CSC340: (Singly) Linked Lists

Main topics:

- Basics of linked lists
- Implementing a linked list as a C++ class.

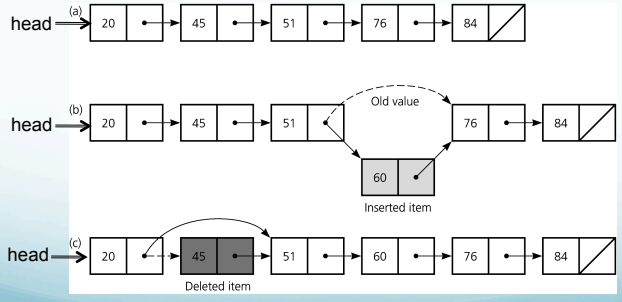
Readings:

- 5th edition: Chapter 4.2
- 6th edition: Chapter 4

Hui Yang
Computer Science Department
San Francisco State University
http://www.cs.sfsu.edu/~huiyang/

Nodes and Linked Lists

- A linked list is a list that can grow and shrink while the program is running
- A linked list is constructed using pointers
- A linked list can be visualized as **nodes** (drawn as boxes) connected to other items by **pointers** (arrows)



Implementing Nodes

- **Nodes are implemented in C++ as structs or classes**
  - Example: A structure to store two data items and a pointer to another node of the same type:

```
struct Node
{
   string item;
   int count;
   Node *next;
};
```

This circular definition is allowed in C++

Head of a List and Node Allocation

- **The box labeled head, is not a node, but a pointer variable that points to a node.**
- **Pointer variable head is declared as:**

Node* head;

- **If head is NULL, the linked list is empty**
- **A node is dynamically allocated**

```
Node *p;       // pointer to node
p = new Node; // allocate node
```

The LinkedList class

- **Data members**

```
class LinkedList{
   private:
      struct node{
           Person pObj; //data members
           node *next;
       };
       node *head;

   public:
      …
};
```

- **Function members**
  - Constructors, accessors, mutators
  - Big-3

Linked List: Creating the First Node

- **The default constructor initializes the list to be empty.**
- **To create the first node, the operator new is used to create a new dynamic variable:**

```
head = new Node;
```

  - Now head points to the first and only node in the list

Inserting a Node to a Specific Position

- **Insert to the beginning of a list**

```
newPtr->next = head;
Head = newPtr;
```

- **To insert a node between two nodes**

```
newPtr->next = cur;
prev->next = newPtr;
```

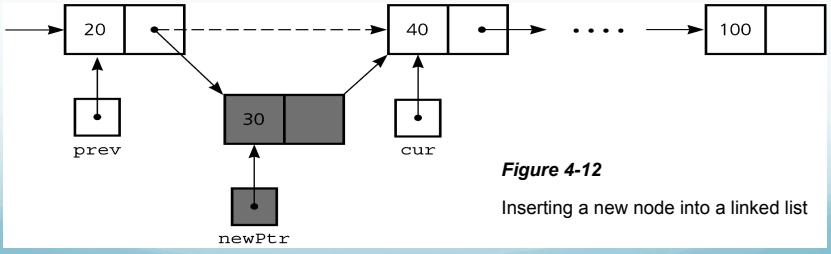


Figure 4-12

Inserting a new node into a linked list

7

Losing Nodes: 2 Common Mistakes

- **Mistake No. 1**
  - You might be tempted to use the head pointer to construct the new node:

```
head = new Node;
head->data = the_number;
```

  - Now to attach the new node to the list
    - The node that head used to point to is now lost!
- **Mistake No. 2**
  - To delete a linked list using the following statement

```
head= NULL;
```

  - The correct way: delete the nodes on the list one by one

8

Insertion: efficiency

- **Notice that inserting into a linked list requires the change of two pointers**
  - This is true regardless of the length of the list
  - Using an array for the list would involve copying as many as all of the array elements to new locations to make room for the new item
- **Inserting into a linked list is often more efficient than inserting into an array**

Searching a Linked List

- **Locate a particular node in a linked list:**
  - We will use a local pointer variable, named here, to move through the list checking for the target
    - The only way to move around a linked list is to follow pointers
  - We will start with here pointing to the first node and move the pointer from node to node following the pointer out of each node

Pseudocode for search

- Declare a pointer variable curr (a cursor)

```
point curr to the first node on the list;
while (the pointee of curr is not the last node)
                                        //curr->next!=NULL
{
   If (curr's pointee contains the target)
       copy the node and return true;

   make curr point to the next node //curr = curr->next;
}

return false; //target doesn't exist
```


Displaying the Contents of a Linked List

- **A traverse operation visits each node in the linked list**
  - A pointer variable `curr` keeps track of the current node

```
for (Node *cur = head; cur != NULL;
     cur = cur->next)
   cout << cur->item << endl;
```

Deleting a Specified Node

- Deleting an interior node
  - prev->next = cur->next;
  - delete cur;
  - cur = NULL;

