CS210: Data Structures

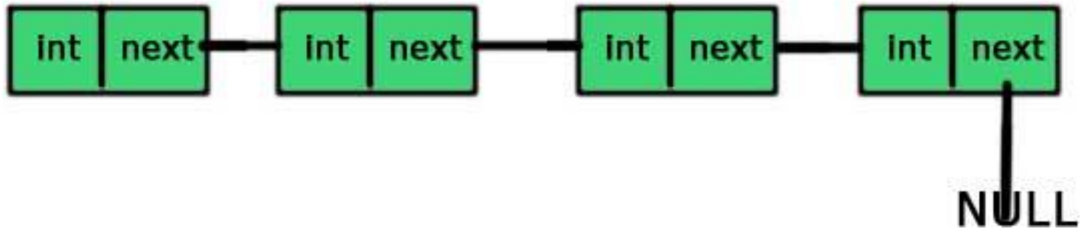
Dr. Balu L. Parne

Introduction to Linked List

Introduction:

- Linked list is one of the most important data structures. We often face situations, where the data is dynamic in nature and number of data can't be predicted or the number of data keeps changing during program execution.
- Linked lists are very useful in this type of situations.

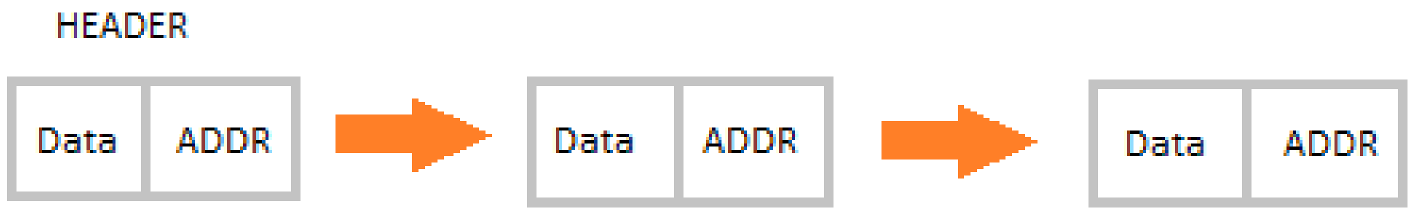
- A linked list is made up of many nodes which are connected in nature. Every node is mainly divided into two parts, one part holds the data and the other part is connected to a different node(next pointer).



- A linked list is a series of connected *nodes*, where each node is a data structure.

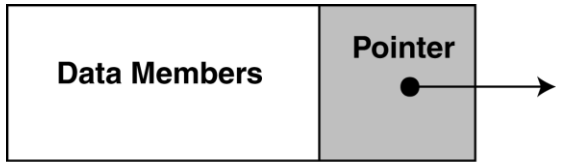
- A linked list can grow or shrink in size as the program runs

- Linked List is a linear data structure and it is very common data structure which consists of group of nodes in a sequence which is divided in two parts. Each node consists of its own data and the address of the next node and forms a chain. Linked Lists are used to create trees and graphs.



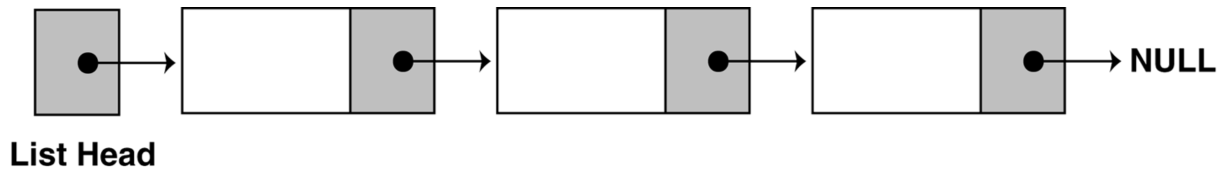
The composition of a Linked List

- Each node in a linked list contains one or more members that represent data.
- In addition to the data, each node contains a pointer, which can point to another node.



The composition of a Linked List

- A linked list is called "linked" because each node in the series has a pointer that points to the next node in the list.



Advantages of Linked Lists :

- They are a dynamic in nature which allocates the memory when required.
- Insertion and deletion operations can be easily implemented.
- Stacks and queues can be easily executed.
- Linked List reduces the access time.
- A linked list can easily grow or shrink in size.
- Insertion and deletion of nodes is quicker with linked lists than with arrays.

