ICS 2105
Data Structures and Algorithms

Linked Lists

Acknowledgement

- Notes adapted from:
  - Data Structures Using C++, Second Edition By D.S. Malik
  - tutorialspoint.com
  - studytonight.com

Learning Outcomes

- By the end of this chapter, the learner should be able to:
  - Define a linked list.
  - Describe operations of a linked list.
  - Implement linked list using C++.
  - Describe applications of linked lists.

Linked Lists

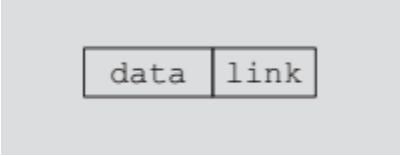
- When data is stored in an array, memory for the components of the array is contiguous i.e. the blocks are allocated one after the other.
- A list of items, called nodes, in which the order of the nodes is determined by the address, called the link, stored in each node.

Linked Lists

- A linked list is a collection of components, called nodes.
  - Every node (except the last node) contains the address of the next node.
- Thus, every node in a linked list has two components:
  - One to store the relevant information, i.e. data.
  - One to store the address, called the link, of the next node in the list.

Linked Lists

- The address of the first node in the list is stored in a separate location, called the head or first.



Linked Lists

- Each node holds its own data and the address of the next node hence forming a chain like structure.
- Linked Lists are used to create trees and graphs.

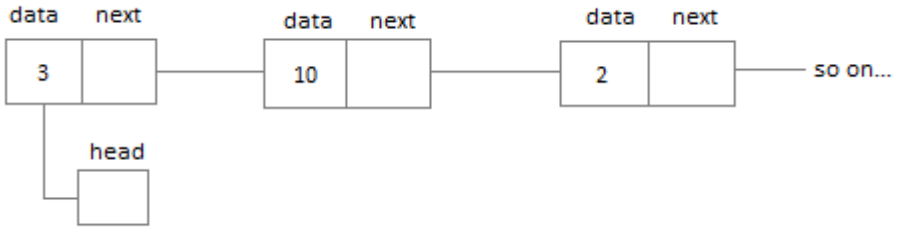
Types of Linked Lists

- There are 3 different implementations of Linked List:
  - Singly Linked List
  - Doubly Linked List
  - Circular Linked List

Singly Linked List

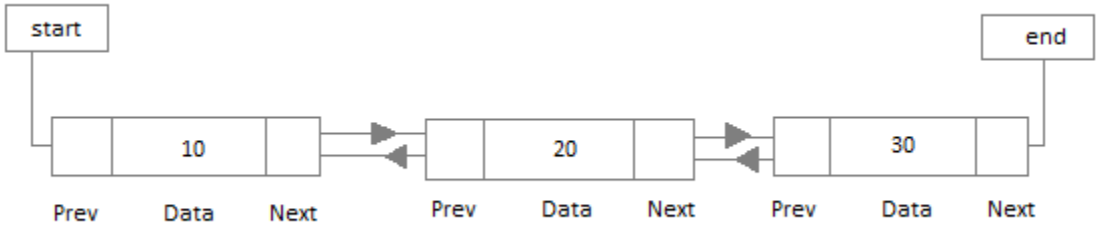
- Singly linked lists contain nodes which have a data part as well as an address part i.e. next, which points to the next node in the sequence of nodes.
- The operations we can perform on singly linked lists are insertion, deletion and traversal.

Singly Linked List



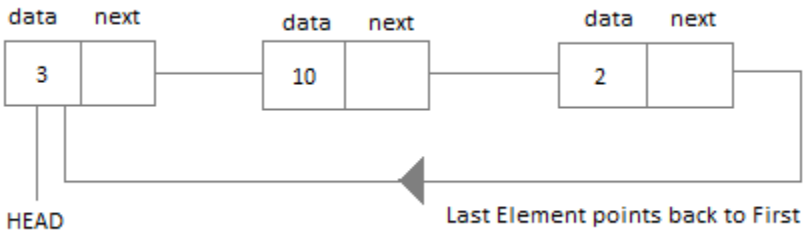
Doubly Linked List

- In a doubly linked list, each node contains a **data** part and two addresses, one for the **previous** node and one for the **next** node.



Circular Linked List

- In circular linked list the last node of the list holds the address of the first node hence forming a circular chain.

