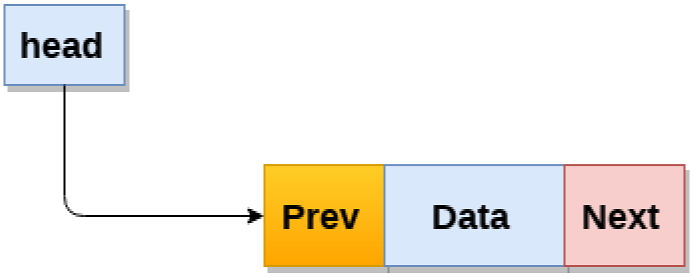
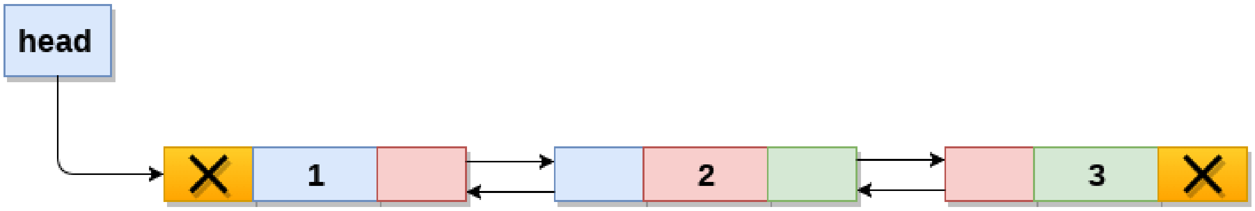
Doubly linked list

Doubly linked list is a complex type of linked list in which a node contains a pointer to the previous as well as the next node in the sequence. Therefore, in a doubly linked list, a node consists of three parts: node data, pointer to the next node in sequence (next pointer) , pointer to the previous node (previous pointer)



Node

A doubly linked list containing three nodes having numbers from 1 to 3 in their data part



Doubly Linked List

In C, structure of a node in doubly linked list can be given as :

```
struct node
{
    struct node *prev;
    int data;
    struct node *next;
}
```

SN	Operation	Description
1	Insertion at beginning	Adding the node into the linked list at beginning.
2	Insertion at end	Adding the node into the linked list to the end.
3	Insertion after specified node	Adding the node into the linked list after the specified node.
4	Deletion at beginning	Removing the node from beginning of the list
5	Deletion at the end	Removing the node from end of the list.
6	Deletion of the node having given data	Removing the node which is present just after the node contair
7	Searching	Comparing each node data with the item to be searched and item in the list if the item found else return null.
8	Traversing	Visiting each node of the list at least once in order to perform s searching, sorting, display, etc.
9	All in one	Create a program for add and use all operation in list.

23

24

Insertion in doubly linked list at beginning

As in doubly linked list, each node of the list contain double pointers therefore we have to maintain more number of pointers in doubly linked list as compare to singly linked list.

There are two scenarios of inserting any element into doubly linked list. Either the list is empty or it contains at least one element. Perform the following steps to insert a node in doubly linked list at beginning.

Allocate the space for the new node in the memory. This will be done by using the following statement.

```
ptr = (struct node *)malloc(sizeof(struct node));
```

Check whether the list is empty or not. The list is empty if the condition head == NULL holds. In that case, the node will be inserted as the only node of the list and therefore the prev and the next pointer of the node will point to NULL and the head pointer will point to this node.

```
ptr->next = NULL;
        ptr->prev=NULL;
        ptr->data=item;
        head=ptr;
```

n the second scenario, the condition head == NULL become false and the node will be inserted in beginning. The next pointer of the node will point to the existing head pointer of the node. The prev pointer of the existing head will point to the new node being inserted.

This will be done by using the following statements.

```
ptr->next = head;
  head→prev=ptr;
```

Since, the node being inserted is the first node of the list and therefore it must contain NULL in its prev pointer. Hence assign null to its previous part and make the head point to this node.

