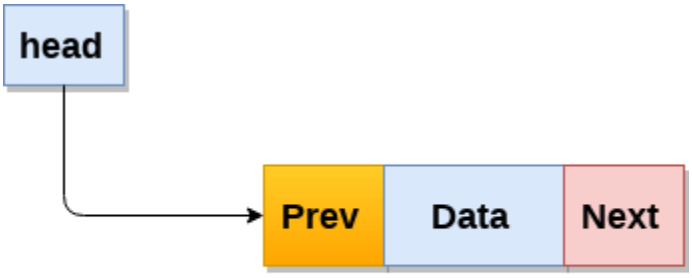
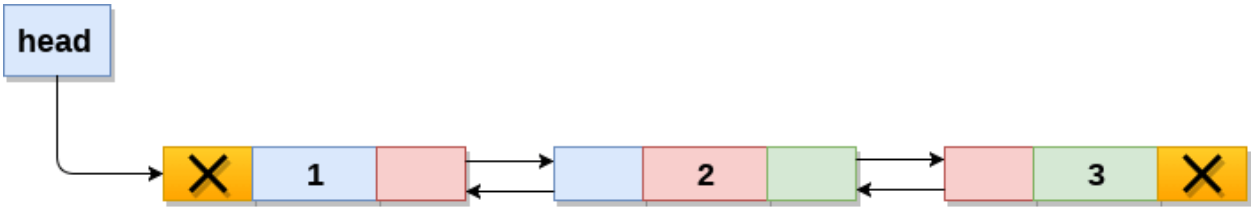
Doubly linked list

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 :

```
1. struct node
2. {
3.     struct node *prev;
4.     int data;
5.     struct node *next;
6. }
```

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 contain
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

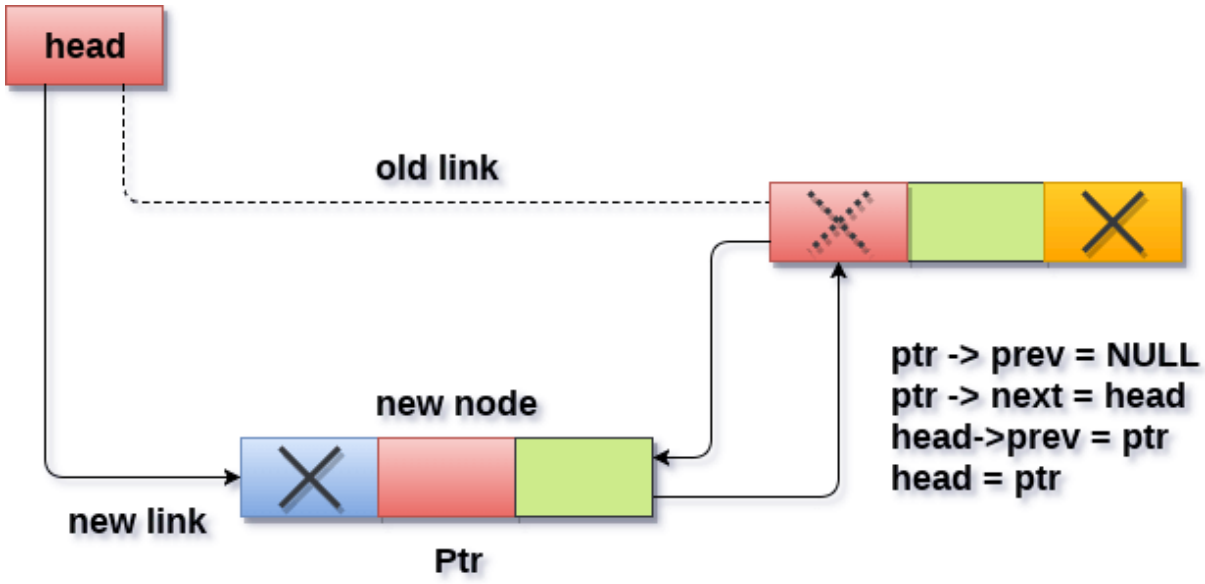
head = ptr

Algorithm :