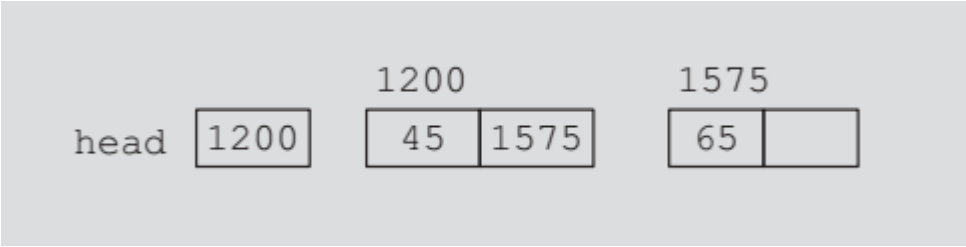


Example of a Linked List

- The arrow in each node indicates that the address of the node to which it is pointing is stored in that node.
- The down arrow in the last node indicates that this link field is NULL.

Linked List and Values of The Links

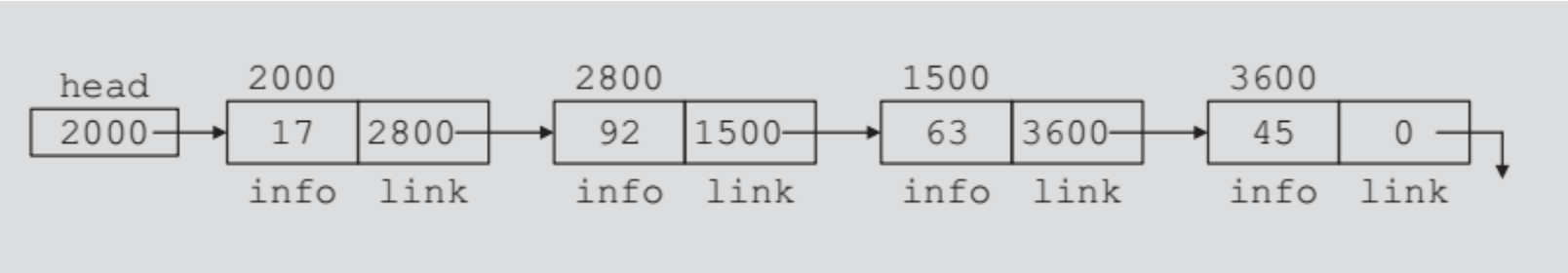
- For a better understanding of this notation, suppose that the first node is at memory location 1200, and the second node is at memory location 1575.



Linked List and Values of The Links

- The value of the head is 1200, the data part of the first node is 45, and the link component of the first node contains 1575, the address of the second node.

Linked list with four nodes



- Suppose that the first node is at location 2000, the second node is at location 2800, the third node is at location 1500, and the fourth node is at location 3600.
- The table on the next slide shows the values of head and some other nodes.

Linked list with four nodes

	Value	Explanation
`head`	2000	
`head->info`	17	Because **`head`** is 2000 and the **`info`** of the node at location 2000 is 17
`head->link`	2800	
`head->link->info`	92	Because **`head->link`** is 2800 and the **`info`** of the node at location 2800 is 92

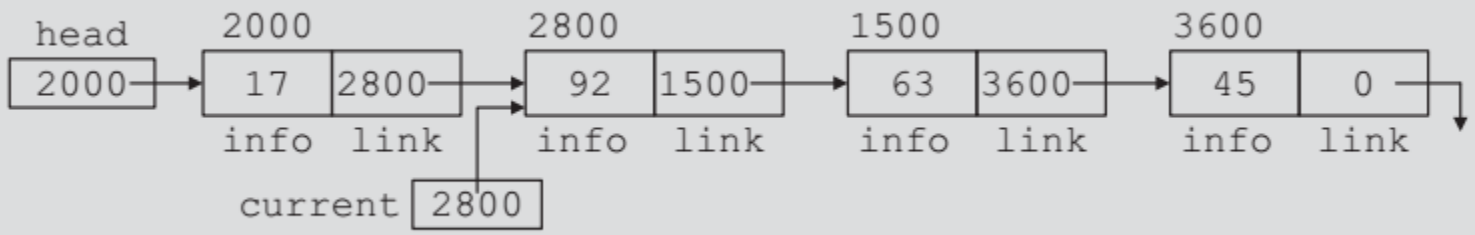
Linked list with four nodes

- Suppose that current is a pointer of the same type as the pointer head.
- Then the statement current = head; copies the value of head into current.