Disadvantages of Linked Lists:

- The memory is wasted as pointers require extra memory for storage.
- No element can be accessed randomly; it has to access each node sequentially.
- Reverse Traversing is difficult in linked list.

Array versus Linked Lists:

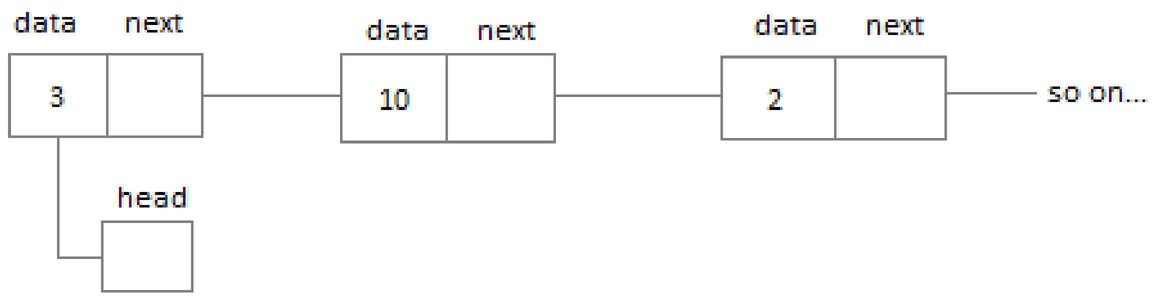
- Linked lists are more complex to code and manage than arrays, but they have some distinct advantages.
  - **Dynamic**: a linked list can easily grow and shrink in size.
    - We don't need to know how many nodes will be in the list. They are created in memory as needed.
    - In contrast, the size of a JAVA array is fixed at compilation time.
  - **Easy and fast insertions and deletions**
    - To insert or delete an element in an array, we need to copy to temporary variables to make room for new elements or close the gap caused by deleted elements.
    - With a linked list, no need to move other nodes. Only need to reset some pointers.

Applications of Linked Lists:

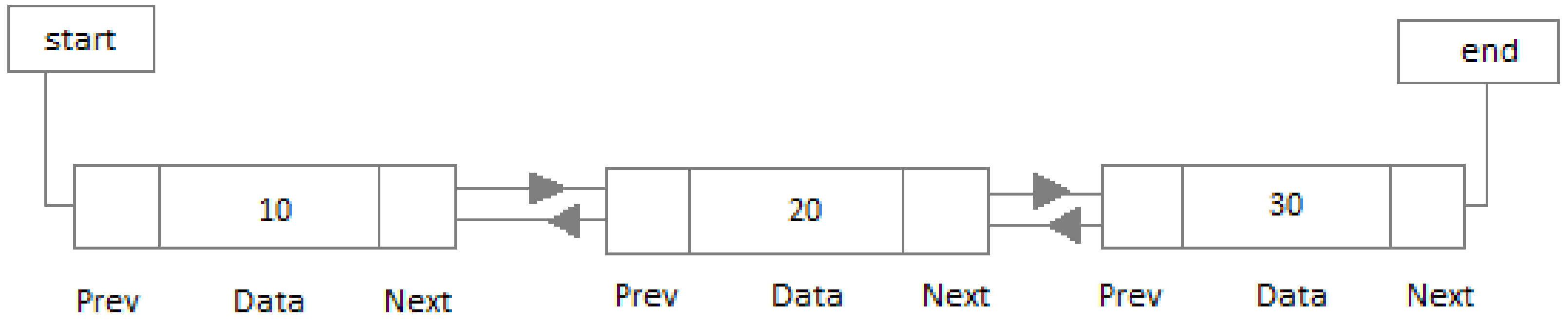
- Linked lists are used to implement stacks, queues, graphs, etc.
- Linked lists let you insert elements at the beginning and end of the list.
- In Linked Lists we don't need to know the size in advance.

Types of Linked Lists:

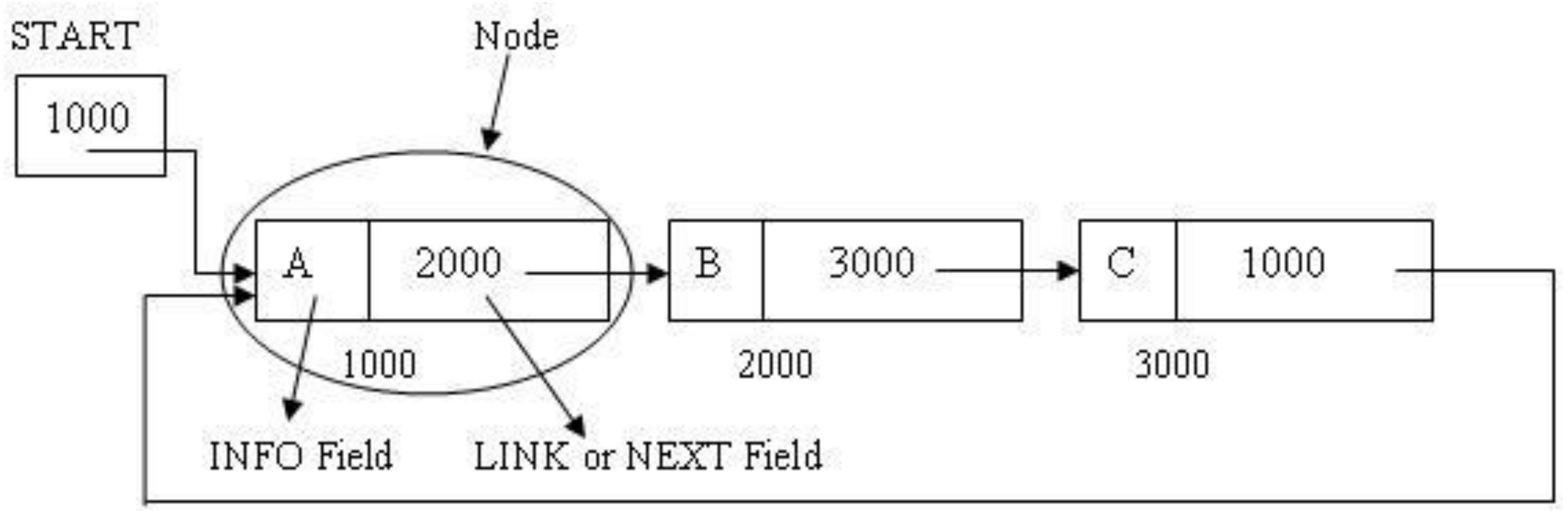
- **Singly Linked List :** Singly linked lists contain nodes which have a data part as well as an address part i.e. next, which points to the next node in sequence of nodes. The operations we can perform on singly linked lists are insertion, deletion and traversal



- **Doubly Linked List :** In a doubly linked list, each node contains two links the first link points to the previous node and the next link points to the next node in the sequence.



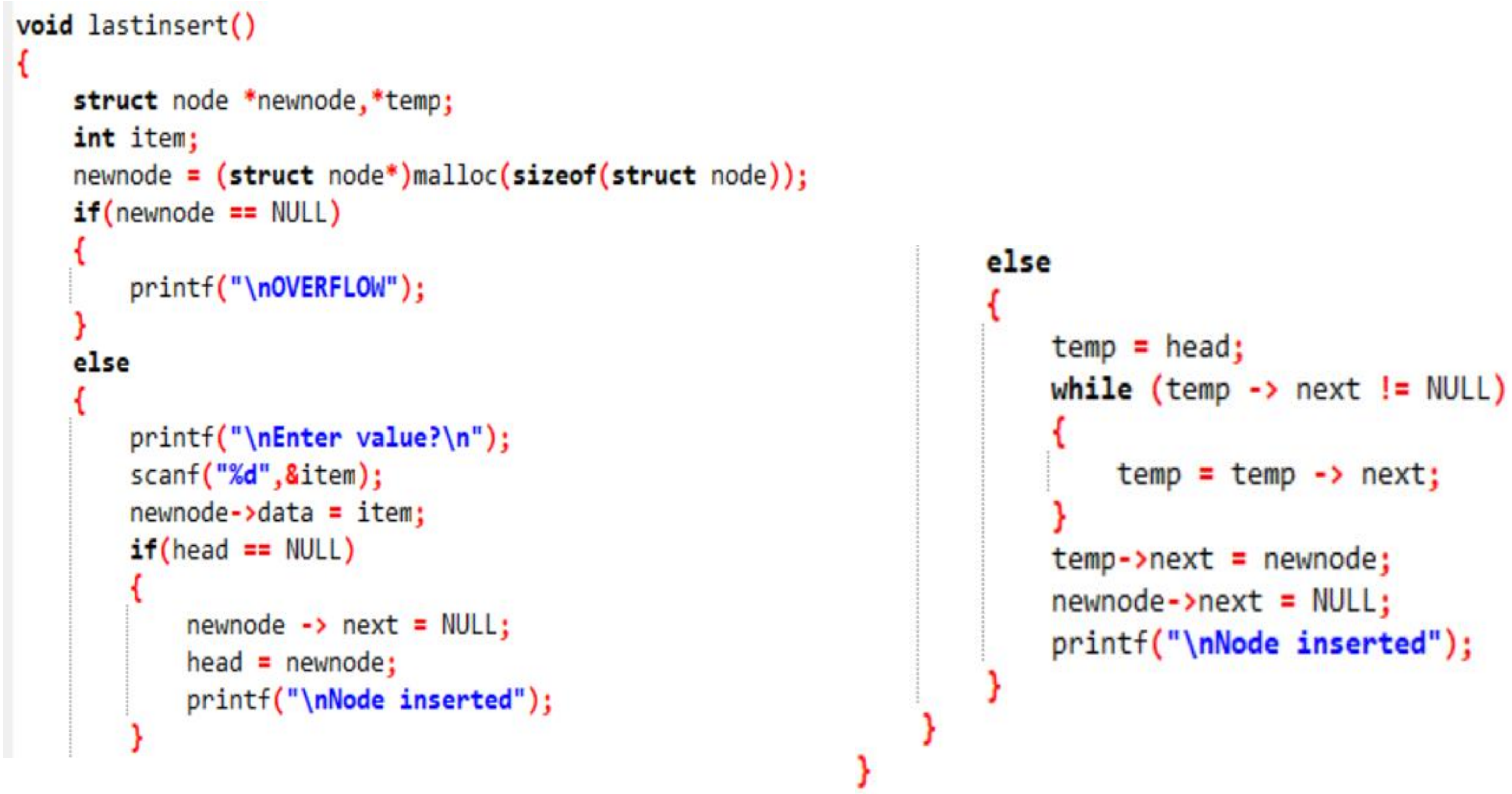
- **Circular Linked List :** In the circular linked list the last node of the list contains the address of the first node and forms a circular chain.



