DATA STRUCTURES

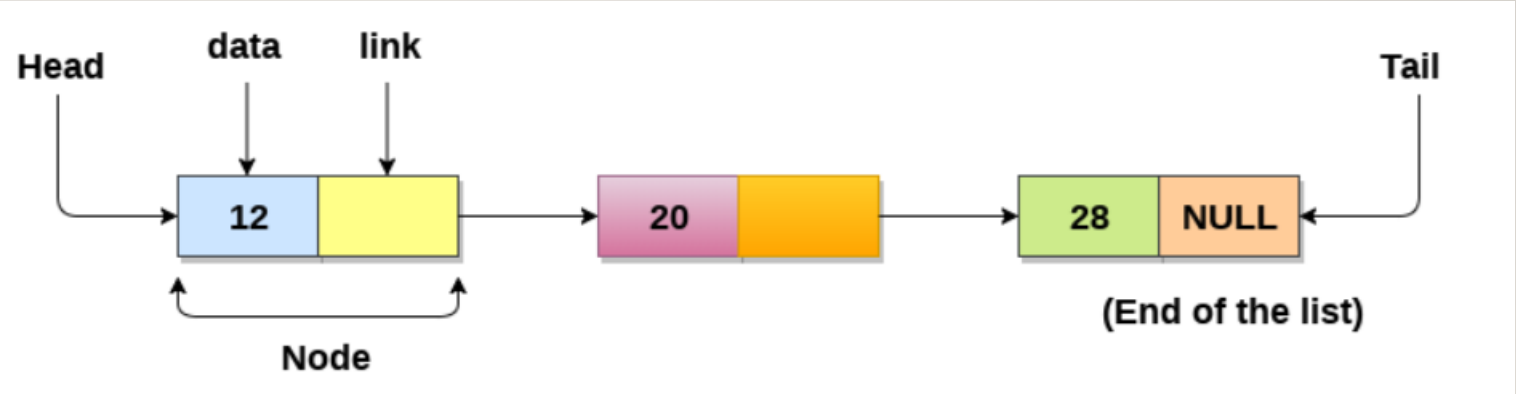
LECTURE-6

LINKED LIST

Dr. Sumitra Kisan

Linked List

- A linked list is a linear data structure that stores a collection of data elements dynamically.
- Nodes represent those data elements, and links or pointers connect each node.
- Each node consists of two fields, the information stored in a linked list and a pointer that stores the address of its next node.
- The last node contains null in its second field because it will point to no node.
- A linked list can grow and shrink its size, as per the requirement.
- It does not waste memory space.



Array contains following limitations:

- The size of array must be known in advance before using it in the program.
- Increasing size of the array is a time taking process. It is almost impossible to expand the size of the array at run time.
- All the elements in the array need to be contiguously stored in the memory. Inserting any element in the array needs shifting of all its predecessors.

Linked list is useful because:

- It allocates the memory dynamically. All the nodes of linked list are non-contiguously stored in the memory and linked together with the help of pointers.
- Sizing is no longer a problem since we do not need to define its size at the time of declaration. List grows as per the program's demand and limited to the available memory space.

Creation of Node and Declaration of Linked Lists

Creating a node means defining its structure, allocating memory to it and its initialization.

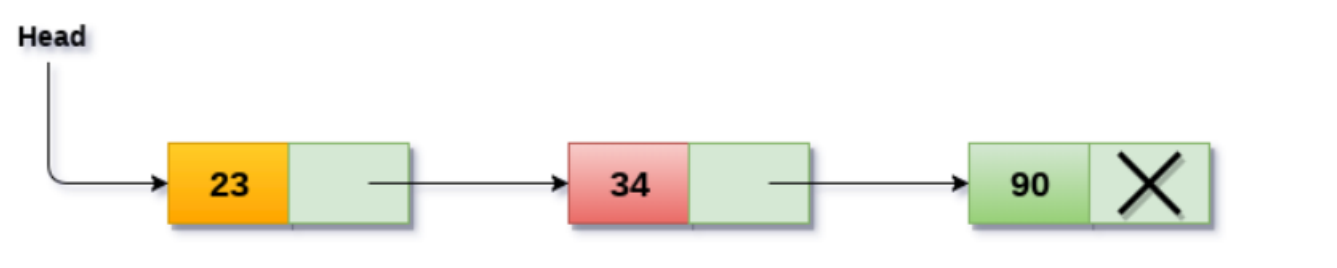
```
struct node
{
        int data;
        struct node * next;
};
struct node * n;
n=(struct node*)malloc(sizeof(struct node*));
```

- It is a declaration of a node that consists of the first variable as data and the next as a pointer, which will keep the address of the next node.
- Malloc function is used to allocate memory for the nodes dynamically.