Linked list with four nodes

- The statement current = current->link; copies the value of current->link, which is 2800, into current.
- Therefore, after this statement executes, current points to the second node in the list.
  - When working with linked lists, these types of statements are used to advance a pointer to the next node in the list

Linked Lists



	Value
current	2800
current->info	92
current->link	1500
current->link->info	63
head->link->link	1500
head->link->link->info	63
head->link->link->link	3600
current->link->link->link	0 (that is, NULL)
current->link->link->link->info	Does not exist (run-time error)

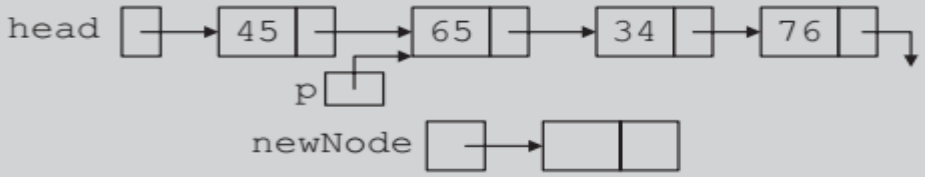
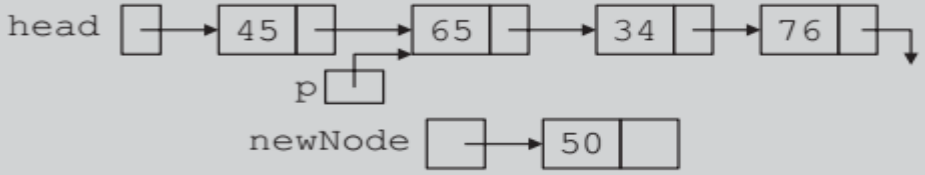
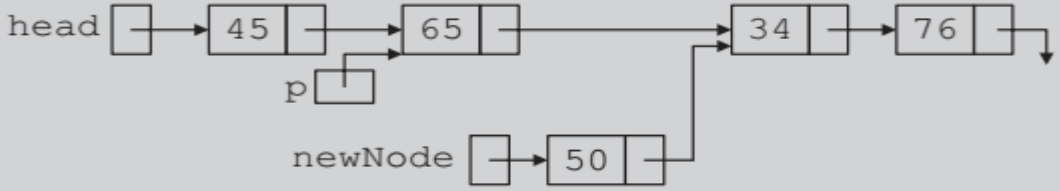
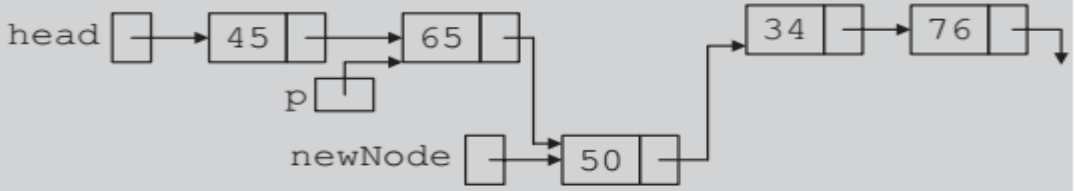
Linked Lists Operations

- Inserting an element.
- Checking whether Linked List is empty or not.
- Searching any element in the Linked List
- Deleting a particular Node from the List


```
class Node {
 public:
 int data;
 //pointer to the next node
 node* next;
  node() {
  data = 0;
  next = NULL;
 }
  node(int x) {
  data = x;
  next = NULL;
 }
} ;
```

Before inserting the node in the list we need to create a class **Node**

Inserting a node in a linked list

Statement	Effect
`newNode = new nodeType;`	
`newNode->info = 50;`	
`newNode->link = p->link;`	
`p->link = newNode;`	

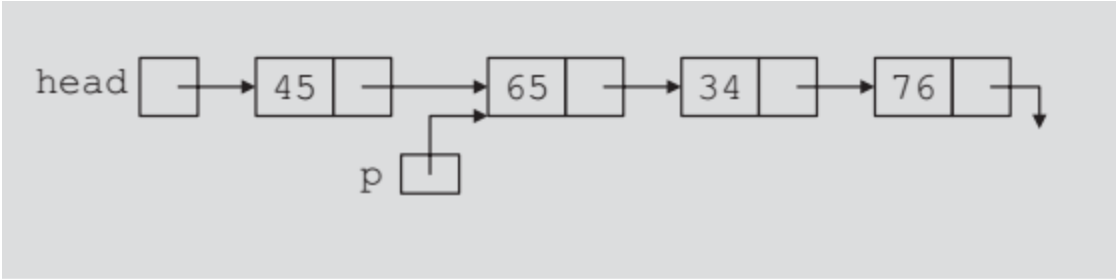
Inserting

- The sequence of statements to insert the node,

```
newNode->link = p->link;
p->link = newNode;
```

is very important because to insert newNode in the list we use only one pointer, p, to adjust the links of the nodes of the linked list.

