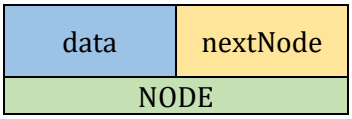
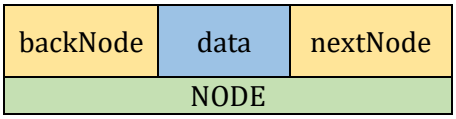
Linked List

A linked list is a list of items where one item of the list points to another item. A Node is used to represent a linked list. A Node is a user defined data type that binds two or more variables together and makes a single unit of variable. An illustration of a node is given below:



(a) Node for Single Linked List

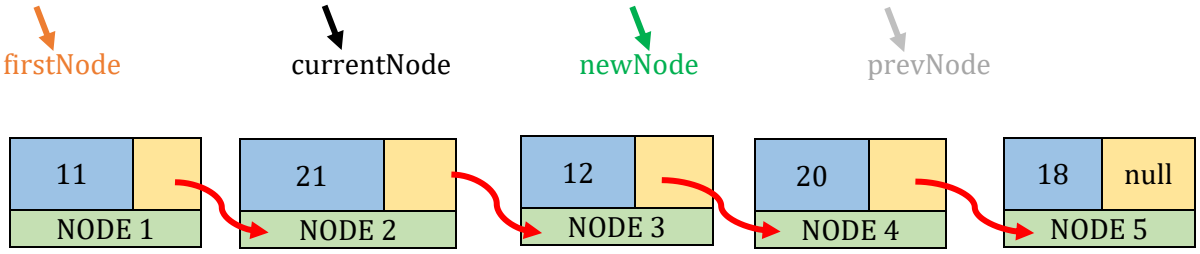


(b) Node for Doubly Linked List

We can use a structure to create a node:

(a) Node for Single Linked List	(b) Node for Doubly Linked List

struct Node
{
int data;
Node *nextNode;
};
	struct Node
	{
	int data;
	Node *nextNode;
	Node *backNode;
	};



a. Creating a Single linked List:

Input and Initializations: int n, struct Node, Node *firstNode, Node *newNode, Node *currentNode.

Process:

1. Create a Node (***newNode***).
2. Input the value of ***data*** for the ***newNode.*** The value of ***nextNode*** for the ***newNode*** will be null.
3. If, the value of ***firstNode*** is null, go to (4). Else, go to (5).
4. The value of ***firstNode*** will be ***newNode*** also, the value of ***currentNode*** will be ***newNode.*** Go to (6).
5. The value of ***nextNode*** for the ***currentNode*** will be ***newNode***. The value of ***currentNode*** will be ***newNode.*** Go to (6).
6. Repeat (1), (2), (3), (4)/(5) for ***n*** times and a linked list with **n** nodes has already been created***.***

Output: The data of all the nodes.

b. Printing a Single Linked List:

1. The value of ***currentNode*** will be ***firstNode***.
2. Print the data of ***currentNode***.
3. The value of ***currentNode*** will be the ***nextNode*** of ***currentNode***.
4. Repeat (2), (3) while the ***currentNode*** is not null.

c. Search an element from a Single Linked List:

Input and Initializations: A linked list, int ***element***, bool ***flag =*** *false*.

Process:

1. The value of ***currentNode*** will be ***firstNode***.
2. If, the value of ***data*** for ***currentNode*** is ***element,*** go to (3), else go to (4).
3. The value of ***flag*** will be *true.* Exit.
4. The value of ***currentNode*** will be the ***nextNode*** of ***currentNode***.
5. While the ***currentNode*** is not NULL, repeat (2), (3) and (4).

Output: If the value of ***flag*** is *true,* print "Found", else print "Not Found".

d. Find the smallest element from a Single Linked List:

Input and Initializations: A linked list, int ***mini = 99999999***.

Process:

1. The value of ***currentNode*** will be ***firstNode***.
2. If, the value of ***data*** for ***currentNode*** is less than ***mini,*** go to (3), else go to (4).
3. The value of ***mini*** will be the value of ***data*** for ***currentNode.***
4. The value of ***currentNode*** will be the ***nextNode*** of ***currentNode***.
5. While the ***currentNode*** is not NULL, repeat (2), (3) and (4).