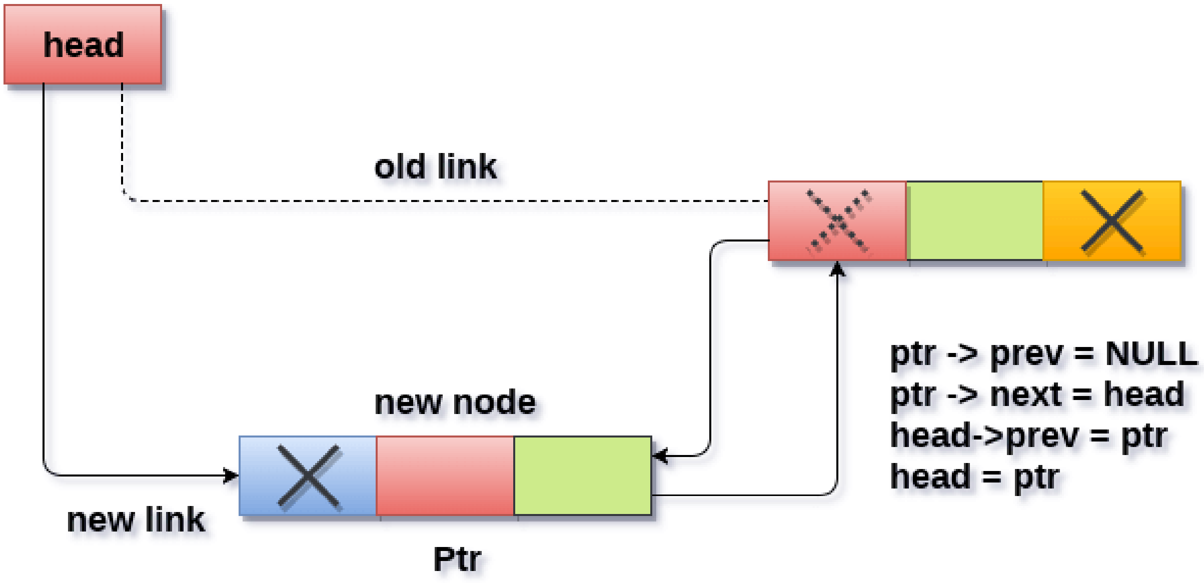
```
ptr→prev =NULL
head = ptr
```

Algorithm :

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

```
        Write OVERFLOW
         Go to Step 9
         [END OF IF]

    o   Step 2: SET NEW_NODE = ptr
    o   Step 3: SET ptr = ptr -> NEXT
    o   Step 4: SET NEW_NODE -> DATA = VAL
    o   Step 5: SET NEW_NODE -> PREV = NULL
    o   Step 6: SET NEW_NODE -> NEXT = START
    o   Step 7: SET head -> PREV = NEW_NODE
    o   Step 8: SET head = NEW_NODE
    o   Step 9: EXIT
```



Insertion into doubly linked list at beginning

C Function

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

```
{
    int data;
    struct node *next;
    struct node *prev;
};
struct node *head;
void main ()
{
    int choice,item;
    do
    {
        printf("\nEnter the item which you want to insert?\n");
        scanf("%d",&item);
        insertbeginning(item);
        printf("\nPress 0 to insert more ?\n");
        scanf("%d",&choice);
    }while(choice == 0);
}
void insertbeginning(int item)
{

   struct node *ptr = (struct node *)malloc(sizeof(struct node));
   if(ptr == NULL)
   {
       printf("\nOVERFLOW");
   }
   else
   {


   if(head==NULL)
   {
      ptr->next = NULL;
      ptr->prev=NULL;
      ptr->data=item;
       head=ptr;
   }
   else
   {
       ptr->data=item;
       ptr->prev=NULL;
       ptr->next = head;
       head->prev=ptr;
       head=ptr;
   }
}

}

```

Insertion in doubly linked list at the end

In order to insert a node in doubly linked list at the end, we must make sure whether the list is empty or it contains any element. Use the following steps in order to insert the node in doubly linked list at the end.

Allocate the memory for the new node. Make the pointer ptr point to the new node being inserted.

```
ptr = (struct node *) malloc(sizeof(struct node));
```

Check whether the list is empty or not. The list is empty if the condition head == NULL holds. In that case, the node will be inserted as the only node of the list and therefore the prev and the next pointer of the node will point to NULL and the head pointer will point to this node.

```
ptr->next = NULL;
   ptr->prev=NULL;
   ptr->data=item;
   head=ptr;
```

In the second scenario, the condition head == NULL become false. The new node will be inserted as the last node of the list. For this purpose, we have to traverse the whole list in order to reach the last node of the list. Initialize the pointer temp to head and traverse the list by using this pointer.

```
Temp = head;
while (temp != NULL)
{
temp = temp → next;
   }
```

the pointer temp point to the last node at the end of this while loop. Now, we just need to make a few pointer adjustments to insert the new node ptr to the list. First, make the next pointer of temp point to the new node being inserted i.e. ptr.

```
temp→next =ptr;
```

make the previous pointer of the node ptr point to the existing last node of the list i.e. temp.