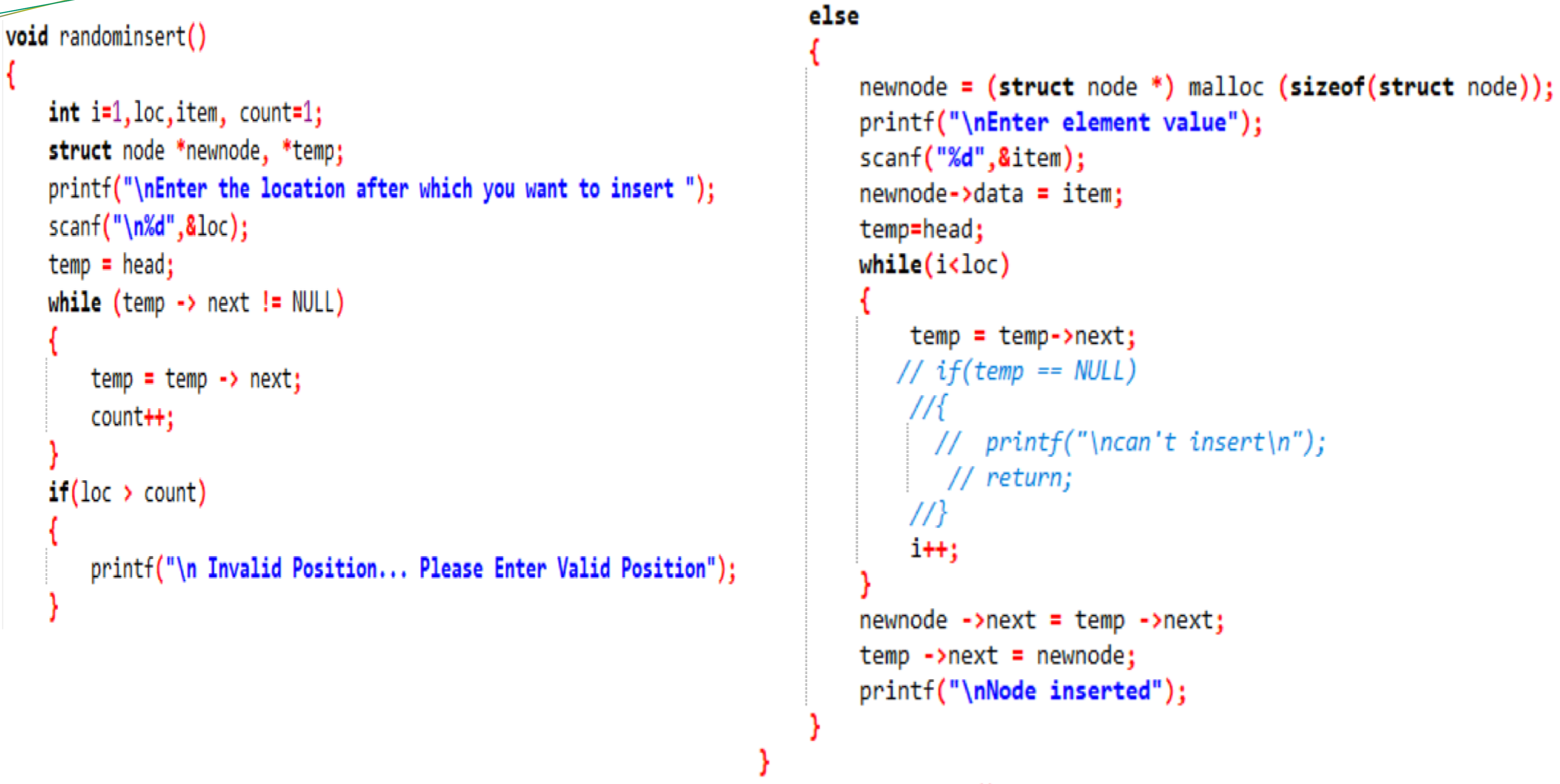
Various Operations:

