C++ Linked lists

- Review:
  - Intro
  - Types: Node struct
  - Traversal: printList
- Some insertion and deletion operations
- Time permitting: finish **`sharedObj.cpp`** an **`copiedObj.cpp`** example from Thur lect (slide 19)

`testlist1.cpp`

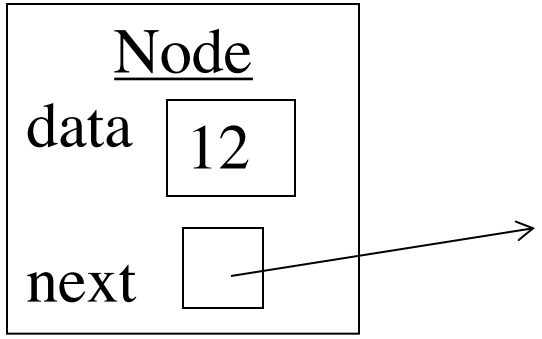
available after class

Announcements

- PA 5 due in about a week+
- No lab assignment this week
- Thur 4/22 Wellness Day:
  - no 4/22 lecture
  - no 4/22 office hours
  - no Th/Fr lab meetings
- Sample final exams have been posted
- Final Exam Review session:
  Mon, May 10, 2 – 4pm

Node with constructors

```
struct Node {
   int data;
   Node *next;
   Node (int item);
   Node (int item, Node *n);
};
Node::Node(int item) {
   data = item;
   next = NULL;
}
Node::Node(int item, Node *n) {
   data = item;
   next = n;
}
```



Example calls:

```
Node *p = new Node(3);
Node *q = new Node(5, p);
Node *r = new Node(12, NULL);
```

Linked list types (cont)

- The type for a linked list variable itself will be **`Node *`**
- Can make a type for this for clarity:

```
typedef Node* ListType;
```

- Some examples:

```
ListType list;
list = NULL;  // create a valid empty list


insertFront(list, 3);


insertFront(list, 7);


insertFront(list, 4);
```

Traversing a linked list

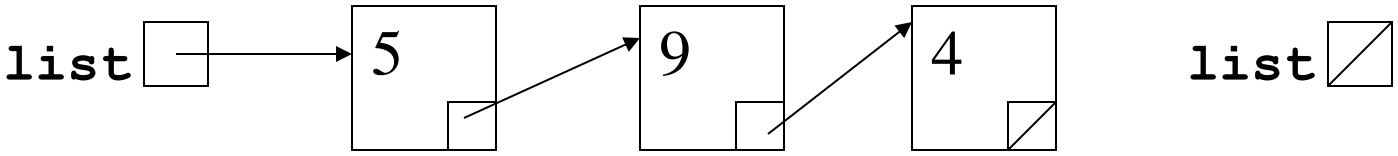
```
void printList(ListType list) {
  Node *p = list;
  while (p != NULL) {
    cout << p->data << " ";
    p = p->next;
  }
  cout << endl;
}
```

What is a well-formed linked list?

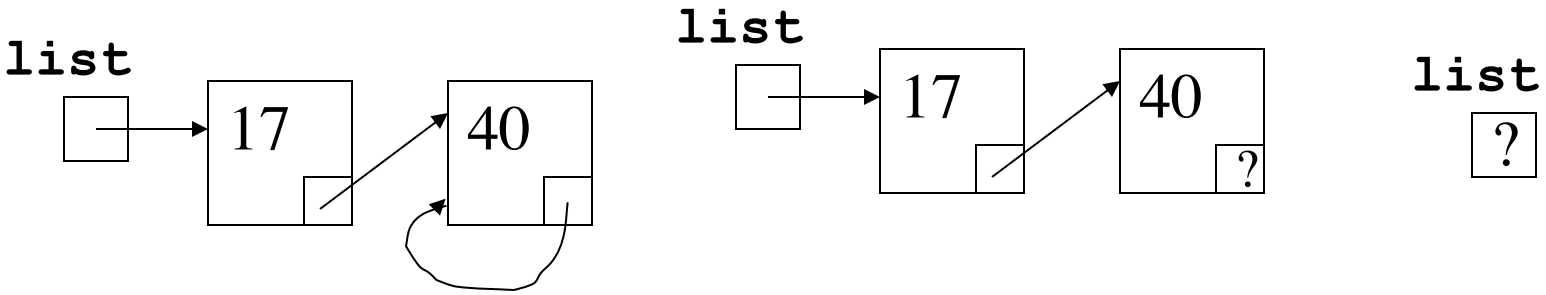
- Precondition for linked list routines:

```
PRE: list must be a well-formed list
void printList(Node *list);
```

- Well-formed examples:



- Not well-formed examples:



Passing linked lists as parameters

- A function that does not change the list:

```
void printList(ListType list);
```

- A function that does change the list:

```
void insertFront(
```

- Example of its use:

```
ListType myList = NULL;
insertFront(myList, 3);
printList(myList);
```

Some additional list operations

Let's now write code to:

- print last element
- insert element in front
- remove first element
- remove last element

```
struct Node {
  int data;
  Node *next;
  Node (int item);
  Node (int item, Node *n);
};

typedef Node * ListType;
```

```
void printLast(ListType list)
```

```
void insertFront(Node * & list, int val)
```

```
// pre: list is not empty
void removeFront(ListType & list) {
```

```
// PRE: list != NULL
void removeLast(ListType & list)
```

```
// Return updated value instead of
// pass by ref.  Sample call:
//  myList = removeLast(myList);
ListType removeLast(ListType list)
```