Types of Linked Lists

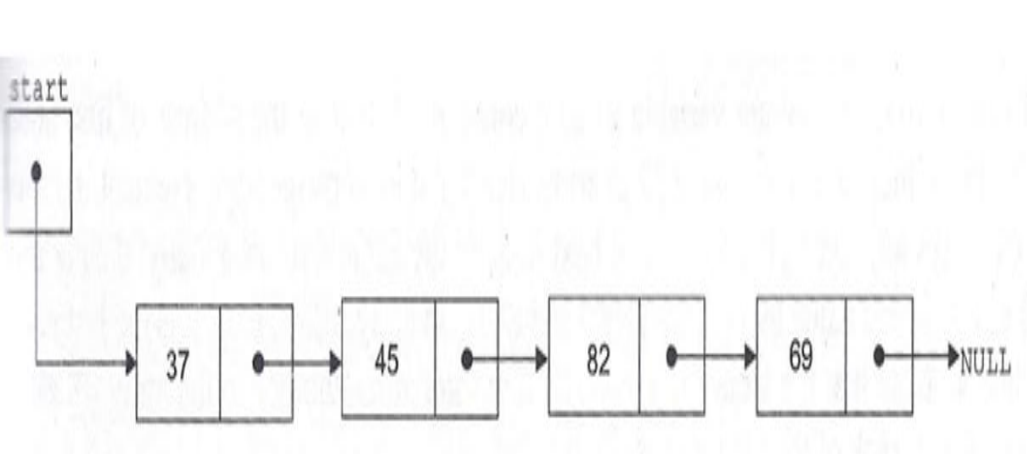
Singly Linked List

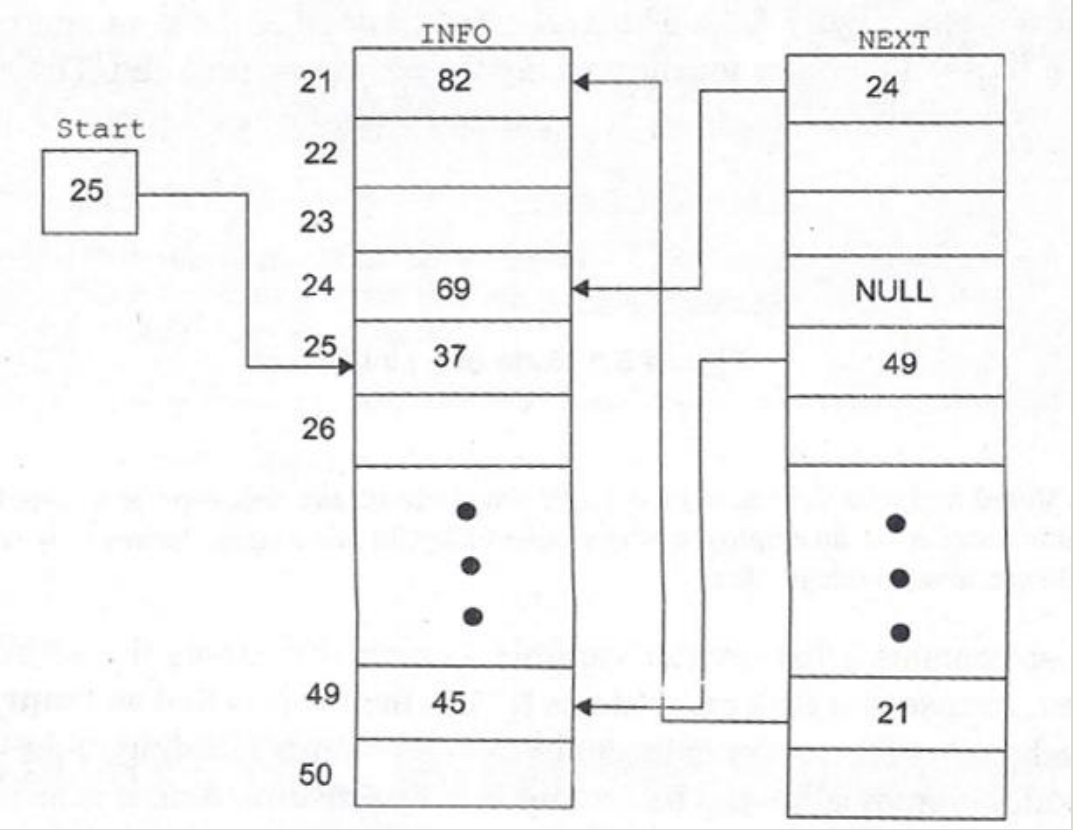
- A singly linked list (also called linear linked list), each node consists of two fields: info and next. The info field contains the data and the next field contains the address of memory location where the next node is stored.
- The last node of the singly linked list contains NULL in its next field that indicates the end of list.
- A linked list contains a list pointer variable Head that stores the address of the first node of the list. In case, the Head contains NULL, the list is called an **empty list or a null list.**



