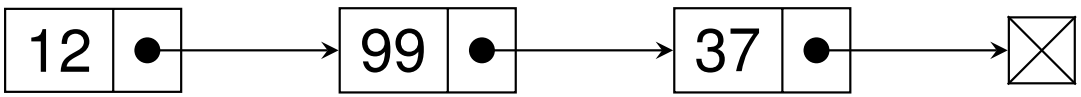
Lab 03: Linked Lists

COP 3503

October 13, 2014

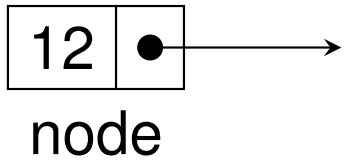
Definition

- Linked lists are a popular and simple data structure
- They are made up of a bunch of *nodes* linked together
- Together these nodes represent a sequence of some sort of *data*



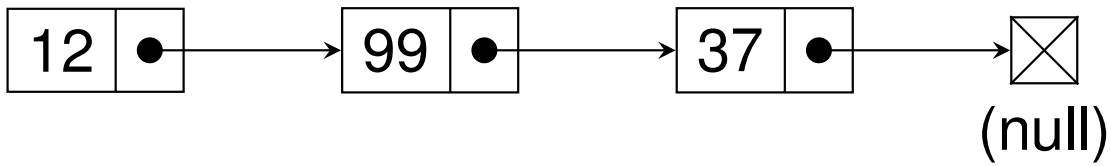
The Node Structure

- A node is made up of some value and a pointer to the next node in the list. The data can be a primitive type or even a pointer to another class!



The above node has a value of 12 and presumably points to another node.

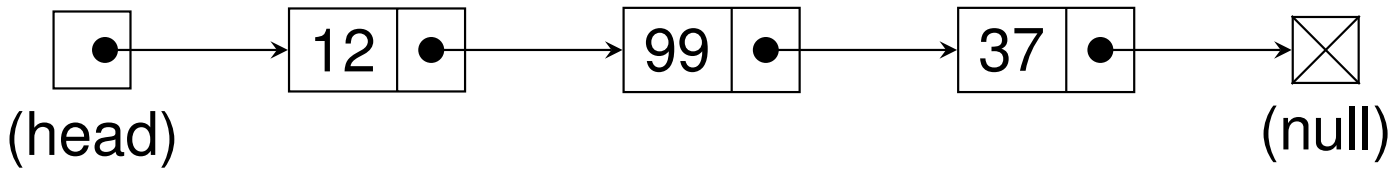
- If no node follows a given node, we point that node to *NULL* or *nullptr*. This usually signfies the end of the list



The above list starts with 12 and ends with 37

The List Structure

- The only thing we need to store for a list is a pointer to the first node
- Once we have this, the idea is to follow the next pointers until you hit the NULL pointer, in which case you know the list is over
- This is typically known as the head of the list



In Code

- We usually break the linked list into two classes: Node and LinkedList.

```
1  class Node{
2    private:
3      typename data; // Typename is any data type
4      Node * next; // Notice it is a pointer!
5    public:
6      // Methods and Stuff
7  }
8
9  class LinkedList{
10   private:
11     Node * head; // Again, we have a node pointer!
12   public:
13     // Methods and stuff
14 }
```

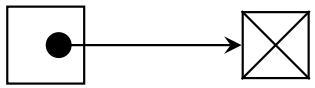
Some Common Methods

- Most methods with linked lists can be done either recursively or iteratively.
- Common methods are:
  - Insert - Add a value to the list
  - Remove - Remove a value from the list
  - Traverse - Traverse the list (visit every single node), doing something useful at each node
  - Member - Check to see if a given value is in the list
  - Split - Split a given list into two seperate lists

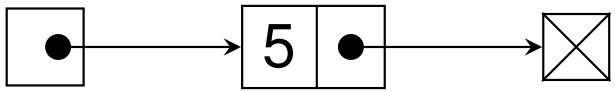
Insert

At a high level, inserting a value just involves finding the end of a list and pointing it to a new node:

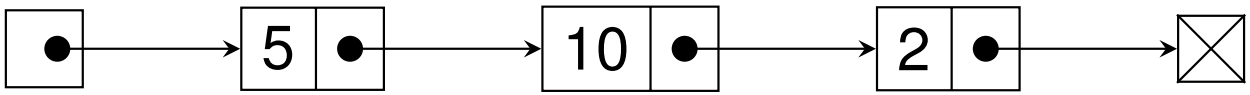
```
LinkedList * ll = new LinkedList();
```



```
ll.insert(5);
```




```
ll.insert(10);
ll.insert(2);
```



Insert Code: Two Ways

Recursive:

```
1  void LinkedList::insert(int value){
2    if(head == NULL) {  // List is empty
3      head = new Node(value);
4      return;
5    }
6    head->insert(value); // recurse starting with head
7  }
8
9  void Node::insert(int value){
10   if(next == NULL) { // Base case: value goes after me!
11     next = new Node(value);
12     return;
13   }
14   next->insert(value); // I'm not the end of the list.
15                        // Keep going
16 }
```

Insert Code: Two Ways

Iterative:

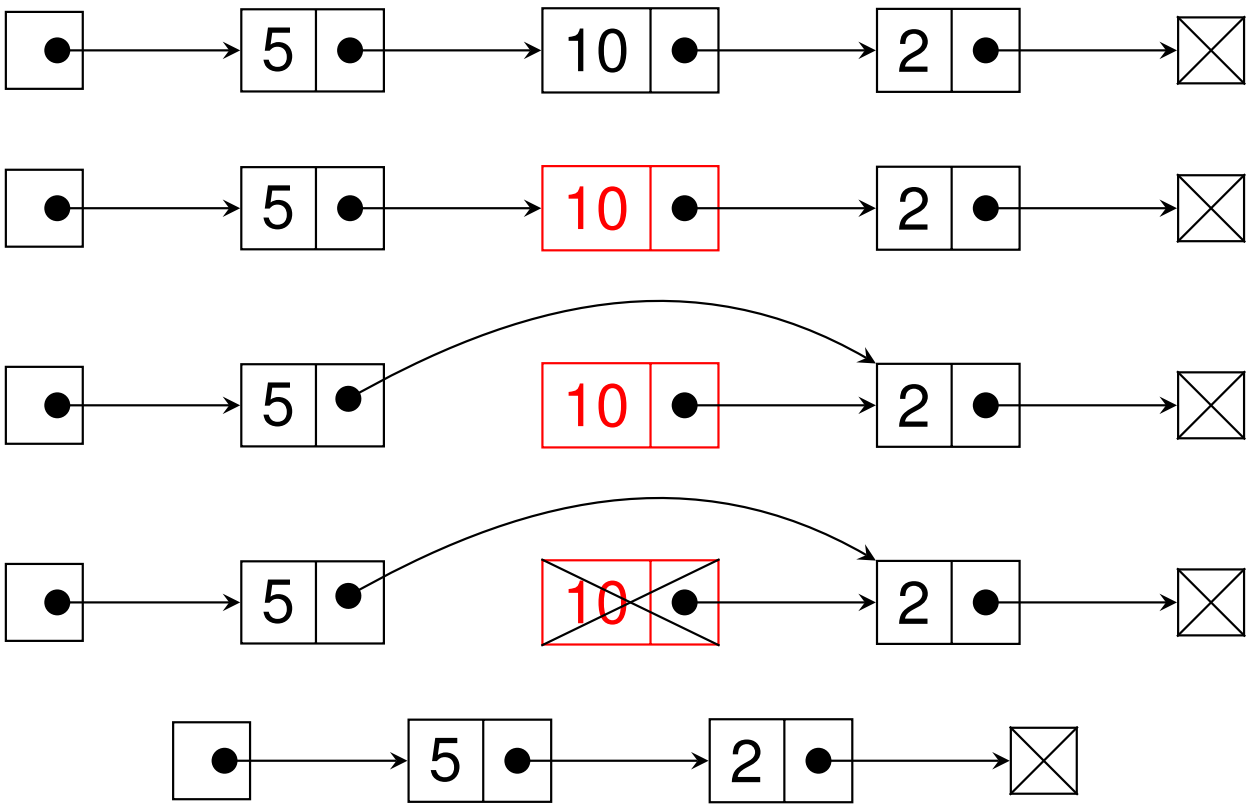
```
1  void LinkedList::insert(int value){
2    if(head == NULL) {  // List is empty
3      head = new Node(value);
4      return;
5    }
6    Node * temp = head;
7    while(temp->next != NULL) { // While we aren't at end
8      temp = temp->next;
9    }
10   // temp now points at the final node in the list
11   temp->next = new Node(value);
12 }
```

Remove Overview

- Base case: Head is null. There is nothing to remove, so the function ends!
- Find the node you want to remove (either iteratively or recursively).
- Once found, you point the node's predecessor's pointer to the node's successor (i.e. previous->next = current->next).
- Then, you must *delete* the node you removed!

Example

```
ll->remove(10);
```



Command Line Arguments

- Command line arguments are a way to pass information to your program when you run it about how it should operate.

```
g++ -c list.cpp -o list.o
```

- g++ is a program, just like any that you write
- The other things on the line are arguments telling g++ how it should run
  - -c means just compile the source code into an object file
  - -o tells g++ what file to stick the result of its compilation in
  - The other two (list.cpp and list.o) are filenames passed to the program
- In your project, you need to take in an argument that can either be best or worst

Command Line in C++

- In C++, we use paramters passed to main to get command line arguemnts

```
1 int main(int argc, char ** argv){
2   for(int i=0; i < argc; i++)
3     cout << "argv[" << i << "] = " << argv[i] << endl;
4   return 0;
5 }
```

- argc is the number of command line arguments passed to the program
- argv is an array of strings that contains each argument
- Note that the name of the executable is the first string passed in the program, so argc is always >= 1

Example Runs

```
1 int main(int argc, char ** argv){
2   for(int i=0; i < argc; i++)
3     cout << "argv[" << i << "] = " << argv[i] << endl;
4   return 0;
5 }
```

```
> ./a.out one two three
argv[0] = ./a.out
argv[1] = one
argv[2] = two
argv[3] = three
> ./a.out worst
argv[0] = ./a.out
argv[1] = worst
```

Other Things

- The strings passed to your program are called cstrings. This is different than the string type in C++
- When using cstrings, you need to use functions like strcmp (for comparison), strlen (for length of a string), and strcat (to concatenate strings). These can all be found in the library <cstring>
- It is important that you check the arguments passed to your program are valid arguments. In this case, that means that there is only one and it is either best or worst

Exercises

Implement a linked list class with the methods we discussed during the lab:

- Insertion
- Removal
- Traversal (print the list out)
- Search

Questions

???