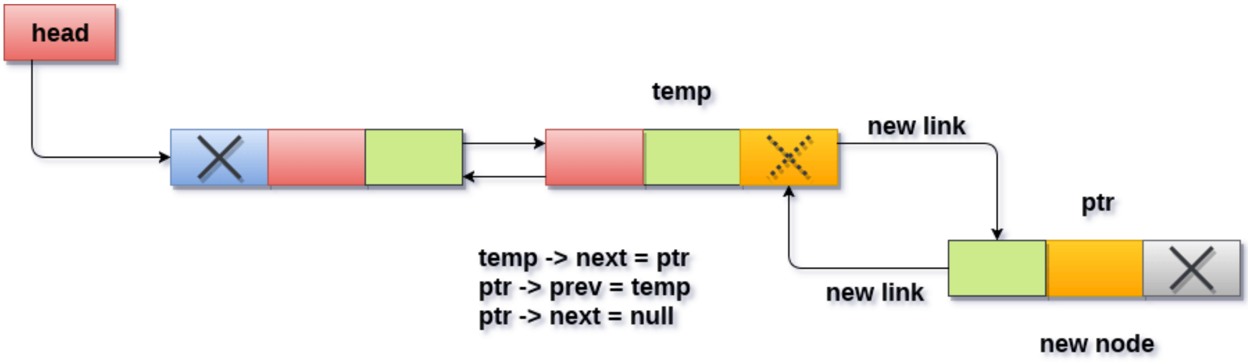
```
ptr → prev = temp;
```

make the next pointer of the node ptr point to the null as it will be the new last node of the list.

```
ptr → next = NULL
```

Algorithm

```
○ Step 1: IF PTR = NULL
      Write OVERFLOW
      Go to Step 11
      [END OF IF]
○ Step 2: SET NEW_NODE = PTR
○ Step 3: SET PTR = PTR -> NEXT// head value null
○ Step 4: SET NEW_NODE -> DATA = VAL
○ Step 5: SET NEW_NODE -> NEXT = NULL
○ Step 6: SET TEMP = START
○ Step 7: Repeat Step 8 while TEMP -> NEXT != NULL
○ Step 8: SET TEMP = TEMP -> NEXT
      [END OF LOOP]
○ Step 9: SET TEMP -> NEXT = NEW_NODE
○ Step 10C: SET NEW_NODE -> PREV = TEMP
○ Step 11: EXIT
```



Insertion into doubly linked list at the end

C Program

```
#include<stdio.h>
#include<stdlib.h>
void insertlast(int);
struct node
{
    int data;
    struct node *next;
    struct node *prev;
};
struct node *head;
void main ()
{
    int choice,item;
    do
    {
        printf("\nEnter the item which you want to insert?\n");
        scanf("%d",&item);
        insertlast(item);
        printf("\nPress 0 to insert more ?\n");
        scanf("%d",&choice);
    }while(choice == 0);
}
void insertlast(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;
           ptr->prev = NULL;
           head = ptr;
       }
       else
       {
          temp = head;
          while(temp->next!=NULL)
          {
              temp = temp->next;
          }
          temp->next = ptr;
          ptr ->prev=temp;
          ptr->next = NULL;
       }
printf("\nNode Inserted\n");

   }
}
```

Insertion in doubly linked list Specified node

In order to insert a node after the specified node in the list, we need to skip the required number of nodes in order to reach the mentioned node and then make the pointer adjustments as required.

Use the following steps for this purpose.

Allocate the memory for the new node. Use the following statements for this.

```
ptr = (struct node *)malloc(sizeof(struct node));
```

Traverse the list by using the pointer temp to skip the required number of nodes in order to reach the specified node.

```
temp=head;
  for(i=0;i<loc;i++)
 {
    temp = temp->next;
    if(temp == NULL) // the temp will be //null if the list doesn't last long //up to mentioned location
    {
        return;
    }
 }
```

The temp would point to the specified node at the end of the for loop. The new node needs to be inserted after this node therefore we need to make a fer pointer adjustments here. Make the next pointer of ptr point to the next node of temp.

```
ptr → next = temp → next;
```

make the prev of the new node ptr point to temp.

```
ptr → prev = temp;
```

make the next pointer of temp point to the new node ptr.

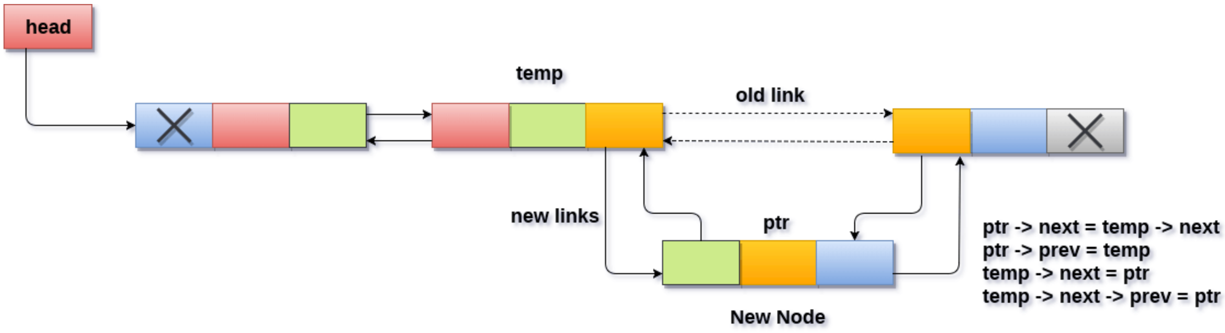
```
temp → next = ptr;
```

make the previous pointer of the next node of temp point to the new node.

```
temp → next → prev = ptr;
```

