Tutorial Link https://codequotient.com/tutorials/Linked-List : Operations - Insertion/5a59de12a4af025f554a0bcb

TUTORIAL

# Linked-List : Operations - Insertion

## Chapter

Topics

## Insertion in Linked List

Let a linked list is having successive nodes A and B. Suppose a node N is to be inserted into the list between nodes A and B. It is called inserting a new node in the list.

**Insertion Algorithms:** To insert an element in linked list we have to correctly manipulate the pointers of linked list nodes. There are three different scenarios for insertions, which are:

(a) Insert a node at the beginning of the list.

(b) Insert a node after the given node with a given location.

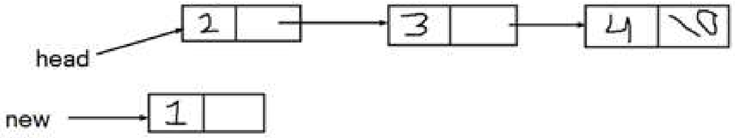
(c) Insert a node at last of the list.

**Insertion at the beginning of a List**

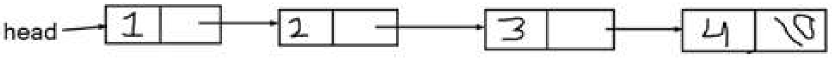
While inserting in the beginning, the new node will point to the first node of list, and then head will point to the new node. So basically two pointers needs to be manipulated. So,

```
NEW[DATA] = INFO        // place the data in new node
NEW[NEXT] = HEAD    // link the next of new node to head
HEAD = NEW        // link the head to new node
```

Following figure illustrate this manipulation: -



Before Insertion



After Insertion at beginning

While modifying these pointers care needs to be taken, that if these pointers are modified in wrong sequence, then it will destroy the list. If we write the above steps in following order: -

```
NEW[DATA] = INFO        // place the data in NEWnode
HEAD = NEW        // link the head to NEWnode
NEW[NEXT] = HEAD    // link the next of NEWnode to HEAD
```

In second step the pointer to existing list will be destroyed as now head is pointing to new node, so in 3rd step new[next] is actually pointing to itself. So, always change the pointers of new node first using existing information and then change the existing links to point to new node.

C

```c
// Insert in Beginning
void insertBeg(struct Node** head, int data){
  struct Node* node = (struct Node*) malloc(sizeof(struct Node));
  node->data  = data;     //  Insert data in new node
  node->next = (*head);   // link new node at beginning of list
  (*head)    = node;  // Change the head to new node.
}

```

Java

```java
// Insert in Beginning
static LinkList insertBeg(LinkList first, int data)
{
  LinkList newLink = new LinkList(data);
  newLink.next = first; // newLink --> old first
  first = newLink; // first --> newLink
  return first;
}
```

Python 3

```python
def insertBeg(head, data):
    new_node = Node(data) # Allocating a new node
    new_node.next = head # link the new node to the beginning of list
    head = new_node # changing the head of the node
    return head;

```

C++

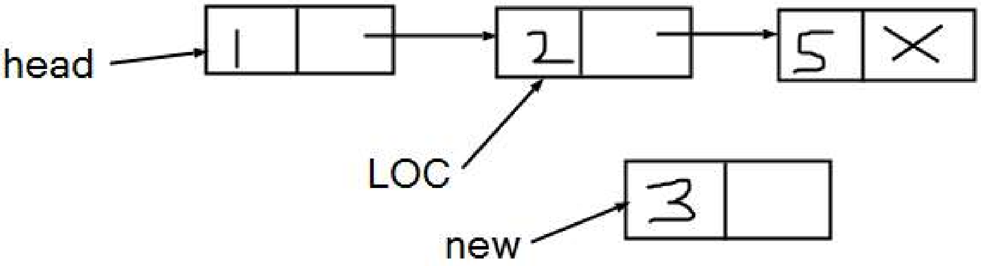
```cpp
// Insert in Beginning
void insertBeg(struct Node** head, int data){
    struct Node* node = new Node(); // Allocating memory
    node->data  = data;     //  Insert data in new node
    node->next = (*head);   // link new node at beginning of list
    (*head)    = node;  // Change the head to new node.
}

```

### Insertion after a Given Node

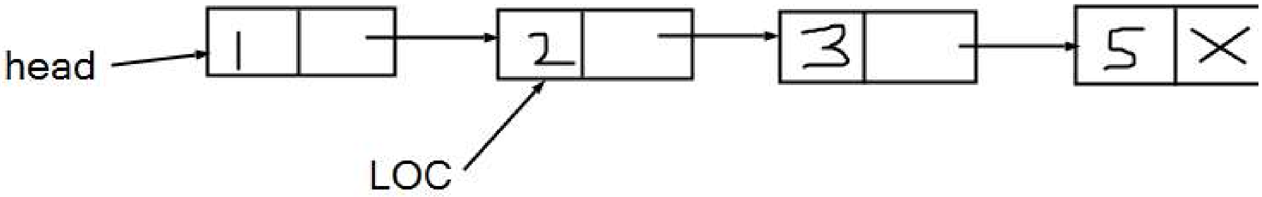
Suppose we are given the value of LOC which indicates the location of the node after which new node is to be inserted. To insert an element after this node, we have to change the next pointer of this node and the next pointer of new node in correct sequence.

```
NEW[DATA] = INFO            // place the data in NEW node
NEW[NEXT] =LOC[NEXT]        // link the next of NEW to next of LOC
LOC[NEXT]  = NEW            // link the next of LOC to NEW
```