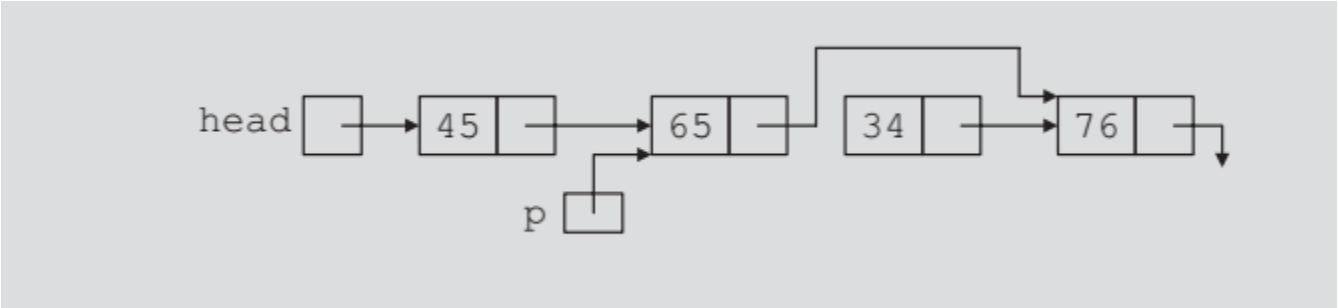
Deletion



- Suppose that the node with info 34 is to be deleted from the list.
- The following statement removes the node from the list:

```
p->link = p->link->link;
```

Deletion



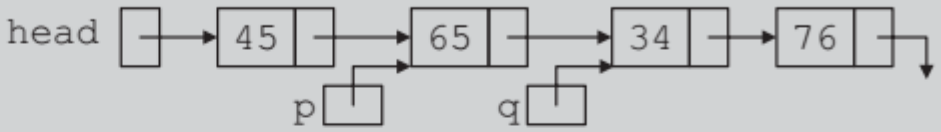
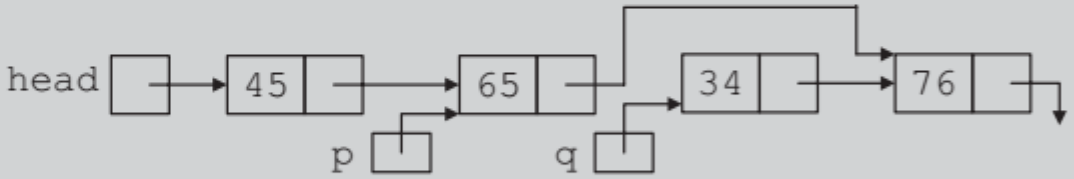
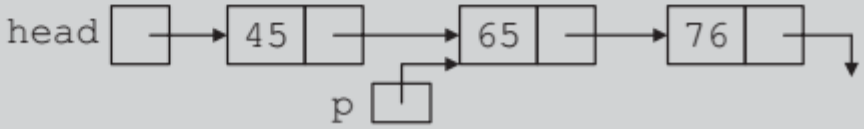
- Resulting List after The Preceding Statement Executes
- The node with info 34 is removed from the list.
- However, the memory is still occupied by this node and this memory is inaccessible.

Deletion

- To deallocate the memory, we need a pointer to this node.
- The following statements delete the node from the list and deallocate the memory occupied by this node:

```
q = p->link;
p->link = q->link;
delete q;
```

Deleting a node from a linked list

Statement	Effect
`q = p->link;`	
`p->link = q->link;`	
`delete q;`	

Advantages of Linked Lists

- They are a dynamic in nature which allocates the memory when required.
- Insertion and deletion operations can be easily implemented.
- Stacks and queues can be easily executed.
- Linked List reduces the access time

Disadvantages of Linked Lists

- The memory is wasted as pointers require extra memory for storage.
- No element can be accessed randomly; it has to access each node sequentially.
- Reverse Traversing is difficult in linked list.

Applications of Linked Lists

- Linked lists are used to implement stacks, queues, graphs, etc.
- Linked lists let you insert elements at the beginning and end of the list.
- In Linked Lists we don't need to know the size in advance.

End of lesson