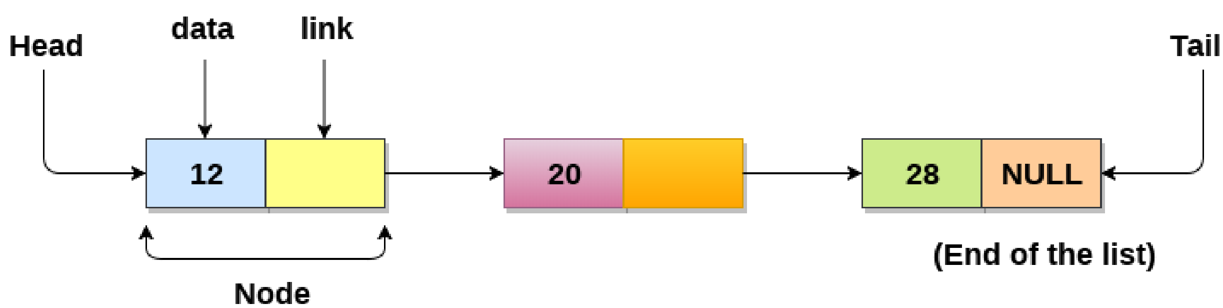
Linked List

- Linked List can be defined as collection of objects called **nodes** that are randomly stored in the memory.
- A node contains two fields i.e. data stored at that particular address and the pointer which contains the address of the next node in the memory.
- The last node of the list contains pointer to the null.



Uses of Linked List

- The list is not required to be contiguously present in the memory. The node can reside any where in the memory and linked together to make a list. This achieves optimized utilization of space.
- list size is limited to the memory size and doesn't need to be declared in advance.
- Empty node can not be present in the linked list.
- We can store values of primitive types or objects in the singly linked list.

Why use linked list over array?

Till now, we were using array data structure to organize the group of elements that are to be stored individually in the memory. However, Array has several advantages and disadvantages which must be known in order to decide the data structure which will be used throughout the program.

Array contains following limitations:

1. The size of array must be known in advance before using it in the program.

2. Increasing size of the array is a time taking process. It is almost impossible to expand the size of the array at run time.
3. 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 the data structure which can overcome all the limitations of an array. Using linked list is useful because,

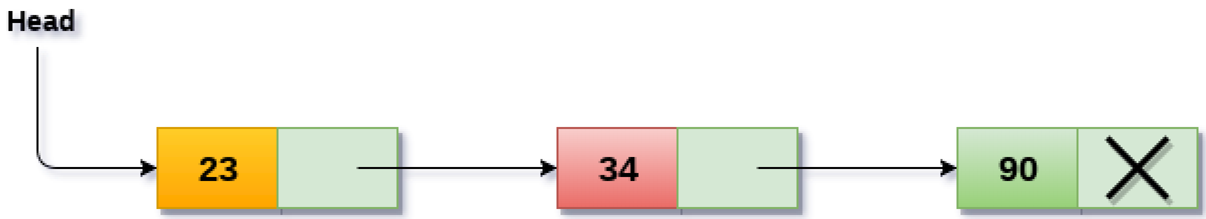
1. It allocates the memory dynamically. All the nodes of linked list are non-contiguously stored in the memory and linked together with the help of pointers.
2. Sizing is no longer a problem since we do not need to define its size at the time of declaration. List grows as per the program's demand and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

Singly linked list or One way chain

Singly linked list can be defined as the collection of ordered set of elements. The number of elements may vary according to need of the program. A node in the singly linked list consist of two parts: data part and link part. Data part of the node stores actual information that is to be represented by the node while the link part of the node stores the address of its immediate successor.

One way chain or singly linked list can be traversed only in one direction. In other words, we can say that each node contains only next pointer, therefore we can not traverse the list in the reverse direction.

Consider an example where the marks obtained by the student in three subjects are stored in a linked list as shown in the figure.



In the above figure, the arrow represents the links. The data part of every node contains the marks obtained by the student in the different subject. The last node in the list is identified by the null pointer which is present in the address part of the last node. We can have as many elements we require, in the data part of the list.

Complexity

Data Structure	Time Complexity							
	Average				**Worst**			
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion
Singly Linked List	θ(n)	θ(n)	θ(1)	θ(1)	O(n)	O(n)	O(1)	O(1)

Operations on Singly Linked List

There are various operations which can be performed on singly linked list. A list of all such operations is given below.

Node Creation

```
1. struct node
2. {
3.     int data;
4.     struct node *next;
5. };
6. struct node *head, *ptr;
7. ptr = (struct node *)malloc(sizeof(struct node *));
```

Insertion

The insertion into a singly linked list can be performed at different positions. Based on the position of the new node being inserted, the insertion is categorized into the following categories.

SN	Operation	Description
1	Insertion at beginning	It involves inserting any element at the front of the list. We just need to a to make the new node as the head of the list.
2	Insertion at end of the list	It involves insertion at the last of the linked list. The new node can be node in the list or it can be inserted as the last one. Different logics are i scenario.
3	Insertion after specified node	It involves insertion after the specified node of the linked list. We need number of nodes in order to reach the node after which the new node will

Insertion in singly linked list at beginning

Inserting a new element into a singly linked list at beginning is quite simple. We just need to make a few adjustments in the node links. There are the following steps which need to be followed in order to inser a new node in the list at beginning.

- Allocate the space for the new node and store data into the data part of the node. This will be done by the following statements.

1. ptr = (struct node *) malloc(sizeof(struct node *));
2. ptr → data = item

- Make the link part of the new node pointing to the existing first node of the list. This will be done by using the following statement.

1. ptr->next = head;

- At the last, we need to make the new node as the first node of the list this will be done by using the following statement.