Memory Representation of Linked List

- To maintain a linked list in memory, two parallel arrays of equal size are used.
- One array(say INFO) is used for the info field and another array (say, NEXT) for the next field of the nodes of the list.

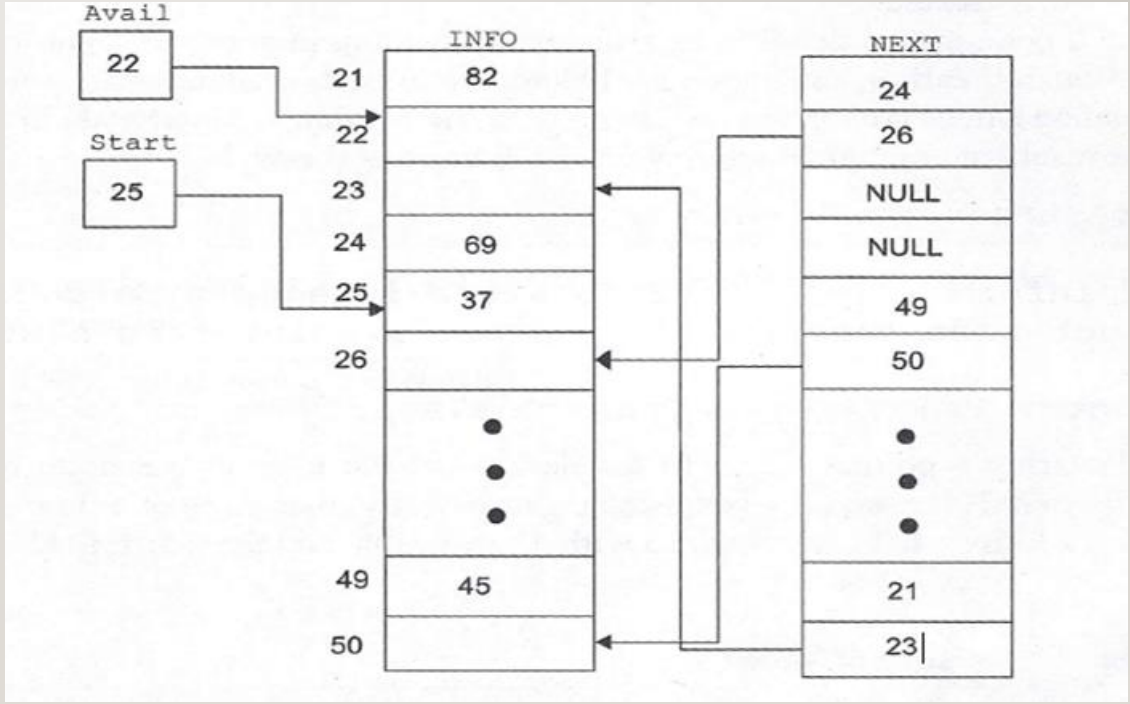




*In this figure, the pointer variable **Start** contains 25, that is, the address of first node of the list, which stores the value 37 in array **INFO** and its corresponding element in array **NEXT** stores 49, that is, the address of next node in the list and so on. Finally, the node at address 24 stores value 69 in array **INFO** and **NULL** in array **NEXT**, thus, it is the last node of the list.*

Memory Allocation

- As memory is allocated dynamically to the linked list, a new node can be inserted anytime in the list.
- For this, the memory manager maintains a special linked list known as free storage list or memory bank or free pool that consists of unused memory cells.
- This list keeps track of the free space available in the memory and a pointer to this list is stored in a pointer variable **Avail**.
- The end of free-storage list is also denoted by storing NULL in the last available block of memory.