Algorithm

```
Step 1: IF PTR = NULL
            Write OVERFLOW
            Go to Step 15
        [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 TEMP = START
Step 6: SET I = 0
Step 7: REPEAT 8 to 10 until I<="" li="">
Step 8: SET TEMP = TEMP -> NEXT
STEP 9: IF TEMP = NULL
STEP 10: WRITE "LESS THAN DESIRED NO. OF ELEMENTS"
            GOTO STEP 15
            [END OF IF]
        [END OF LOOP]
Step 11: SET NEW_NODE -> NEXT = TEMP -> NEXT
Step 12: SET NEW_NODE -> PREV = TEMP
Step 13 : SET TEMP -> NEXT = NEW_NODE
Step 14: SET TEMP -> NEXT -> PREV = NEW_NODE
Step 15: EXIT
```



Insertion into doubly linked list after specified node

C Function

```
#include<stdio.h>
#include<stdlib.h>
void insert_specified(int);
void create(int);
struct node
{
    int data;
    struct node *next;
    struct node *prev;
};
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
        {
            insert_specified(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");
   }
   else
   {


   if(head==NULL)
   {
       ptr->next = NULL;
       ptr->prev=NULL;
       ptr->data=item;
       head=ptr;
   }
   else
   {
       ptr->data=item;printf("\nPress 0 to insert more ?\n");
       ptr->prev=NULL;
```

```
        ptr->next = head;
        head->prev=ptr;
        head=ptr;
    }
     printf("\nNode Inserted\n");
}

}
void insert_specified(int item)
{

    struct node *ptr = (struct node *)malloc(sizeof(struct node));
    struct node *temp;
    int i, loc;
    if(ptr == NULL)
    {
        printf("\n OVERFLOW");
    }
    else
    {
        printf("\nEnter the location\n");
        scanf("%d",&loc);
        temp=head;
        for(i=0;i<loc;i++)
        {
            temp = temp->next;
            if(temp == NULL)
            {
                printf("\ncan't insert\n");
                return;
            }
        }
        ptr->data = item;
        ptr->next = temp->next;
        ptr -> prev = temp;
        temp->next = ptr;
        temp->next->prev=ptr;
        printf("Node Inserted\n");
    }
}
```

Deletion at beginning

Deletion in doubly linked list at the beginning is the simplest operation. We just need to copy the head pointer to pointer ptr and shift the head pointer to its next.

```
Ptr = head;
   head = head → next;
```

now make the prev of this new head node point to NULL. This will be done by using the following statements.

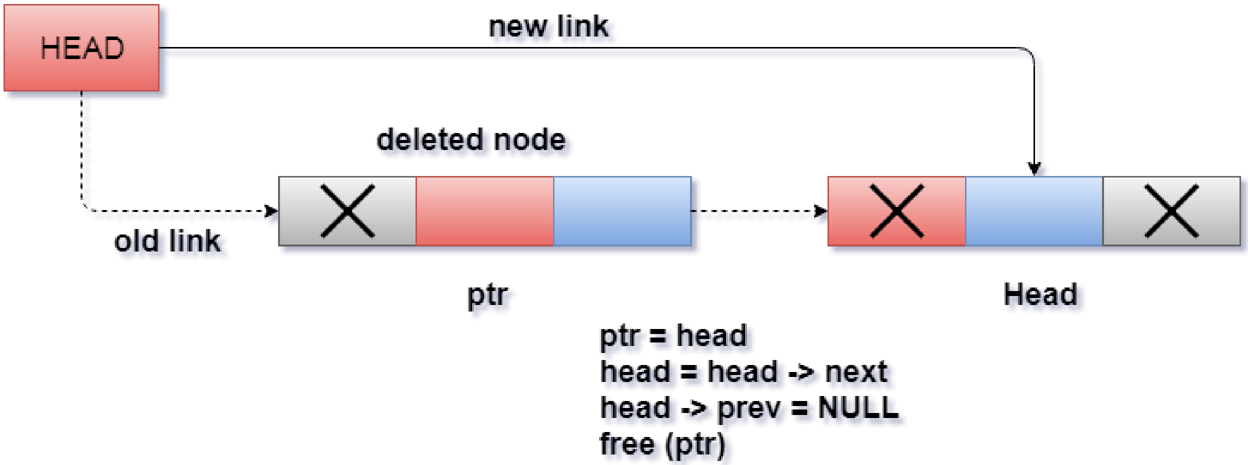
```
head → prev = NULL
```

Now free the pointer ptr by using the free function.

```
free(ptr)
```

Algorithm

```
○ STEP 1: IF HEAD = NULL
      WRITE UNDERFLOW
      GOTO STEP 6
○ STEP 2: SET PTR = HEAD
○ STEP 3: SET HEAD = HEAD → NEXT
○ STEP 4: SET HEAD → PREV = NULL
○ STEP 5: FREE PTR
○ STEP 6: EXIT
```



