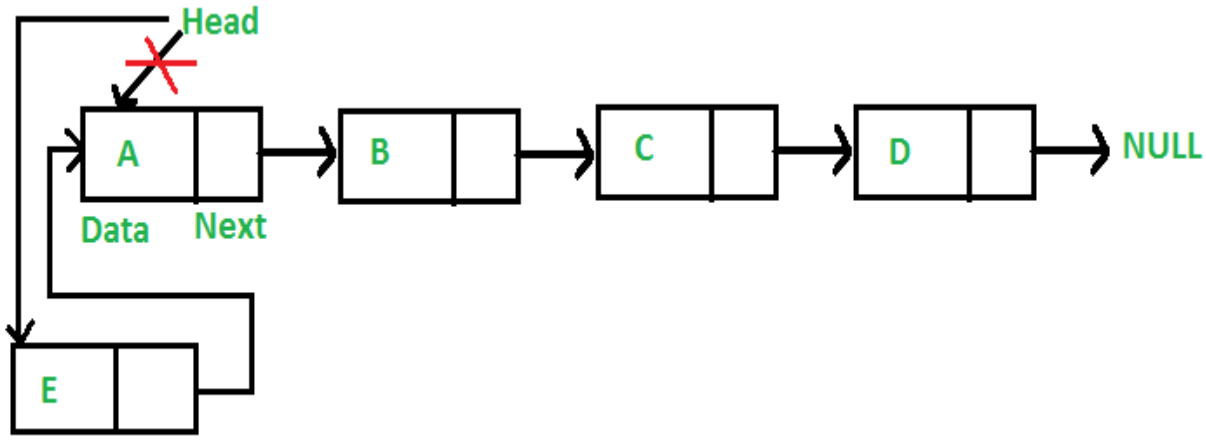
Linked List | Inserting a node

A node can be added in three ways

1) At the front of the linked list
2) After a given node.
3) At the end of the linked list.

Add a node at the front: (A 4 steps process)

The new node is always added before the head of the given Linked List. The newly added node becomes the new head of the Linked List. For example if the given Linked List is 2->3->4->5 and we add a node with value 1 at the front, then the Linked List becomes 1->2->3->4->5. Let us call the function that adds at the front of the list is insertAtFront(). The insertAtFront() must receive a pointer to the head pointer, because insert must change the head pointer to point to the new node.



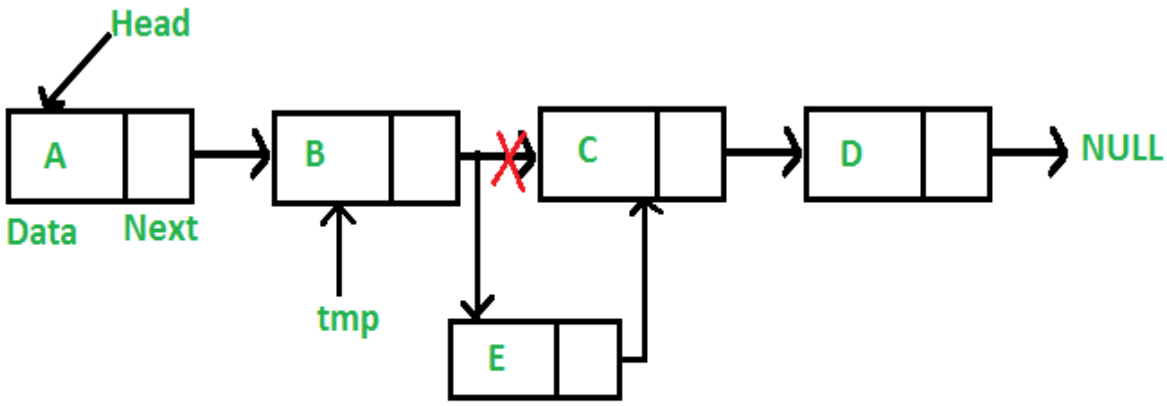
Following are the 4 steps to add node at the front.

```
// A linked list node
struct Node
{
  int data;
  struct Node *next;
};


/* Given a reference (pointer to pointer) to the head of a list
   and an int,  inserts a new node on the front of the list. */
void insertAtFront(struct Node** head_ref, int new_data)
{
    /* 1. allocate node */
```

```
    struct Node* new_node = (struct Node*) malloc(sizeof(struct Node));

    /* 2. put in the data  */
    new_node->data = new_data;

    /* 3. Make next of new node as head */
    new_node->next = (*head_ref);

    /* 4. move the head to point to the new node */
    (*head_ref = new_node;
}
```

Add a node after a given node: (5 steps process)

We are given pointer to a node, and the new node is inserted after the given node.