- *In this figure, Avail contains 22, hence, INFO[22] is the starting point of the free storage list.*
- *Since NEXT [22] contains 26, INFO [26] is the next free memory location.*
- *Similarly, other free spaces can be accessed and the NULL in NEXT [23] indicates the end of free-storage list.*

Operations

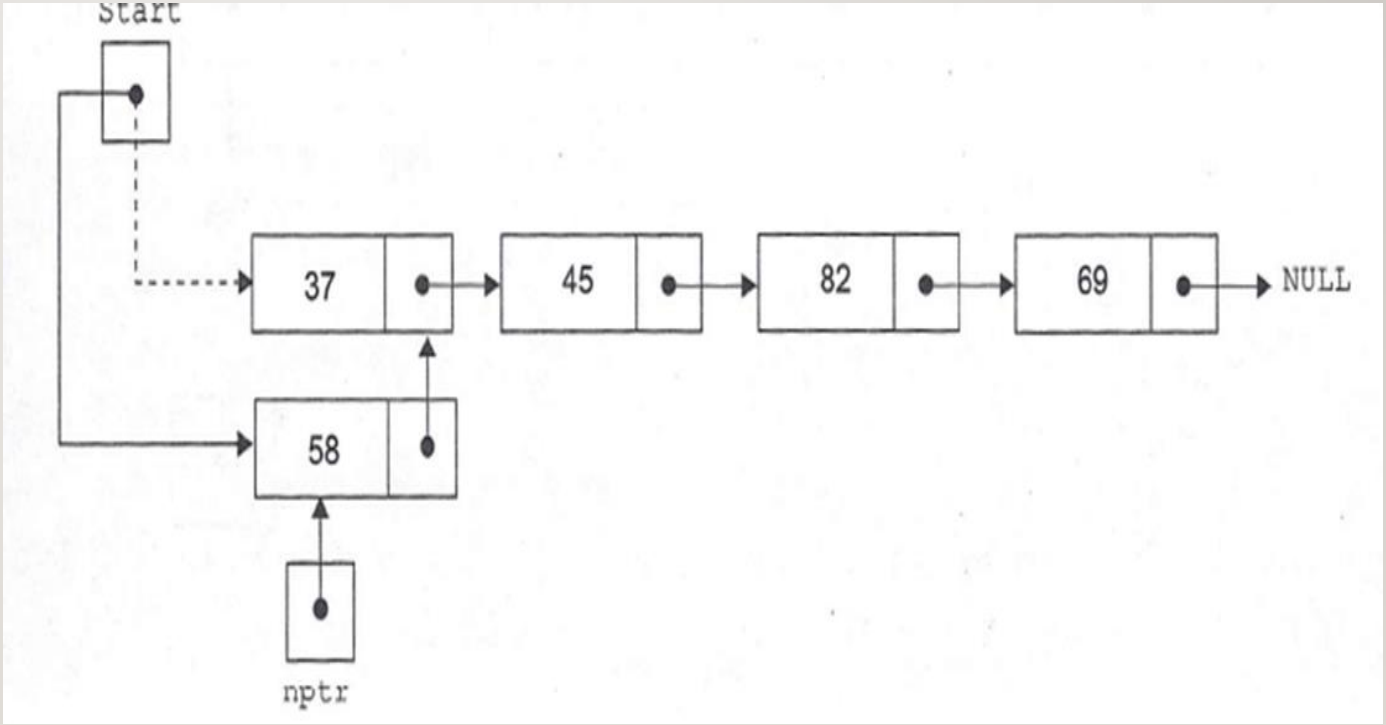
Creation of node

```
create_node()

1.Allocate memory for nptr    //nptr is a pointer to new node
2.If nptr = NULL
      Print "Overflow: Memory not allocated!" and go to step 7
  End If
3.Read item  //item is the value to be inserted in the new node
4.Set nptr->info = item
5.Set nptr->next = NULL
6.Return nptr    //returning pointer nptr
7.End
```

Insertion in Beginning:

To insert a node in the beginning of list, the next field of new node (pointed by nptr) is made to point to the existing first node and the Start pointer is modified to point to the new node.