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

Write OVERFLOW
Go to Step 9
[END OF IF]

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



Insertion into doubly linked list at beginning

C Function

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

```
75  {
76      int data;
77      struct node *next;
78      struct node *prev;
79  };
80  struct node *head;
81  void main ()
82  {
83      int choice,item;
84      do
85      {
86          printf("\nEnter the item which you want to insert?\n");
87          scanf("%d",&item);
88          insertbeginning(item);
89          printf("\nPress 0 to insert more ?\n");
90          scanf("%d",&choice);
91      }while(choice == 0);
92  }
93  void insertbeginning(int item)
94  {
95
96      struct node *ptr = (struct node *)malloc(sizeof(struct node));
97      if(ptr == NULL)
98      {
99          printf("\nOVERFLOW");
100     }
101     else
102     {
103
104
```

```
105     if(head==NULL)
106     {
107        ptr->next = NULL;
108        ptr->prev=NULL;
109        ptr->data=item;
110        head=ptr;
111     }
112     else
113     {
114        ptr->data=item;
115        ptr->prev=NULL;
116        ptr->next = head;
117        head->prev=ptr;
118        head=ptr;
119     }
120  }
121
122  }
123
```

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

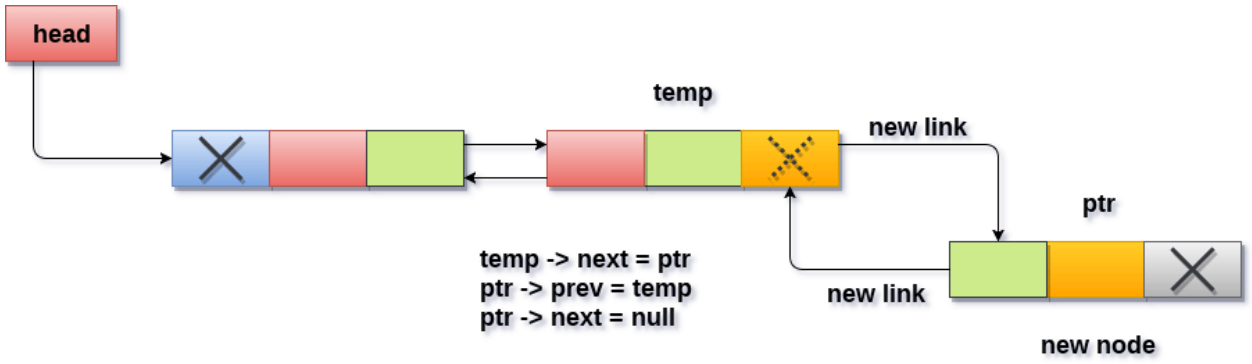
- **Step 1:** IF PTR = NULL

  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)
  {
```

```
207         printf("\nOVERFLOW");
208
209     }
210     else
211     {
212
213         ptr->data=item;
214         if(head == NULL)
215         {
216             ptr->next = NULL;
217             ptr->prev = NULL;
218             head = ptr;
219         }
220         else
221         {
222           temp = head;
223           while(temp->next!=NULL)
224           {
225              temp = temp->next;
226           }
227           temp->next = ptr;
228           ptr ->prev=temp;
229           ptr->next = NULL;
230         }
231 printf("\nNode Inserted\n");
232
233     }
234 }
```

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