Deletion in doubly linked list from beginning

C Function

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

struct node
{
   int data;
   struct node *next;
   struct node *prev;
};
struct node *head;
void main ()
{
   int choice,item;
   do
   {
      printf("1.Append List\n2.show list \n3.Delete node from beginning\n4.Exit\n\tEnter your choice?");
      scanf("%d",&choice);
      switch(choice)
      {
         case 1:
                  printf("\nEnter the item\n");
                  scanf("%d",&item);
```

```
                    create(item);
         break;
                                 case 2:
                                            show();
                                 break;
         case 3:
                    deletion_beginning();
         break;
         case 4:
                 exit(0);
         break;
         default:
         printf("\nPlease enter valid choice\n");
      }

   }while(choice != 4);
}
void create(int item)
{

  struct node *ptr = (struct node *)malloc(sizeof(struct node));
  if(ptr == NULL)
  {
     printf("\nOVERFLOW\n");
  }
  else
  {


  if(head==NULL)
  {
     ptr->next = NULL;
     ptr->prev=NULL;
     ptr->data=item;
     head=ptr;
  }
  else
  {
     ptr->data=item;printf("\nPress 0 to insert more ?\n");
     ptr->prev=NULL;
     ptr->next = head;
     head->prev=ptr;
     head=ptr;
  }
   printf("\nNode Inserted\n");
}

}
```

```


void show()
{
        struct node *ptr;
        if(head==NULL)
        {
                printf("\nlist is empty\n");
        }
        else
        {
                ptr=head;
                while(ptr!=NULL)
                {
                        printf("%d\n",ptr->data);
                        ptr=ptr->next;
                }
        }
}

void deletion_beginning()
{
   struct node *ptr;
   if(head == NULL)
   {
      printf("\n UNDERFLOW");
   }
   else if(head->next == NULL)
   {
      head = NULL;
      free(head);
      printf("\nnode deleted\n");
   }
   else
   {
      ptr = head;
      head = head -> next;
      head -> prev = NULL;
      free(ptr);
      printf("\nnode deleted\n");
   }
}

```

Deletion in doubly linked list at the end

Deletion of the last node in a doubly linked list needs traversing the list in order to reach the last node of the list and then make pointer adjustments at that position.

In order to delete the last node of the list, we need to follow the following steps.

If the list is already empty then the condition head == NULL will become true and therefore the operation can not be carried on.

If there is only one node in the list then the condition head → next == NULL become true. In this case, we just need to assign the head of the list to NULL and free head in order to completely delete the list.

Otherwise, just traverse the list to reach the last node of the list. This will be done by using the following statements.

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

- The ptr would point to the last node of the ist at the end of the for loop. Just make the next pointer of the previous node of **ptr** to **NULL**.

1. ptr → prev → next = NULL

free the pointer as this the node which is to be deleted.

1. free(ptr)

Algorithm

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

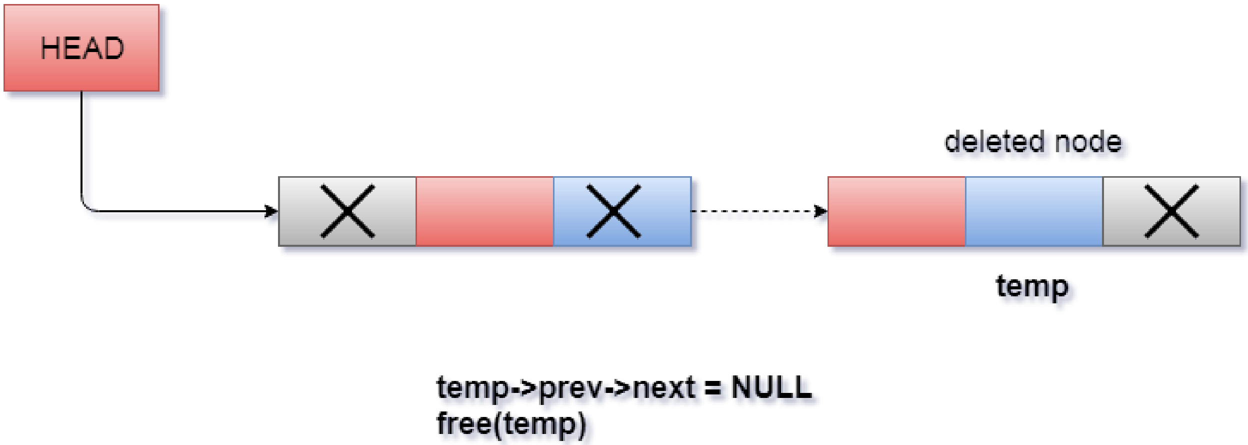
  Write UNDERFLOW
  Go to Step 7
  [END OF IF]

- **Step 2:** SET TEMP = HEAD
- **Step 3:** REPEAT STEP 4 WHILE TEMP->NEXT != NULL

- **Step 4:** SET TEMP = TEMP->NEXT

  [END OF LOOP]

- **Step 5:** SET TEMP ->PREV-> NEXT = NULL
- **Step 6:** FREE TEMP
- **Step 7:** EXIT



Deletion in doubly linked list at the end

C Function

