**Delete First Node  
Delete First ( ):  
Description:** Here START is a pointer variable which contains the address of first node. ITEM is the value to be deleted.

1. If (START == NULL) Then [Check whether list is empty]  
2. Print: Linked-List is empty.  
3. Else  
4. PTR = START  
5. ITEM = START->INFO[Assign INFO of first node to ITEM]  
6. START == START->LINK[START now points to 2nd node]  
7. Delete PTR [Delete first node]  
8. Print: ITEM deleted  
    [End of If]  
9. Exit

**Delete Last Node  
Delete Last ( ):  
Description:**Here START is a pointer variable which contains the address of first node. PTR is a pointer variable which contains address of node to be deleted. PREV is a pointer variable which points to previous node. ITEM is the value to be deleted.

1. If (START == NULL) Then [Check whether list is empty]  
2. Print: Linked-List is empty.  
3. Else  
4. PTR = START, PREV = START  
5. Repeat While (PTR->LINK != NULL)  
6. PREV = PTR[Assign PTR to PREV]  
7. PTR = PTR->LINK [Move PTR to next node]  
    [End of While Loop]  
8. ITEM = PTR->INFO[Assign INFO of last node to ITEM]  
9. If (START->LINK == NULL) Then [If only one node is left]  
10. START = NULL [Assign NULL to START]  
11. Else  
9. PREV->LINK = NULL[Assign NULL to link field of second last node]  
     [End of Step 9 If]  
10. Delete PTR  
11. Print: ITEM deleted  
      [End of Step 1 If]  
12. Exit

**Delete Specific Node  
Delete Specific ( ):  
Description:**Here START is a pointer variable which contains the address of first node. PTR is a pointer variable which contains address of node to be deleted. PREV is a pointer variable which points to previous node. ITEM is the value to be deleted.

1. If (START == NULL) Then[Check whether list is empty]  
2. Print: Linked-List is empty.  
3. Else If (START->INFO == ITEM) Then[Check if ITEM is in 1st node]  
4. PTR = START  
5. START = START->LINK [START now points to 2nd node]  
6. Delete PTR  
7. Else  
8. PTR = START, PREV = START  
9. Repeat While (PTR != NULL)  
10. If (PTR->INFO == ITEM) Then[If ITEM matches with PTR->INFO]  
11. PREV->LINK = PTR->LINK[Assign LINK field of PTR to PREV]  
12. Delete PTR  
13. Else  
14 PREV = PTR [Assign PTR to PREV]  
15. PTR = PTR->LINK[Move PTR to next node]  
      [End of Step 10 If]  
      [End of While Loop]  
16. Print: ITEM deleted  
      [End of Step 1 If]  
17. Exit

**Insert First Node  
Insert First ( ):  
Description:**Here START is a pointer variable which contains the address of first node. ITEM is the value to be inserted.

1. If (START == NULL) Then  
2. START = New Node[Create a new node]  
3. START->INFO = ITEM[Assign ITEM to INFO field]  
4. START->LINK = NULL[Assign NULL to LINK field]  
5. Else  
6. Set PTR = START [Initialize PTR with START]  
7. START = New Node [Create a new node]  
8. START->INFO = ITEM [Assign ITEM to INFO field]  
9. START->LINK = PTR[Assign PTR to LINK field]  
    [End of If]  
10. Exit

**Insert Item in a Sorted List**  
**Insert Sorted ( ):  
Description:**Here START is a pointer variable which contains the address of first node. PREV is a pointer variable which contains address of previous node. ITEM is the value to be inserted.

1. If (START == NULL) Then [Check whether list is empty]  
2. START = New Node[Create a new node]  
3. START->INFO = ITEM [Assign ITEM to INFO field]  
4. START->LINK = NULL [Assign NULL to LINK field]  
5. Else  
6. If (ITEM < START->INFO) Then [Check whether ITEM is less then value in first node]  
7. PTR = START  
8. START = New Node  
9. START->INFO = ITEM  
10. START->LINK = PTR  
11. Else  
12. Set PTR = START, PREV = START  
13. Repeat While (PTR != NULL)  
14. If (ITEM < PTR->INFO) Then  
15. PREV->LINK = New Node  
16. PREV = PREV->LINK  
17. PREV->INFO = ITEM  
18. PREV->LINK = PTR  
19. Return  
20. Else If (PTR->LINK == NULL) Then [Check whether PTR reaches last node]  
21. PTR->LINK = New Node  
22. PTR = PTR->LINK  
23. PTR->INFO = ITEM  
24. PTR->LINK = NULL  
25. Return  
26. Else  
27. PREV = PTR  
28. PTR = PTR->LINK  
      [End of Step 14 If]  
      [End of While Loop]  
      [End of Step 6 If]  
      [End of Step 1 If]  
29. Exit

**Insert Last Node  
Insert Last ( ):  
Description:**Here START is a pointer variable which contains the address of first node. ITEM is the value to be inserted.

