Insertion in Singly Linked List

> *Given the head node of a linked list, the task is to insert a new node in this already created linked list.*

There can be many different situations that may arise while inserting a node in a Linked List. Three most frequent situations are:

1. Inserting a node at the start of the List.
2. Inserting a node after any given node in the List.
3. Inserting a node at the end of the List.

We have seen that a linked list node can be represented using structures and classes as:

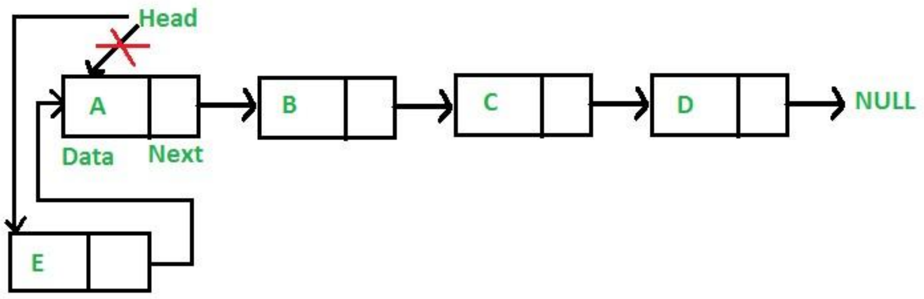
C++ Java

```

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

```

Let us now take a look at each of the above three listed methods of inserting a node in the Linked List:

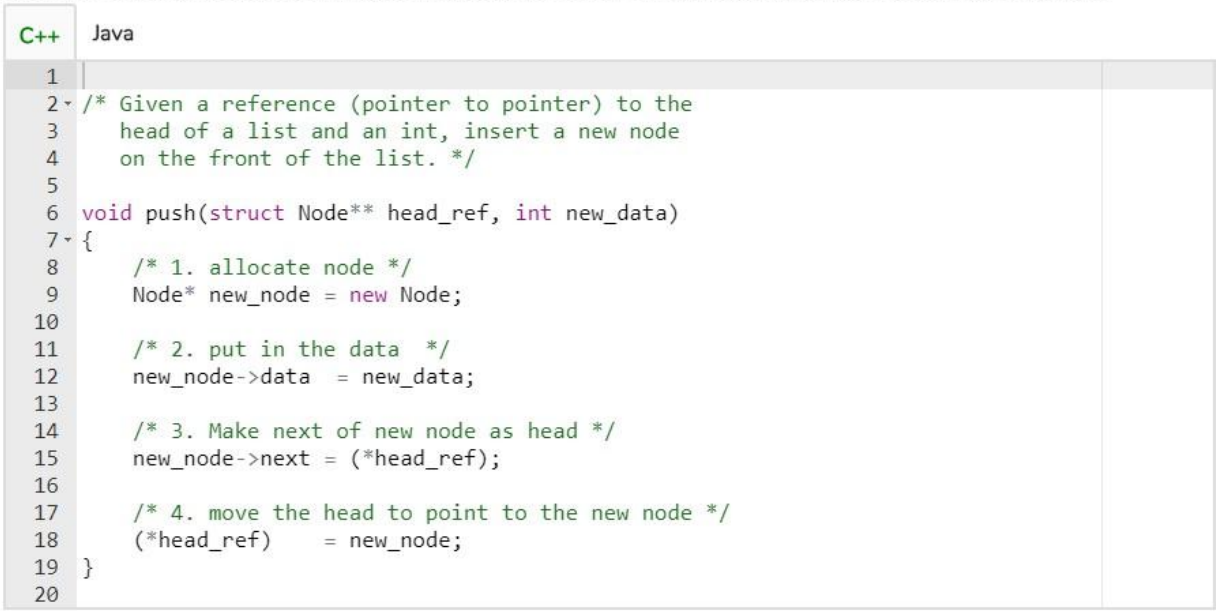
- **Inserting a Node at Beginning**: Inserting a node at the start of the list is a ***four-step*** process. In this process, the new node is always added before the head of the given Linked List and the newly added node becomes the new head of the Linked List.

  For example, if the given Linked List is ***10->15->20->25*** and we add an item **5** at the front, then the Linked List becomes ***5->10->15->20->25***.



Let us call the function that adds a new node at the front of the list as *push()*. The push() function must receive a pointer to the head node, because the function must change the head pointer to point to the new node.