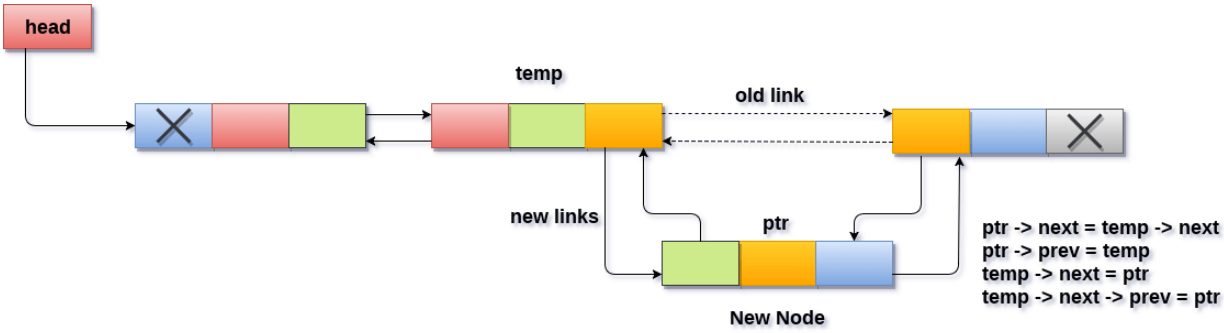
- **Step 1:** IF PTR = NULL

  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);
      }
```

```
314             else
315             {
316                insert_specified(item);
317             }
318             printf("\nPress 0 to insert more ?\n");
319             scanf("%d",&choice);
320          }while(choice == 0);
321       }
322       void create(int item)
323          {
324          struct node *ptr = (struct node *)malloc(sizeof(struct node));
325          if(ptr == NULL)
326          {
327             printf("\nOVERFLOW");
328          }
329          else
330          {
331
332
333          if(head==NULL)
334          {
335             ptr->next = NULL;
336             ptr->prev=NULL;
337             ptr->data=item;
338             head=ptr;
339          }
340          else
341          {
342             ptr->data=item;printf("\nPress 0 to insert more ?\n");
343             ptr->prev=NULL;
```

```
344         ptr->next = head;
345         head->prev=ptr;
346         head=ptr;
347       }
348        printf("\nNode Inserted\n");
349    }
350
351    }
352    void insert_specified(int item)
353    {
354
355      struct node *ptr = (struct node *)malloc(sizeof(struct node));
356      struct node *temp;
357      int i, loc;
358      if(ptr == NULL)
359      {
360         printf("\n OVERFLOW");
361      }
362      else
363      {
364         printf("\nEnter the location\n");
365         scanf("%d",&loc);
366         temp=head;
367         for(i=0;i<loc;i++)
368         {
369            temp = temp->next;
370            if(temp == NULL)
371            {
372               printf("\ncan't insert\n");
373               return;
```

```
374             }
375         }
376         ptr->data = item;
377         ptr->next = temp->next;
378         ptr -> prev = temp;
379         temp->next = ptr;
380         temp->next->prev=ptr;
381         printf("Node Inserted\n");
382       }
383     }
```

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.

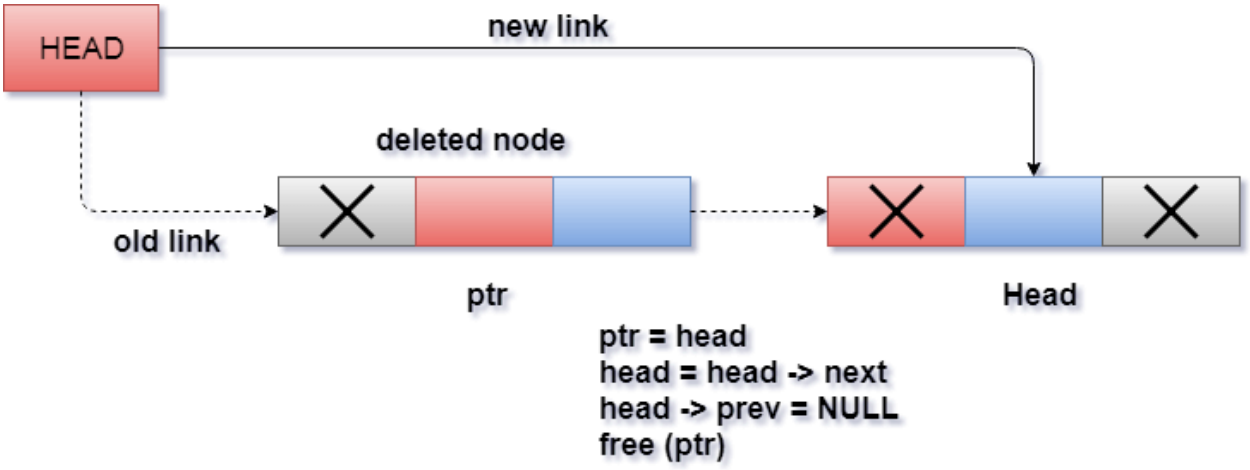
free(ptr)