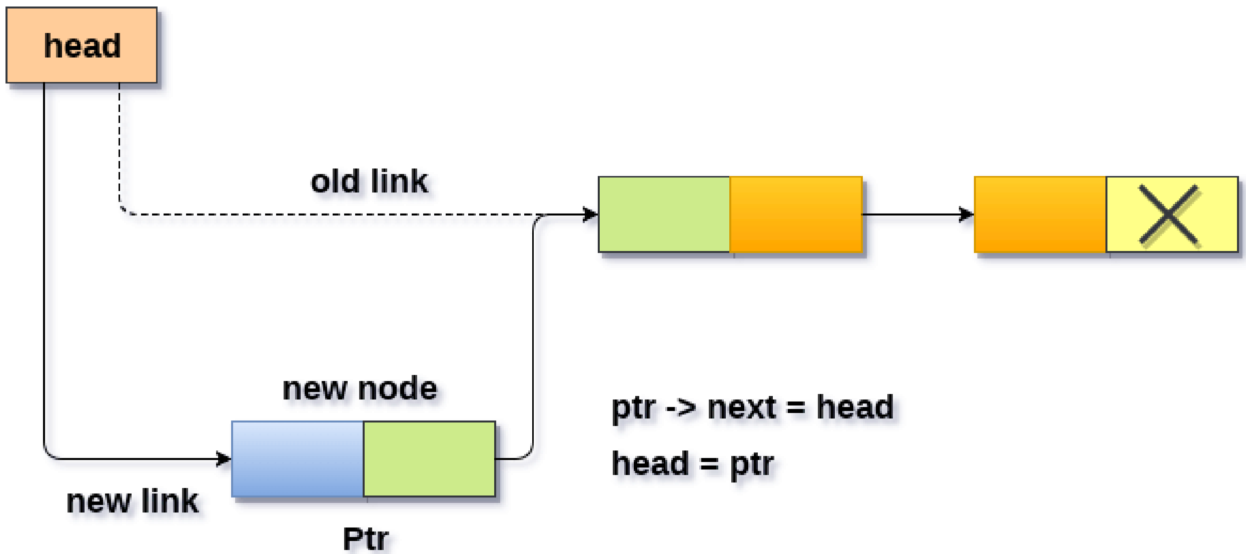
1. head = ptr;

Algorithm

- **Step 1:** IF PTR = NULL

  Write OVERFLOW
  Go to Step 7
  [END OF IF]

- **Step 2:** SET NEW_NODE = PTR
- **Step 3:** SET PTR = PTR → NEXT
- **Step 4:** SET NEW_NODE → DATA = VAL
- **Step 5:** SET NEW_NODE → NEXT = HEAD
- **Step 6:** SET HEAD = NEW_NODE
- **Step 7:** EXIT



C Function

```
#include<stdio.h>
#include<stdlib.h>
void beginsert(int);
struct node
{
   int data;
   struct node *next;
};
struct node *head;
void main ()
{
   int choice,item;
   do
   {
      printf("\nEnter the item which you want to insert?\n");
      scanf("%d",&item);
```

```
        beginsert(item);
        printf("\nPress 0 to insert more ?\n");
        scanf("%d",&choice);
    }while(choice == 0);
}
void beginsert(int item)
    {
        struct node *ptr = (struct node *)malloc(sizeof(struct node *));
        if(ptr == NULL)
        {
            printf("\nOVERFLOW\n");
        }
        else
        {
            ptr->data = item;
            ptr->next = head;
            head = ptr;
            printf("\nNode inserted\n");
        }

    }

```

Insertion in singly linked list at the end

In order to insert a node at the last, there are two following scenarios which need to be mentioned.

1. The node is being added to an empty list
2. The node is being added to the end of the linked list

in the first case,

- The condition (head == NULL) gets satisfied. Hence, we just need to allocate the space for the node by using malloc statement in C. Data and the link part of the node are set up by using the following statements.

1. ptr->data = item;
2. ptr -> next = NULL;

- Since, **ptr** is the only node that will be inserted in the list hence, we need to make this node pointed by the head pointer of the list. This will be done by using the following Statements.

1. Head = ptr

In the second case,

- The condition **Head = NULL** would fail, since Head is not null. Now, we need to declare a temporary pointer temp in order to traverse through the list. **temp** is made to point the first node of the list.

1. Temp = head

- Then, traverse through the entire linked list using the statements:

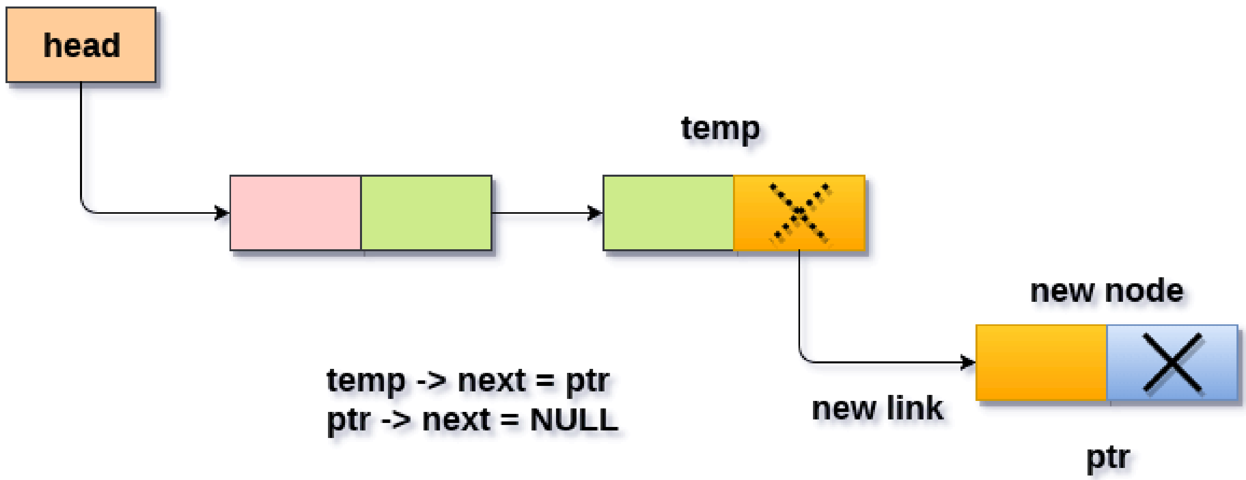
1. **while** (temp→ next != NULL)
2. temp = temp → next;