1. If (START == NULL) Then[Check whether list is empty]  
2. START == New Node  
3. START->INFO = ITEM  
4. START->LINK = NULL  
5. Else  
6. Set PTR = START[Initialize PTR with START]  
7. Repeat While (PTR->LINK != NULL)  
8. PTR = PTR->LINK[Until PTR reaches last node]  
    [End of While Loop]  
9. PTR->LINK = New Node[Assign address of new node to PTR]  
10. PTR = PTR->LINK [Move PTR to next node]  
11. PTR->INFO = ITEM  
12. PTR->LINK = NULL  
      [End of If]  
13. Exit

**Insert Specific Node  
Insert Specific ( ):  
Description:**Here START is a pointer variable which contains the address of first node. NEW is a pointer variable which will contain address of new node. N is the value after which new node is to be inserted and ITEM is the value to be inserted.

1. If (START == NULL) Then  
2. Print: Linked-List is empty. It must have at least one node  
3. Else  
4. Set PTR = START, NEW = START  
5. Repeat While (PTR != NULL)  
6. If (PTR->INFO == N) Then  
7. NEW = New Node  
8. NEW->INFO = ITEM  
9. NEW->LINK = PTR->LINK  
10. PTR->LINK = NEW  
11. Print: ITEM inserted  
12. ELSE  
13. PTR = PTR->LINK  
     [End of Step 6 If]  
     [End of While Loop]  
     [End of Step 1 If]  
14. Exit

**Reverse a Linked List**  **Reverse ( ):  
Description:** Here START is a pointer variable which contains the address of first node. PTR will point to the current node and PREV will point to the previous node. REV will maintain the reverse list.

1. Set PTR = START, PREV = NULL  
2. Repeat While (PTR != NULL)  
3. REV = PREV  
4. PREV = PTR  
5. PTR = PTR->LINK  
6. PREV->LINK = REV  
    [End of While Loop]  
7. START = PREV  
8. Exit

**Search a Sorted Linked List**  
**Search Sorted ( ):  
Description:**Here START is a pointer variable which contains the address of first node. ITEM is the value to be searched.

1. Set PTR = START, LOC = 1[Initialize PTR and LOC]  
2. Repeat While (PTR != NULL)  
3. If (ITEM > PTR->INFO) Then [Check if ITEM is greater then INFO field]  
4. PTR = PTR->LINK [Move PTR to next node]  
5. LOC = LOC + 1 [Increment LOC]  
6. Else If (ITEM == PTR->INFO) Then[Check if ITEM matches with INFO field]  
7. Print: ITEM is present at location LOC  
8. Return  
9. Else  
10. Print: ITEM is not present in the list  
11. Return  
     [End of If]  
     [End of While Loop]  
12. Exit

**Search Unsorted Linked List**  
**Search Unsorted ( ):  
Description:** Here START is a pointer variable which contains the address of first node. ITEM is the value to be searched.

1. Set PTR = START, LOC = 1 [Initialize PTR and LOC]  
2. Repeat While (PTR != NULL)  
3. If (ITEM == PTR->INFO) Then [Check if ITEM matches with INFO field]  
4. Print: ITEM is present at location LOC  
5. Return  
6. Else  
7. PTR = PTR->LINK[Move PTR to next node]  
8. LOC = LOC + 1[Increment LOC]  
    [End of If]  
    [End of While Loop]  
9. Print: ITEM is not present in the list  
10. Exit

**Traverse a Linked List  
Traverse ( ):  
Description:**Here START is a pointer variable which contains the address of first node. PROCESS is any operation that is to be performed on the node.

1. Set PTR = START  
2. Repeat While (PTR != NULL)  
3. Apply PROCESS to PTR->INFO  
4. PTR = PTR->LINK  
    [End of While Loop]  
5. Exit

[Next](http://joseblog.netau.net/Computer_Science/data_structures/queues.php)

**Category**

* *[Arrays](http://joseblog.netau.net/Computer_Science/data_structures/arrays.php)*
* *[Linked Lists](http://joseblog.netau.net/Computer_Science/data_structures/linked-lists.php)*
* *[Queues & Stacks](http://joseblog.netau.net/Computer_Science/data_structures/queues.php)*
* *[Binary Search Trees](http://joseblog.netau.net/Computer_Science/data_structures/binary_search_trees.php)*