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);
```

```
435                     create(item);
436             break;
437                             case 2:
438                                     show();
439                             break;
440             case 3:
441                     deletion_beginning();
442             break;
443             case 4:
444                     exit(0);
445             break;
446             default:
447             printf("\nPlease enter valid choice\n");
448           }
449
450        }while(choice != 4);
451     }
452     void create(int item)
453     {
454
455       struct node *ptr = (struct node *)malloc(sizeof(struct node));
456       if(ptr == NULL)
457       {
458          printf("\nOVERFLOW\n");
459       }
460       else
461       {
462
463
464       if(head==NULL)
465       {
466          ptr->next = NULL;
467          ptr->prev=NULL;
468          ptr->data=item;
469          head=ptr;
470       }
471       else
472       {
473          ptr->data=item;printf("\nPress 0 to insert more ?\n");
474          ptr->prev=NULL;
475          ptr->next = head;
476          head->prev=ptr;
477          head=ptr;
478       }
479        printf("\nNode Inserted\n");
480     }
481
482     }
```

```
483
484
485     void show()
486     {
487             struct node *ptr;
488             if(head==NULL)
489             {
490                     printf("\nlist is empty\n");
491             }
492             else
493             {
494                     ptr=head;
495                     while(ptr!=NULL)
496                     {
497                             printf("%d\n",ptr->data);
498                             ptr=ptr->next;
499                     }
500             }
501     }
502
503     void deletion_beginning()
504     {
505        struct node *ptr;
506        if(head == NULL)
507        {
508           printf("\n UNDERFLOW");
509        }
510        else if(head->next == NULL)
511        {
512           head = NULL;
513           free(head);
514           printf("\nnode deleted\n");
515        }
516        else
517        {
518           ptr = head;
519           head = head -> next;
520           head -> prev = NULL;
521           free(ptr);
522           printf("\nnode deleted\n");
523        }
524     }
525
```

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

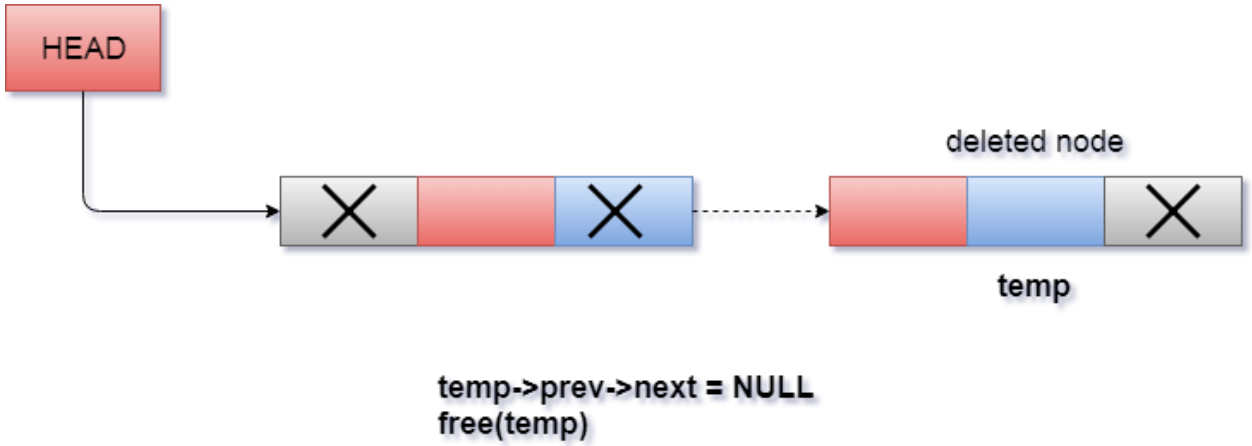
  Write UNDERFLOW
  Go to Step 7
  [END OF IF]

- **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