- At the end of the loop, the temp will be pointing to the last node of the list. Now, allocate the space for the new node, and assign the item to its data part. Since, the new node is going to be the last node of the list hence, the next part of this node needs to be pointing to the **null**. We need to make the next part of the temp node (which is currently the last node of the list) point to the new node (ptr) .

```
temp = head;
      while (temp -> next != NULL)
      {
         temp = temp -> next;
      }
      temp->next = ptr;
      ptr->next = NULL;
```

Algorithm

- **Step 1:** IF PTR = NULL Write OVERFLOW Go to Step 1
  [END OF IF]
- **Step 2:** SET NEW_NODE = PTR
- **Step 3:** SET PTR = PTR - > NEXT
- **Step 4:** SET NEW_NODE - > DATA = VAL
- **Step 5:** SET NEW_NODE - > NEXT = NULL
- **Step 6:** SET PTR = HEAD
- **Step 7:** Repeat Step 8 while PTR - > NEXT != NULL
- **Step 8:** SET PTR = PTR - > NEXT
  [END OF LOOP]
- **Step 9:** SET PTR - > NEXT = NEW_NODE
- **Step 10:** EXIT



Inserting node at the last into a non-empty list

C Function

```
#include<stdio.h>
#include<stdlib.h>
void lastinsert(int);
struct node
{
    int data;
    struct node *next;
};
struct node *head;
void main ()
{
    int choice,item;
    do
    {
        printf("\nEnter the item which you want to insert?\n");
```

```
        scanf("%d",&item);
        lastinsert(item);
        printf("\nPress 0 to insert more ?\n");
        scanf("%d",&choice);
    }while(choice == 0);
}
void lastinsert(int item)
    {
        struct node *ptr = (struct node*)malloc(sizeof(struct node));
        struct node *temp;
        if(ptr == NULL)
        {
            printf("\nOVERFLOW");
        }
        else
        {
            ptr->data = item;
            if(head == NULL)
            {
                ptr -> next = NULL;
                head = ptr;
                printf("\nNode inserted");
            }
            else
            {
                temp = head;
                while (temp -> next != NULL)
                {
                    temp = temp -> next;
```

```
                }
                temp->next = ptr;
                ptr->next = NULL;
                printf("\nNode inserted");

            }
        }
    }
```

Output

```

Enter the item which you want to insert?
12

Node inserted
Press 0 to insert more ?
0

Enter the item which you want to insert?
23

Node inserted
Press 0 to insert more ?
2

```

Insertion in singly linked list after specified Node

- In order to insert an element after the specified number of nodes into the linked list, we need to skip the desired number of elements in the list to move the pointer at the position after which the node will be inserted. This will be done by using the following statements.

```
emp=head;
        for(i=0; i<loc; i++)
        {
            temp = temp->next;
            if(temp == NULL)
            {
                return;
            }

        }
```

- Allocate the space for the new node and add the item to the data part of it. This will be done by using the following statements.

```
ptr = (struct node *) malloc (sizeof(struct node));
      ptr->data = item;
```

- Now, we just need to make a few more link adjustments and our node at will be inserted at the specified position. Since, at the end of the loop, the loop pointer temp would be pointing to the node after which the new node will be inserted. Therefore, the next part of the new node ptr must contain the address of the next part of the temp (since, ptr will be in between temp and the next of the temp). This will be done by using the following statements.

```
ptr→ next = temp → next
```

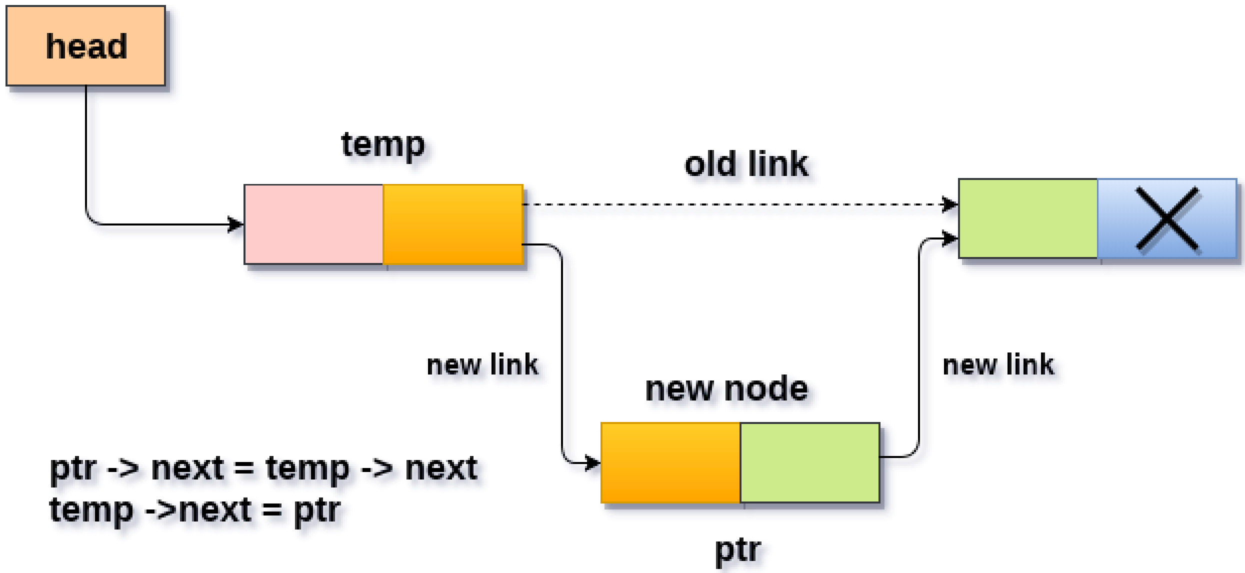
now, we just need to make the next part of the temp, point to the new node ptr. This will insert the new node ptr, at the specified position.