1. Call create_node()
2. Set nptr->next= start
3. Set start = nptr
4. End



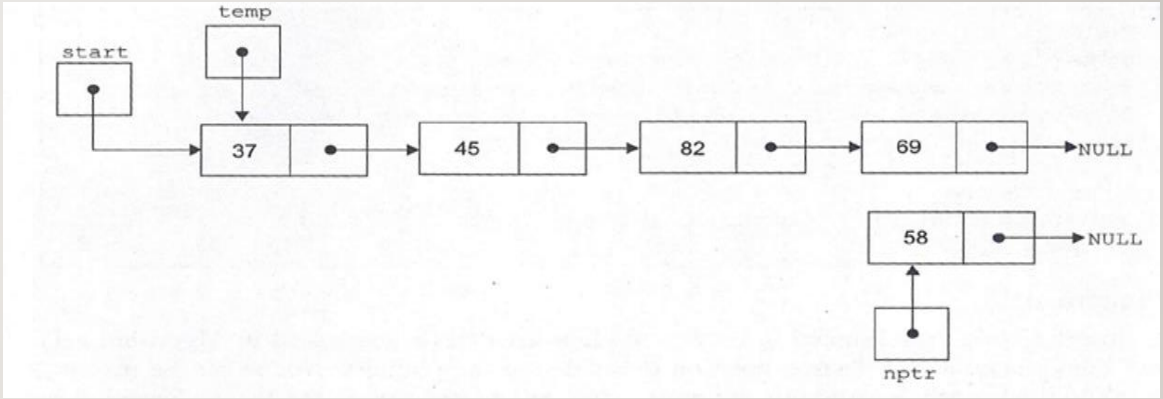
```
#include <stdio.h>
#include <stdlib.h>
struct Node {// Define a node in the linked list
                int data; // Data stored in the node
                 struct Node*
                   next;};
struct Node* createNode(int new_data)
{
struct Node* new_node
       = (struct Node*)malloc(sizeof(struct Node));
new_node->data = new_data; // Initialize the node's data
new_node->next = NULL; // Set the next pointer to NULL
return new_node; // Return the newly created node
}
struct Node* insertAtFront(struct Node* head, int new_data)
{
struct Node* new_node = createNode(new_data);
new_node->next = head;
return new_node;
}
```

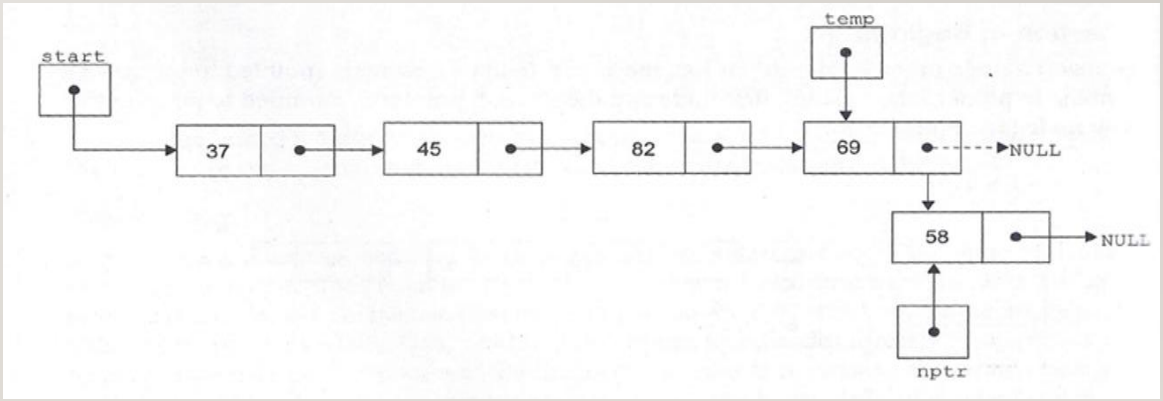
```
void printList(struct Node* head)
{
struct Node* curr = head;
while (curr != NULL) {
printf(" %d", curr->data);
curr = curr->next;  }
printf("\n");
}
int main()
{
    // Create the linked list 2->3->4->5
    struct Node* head = createNode(2);
    head->next = createNode(3);
    head->next->next = createNode(4);
    head->next->next->next = createNode(5);
printf("Original Linked List:");
    printList(head);
printf("After inserting Nodes at the front:");
    int data = 1;
    head = insertAtFront(head, data);
printList(head);
 return 0;
}
```