```
562  C Function
563  #include<stdio.h>
564  #include<stdlib.h>
565  void create(int);
566  void last_delete();
567  struct node
568  {
569     int data;
570     struct node *next;
571     struct node *prev;
572  };
573  struct node *head;
574  void main ()
575  {
576     int choice,item;
577     do
578     {
579        printf("1.Append List\n2.Delete node from end\n3.Exit\n4.Enter your choice?");
580        scanf("%d",&choice);
581        switch(choice)
582        {
583           case 1:
584           printf("\nEnter the item\n");
585           scanf("%d",&item);
586           create(item);
587           break;
588           case 2:
589           last_delete();
590           break;
591           case 3:
```

```
592                 exit(0);
593                 break;
594                 default:
595                 printf("\nPlease enter valid choice\n");
596             }
597
598         }while(choice != 3);
599     }
600     void create(int item)
601     {
602
603       struct node *ptr = (struct node *)malloc(sizeof(struct node));
604       if(ptr == NULL)
605       {
606          printf("\nOVERFLOW\n");
607       }
608       else
609       {
610
611
612       if(head==NULL)
613       {
614          ptr->next = NULL;
615          ptr->prev=NULL;
616          ptr->data=item;
617          head=ptr;
618       }
619       else
620       {
621          ptr->data=item;
```

```
622             ptr->prev=NULL;
623             ptr->next = head;
624             head->prev=ptr;
625             head=ptr;
626           }
627           printf("\nNode Inserted\n");
628       }
629
630       }
631       void last_delete()
632       {
633           struct node *ptr;
634           if(head == NULL)
635           {
636               printf("\n UNDERFLOW\n");
637           }
638           else if(head->next == NULL)
639           {
640               head = NULL;
641               free(head);
642               printf("\nNode Deleted\n");
643           }
644           else
645           {
646               ptr = head;
647               if(ptr->next != NULL)
648               {
649                   ptr = ptr -> next;
650               }
651               ptr -> prev -> next = NULL;
```

```
652             free(ptr);
653             printf("\nNode Deleted\n");
654         }
655     }
656
```

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

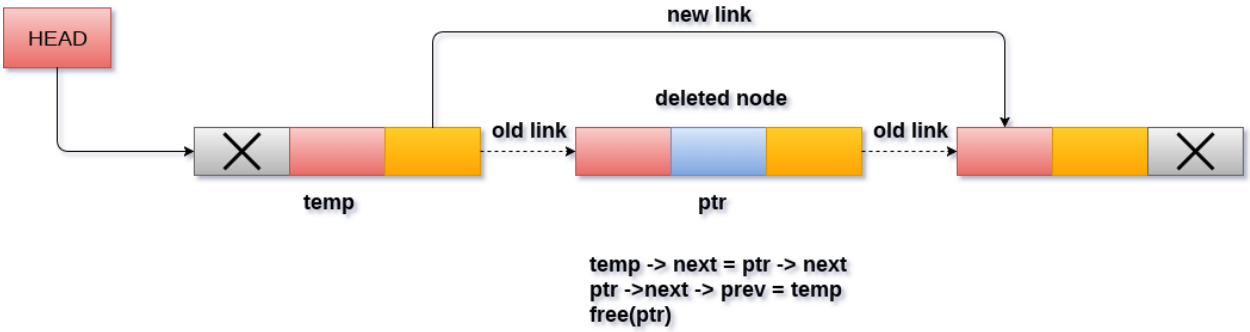
- **Step 1:** IF HEAD = NULL

  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)
   {
```

```
740         printf("\nOVERFLOW\n");
741       }
742       else
743       {
744
745
746       if(head==NULL)
747       {
748          ptr->next = NULL;
749          ptr->prev=NULL;
750          ptr->data=item;
751          head=ptr;
752       }
753       else
754       {
755          ptr->data=item;
756          ptr->prev=NULL;
757          ptr->next = head;
758          head->prev=ptr;
759          head=ptr;
760       }
761        printf("\nNode Inserted\n");
762    }
763
764    }
765    void delete_specified( )
766    {
767        struct node *ptr, *temp;
768        int val;
769        printf("Enter the value");
```

```
770         scanf("%d",&val);
771         temp = head;
772         while(temp -> data != val)
773         temp = temp -> next;
774         if(temp -> next == NULL)
775         {
776            printf("\nCan't delete\n");
777         }
778         else if(temp -> next -> next == NULL)
779         {
780            temp ->next = NULL;
781            printf("\nNode Deleted\n");
782         }
783         else
784         {
785            ptr = temp -> next;
786            temp -> next = ptr -> next;
787            ptr -> next -> prev = temp;
788            free(ptr);
789            printf("\nNode Deleted\n");
790         }
791      }
792
```