1. temp ->next = ptr;

Algorithm

- **STEP 1:** IF PTR = NULL
  WRITE OVERFLOW
  GOTO STEP 12
  END OF IF
- **STEP 2:** SET NEW_NODE = PTR
- **STEP 3:** NEW_NODE → DATA = VAL
- **STEP 4:** SET TEMP = HEAD
- **STEP 5:** SET I = 0
- **STEP 6:** REPEAT STEP 5 AND 6 UNTIL I
- **STEP 7:** TEMP = TEMP → NEXT
- **STEP 8:** IF TEMP = NULL

  WRITE "DESIRED NODE NOT PRESENT"
  GOTO STEP 12
  END OF IF
  END OF LOOP
- **STEP 9:** PTR → NEXT = TEMP → NEXT
- **STEP 10:** TEMP → NEXT = PTR
- **STEP 11:** SET PTR = NEW_NODE
- **STEP 12:** EXIT



C Function

```
#include<stdio.h>
#include<stdlib.h>
void randominsert(int);
void create(int);
struct node
{
    int data;
    struct node *next;
};
struct node *head;
void main ()
{
    int choice,item,loc;
    do
    {
        printf("\nEnter the item which you want to insert?\n");
        scanf("%d",&item);
```

```
        if(head == NULL)
        {
            create(item);
        }
        else
        {
            randominsert(item);
        }
        printf("\nPress 0 to insert more ?\n");
        scanf("%d",&choice);
    }while(choice == 0);
}
void create(int item)
{

        struct node *ptr = (struct node *)malloc(sizeof(struct node *));
        if(ptr == NULL)
        {
            printf("\nOVERFLOW\n");
        }
        else
        {
            ptr->data = item;
            ptr->next = head;
            head = ptr;
            printf("\nNode inserted\n");
        }
}
void randominsert(int item)
```

```
{
    struct node *ptr = (struct node *) malloc (sizeof(struct node));
    struct node *temp;
    int i,loc;
    if(ptr == NULL)
    {
        printf("\nOVERFLOW");
    }
    else
    {

        printf("Enter the location");
        scanf("%d",&loc);
        ptr->data = item;
        temp=head;
        for(i=0;i<loc;i++)
        {
            temp = temp->next;
            if(temp == NULL)
            {
                printf("\ncan't insert\n");
                return;
            }

        }
        ptr ->next = temp ->next;
        temp ->next = ptr;
        printf("\nNode inserted");
    }
```

```
    }
Output

Enter the item which you want to insert?
12

Node inserted

Press 0 to insert more ?
2
```

Deletion and Traversing

The Deletion of a node from a singly linked list can be performed at different positions. Based on the position of the node being deleted, the operation is categorized into the following categories.

SN	Operation	Description
1	Deletion at beginning	It involves deletion of a node from the beginning of the list. This is the simplest operation among all. It just need a few adjustments in the node pointers.
2	Deletion at the end of the list	It involves deleting the last node of the list. The list can either be empty or full. Different logic is implemented for the different scenarios.
3	Deletion after specified node	It involves deleting the node after the specified node in the list. we need to skip the desired number of nodes to reach the node after which the node will be deleted. This requires traversing through the list.
4	Traversing	In traversing, we simply visit each node of the list at least once in order to perform some specific operation on it, for example, printing data part of each node present in the list.
5	Searching	In searching, we match each element of the list with the given element. If the element is found on any of the location then location of that element is returned otherwise null is returned. .

Deletion in singly linked list at beginning

Deleting a node from the beginning of the list is the simplest operation of all. It just need a few adjustments in the node pointers. Since the first node of the list is to be deleted, therefore, we just need to make the head, point to the next of the head. This will be done by using the following statements.

```
1. ptr = head;
2.      head = ptr->next;
```

Now, free the pointer ptr which was pointing to the head node of the list. This will be done by using the following statement.

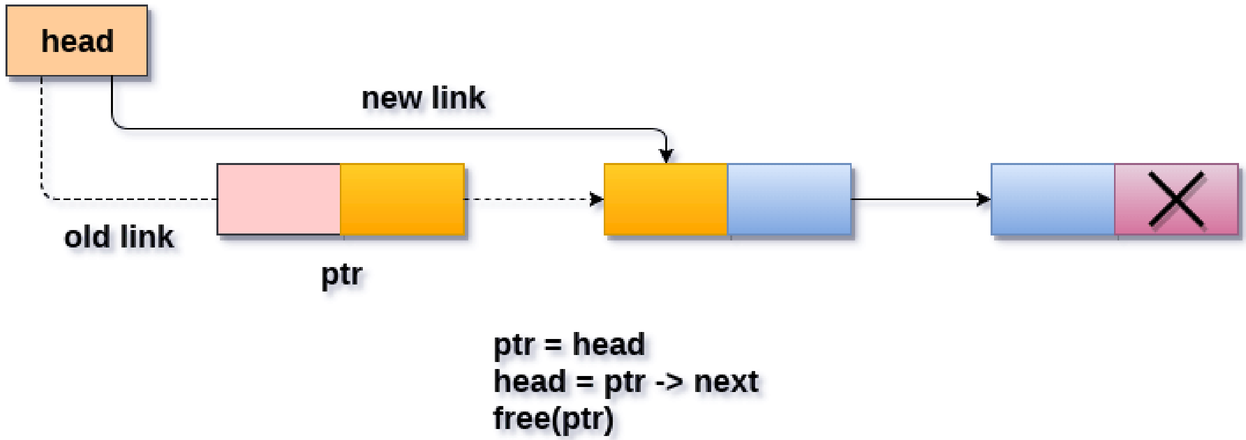
```
1. free(ptr)
```

Algorithm

- **Step 1:** IF HEAD = NULL

  Write UNDERFLOW
  Go to Step 5
  [END OF IF]