- Insertion(Beginning/End/positions)
- Finding whether Linked List is empty or not.
- Searching any element in the Linked List
- Deleting a particular Node from the List

```
void beginsert()
{
    struct node *newnode;
    int item;
    newnode = (struct node *) malloc(sizeof(struct node));
    if(newnode == NULL)
    {
        printf("\nOVERFLOW");
    }
    else
    {
        printf("\nEnter value\n");
        scanf("%d",&item);
        newnode->data = item;
        newnode->next = head;
        head = newnode;
        printf("\nNode inserted");
    }
}
```



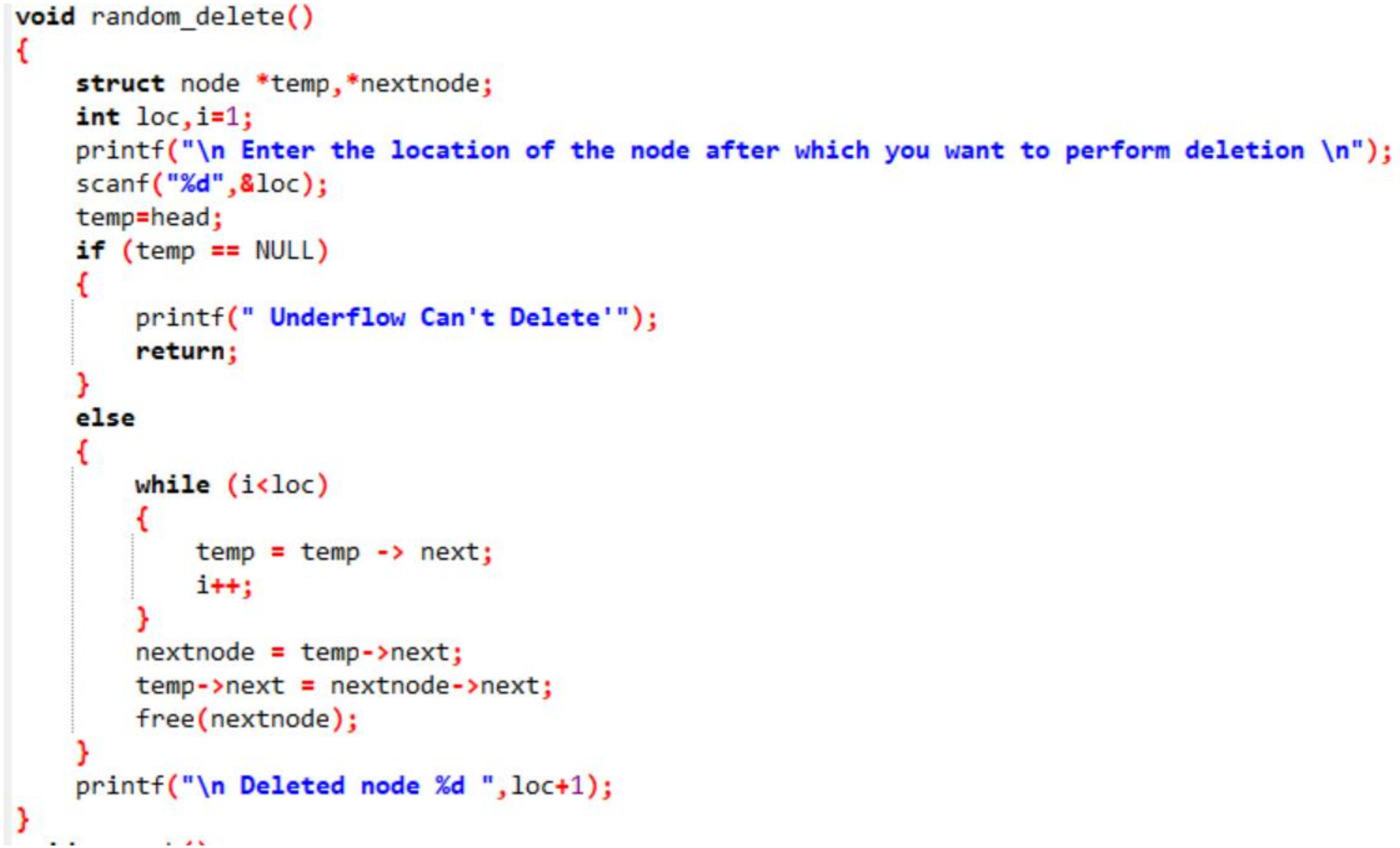
```c
void lastinsert()
{
    struct node *newnode,*temp;
    int item;
    newnode = (struct node*)malloc(sizeof(struct node));
    if(newnode == NULL)
    {
        printf("\nOVERFLOW");
    }
    else
    {
        printf("\nEnter value?\n");
        scanf("%d",&item);
        newnode->data = item;
        if(head == NULL)
        {
            newnode -> next = NULL;
            head = newnode;
            printf("\nNode inserted");
        }
        else
        {
            temp = head;
            while (temp -> next != NULL)
            {
                temp = temp -> next;
            }
            temp->next = newnode;
            newnode->next = NULL;
            printf("\nNode inserted");
        }
    }
}
```



```c
void randominsert()
{
    int i=1,loc,item, count=1;
    struct node *newnode, *temp;
    printf("\nEnter the location after which you want to insert ");
    scanf("\n%d",&loc);
    temp = head;
    while (temp -> next != NULL)
    {
        temp = temp -> next;
        count++;
    }
    if(loc > count)
    {
        printf("\n Invalid Position... Please Enter Valid Position");
    }
    else
    {
        newnode = (struct node *) malloc (sizeof(struct node));