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);
```

```
908                 flag=0;
909                 break;
910             }
911             else
912             {
913                 flag=1;
914             }
915             i++;
916             ptr = ptr -> next;
917         }
918         if(flag==1)
919         {
920             printf("\nItem not found\n");
921         }
922     }
923 }
924
```

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
```

```
969         {
970             printf("1.Append List\n2.Traverse\n3.Exit\n4.Enter your choice?");
971             scanf("%d",&choice);
972             switch(choice)
973             {
974                 case 1:
975                 printf("\nEnter the item\n");
976                 scanf("%d",&item);
977                 create(item);
978                 break;
979                 case 2:
980                 traverse();
981                 break;
982                 case 3:
983                 exit(0);
984                 break;
985                 default:
986                 printf("\nPlease enter valid choice\n");
987             }
988
989          }while(choice != 3);
990     }
991     void create(int item)
992     {
993
994       struct node *ptr = (struct node *)malloc(sizeof(struct node));
995       if(ptr == NULL)
996       {
997           printf("\nOVERFLOW\n");
998       }
999       else
1000      {
1001
1002
1003      if(head==NULL)
1004      {
1005          ptr->next = NULL;
1006          ptr->prev=NULL;
1007          ptr->data=item;
1008          head=ptr;
1009      }
1010      else
1011      {
1012          ptr->data=item;printf("\nPress 0 to insert more ?\n");
1013          ptr->prev=NULL;
1014          ptr->next = head;
1015          head->prev=ptr;
1016          head=ptr;
```

```
1017        }
1018         printf("\nNode Inserted\n");
1019    }
1020
1021    }
1022    int traverse()
1023    {
1024        struct node *ptr;
1025        if(head == NULL)
1026        {
1027            printf("\nEmpty List\n");
1028        }
1029        else
1030        {
1031            ptr = head;
1032            while(ptr != NULL)
1033            {
1034                printf("%d\n",ptr->data);
1035                ptr=ptr->next;
1036            }
1037        }
1038    }
1039
```

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:
```

```
1085              deletion_beginning();
1086              break;
1087              case 5:
1088              deletion_last();
1089              break;
1090              case 6:
1091              deletion_specified();
1092              break;
1093              case 7:
1094              search();
1095              break;
1096              case 8:
1097              display();
1098              break;
1099              case 9:
1100              exit(0);
1101              break;
1102              default:
1103              printf("Please enter valid choice..");
1104           }
1105       }
1106    }
1107    void insertion_beginning()
1108    {
1109      struct node *ptr;
1110      int item;
1111      ptr = (struct node *)malloc(sizeof(struct node));
1112      if(ptr == NULL)
1113      {
1114         printf("\nOVERFLOW");
1115      }
1116      else
1117      {
1118       printf("\nEnter Item value");
1119       scanf("%d",&item);
1120
1121      if(head==NULL)
1122      {
1123         ptr->next = NULL;
1124         ptr->prev=NULL;
1125         ptr->data=item;
1126         head=ptr;
1127      }
1128      else
1129      {
1130         ptr->data=item;
1131         ptr->prev=NULL;
1132         ptr->next = head;
```