Output: The value of ***mini***.

e. Insert a node at the first position of a Single Linked List:

Input and Initializations: A Linked List.
Process:

1. Create a new Node (***newNode***).
2. Input the value of ***data*** for the ***newNode***. The value of ***nextNode*** will be NULL.
3. The value of ***nextNode*** for the ***newNode*** will be ***firstNode.***
4. The value of ***firstNode*** will be ***newNode.***

Output: The data of all the nodes.

f. Insert a node at the last position of a Single Linked List:

1. Create a new Node (***newNode***).
2. Input the value of ***data*** for the ***newNode***. The value of ***nextNode*** will be NULL.
3. The value of ***currentNode*** will be the ***firstNode.***
4. If the value of ***nextNode*** for ***currentNode*** is not null, go to (5), else go to (7).
5. The value of ***currentNode*** will be ***nextNode*** of ***currentNode.***
6. Repeat (4) and (5).
7. The value of ***nextNode*** for ***currentNode*** will be ***newNode***. The value of ***currentNode*** will be ***newNode***.

g. Insert a node somewhere in the middle of a Single Linked List:

1. Enter the element (***prevElement***) after which the node will be inserted.
2. Search ***prevElement*** from the list. If it is found, go to (3), else Exit.
3. Create a new Node (***newNode***).
4. Input the value of ***data*** for the ***newNode***. The value of ***nextNode*** will be NULL.
5. The value of ***nextNode*** for ***newNode*** will be the ***nextNode*** of ***currentNode***.
6. The value of ***nextNode*** for ***currentNode*** will be the ***newNode.***

h. Delete the first node of a Single Linked List:

1. The value of ***currentNode*** will be ***firstNode***.
2. The value of ***firstNode*** will be the ***nextNode*** of ***currentNode***.
3. Delete ***currentNode***.

i. Delete the Last node of a Single Linked List:

1. The value of ***currentNode*** will be ***firstNode***.
2. If the value of ***nextNode*** for ***currentNode*** is not null, go to (3). Else go to (5).
3. The value of ***prevNode*** will be ***currentNode***. The value of ***currentNode*** will be the ***nextNode*** of ***currentNode***.
4. Repeat (2) and (3).
5. The value of ***nextNode*** of ***prevNode*** will be null.
6. Delete ***currentNode***.

j. Delete Node from the middle of a Single Linked List:

1. Enter the element (***element***) for the node which will be deleted.
2. Search the ***element*** from the list. If found, go to (3). Else, go to (10).
3. The value of ***currentNode*** will be ***firstNode***.
4. If the ***data*** of ***currentNode*** is not ***element,*** go to (5), else go to (8).
5. The value of ***prevNode*** will be the value of ***currentNode***.
6. The value of ***currentNode*** will be the ***nextNode*** of ***currentNode***.
7. Repeat (4), (5) and (6).
8. The value of ***nextNode*** for ***prevNode*** will be the value of ***nextNode*** of ***currentNode***.
9. Delete the ***currentNode***. Exit.
10. Print “Not Found and cannot be removed.”

k. Creating a Doubly linked List:

Input and Initializations: int n, struct Node, Node *firstNode, Node *newNode, Node *currentNode.

Process:

1. Create a Node (***newNode***).
2. Input the value of ***data*** for the ***newNode.*** The value of ***nextNode*** for the ***newNode*** will be null, the value of ***backNode*** for ***newNode*** will be null.
3. If, the value of ***firstNode*** is null, go to (4). Else, go to (5).
4. The value of ***firstNode*** will be ***newNode*** also, the value of ***currentNode*** will be ***newNode.*** Go to (6).
5. The value of ***nextNode*** for the ***currentNode*** will be ***newNode***. The ***backNode*** for the ***newNode*** will be ***currentNode***. The value of ***currentNode*** will be ***newNode.*** Go to (6).
6. Repeat (1), (2), (3), (4)/(5) for ***n*** times and a linked list with **n** nodes has already been created***.***

Output: The data of all the nodes.

l. Creating a Circular Linked List:

Input and Initializations: int n, struct Node, Node *firstNode, Node *newNode, Node *currentNode.

Process:

1. Create a Node (***newNode***).
2. Input the value of ***data*** for the ***newNode.*** The value of ***nextNode*** for the ***newNode*** will be null.
3. If, the value of ***firstNode*** is null, go to (4). Else, go to (5).
4. The value of ***firstNode*** will be ***newNode*** also, the value of ***currentNode*** will be ***newNode.*** Go to (6).
5. The value of ***nextNode*** for the ***currentNode*** will be ***newNode***. The ***nextNode*** for the ***newNode*** will be ***firstNode***. The value of ***currentNode*** will be ***newNode.*** Go to (6).
6. Repeat (1), (2), (3), (4)/(5) for ***n*** times and a linked list with **n** nodes has already been created***.***

Output: The data of all the nodes.