```
#include<stdio.h>
#include<stdlib.h>
void create(int);
void last_delete();
struct node
{
    int data;
    struct node *next;
    struct node *prev;
};
struct node *head;
void main ()
{
    int choice,item;
    do
    {
        printf("1.Append List\n2.Delete node from end\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:
            last_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
   {


   if(head==NULL)
   {
       ptr->next = NULL;
       ptr->prev=NULL;
       ptr->data=item;
       head=ptr;
   }
   else
   {
       ptr->data=item;
       ptr->prev=NULL;
       ptr->next = head;
       head->prev=ptr;
       head=ptr;
   }
    printf("\nNode Inserted\n");
}

}
void last_delete()
{
    struct node *ptr;
    if(head == NULL)
    {
        printf("\n UNDERFLOW\n");
    }
    else if(head->next == NULL)
    {
        head = NULL;
        free(head);
        printf("\nNode Deleted\n");
    }
    else
    {
        ptr = head;
        if(ptr->next != NULL)
        {
            ptr = ptr -> next;
        }
        ptr -> prev -> next = NULL;
        free(ptr);
        printf("\nNode Deleted\n");
    }
}

```

Deletion in doubly linked list after the specified node

In order to delete the node after the specified data, we need to perform the following steps.

Copy the head pointer into a temporary pointer temp.

```
temp = head
```

Traverse the list until we find the desired data value.

```
while(temp -> data != val)
temp = temp -> next;
```

Check if this is the last node of the list. If it is so then we can't perform deletion.

```
if(temp -> next == NULL)
   {
return;
   }
```

Check if the node which is to be deleted, is the last node of the list, if it so then we have to make the next pointer of this node point to null so that it can be the new last node of the list.

```
if(temp -> next -> next == NULL)
   {
       temp ->next = NULL;
   }
```

Otherwise, make the pointer ptr point to the node which is to be deleted. Make the next of temp point to the next of ptr. Make the previous of next node of ptr point to temp. free the ptr.

```
ptr = temp -> next;
       temp -> next = ptr -> next;
       ptr -> next -> prev = temp;
       free(ptr);
```

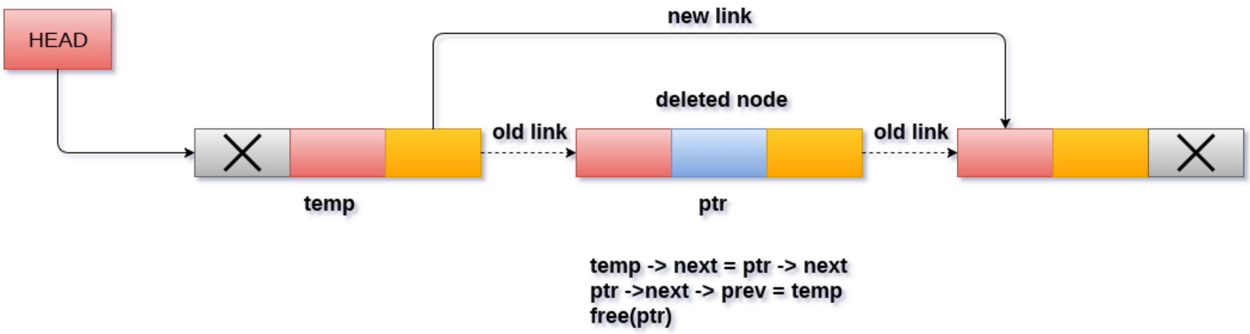
Algorithm

```
   ○ Step 1: IF HEAD = NULL
        Write UNDERFLOW
       Go to Step 9
      [END OF IF]
```

- **Step 2:** SET TEMP = HEAD
- **Step 3:** Repeat Step 4 while TEMP -> DATA != ITEM
- **Step 4:** SET TEMP = TEMP -> NEXT

  [END OF LOOP]

- **Step 5:** SET PTR = TEMP -> NEXT
- **Step 6:** SET TEMP -> NEXT = PTR -> NEXT
- **Step 7:** SET PTR -> NEXT -> PREV = TEMP
- **Step 8:** FREE PTR
- **Step 9:** EXIT



Deletion of a specified node in doubly linked list

C Function