Insertion at End

- To insert a node at the end of a linked list, the list is traversed up to the last node and the next field of this node is modified to point to the new node.
- If the linked list is initially empty, then the new node becomes the first node and Start points to it.

```
Insert_end(start)
1.  Call create_node()
2. If start = NULL
      Set start =nptr
    Else
      Set temp = start
      while temp->next! =NULL
          Set temp = temp->next
   End while
  Set temp->next = nptr
End If
3. End
```



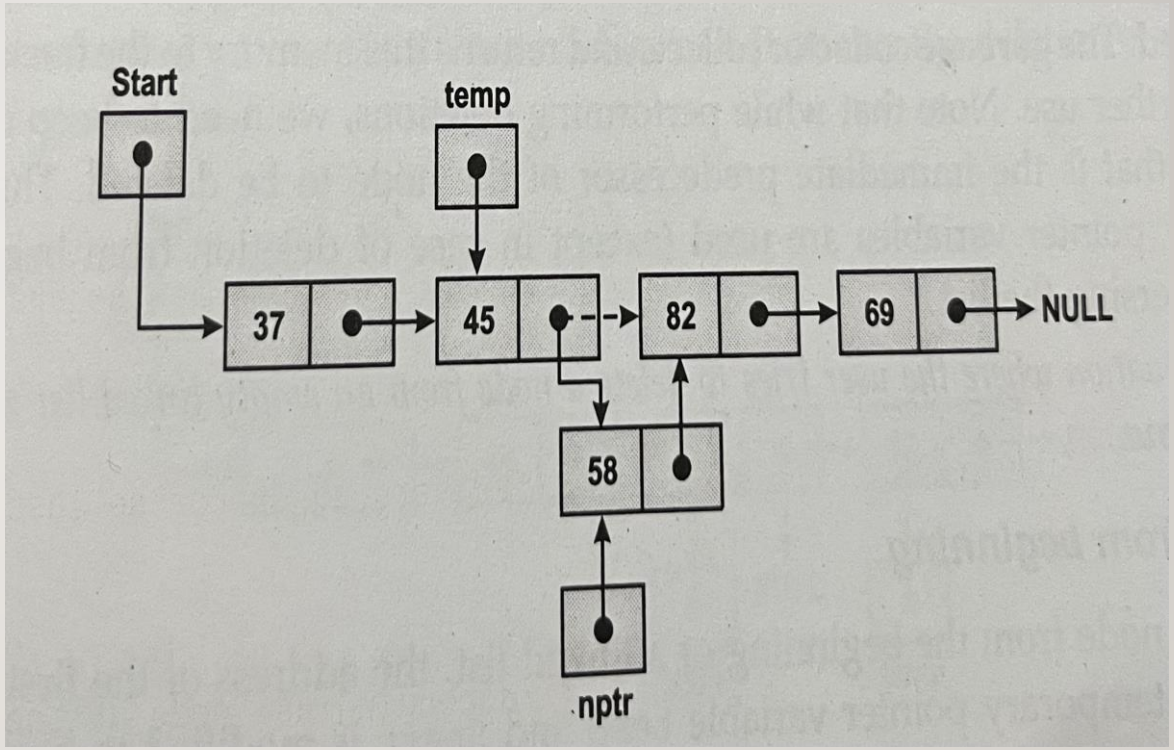


```
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node* next; };
struct Node* createNode(int new_data) {
    struct Node* new_node =
      (struct Node*)malloc(sizeof(struct Node));
    new_node->data = new_data;
    new_node->next = NULL;
    return new_node; }
struct Node* append(struct Node* head, int new_data) {
  struct Node* new_node = createNode(new_data);
if (head == NULL)  {
        return new_node;
    }
struct Node* last = head;

while (last->next != NULL) {
        last = last->next;
    }
last->next = new_node;
return head; }
```

```
void printList(struct Node* node) {
    while (node != NULL) {
        printf(" %d", node->data);
        node = node->next;
    }
}
int main() {

    // Create a hard-coded linked list:
    // 2 -> 3 -> 4 -> 5 -> 6
    struct Node* head = createNode(2);
    head->next = createNode(3);
    head->next->next = createNode(4);
    head->next->next->next = createNode(5);
    head->next->next->next->next = createNode(6);
 printf("Created Linked list is:");
    printList(head);
head = append(head, 1);
printf("\nAfter inserting 1 at the end:");
    printList(head);
return 0;
}
```