- **Step 2:** SET PTR = HEAD
- **Step 3:** SET HEAD = HEAD -> NEXT
- **Step 4:** FREE PTR
- **Step 5:** EXIT



Deleting a node from the beginning

C function

```
#include<stdio.h>
#include<stdlib.h>
void create(int);
void begdelete();
struct node
{
    int data;
    struct node *next;
};
struct node *head;
void main ()
{
    int choice,item;
    do
    {
        printf("\n1.Append List\n2.Delete node\n3.Exit\n4.Enter your choice?");
```

```
        scanf("%d",&choice);
        switch(choice)
        {
          case 1:
          printf("\nEnter the item\n");
          scanf("%d",&item);
          create(item);
          break;
          case 2:
          begdelete();
          break;
          case 3:
          exit(0);
          break;
          default:
          printf("\nPlease enter valid choice\n");
        }

      }while(choice != 3);
    }
    void create(int item)
      {
        struct node *ptr = (struct node *)malloc(sizeof(struct node *));
        if(ptr == NULL)
        {
          printf("\nOVERFLOW\n");
        }
        else
        {
```

```
            ptr->data = item;
            ptr->next = head;
            head = ptr;
            printf("\nNode inserted\n");
        }

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

Output

```
1.Append List
2.Delete node
3.Exit
4.Enter your choice?1
```

```
Enter the item
23

Node inserted

1.Append List
2.Delete node
3.Exit
4.Enter your choice?2

 Node deleted from the begining ...

```

Deletion in singly linked list at the end

There are two scenarios in which, a node is deleted from the end of the linked list.

1. There is only one node in the list and that needs to be deleted.
2. There are more than one node in the list and the last node of the list will be deleted.

In the first scenario,

the condition head → next = NULL will survive and therefore, the only node head of the list will be assigned to null. This will be done by using the following statements.

```
1. ptr = head
2.      head = NULL
3.      free(ptr)
```

In the second scenario,

The condition head → next = NULL would fail and therefore, we have to traverse the node in order to reach the last node of the list.

For this purpose, just declare a temporary pointer temp and assign it to head of the list. We also need to keep track of the second last node of the list. For this purpose, two pointers ptr and ptr1 will be used where ptr will point to the last node and ptr1 will point to the second last node of the list.

this all will be done by using the following statements.

```
1. ptr = head;
2.          while(ptr->next != NULL)
3.          {
4.            ptr1 = ptr;
5.            ptr = ptr ->next;
6.          }
```

Now, we just need to make the pointer ptr1 point to the NULL and the last node of the list that is pointed by ptr will become free. It will be done by using the following statements.

1. ptr1->next = NULL;
2. free(ptr);

Algorithm

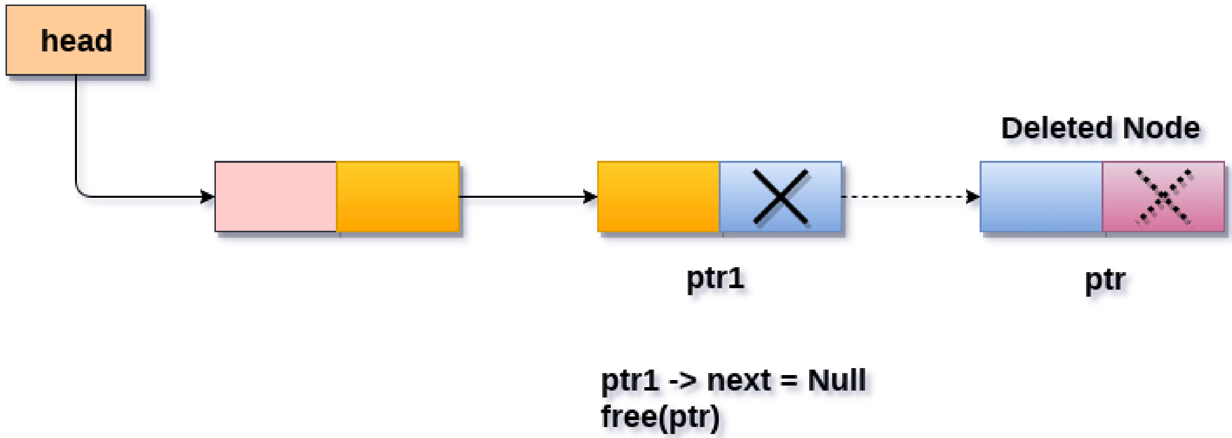
- **Step 1:** IF HEAD = NULL

  Write UNDERFLOW
  Go to Step 8
  [END OF IF]

- **Step 2:** SET PTR = HEAD
- **Step 3:** Repeat Steps 4 and 5 while PTR -> NEXT!= NULL
- **Step 4:** SET PREPTR = PTR
- **Step 5:** SET PTR = PTR -> NEXT

  [END OF LOOP]

- **Step 6:** SET PREPTR -> NEXT = NULL
- **Step 7:** FREE PTR
- **Step 8:** EXIT



Deleting a node from the last

C Function :