Following figure illustrate this manipulation: -



BEFORE INSERTION



AFTER INSERTION

Again take care while changing the pointers as changing in wrong sequence will lead to removal of list pointers.

```C
void insertAfter(struct Node* prev, int data)
{
  if (prev == NULL)
  {
    printf("the given previous node cannot be NULL");
    return;
  }
  struct Node* node =(struct Node*) malloc(sizeof(struct Node));
  node->data  = data;     //  Insert data in new node
  node->next = prev->next;   // link new node after prev node
  prev->next = node;  // Link the previous node to new node.
}

```

```Java
// insert after prev
static LinkList insertAfter(LinkList prev, int data)
  {
    if (prev == null)
    {
      System.out.println("the given previous node cannot be NULL");
      return null;
    }
    LinkList newLink = new LinkList(data);
    newLink.next = prev.next;   // link new node after prev node
    prev.next = newLink;  // Link the previous node to new node.
    return prev;
  }
```

Python 3

```
# Inserting after a given node
def insertAfter(prev_node, data):
    if prev_node is None:
        print('The previous node cannot be null');
        return;
    new_node = Node(data)
    new_node.next = prev_node.next
    prev_node.next = new_node
```

C++

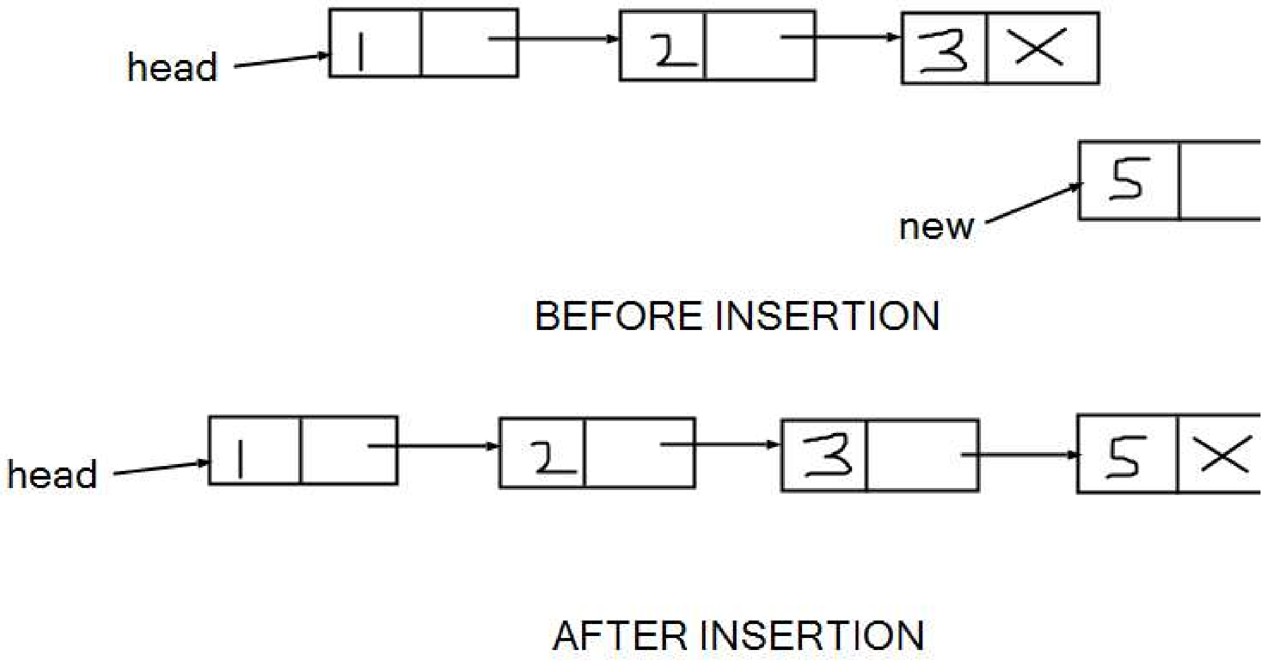
```
// Insert after a given node
void insertAfter(struct Node* prev, int data){
    if (prev == NULL){
        cout<<"The given previous node cannot be NULL";
        return;
    }
    struct Node* node = new Node();
    node->data  = data;     //  Insert data in new node
    node->next = prev->next;   // link new node after prev node
    prev->next = node;  // Link the previous node to new node.
}
```