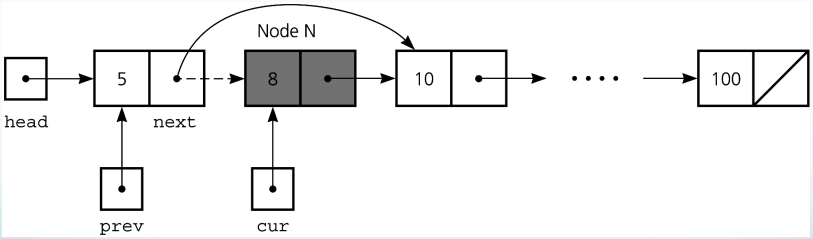


Figure 4-10 Deleting a node from a linked list

Deleting the First Node

- Syntax

```
head = head->next;
delete cur;
```

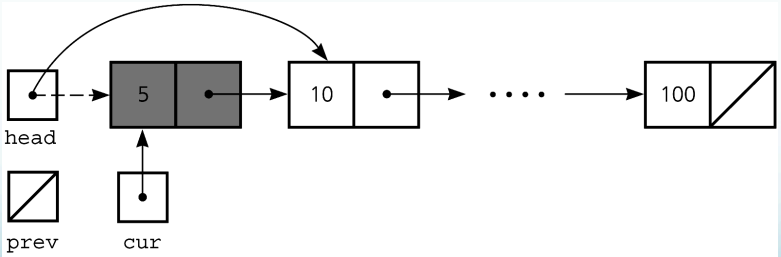


Figure 4-11 Deleting the first node

Big-three

- **Destructor**
  - Delete the entire list one node at a time
- **Copy constructor**
  - Need to allocate space one node at a time to construct a copy of the linked list passed in by parameter
- **Overloaded assignment operator**
  - Without properly overloading the assignment operator

    head2 = head1;

    causes head2 and head1 to point to the same list.

Variations on Linked Lists

- **Many other data structures can be constructed using nodes and pointers**
- **Doubly-linked List**
  - Each node has two links, one to the next node and one to the previous node
  - Allows easy traversal of the list in both directions

```
class Node {
   public:
            int data;
            Node *forward_link;
            Node *back_link;
            Node();
};
```

Variations: Doubly Linked Lists

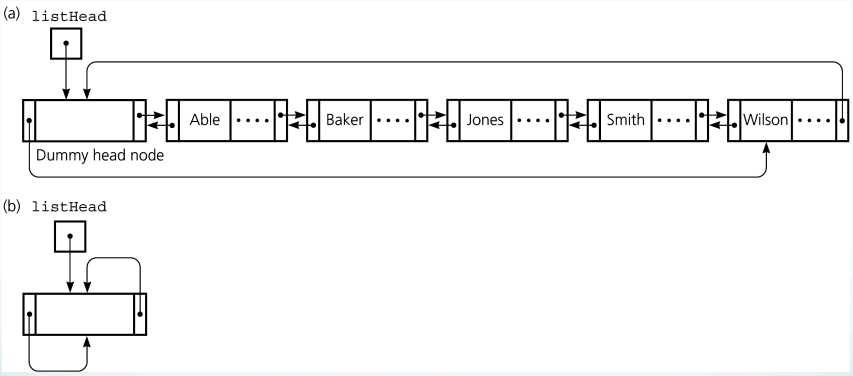


Figure 4-29 (a) A circular doubly linked list with a dummy head node
(b) An empty list with a dummy head node

17

Variations: Circular Linked Lists

- Last node points to the first node
- Every node has a successor
- No node in a circular linked list contains NULL

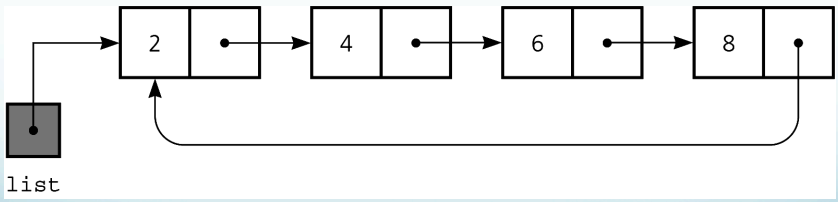


Figure 4-25 A circular linked list

18

Binary Tree

- **A tree is a data structure that looks like an upside-down tree with the root at the top**
  - No cycles
- **In a binary tree each node has at most two links**

```
struct TreeNode
{
  int data;
  TreeNode *left_link;
  TreeNode *right_link;
};
```


Summary

- **Linked lists: basic concepts**
- **Manage a linked list**
  - Create a list
  - Insert a node to a list
  - Search a node on a list
  - Traverse a list
  - Delete a node from a list
  - Destroy a list

Quiz

1. Inserting a node to a linked list is more efficient than inserting a node to an array.
2. One can delete a linked list by simply setting the head pointer to NULL.
3. When operating on a linked list, one should always remember to keep the head pointer pointing to the first node.
4. Even though a linked list is sorted, performing sequential search will be more efficient than binary search.