Below is the 4 step process of adding a new node at the front of Linked List declared at the beginning of this post:

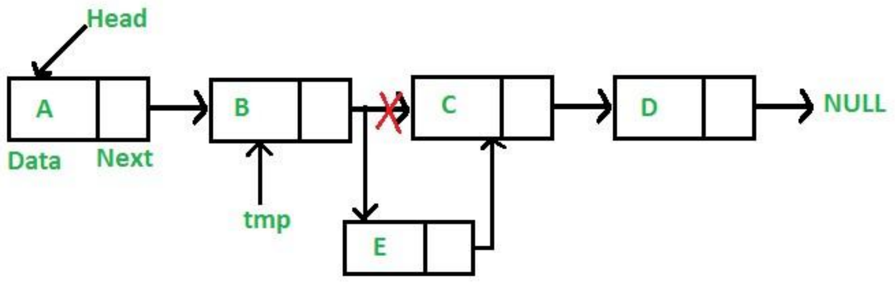


```cpp

/* Given a reference (pointer to pointer) to the
   head of a list and an int, insert a new node
   on the front of the list. */

void push(struct Node** head_ref, int new_data)
{
    /* 1. allocate node */
    Node* new_node = new 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;
}

```

*The **time complexity** of inserting a node at start of the List is O(1).*

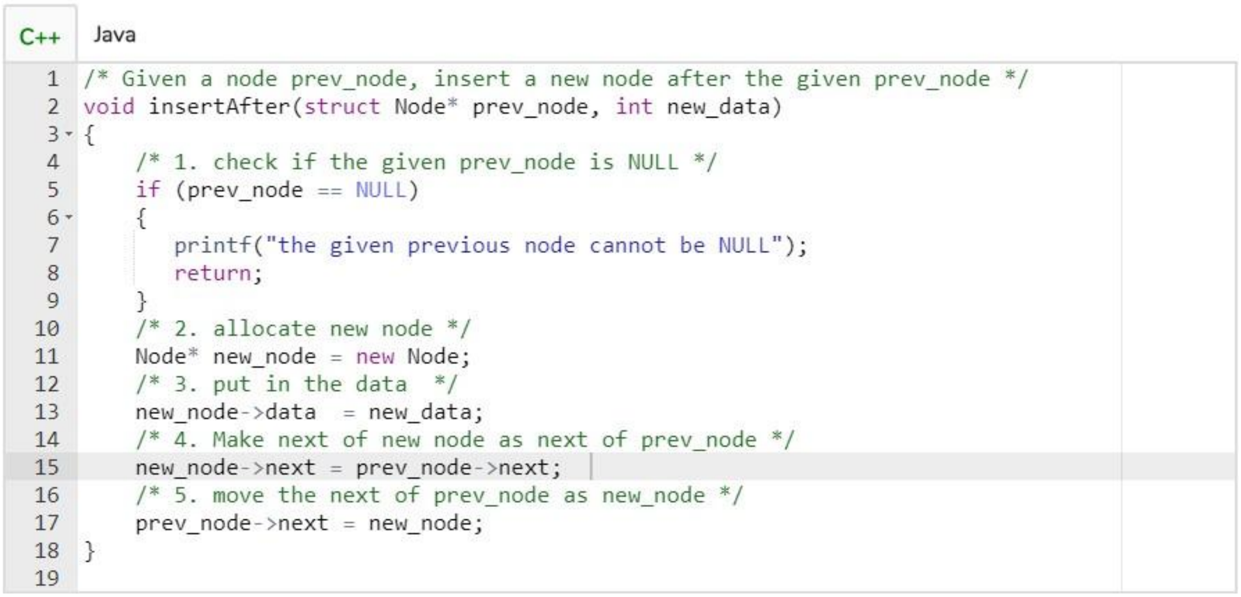
- **Inserting a Node after given Node**: Inserting a Node after a given Node is also similar to the above process. One have to first allocate the new Node and change the next pointer of the newly created node to the next of the previous node and the next pointer of the previous node to point to the newly created node.

  Below is the pictorial representation of the complete process:



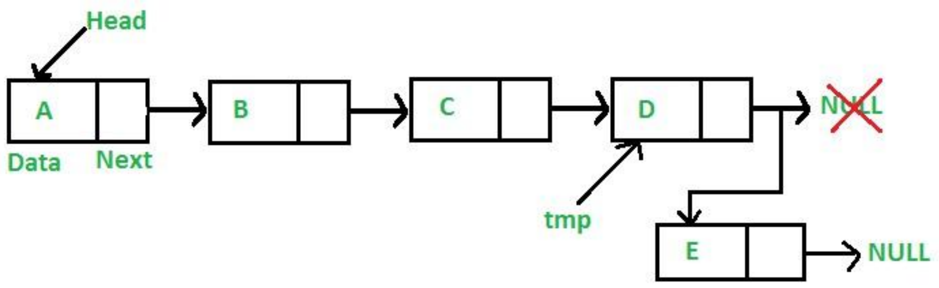
Let us call the function that adds a new node after a given node in the list as insertAfter(). The *insertAfter()* function must receive a pointer to the previous node after which the new node is to be inserted.