```
#include<stdio.h>
#include<stdlib.h>
void create(int);
void end_delete();
struct node
{
    int data;
    struct node *next;
};
struct node *head;
void main ()
{
    int choice,item;
    do
    {
        printf("\n1.Append List\n2.Delete node\n3.Exit\n4.Enter your choice?");
```

```
        scanf("%d",&choice);
        switch(choice)
        {
          case 1:
          printf("\nEnter the item\n");
          scanf("%d",&item);
          create(item);
          break;
          case 2:
          end_delete();
          break;
          case 3:
          exit(0);
          break;
          default:
          printf("\nPlease enter valid choice\n");
        }

    }while(choice != 3);
}
void create(int item)
    {
        struct node *ptr = (struct node *)malloc(sizeof(struct node *));
        if(ptr == NULL)
        {
          printf("\nOVERFLOW\n");
        }
        else
        {
```

```
            ptr->data = item;
            ptr->next = head;
            head = ptr;
            printf("\nNode inserted\n");
        }

    }
void end_delete()
    {
        struct node *ptr,*ptr1;
        if(head == NULL)
        {
            printf("\nlist is empty");
        }
        else if(head -> next == NULL)
        {
            head = NULL;
            free(head);
            printf("\nOnly node of the list deleted ...");
        }

        else
        {
            ptr = head;
            while(ptr->next != NULL)
                {
                    ptr1 = ptr;
                    ptr = ptr ->next;
                }
```

```
                ptr1->next = NULL;
                free(ptr);
                printf("\n Deleted Node from the last ...");
            }
        }
```

Output

```

1.Append List
2.Delete node
3.Exit
4.Enter your choice?1

Enter the item
12

Node inserted

1.Append List
2.Delete node
3.Exit
4.Enter your choice?2

Only node of the list deleted ...
```

Deletion in singly linked list after the specified node :

In order to delete the node, which is present after the specified node, we need to skip the desired number of nodes to reach the node after which the node will be deleted. We need to keep track of the two nodes. The one which is to be deleted the other one if the node which is present before that node. For this purpose, two pointers are used: ptr and ptr1.

Use the following statements to do so.

```
ptr=head;
    for(i=0;i<loc;i++)
    {
      ptr1 = ptr;
       ptr = ptr->next;

       if(ptr == NULL)
      {
          printf("\nThere are less than %d elements in the list..",loc);
         return;
       }
    }
```

Now, our task is almost done, we just need to make a few pointer adjustments. Make the next of ptr1 (points to the specified node) point to the next of ptr (the node which is to be deleted).

This will be done by using the following statements.

```
ptr1 ->next = ptr ->next;
    free(ptr);
```

Algorithm

- **STEP 1:** IF HEAD = NULL

WRITE UNDERFLOW GOTO STEP 10

END OF IF

- **STEP 2:** SET TEMP = HEAD
- **STEP 3:** SET I = 0
- **STEP 4:** REPEAT STEP 5 TO 8 UNTIL I
- **STEP 5:** TEMP1 = TEMP
- **STEP 6:** TEMP = TEMP → NEXT
- **STEP 7:** IF TEMP = NULL

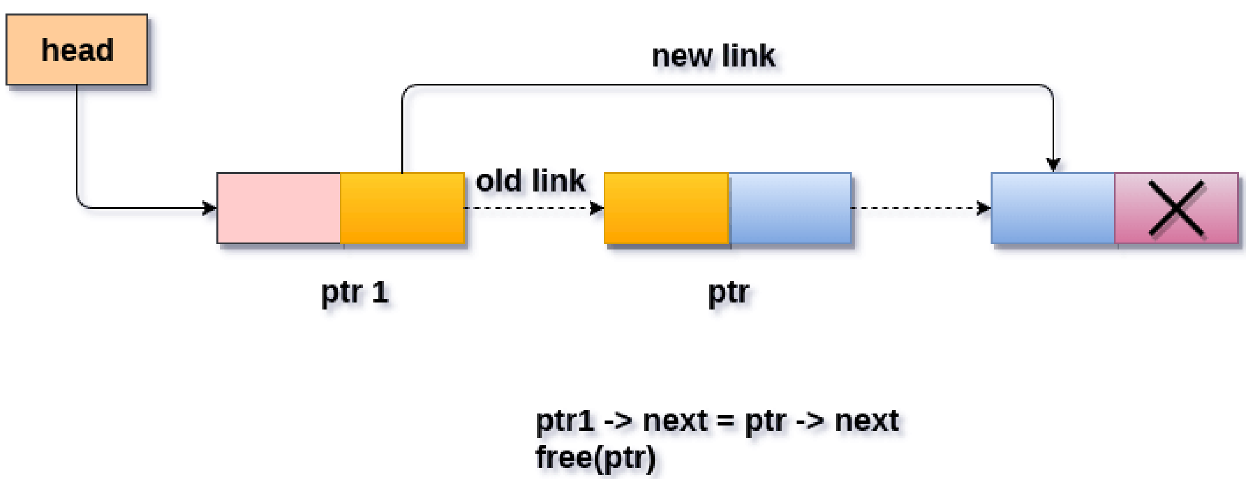
WRITE "DESIRED NODE NOT PRESENT" GOTO STEP 12

END OF IF

- **STEP 8:** I = I+1

END OF LOOP

- **STEP 9:** TEMP1 → NEXT = TEMP → NEXT
- **STEP 10:** FREE TEMP
- **STEP 11:** EXIT



Deletion a node from specified position

C function

```
#include<stdio.h>
#include<stdlib.h>
void create(int);
void delete_specified();
```

```
struct node
{
  int data;
  struct node *next;
};
struct node *head;
void main ()
{
  int choice,item;
  do
  {
    printf("\n1.Append List\n2.Delete node\n3.Exit\n4.Enter your choice?");
    scanf("%d",&choice);
    switch(choice)
    {
      case 1:
      printf("\nEnter the item\n");
      scanf("%d",&item);
      create(item);
      break;
      case 2:
      delete_specified();
      break;
      case 3:
      exit(0);
      break;
      default:
      printf("\nPlease enter valid choice\n");
    }
```