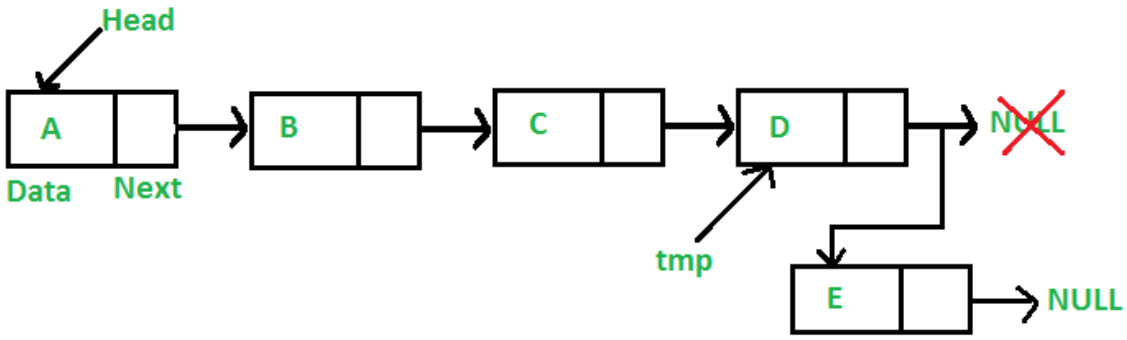



```
/* Given a node prev_node, insert a new node after the given
   prev_node */
void insertAfter(struct Node* prev_node, int new_data)
{
    /*1. check if the given prevnode is NULL */
    if (prev_node == NULL)
    {
       printf("the given previous node cannot be NULL");
       return;
    }

    /* 2. allocate new node */
    struct Node* new_node =(struct Node*) malloc(sizeof(struct Node));

    /* 3. put in the data  */
    new_node->data  = new_data;

    /* 4. Make next of new node as next of prev_node */
```

```
    new_node->next = prev_node->next;

    /* 5. move the next of prev_node as new_node */
    prev_node->next = new_node;
}
```

Add a node at the end: (6 steps process)

The new node is always added after the last node of the given Linked List. For example if the given Linked List is 1->2->3->4->5 and we add a new node with value 6 at the end, then the Linked List becomes 1->2->3->4->5 ->6. Because a Linked List is typically represented by the head of it, we have to traverse the list (iterate over nodes of the list) till end and then change the next of last node to new node.




```
/* Given a reference (pointer to pointer) to the head
   of a list and an int, appends a new node at the end  */
void append(struct Node** head_ref, int new_data)
{
    /* 1. allocate node */
    struct Node* new_node = (struct Node*) malloc(sizeof(struct Node));

    struct Node *last = *head_ref;  /* used in step 5*/

    /* 2. put in the data  */
    new_node->data  = new_data;

    /* 3. This new node is going to be the last node, so make next
          of it as NULL*/
    new_node->next = NULL;

    /* 4. If the Linked List is empty, then make the new node as head
*/
    if (*head_ref == NULL)
    {
       *head_ref = new_node;
       return;
    }
```

```
    /* 5. Else traverse till the last node */
    while (last->next != NULL)
        last = last->next;

    /* 6. Change the next of last node */
    last->next = new_node;
    return;
}
```

The complete program that uses all of the above methods to create a linked list and insert nodes in different locations of the linked list is given below.

```
// A complete working C program to demonstrate all insertion methods
// on Linked List
#include <stdio.h>
#include <stdlib.h>

// A linked list node
struct Node
{
  int data;
  struct Node *next;
};

/* Given a reference (pointer to pointer) to the head of a list and
   an int, inserts a new node on the front of the list. */
void insertAtFront(struct Node** head_ref, int new_data)
{
    /* 1. allocate node */
    struct Node* new_node = (struct Node*) malloc(sizeof(struct Node));

    /* 2. put in the data  */
    new_node->data  = new_data;

    /* 3. Make next of new node as head */
    new_node->next = (*head_ref);

    /* 4. move the head to point to the new node */
    (*head_ref)    = new_node;
}

/* Given a node prev_node, insert a new node after the given
   prev_node */
void insertAfter(struct Node* prev_node, int new_data)
{
    /*1. check if the given prev_node is NULL */
    if (prev_node == NULL)
    {
      printf("the given previous node cannot be NULL");
```

```
      return;
    }

    /* 2. allocate new node */
    struct Node* new_node =(struct Node*) malloc(sizeof(struct Node));

    /* 3. put in the data  */
    new_node->data  = new_data;

    /* 4. Make next of new node as next of prev_node */
    new_node->next = prev_node->next;

    /* 5. move the next of prev_node as new_node */
    prev_node->next = new_node;
}

/* Given a reference (pointer to pointer) to the head
   of a list and an int, appends a new node at the end  */
void append(struct Node** head_ref, int new_data)
{
    /* 1. allocate node */
    struct Node* new_node = (struct Node*) malloc(sizeof(struct Node));

    struct Node *last = *head_ref;  /* used in step 5*/

    /* 2. put in the data  */
    new_node->data  = new_data;

    /* 3. This new node is going to be the last node, so make next of
          it as NULL*/
    new_node->next = NULL;

    /* 4. If the Linked List is empty, then make the new node as head
*/
    if (*head_ref == NULL)
    {
       *head_ref = new_node;
       return;
    }

    /* 5. Else traverse till the last node */
    while (last->next != NULL)
        last = last->next;

    /* 6. Change the next of last node */
    last->next = new_node;
    return;
}
```

```
// This function prints contents of linked list starting from head
void printList(struct Node *node)
{
  while (node != NULL)
  {
     printf(" %d ", node->data);
     node = node->next;
  }
}

/* Driver program to test above functions*/
int main()
{
  /* Start with the empty list */
  struct Node* head = NULL;

  // Insert 6.  So linked list becomes 6->NULL
  append(&head, 6);

  // Insert 7 at the beginning. So linked list becomes 7->6->NULL
  insertAtFront(&head, 7);

  // Insert 1 at the beginning. So linked list becomes 1->7->6->NULL
  insertAtFront(&head, 1);

  // Insert 4 at the end. So linked list becomes 1->7->6->4->NULL
  append(&head, 4);

  // Insert 8, after 7. So linked list becomes 1->7->8->6->4->NULL
  insertAfter(head->next, 8);

  printf("\n Created Linked list is: ");
  printList(head);

  return 0;
}
```

Output:

```
Created Linked list is:  1  7  8  6  4
```