```
#include<stdio.h>
#include<stdlib.h>
void create(int);
void delete_specified();
struct node
{
    int data;
    struct node *next;
    struct node *prev;
};
struct node *head;
void main ()
```

```
{
   int choice,item;
   do
   {
      printf("1.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
  {


  if(head==NULL)
  {
      ptr->next = NULL;
      ptr->prev=NULL;
      ptr->data=item;
      head=ptr;
  }
  else
  {
      ptr->data=item;
      ptr->prev=NULL;
      ptr->next = head;
      head->prev=ptr;
      head=ptr;
  }
   printf("\nNode Inserted\n");
}

}
void delete_specified( )
{
   struct node *ptr, *temp;
   int val;
   printf("Enter the value");
   scanf("%d",&val);
   temp = head;
   while(temp -> data != val)
   temp = temp -> next;
   if(temp -> next == NULL)
   {
       printf("\nCan't delete\n");
   }
   else if(temp -> next -> next == NULL)
   {
       temp ->next = NULL;
       printf("\nNode Deleted\n");
   }
   else
   {
       ptr = temp -> next;
       temp -> next = ptr -> next;
       ptr -> next -> prev = temp;
       free(ptr);
       printf("\nNode Deleted\n");
   }
}
```

Searching for a specific node in Doubly Linked List

We just need traverse the list in order to search for a specific element in the list. Perform following operations in order to search a specific operation.

Copy head pointer into a temporary pointer variable ptr.

ptr = head

declare a local variable I and assign it to 0.

i=0

Traverse the list until the pointer ptr becomes null. Keep shifting pointer to its next and increasing i by +1.

Compare each element of the list with the item which is to be searched.

If the item matched with any node value then the location of that value I will be returned from the function else NULL is returned.

Algorithm

```
○ Step 1: IF HEAD == NULL
    WRITE "UNDERFLOW"
   GOTO STEP 8
   [END OF IF]
○ Step 2: Set PTR = HEAD
○ Step 3: Set i = 0
○ Step 4: Repeat step 5 to 7 while PTR != NULL
○ Step 5: IF PTR → data = item
  return i
  [END OF IF]
○ Step 6: i = i + 1
○ Step 7: PTR = PTR → next
○ 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 *prev;
};
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");
   }
   else
   {


   if(head==NULL)
   {
       ptr->next = NULL;
       ptr->prev=NULL;
       ptr->data=item;
       head=ptr;
   }
   else
   {
       ptr->data=item;printf("\nPress 0 to insert more ?\n");
       ptr->prev=NULL;
       ptr->next = head;
       head->prev=ptr;
       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("\nitem found at location %d ",i+1);
                flag=0;
                break;
            }
            else
            {
                flag=1;
            }
            i++;
            ptr = ptr -> next;
        }
        if(flag==1)
        {
            printf("\nItem not found\n");
        }
    }
}
```

Traversing in doubly linked list

raversing is the most common operation in case of each data structure. For this purpose, copy the head pointer in any of the temporary pointer ptr.

Ptr = head

then, traverse through the list by using while loop. Keep shifting value of pointer variable ptr until we find the last node. The last node contains null in its next part.

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

Although, traversing means visiting each node of the list once to perform some specific operation. Here, we are printing the data associated with each node of the list.

Algorithm

```
○ Step 1: IF HEAD == NULL
     WRITE "UNDERFLOW"
     GOTO STEP 6
     [END OF IF]
○ Step 2: Set PTR = HEAD
○ Step 3: Repeat step 4 and 5 while PTR != NULL
○ Step 4: Write PTR → data
○ Step 5: PTR = PTR → next
○ Step 6: Exit
```

C Function

```
#include<stdio.h>
#include<stdlib.h>
void create(int);
int traverse();
struct node
{
   int data;
   struct node *next;
   struct node *prev;
};
struct node *head;
void main ()
{
   int choice,item;
   do
```

```
    {
        printf("1.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
   {


   if(head==NULL)
   {
       ptr->next = NULL;
       ptr->prev=NULL;
       ptr->data=item;
       head=ptr;
   }
   else
   {
       ptr->data=item;printf("\nPress 0 to insert more ?\n");
       ptr->prev=NULL;
       ptr->next = head;
       head->prev=ptr;
       head=ptr;
   }
    printf("\nNode Inserted\n");
}