```
1133          head->prev=ptr;
1134          head=ptr;
1135       }
1136       printf("\nNode inserted\n");
1137    }
1138
1139    }
1140    void insertion_last()
1141    {
1142       struct node *ptr,*temp;
1143       int item;
1144       ptr = (struct node *) malloc(sizeof(struct node));
1145       if(ptr == NULL)
1146       {
1147          printf("\nOVERFLOW");
1148       }
1149       else
1150       {
1151          printf("\nEnter value");
1152          scanf("%d",&item);
1153           ptr->data=item;
1154          if(head == NULL)
1155          {
1156             ptr->next = NULL;
1157             ptr->prev = NULL;
1158             head = ptr;
1159          }
1160          else
1161          {
1162            temp = head;
1163            while(temp->next!=NULL)
1164            {
1165               temp = temp->next;
1166            }
1167            temp->next = ptr;
1168            ptr ->prev=temp;
1169            ptr->next = NULL;
1170            }
1171
1172          }
1173        printf("\nnode inserted\n");
1174       }
1175    void insertion_specified()
1176    {
1177       struct node *ptr,*temp;
1178       int item,loc,i;
1179       ptr = (struct node *)malloc(sizeof(struct node));
1180       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");
     }
```

```
1229
1230    }
1231    void deletion_last()
1232    {
1233       struct node *ptr;
1234       if(head == NULL)
1235       {
1236          printf("\n UNDERFLOW");
1237       }
1238       else if(head->next == NULL)
1239       {
1240          head = NULL;
1241          free(head);
1242          printf("\nnode deleted\n");
1243       }
1244       else
1245       {
1246          ptr = head;
1247          if(ptr->next != NULL)
1248          {
1249             ptr = ptr -> next;
1250          }
1251          ptr -> prev -> next = NULL;
1252          free(ptr);
1253          printf("\nnode deleted\n");
1254       }
1255    }
1256    void deletion_specified()
1257    {
1258       struct node *ptr, *temp;
1259       int val;
1260       printf("\n Enter the data after which the node is to be deleted : ");
1261       scanf("%d", &val);
1262       ptr = head;
1263       while(ptr -> data != val)
1264       ptr = ptr -> next;
1265       if(ptr -> next == NULL)
1266       {
1267          printf("\nCan't delete\n");
1268       }
1269       else if(ptr -> next -> next == NULL)
1270       {
1271          ptr ->next = NULL;
1272       }
1273       else
1274       {
1275          temp = ptr -> next;
1276          ptr -> next = temp -> next;
```

```
1277            temp -> next -> prev = ptr;
1278            free(temp);
1279            printf("\nnode deleted\n");
1280        }
1281    }
1282    void display()
1283    {
1284        struct node *ptr;
1285        printf("\n printing values...\n");
1286        ptr = head;
1287        while(ptr != NULL)
1288        {
1289            printf("%d\n",ptr->data);
1290            ptr=ptr->next;
1291        }
1292    }
1293    void search()
1294    {
1295        struct node *ptr;
1296        int item,i=0,flag;
1297        ptr = head;
1298        if(ptr == NULL)
1299        {
1300            printf("\nEmpty List\n");
1301        }
1302        else
1303        {
1304            printf("\nEnter item which you want to search?\n");
1305            scanf("%d",&item);
1306            while (ptr!=NULL)
1307            {
1308                if(ptr->data == item)
1309                {
1310                    printf("\nitem found at location %d ",i+1);
1311                    flag=0;
1312                    break;
1313                }
1314                else
1315                {
1316                    flag=1;
1317                }
1318                i++;
1319                ptr = ptr -> next;
1320            }
1321            if(flag==1)
1322            {
1323                printf("\nItem not found\n");
1324            }
```

```
    }

}
```

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