Below is the complete process of adding a new node after a given node in the Linked List declared at the beginning of this post:



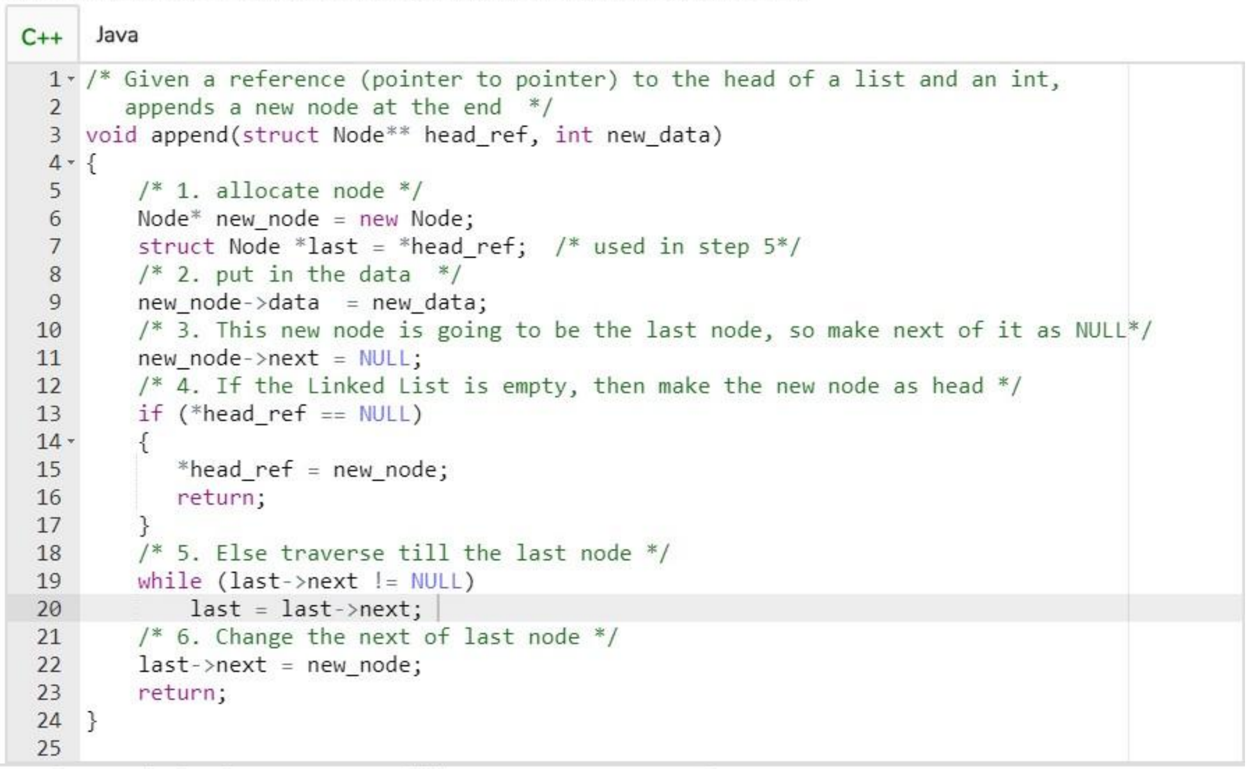
```cpp
/* Given a node prev_node, insert a new node after the given prev_node */
void insertAfter(struct Node* prev_node, int new_data)
{
    /* 1. check if the given prev_node is NULL */
    if (prev_node == NULL)
    {
       printf("the given previous node cannot be NULL");
       return;
    }
    /* 2. allocate new node */
    Node* new_node = new 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;
}

```

- **Inserting a Node at the End**: Inserting a new Node at the last of a Linked list is generally a six step process in total. The new node is always added after the last node of the given Linked List. For example if the given Linked List is ***5->10->15->20->25*** and we add an item **30** at the end, then the Linked List becomes ***5->10->15->20->25->30***.



Since a Linked List is typically represented by the head of it, we have to first traverse the list till the end in order to get the pointer pointing to the last node and then change the next of last node to new node.

Below is the complete 6 step process of adding a new Node at the end of the list:



```
/* 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 */
    Node* new_node = new 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 **time complexity** of this operation is **O(N)** where N is the number of nodes in the Linked List as one have to traverse the complete list in order to find the last node.*