        printf("\nEnter element value");
        scanf("%d",&item);
        newnode->data = item;
        temp=head;
        while(i<loc)
        {
            temp = temp->next;
          // if(temp == NULL)
           //{
             //  printf("\ncan't insert\n");
              // return;
           //}
           i++;
        }
        newnode ->next = temp ->next;
        temp ->next = newnode;
        printf("\nNode inserted");
    }
}
```

```
void begin_delete()
{
    struct node *temp;
    if(head == NULL)
    {
        printf("\n List is empty \n");
    }
    else
    {
        temp = head;
        head = temp->next;
        free(temp);
        printf("\n Node deleted from the begining ...\n");
    }
}
```

```
void last_delete()
{
    struct node *temp,*prevnode;
    if(head == NULL)
    {
        printf("\n List is empty");
    }
    else if(head -> next == NULL)
    {
        head = NULL;
        free(head);
        printf("\n Only node of the list deleted ...\n");
    }
    else
    {
        temp = head;
        while(temp->next != NULL)
        {
            prevnode = temp;
            temp = temp ->next;
        }
        prevnode->next = NULL;
        free(temp);
        printf("\n Deleted Node from the last ...\n");
    }
}
```



```c
void random_delete()
{
    struct node *temp,*nextnode;
    int loc,i=1;
    printf("\n Enter the location of the node after which you want to perform deletion \n");
    scanf("%d",&loc);
    temp=head;
    if (temp == NULL)
    {
        printf(" Underflow Can't Delete'");
        return;
    }
    else
    {
        while (i<loc)
        {
            temp = temp -> next;
            i++;
        }
        nextnode = temp->next;
        temp->next = nextnode->next;
        free(nextnode);
    }
    printf("\n Deleted node %d ",loc+1);
}
```

Thank You.