}
int traverse()
{
    struct node *ptr;
    if(head == NULL)
    {
        printf("\nEmpty List\n");
    }
    else
    {
        ptr = head;
        while(ptr != NULL)
        {
            printf("%d\n",ptr->data);
            ptr=ptr->next;
        }
    }
}
```

All in one list

Menu Driven Program in C to implement all the operations of doubly linked list

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
   struct node *prev;
   struct node *next;
   int data;
};
struct node *head;
void insertion_beginning();
void insertion_last();
void insertion_specified();
void deletion_beginning();
void deletion_last();
void deletion_specified();
void display();
void search();
void main ()
{
int choice =0;
   while(choice != 9)
   {
      printf("\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 the node after the given data\n7.Search\n8.Show\n9.Exit\n");
      printf("\nEnter your choice?\n");
      scanf("\n%d",&choice);
      switch(choice)
      {
         case 1:
         insertion_beginning();
         break;
         case 2:
               insertion_last();
         break;
         case 3:
         insertion_specified();
         break;
         case 4:
```

```
        deletion_beginning();
        break;
        case 5:
        deletion_last();
        break;
        case 6:
        deletion_specified();
        break;
        case 7:
        search();
        break;
        case 8:
        display();
        break;
        case 9:
        exit(0);
        break;
        default:
        printf("Please enter valid choice..");
      }
   }
}
void insertion_beginning()
{
  struct node *ptr;
  int item;
  ptr = (struct node *)malloc(sizeof(struct node));
  if(ptr == NULL)
  {
     printf("\nOVERFLOW");
  }
  else
  {
   printf("\nEnter Item value");
   scanf("%d",&item);

  if(head==NULL)
  {
     ptr->next = NULL;
     ptr->prev=NULL;
     ptr->data=item;
     head=ptr;
  }
  else
  {
     ptr->data=item;
     ptr->prev=NULL;
     ptr->next = head;
```

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

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

    }
   printf("\nnode inserted\n");
   }
void insertion_specified()
{
  struct node *ptr,*temp;
  int item,loc,i;
  ptr = (struct node *)malloc(sizeof(struct node));
  if(ptr == NULL)
```

```
    {
        printf("\n OVERFLOW");
    }
    else
    {
        temp=head;
        printf("Enter the location");
        scanf("%d",&loc);
        for(i=0;i<loc;i++)
        {
            temp = temp->next;
            if(temp == NULL)
            {
                printf("\n There are less than %d elements", loc);
                return;
            }
        }
        printf("Enter value");
        scanf("%d",&item);
        ptr->data = item;
        ptr->next = temp->next;
        ptr -> prev = temp;
        temp->next = ptr;
        temp->next->prev=ptr;
        printf("\nnode inserted\n");
    }
}
void deletion_beginning()
{
    struct node *ptr;
    if(head == NULL)
    {
        printf("\n UNDERFLOW");
    }
    else if(head->next == NULL)
    {
        head = NULL;
        free(head);
        printf("\nnode deleted\n");
    }
    else
    {
        ptr = head;
        head = head -> next;
        head -> prev = NULL;
        free(ptr);
        printf("\nnode deleted\n");
    }
```

```

}
void deletion_last()
{
    struct node *ptr;
    if(head == NULL)
    {
        printf("\n UNDERFLOW");
    }
    else if(head->next == NULL)
    {
        head = NULL;
        free(head);
        printf("\nnode deleted\n");
    }
    else
    {
        ptr = head;
        if(ptr->next != NULL)
        {
            ptr = ptr -> next;
        }
        ptr -> prev -> next = NULL;
        free(ptr);
        printf("\nnode deleted\n");
    }
}
void deletion_specified()
{
    struct node *ptr, *temp;
    int val;
    printf("\n Enter the data after which the node is to be deleted : ");
    scanf("%d", &val);
    ptr = head;
    while(ptr -> data != val)
    ptr = ptr -> next;
    if(ptr -> next == NULL)
    {
        printf("\nCan't delete\n");
    }
    else if(ptr -> next -> next == NULL)
    {
        ptr ->next = NULL;
    }
    else
    {
        temp = ptr -> next;
        ptr -> next = temp -> next;
        temp -> next -> prev = ptr;
        free(temp);
        printf("\nnode deleted\n");
    }
}
void display()
{
    struct node *ptr;
    printf("\n printing values...\n");
    ptr = head;
    while(ptr != NULL)
    {
        printf("%d\n",ptr->data);
        ptr=ptr->next;
    }
}
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("\nitem found at location %d ",i+1);
                flag=0;
                break;
            }
            else
            {
                flag=1;
            }
            i++;
            ptr = ptr -> next;
        }
        if(flag==1)
        {
            printf("\nItem not found\n");
        }
    }

}












Arr[]={1,2,3,4}
X=10
1+2+3+4=10
Arr=x
1+1+1+1+1+1+1+1+1+1+1+1
Arr=x
1+3+3+3=10
Arr=x
(1+1)^7++3=10
Arr=x


```