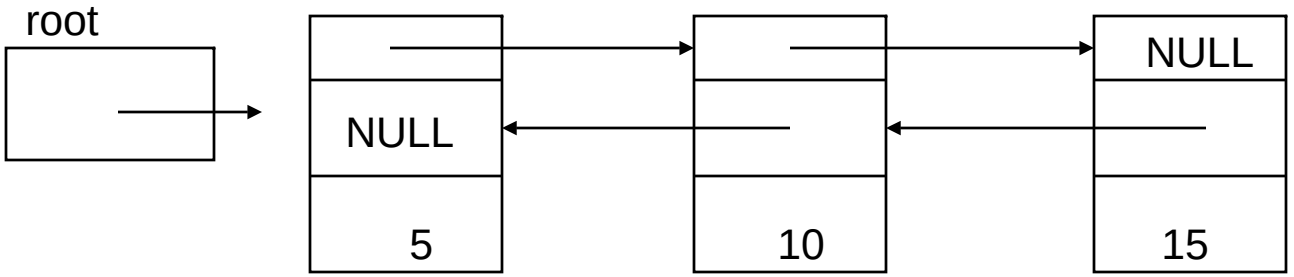
Deleting From A Single Linked List

```
int delete(Node **head, int new_value) {
        Node *prev = NULL;
        Node *current = *head;
        while (current && (current->value != new_value))  {
            prev = current;
            current = current->next;
        }
        if (!current) {
            return -1;          /* not found */
        }
        if (prev) {
            prev->next = current->next;
        } else {
            *head = current->next;      /* deleted first item */
        }
        free (current);
        return 0;
}
```

Doubled Linked Lists

- Each nodes
  - contains a value
  - a pointer the next **and** previous element
- Typical Declaration:

```
typedef struct NODE {
    struct NODE *next;
    struct NODE *prev;
    int value;
} Node;
```



Insert into a doubly Linked list

- Think about 4 cases:
  - Middle of the list
    - Update previous element's next pointer
    - Update next elements previous pointer
  - End of the list
    - Update previous element's next pointer
  - Start of the list
    - Update head of list
    - Update old head of list's previous element
  - An initially empty list
    - Update head of the list

Inserting Into A Doubly Linked List

```
int insert(Node **head, int new_value) {
        Node *prev = NULL, *newItem;
        Node *current = *head;
        while (current && current->value < new_value)  {
                prev = current;
                current = current->next;
        }
        newItem = (Node *) malloc(sizeof(Node));
        if (!newItem) return -1;
        newItem->value = new_value;
        newItem->next = current;
        newItem->prev = prev;
        if (!prev) {
                *head = newItem;
        } else {
                 prev->next = newItem;
        }
        if (current) {
            current->prev = next;
        }
}
```

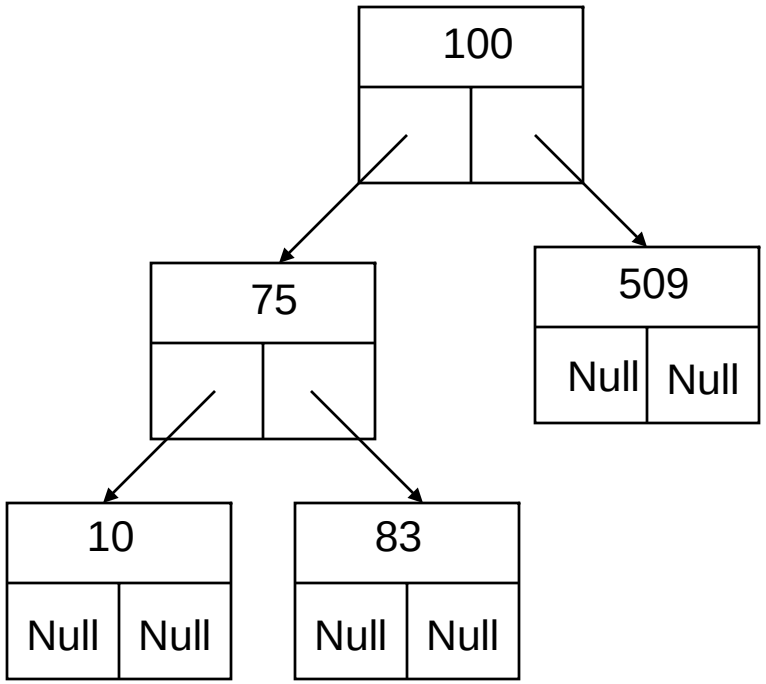
Deleting from a Doubly Linked List

```
int delete(Node **head, int new_value) {
        Node *prev = NULL;
        Node *current = *head;
        while (current && (current->value != new_value))  {
                prev = current;
                current = current->next;
        }
        if (!current)     return -1;          /* not found */
        if (prev) {
            prev->next = current->next;
        } else {
            *head = current->next;        /* deleted first item */
        }
        if (current->next) {
            current->next->prev = prev;
        }
        free (current);
        return 0;
}
```

Binary Trees

- Each nodes
  - contains a value
  - a pointer a left child and a right child
- Typical Declaration:

```
typedef struct NODE {
    struct NODE *left;
    struct NODE *right;
    int value;
} Node;
```



Lookup In A Binary Search Tree

- If the element is less than the current node
  - Look in left child tree
- If the element is greater than current node
  - Look in the right child tree
- Example (assumes values >= 0):

```
int Lookup(Node *root, int value) {
    if (!root) return -1;
    if (root->value == value) {
        return value;
    } else if (root->value > value) {
        return Lookup(root->left, value);
    } else {
        return Lookup(root->right, value);
    }
}
```