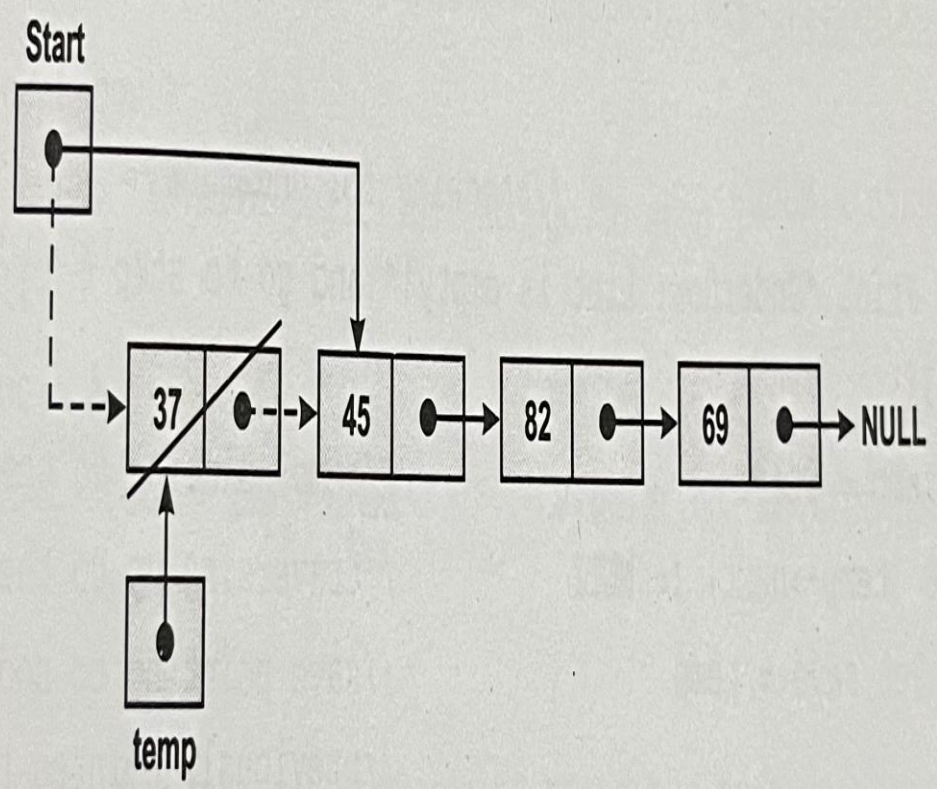
Insertion at a Specified Position

```
insert_pos(Start)

1.  Call create_node()              //creating a new node pointed to by nptr
2.  Set temp = Start
3.  Read pos                        //position at which the new node is
                                    //to be inserted
4.  Call count_node(temp)           //counting total number of nodes in
                                    //count variable
5.  If (pos > count + 1 OR pos = 0)
        Print "Invalid position!" and go to step 7
    End If
6.  If pos = 1
        Set nptr->next = Start
        Set Start = nptr            //inserting new node as the first node
    Else
        Set i = 1
        While i < pos - 1 //traversing up to the node at pos-1 position
           Set temp = temp->next
           Set i = i + 1
    End While
    Set nptr->next = temp->next             //inserting new node at
                                            //pos position
    Set temp->next = nptr
    End If
7.  End
```



Deleting the First Node

```
delete_beg(Start)

1.  If Start = NULL                    //checking for underflow
        Print "Underflow: List is empty!" and go to step 5
    End If
2.  Set temp = Start                   //temp pointing to the first node
3.  Set Start = temp->next             //moving Start to point to the
                                           //second node
4.  Deallocate temp                    //deallocating memory
5.  End
```



Deleting the Last Node

```
delete_end(Start)

1.  If Start = NULL                    //checking for underflow
        Print "Underflow: List is empty!" and go to step 6
    End If
2.  Set temp = Start                   //temp pointing to the first node
3.  While (temp->next) != NULL         //traversing up to the last node
        Set save = temp                //save pointing to node
                                       //previously pointed to by temp
        Set temp = temp->next          //moving temp to point to next node
    End While
4.  Set save->next = NULL              //making new last node to
                                       //point to NULL
5.  Deallocate temp                    //deallocating memory
6.  End
```