### Insertion at the End

Suppose we have to add an element at end of list. In this case, we have to traverse the list from head till the last element. Now new element can be added after the last element by changing the next pointer of last node to point to NEW node and Next pointer of NEW node will point to NULL in this case as this will be the last node now.

```
TRAVERSE WHOLE LIST and FIND THE LAST LOC
NEW[DATA] = INFO            // place the data in NEW node
NEW[NEXT] = NULL        // link the next of NEW to NULL
LAST[NEXT] = NEW        // link the next of LAST to NEW
```

Following figure illustrate this manipulation: -



C

```c
void insertEnd(struct Node** head, int data)
{
  struct Node* node = (struct Node*) malloc(sizeof(struct Node));
  struct Node *last = *head;
  node->data  = data;     //  Insert data in new node
  node->next = NULL;   // link new node to NULL as it is last node
  if (*head == NULL)  // if list is empty add in beginning.
  {
    *head = node;
    return;
  }
  while (last->next != NULL)  // Find the last node
    last = last->next;
  last->next = node;  // Add the node after the last node of list
  return;
}

```

Java

```java
// insert at end
 static LinkList insertEnd(LinkList head, int data)
  {
    LinkList newLink = new LinkList(data);
    LinkList last = head;
    newLink.next = null;   // link new node to NULL as it is last node
    if (head == null)  // if list is empty add in beginning.
    {
      head = newLink;
      return head;
    }
    while (last.next != null)  // Find the last node
      last = last.next;
    last.next = newLink;  // Add the node after the last node of list
    return head;
  }
```

Python 3

```
# Inserting at the end
def insertEnd(head,data):
    new_node = Node(data)
    if head is None:
        head = new_node
        return
    last = head
    while (last.next):
       last = last.next
    last.next =  new_node
    return head;

```

C++

```
// Function to insert at end of linked list
void insertEnd(struct Node** head, int data){
    struct Node* node = new Node();
    struct Node *last = *head;
    node->data  = data;     //  Insert data in new node
    node->next = NULL;   // link new node to NULL as it is last node
    if (*head == NULL)  // if list is empty add in beginning.
    {
        *head = node;
        return;
    }
    while (last->next != NULL)  // Find the last node
        last = last->next;
    last->next = node;  // Add the node after the last node of list
    return;
}
```

All these three insertions can be simulated in a single go as below: -

C

```c
int main()
{
  struct Node* head = NULL;
  printf("Linked List = ");
  printList(head);
  insertBeg(&head, 6);    // At Beginning
  printf("Linked List = ");
  printList(head);
  insertBeg(&head, 2);    // At Beginning
  printf("Linked List = ");
  printList(head);
  insertAfter(head, 3);   // After Head node
  printf("Linked List = ");
  printList(head);
  insertEnd(&head, 8);    // At End
  printf("Linked List = ");
  printList(head);
  insertAfter(head->next, 4); // After 2nd Node
  printf("Linked List = ");
  printList(head);
  return 0;
}

```

Java

```java
public static void main(String[] args)
  {
    LinkList head = null;
    System.out.print("Linked List = ");
    traverse(head);
    head = insertBeg(head, 6);    // At Beginning
    System.out.print("Linked List = ");
    traverse(head);
    head = insertBeg(head, 2);    // At Beginning
    System.out.print("Linked List = ");
    traverse(head);
    head = insertAfter(head, 3);   // After after
    System.out.print("Linked List = ");
    traverse(head);
    head = insertEnd(head, 8);   // After at End
    System.out.print("Linked List = ");
    traverse(head);
    insertAfter(head.next, 4);   // After after
    System.out.print("Linked List = ");
    traverse(head);
  }
}
```

Python 3

```
if __name__ == "__main__":
    head = None;
    print('Linked List = ',end = ' ');
    printList(head);
    head = insertBeg(head,6);
    print('Linked List = ',end = ' ');
    printList(head);
    head = insertBeg(head,2);
    print('Linked List = ',end = ' ');
    printList(head);

    insertAfter(head,3);
    print('Linked List = ',end = ' ');
    printList(head);

    head = insertEnd(head,8);
    print('Linked List = ',end = ' ');
    printList(head);

    insertAfter(head.next,4);
    print('Linked List = ',end = ' ');
    printList(head);
```

C++

```
int main(){
    struct Node* head = NULL;
    cout<<"Linked List = ";
    printList(head);
    insertBeg(&head, 6);    // At Beginning
    cout<<"Linked List = ";
    printList(head);
    insertBeg(&head, 2);    // At Beginning
    cout<<"Linked List = ";
    printList(head);

    insertAfter(head, 3);// After head
    cout<<"Linked List = ";
    printList(head);
    insertEnd(&head,8); // At end
    cout<<"Linked List = ";
    printList(head);

    insertAfter(head->next, 4); // After 2nd node
    cout<<"Linked List = ";
    printList(head);
    return 0;
}
```

```
Linked List =
Linked List = 6 ->
Linked List = 2 -> 6 ->
Linked List = 2 -> 3 -> 6 ->
Linked List = 2 -> 3 -> 6 -> 8 ->
Linked List = 2 -> 3 -> 4 -> 6 -> 8 ->
```