```
    }while(choice != 3);
}
void create(int item)
    {
        struct node *ptr = (struct node *)malloc(sizeof(struct node *));
        if(ptr == NULL)
        {
            printf("\nOVERFLOW\n");
        }
        else
        {
            ptr->data = item;
            ptr->next = head;
            head = ptr;
            printf("\nNode inserted\n");
        }

    }
void delete_specified()
    {
        struct node *ptr, *ptr1;
        int loc,i;
        scanf("%d",&loc);
        ptr=head;
        for(i=0;i<loc;i++)
        {
            ptr1 = ptr;
            ptr = ptr->next;
```

```
        if(ptr == NULL)
        {
            printf("\nThere are less than %d elements in the list..\n",loc);
            return;
        }
    }
    ptr1 ->next = ptr ->next;
    free(ptr);
    printf("\nDeleted %d node ",loc);
}
```

Output

```
1.Append List
2.Delete node
3.Exit
4.Enter your choice?1

Enter the item
12

Node inserted

1.Append List
2.Delete node
3.Exit
4.Enter your choice?1

Enter the item
```

```
23

Node inserted

1.Append List
2.Delete node
3.Exit
4.Enter your choice?2
12

There are less than 12 elements in the list..

1.Append List
2.Delete node
3.Exit
4.Enter your choice?2
1

Deleted 1 node

```

Traversing in singly linked list

Traversing is the most common operation that is performed in almost every scenario of singly linked list. Traversing means visiting each node of the list once in order to perform some operation on that. This will be done by using the following statements.

```
ptr = head;
    while (ptr!=NULL)
      {
        ptr = ptr -> next;
      }
```

Algorithm

- **STEP 1:** SET PTR = HEAD
- **STEP 2:** IF PTR = NULL

  WRITE "EMPTY LIST"
  GOTO STEP 7
  END OF IF

- **STEP 4:** REPEAT STEP 5 AND 6 UNTIL PTR != NULL
- **STEP 5:** PRINT PTR→ DATA
- **STEP 6:** PTR = PTR → NEXT

  [END OF LOOP]

- **STEP 7:** EXIT

C function

```
#include<stdio.h>
#include<stdlib.h>
void create(int);
void traverse();
struct node
```

```
{
   int data;
   struct node *next;
};
struct node *head;
void main ()
{
   int choice,item;
   do
   {
      printf("\n1.Append List\n2.Traverse\n3.Exit\n4.Enter your choice?");
      scanf("%d",&choice);
      switch(choice)
      {
         case 1:
         printf("\nEnter the item\n");
         scanf("%d",&item);
         create(item);
         break;
         case 2:
         traverse();
         break;
         case 3:
         exit(0);
         break;
         default:
         printf("\nPlease enter valid choice\n");
      }

```

```
    }while(choice != 3);
}
void create(int item)
    {
        struct node *ptr = (struct node *)malloc(sizeof(struct node *));
        if(ptr == NULL)
        {
            printf("\nOVERFLOW\n");
        }
        else
        {
            ptr->data = item;
            ptr->next = head;
            head = ptr;
            printf("\nNode inserted\n");
        }

    }
void traverse()
    {
        struct node *ptr;
        ptr = head;
        if(ptr == NULL)
        {
            printf("Empty list..");
        }
        else
        {
            printf("printing values . . . . .\n");
```

```
        while (ptr!=NULL)
        {
            printf("\n%d",ptr->data);
            ptr = ptr -> next;
        }
    }
}
```

Output

```

1.Append List
2.Traverse
3.Exit
4.Enter your choice?1

Enter the item
23

Node inserted

1.Append List
2.Traverse
3.Exit
4.Enter your choice?1

Enter the item
233

Node inserted

```

```
1.Append List
2.Traverse
3.Exit
4.Enter your choice?2
printing values . . . . .

233
23

```

Searching in singly linked list

Searching is performed in order to find the location of a particular element in the list. Searching any element in the list needs traversing through the list and make the comparison of every element of the list with the specified element. If the element is matched with any of the list element then the location of the element is returned from the function.

Algorithm

- **Step 1:** SET PTR = HEAD
- **Step 2:** Set I = 0
- **STEP 3:** IF PTR = NULL

    WRITE "EMPTY LIST"
    GOTO STEP 8
    END OF IF

- **STEP 4:** REPEAT STEP 5 TO 7 UNTIL PTR != NULL
- **STEP 5:** if ptr → data = item

    write i+1
    End of IF

- **STEP 6:** I = I + 1
- **STEP 7:** PTR = PTR → NEXT

    [END OF LOOP]

- **STEP 8:** EXIT

C function

```
#include<stdio.h>
#include<stdlib.h>
void create(int);
```

```
void search();
struct node
{
    int data;
    struct node *next;
};
struct node *head;
void main ()
{
    int choice,item,loc;
    do
    {
        printf("\n1.Create\n2.Search\n3.Exit\n4.Enter your choice?");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1:
            printf("\nEnter the item\n");
            scanf("%d",&item);
            create(item);
            break;
            case 2:
            search();
            case 3:
            exit(0);
            break;
            default:
            printf("\nPlease enter valid choice\n");
        }
```

```
    }while(choice != 3);
}
    void create(int item)
    {
        struct node *ptr = (struct node *)malloc(sizeof(struct node *));
        if(ptr == NULL)
        {
            printf("\nOVERFLOW\n");
        }
        else
        {
            ptr->data = item;
            ptr->next = head;
            head = ptr;
            printf("\nNode inserted\n");
        }