Insert Into a Binary Tree

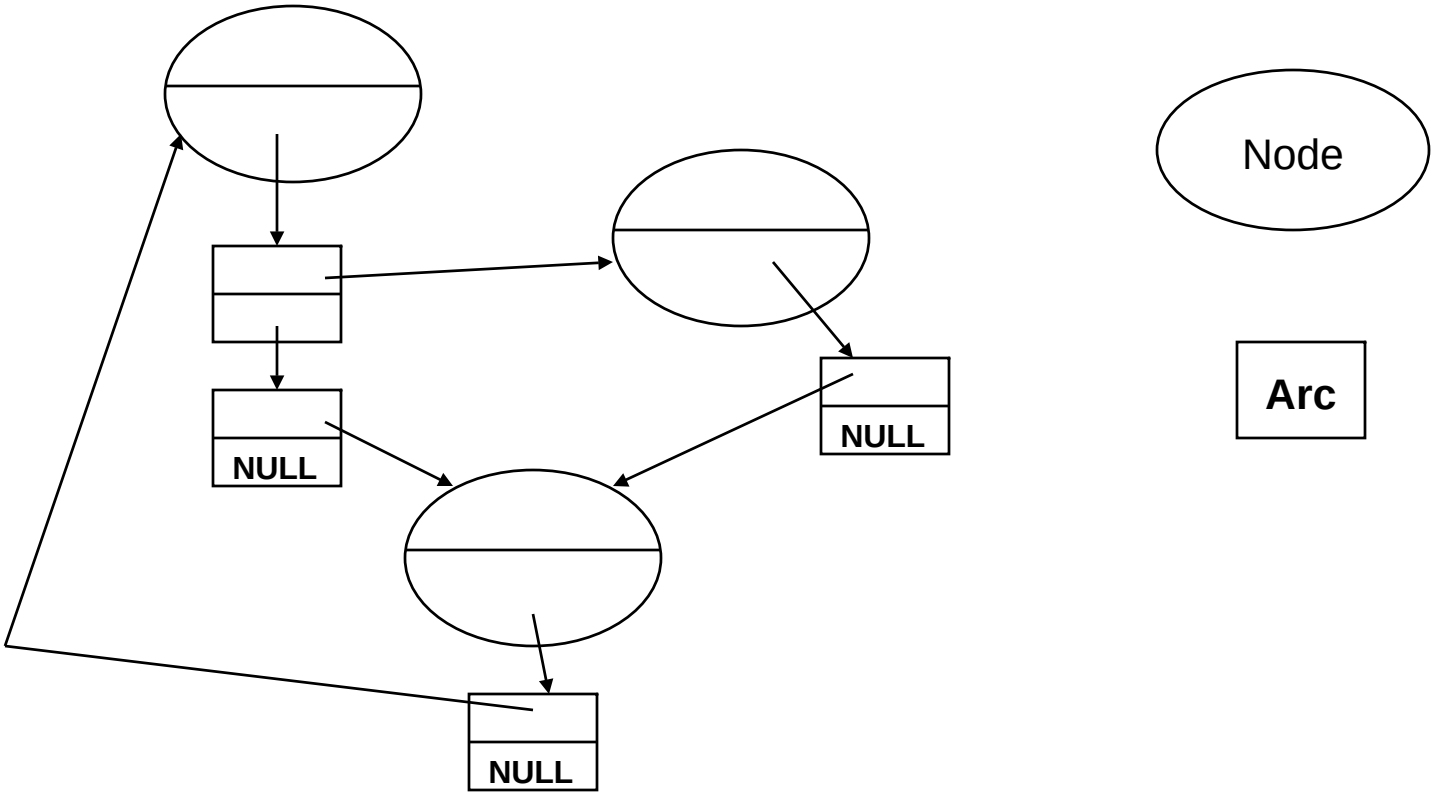
```
int insert(Node **root, int value) {
        Node *new;
        if (!*root) {
            new = (Node *) malloc(sizefof(Node));
            if (!new) return -1;
            *root = new;
            new->left = new->right = NULL;
            new->value = value;
            return 0;
        } if ((*root)->value > value) {
            return insert(&((*root)->left), value);
        } else {
            return insert(&((*root)->right), value);
        }
}
```

Graphs - Directed

- Examples
  - List of web pages that link to each other
- Nodes can
  - have an arbitrary number of out arcs
  - have an arbitrary number of inbound arcs
- Declarations

```
typedef struct ARC {
    struct NODE *nodePtr;
    struct ARC *next;
} Arc;
typedef struct NODE {
    Arc *outArcs;
    int value;
} Node;
```

Graphs



Adding Node To a Graph

```
Node *AddNode(int value) {
    Node *new;
    new = (Node *) malloc(sizeof(Node));
    new->value = value;
    new->outArcs = NULL;
}
```

Adding Items To a Graph

```
int AddArc(Node *from, Node *to) {
        Arc *new, *prev, *current;
        prev = NULL;
        for (current=from->outArcs; current; current=current->next) {
            if (current->nodePtr == to) return 1;  /* already defined this arc */
            prev = current;
        }
        new = (Arc *) malloc(sizeof(Arc));
        if (!new) return -1;
        new->nodePtr = to;
        new->next = NULL;
        if (!prev)  {
              from->outArcs = new;  /* first out arch for this node */
        } else {
             prev->next = new;
        }
    }
```