CMPSC 24: Lecture 4
Implementing Linked Lists in C++

Divyakant Agrawal
Department of Computer Science
UC Santa Barbara

LINKED LISTS

Linked List Using Pointers

struct NodeType {
    int info;
    NodeType* next;
} 

NodeType* list;
NodeType* location;
Be sure you understand the differences among location, *location, and location->info.

Linked List Operations

- Set list to empty
- Add a new node to list
- Count the number of nodes in list
- Remove the last node from list

Structure Definition

```cpp
struct Node {
    int data;
    Node* next;

    Node (int item);
    Node (int item, Node* nextNode);
}

Node::Node(int item): data(item), next(0) {
}

Node::Node(int item, Node* nextNode):
    data(item), next(nextNode) {
}
```
Declaring the Linked List

Node* MyList;

Finding an Element in the List

bool find(int key)
{
    bool found=false;
    Node* p = MyList;
    // traverse the list to find the item
    // DEVELOP THE CODE IN THE CLASS
}

Inserting an Element at the End of the list

void insert(int key)
{
    Node* newNode = new Node(key);
    // lets keep things simple
    // add to the end of the list
    // traverse the list to get to the last node in the list
    // What about boundary conditions?
    // DEVELOP THE CODE IN THE CLASS
}
Deleting an Element from the list

```cpp
void delete(int key) {
    // traverse the list to find the node that matches "key"
    // we need to remove this node from the list
    // why is this more complex?
    // break down the complexity of the procedure into different cases
    // what are the different cases?
    // develop the implementation in class
}
```

Variation of the implementation

• Assuming that insert/append is the common case – can we make it more efficient?

• What if we need to insert an element in the middle?
Example linkedlist.h

```cpp
#include <iostream> // compiler directives
using namespace std; // compiler directives

Class linkedlist{
private:
struct node{
    int data;
    node *link;
} *list;
public:
    linkedlist(); // Constructor
    void append(int num); // append a node with "num"
    void delete(int num); // delete the node containing "num"
    bool find(int num); // TRUE if "num" exists in the list
    ~linkedlist(); // Destructor (to ensure no memory leak)
};
```

Skeleton of linkedlist.cpp

```cpp
linkedlist::linkedlist() {
    p = NULL;
}

void linkedlist::append(int num) {
    // the code we developed goes in here
    ...
}

void linkedlist::delete(int num) {
    // the code we developed goes in here
}

bool linkedlist::find(int num) {
    // the code we developed goes in here
    ...
}

Linkedlist::~linkedlist() {
    // ask the students what should happen here?
    ...
}
```
int main()
{
    linkedlist myList;
    myList.append(35);
    myList.append(45);
    myList.append(37);
    if (myList.find(36)) cout << "36 on the list???" << endl;
    else cout << "36 not on the list!!!" << endl;

    // you should get the idea...
    // would it be nice to have myList.display() ??? Left as an exercise!!!
}