    }
void search()
{
    struct node *ptr;
    int item,i=0,flag;
    ptr = head;
    if(ptr == NULL)
    {
        printf("\nEmpty List\n");
    }
    else
```

```
    {
      printf("\nEnter item which you want to search?\n");
      scanf("%d",&item);
      while (ptr!=NULL)
      {
        if(ptr->data == item)
        {
          printf("item found at location %d ",i+1);
          flag=0;
        }
        else
        {
          flag=1;
        }
        i++;
        ptr = ptr -> next;
      }
      if(flag==1)
      {
        printf("Item not found\n");
      }
    }

}
```

Output

```
1.Create
2.Search
3.Exit
```

```
4.Enter your choice?1

Enter the item
23

Node inserted

1.Create
2.Search
3.Exit
4.Enter your choice?1

Enter the item
34

Node inserted

1.Create
2.Search
3.Exit
4.Enter your choice?2

Enter item which you want to search?
34
item found at location 1

```

1088

Linked List in C: Menu Driven Program all in one

```c
#include<stdio.h>
#include<stdlib.h>
struct node
{
    int data;
    struct node *next;
};
struct node *head;

void beginsert ();
void lastinsert ();
void randominsert();
void begin_delete();
void last_delete();
void random_delete();
void display();
void search();
void main ()
{
    int choice =0;
    while(choice != 9)
    {
        printf("\n\n*********Main Menu*********\n");
        printf("\nChoose one option from the following list ...\n");
        printf("\n===================================================\n");
        printf("\n1.Insert in begining\n2.Insert at last\n3.Insert at any random location\n4.Delete from
Beginning\n
```

```
        5.Delete from last\n6.Delete node after specified location\n7.Search for an
element\n8.Show\n9.Exit\n");
        printf("\nEnter your choice?\n");
        scanf("\n%d",&choice);
        switch(choice)
        {
            case 1:
            beginsert();
            break;
            case 2:
            lastinsert();
            break;
            case 3:
            randominsert();
            break;
            case 4:
            begin_delete();
            break;
            case 5:
            last_delete();
            break;
            case 6:
            random_delete();
            break;
            case 7:
            search();
            break;
            case 8:
            display();
```

```
            break;
            case 9:
            exit(0);
            break;
            default:
            printf("Please enter valid choice..");
        }
    }
}
void beginsert()
{
    struct node *ptr;
    int item;
    ptr = (struct node *) malloc(sizeof(struct node *));
    if(ptr == NULL)
    {
        printf("\nOVERFLOW");
    }
    else
    {
        printf("\nEnter value\n");
        scanf("%d",&item);
        ptr->data = item;
        ptr->next = head;
        head = ptr;
        printf("\nNode inserted");
    }

}
```

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

```
            printf("\nNode inserted");

        }
    }
}
void randominsert()
{
    int i,loc,item;
    struct node *ptr, *temp;
    ptr = (struct node *) malloc (sizeof(struct node));
    if(ptr == NULL)
    {
        printf("\nOVERFLOW");
    }
    else
    {
        printf("\nEnter element value");
        scanf("%d",&item);
        ptr->data = item;
        printf("\nEnter the location after which you want to insert ");
        scanf("\n%d",&loc);
        temp=head;
        for(i=0;i<loc;i++)
        {
            temp = temp->next;
            if(temp == NULL)
            {
                printf("\ncan't insert\n");
                return;
```

```
            }

        }
        ptr ->next = temp ->next;
        temp ->next = ptr;
        printf("\nNode inserted");
    }
}
void begin_delete()
{
    struct node *ptr;
    if(head == NULL)
    {
        printf("\nList is empty\n");
    }
    else
    {
        ptr = head;
        head = ptr->next;
        free(ptr);
        printf("\nNode deleted from the begining ...\n");
    }
}
void last_delete()
{
    struct node *ptr,*ptr1;
    if(head == NULL)
    {
        printf("\nlist is empty");
```

```
    }
    else if(head -> next == NULL)
    {
        head = NULL;
        free(head);
        printf("\nOnly node of the list deleted ...\n");
    }

    else
    {
        ptr = head;
        while(ptr->next != NULL)
        {
            ptr1 = ptr;
            ptr = ptr ->next;
        }
        ptr1->next = NULL;
        free(ptr);
        printf("\nDeleted Node from the last ...\n");
    }
}
void random_delete()
{
    struct node *ptr,*ptr1;
    int loc,i;
    printf("\n Enter the location of the node after which you want to perform deletion \n");
    scanf("%d",&loc);
    ptr=head;
    for(i=0;i<loc;i++)
```

```
        {
            ptr1 = ptr;
            ptr = ptr->next;

            if(ptr == NULL)
            {
                printf("\nCan't delete");
                return;
            }
        }
        ptr1 ->next = ptr ->next;
        free(ptr);
        printf("\nDeleted node %d ",loc+1);
    }
void search()
{
    struct node *ptr;
    int item,i=0,flag;
    ptr = head;
    if(ptr == NULL)
    {
        printf("\nEmpty List\n");
    }
    else
    {
        printf("\nEnter item which you want to search?\n");
        scanf("%d",&item);
        while (ptr!=NULL)
        {
```

```
            if(ptr->data == item)
            {
                printf("item found at location %d ",i+1);
                flag=0;
            }
            else
            {
                flag=1;
            }
            i++;
            ptr = ptr -> next;
        }
        if(flag==1)
        {
            printf("Item not found\n");
        }
    }

}

void display()
{
    struct node *ptr;
    ptr = head;
    if(ptr == NULL)
    {
        printf("Nothing to print");
    }
    else
```

```
        {
            printf("\nprinting values . . . . .\n");
            while (ptr!=NULL)
            {
                printf("\n%d",ptr->data);
                ptr = ptr -> next;
            }
        }
    }
```

1360