Parshvanath Charitable Trust's
A. P. SHAH INSTITUTE OF TECHNOLOGY
(Approved by AICTE New Delhi & Govt. of Maharashtra, Affiliated to University of Mumbai)
(Religious Jain Minority)

Department of Information Technology

Chapter-4
Linked List

Introduction to Linked List, Basic concept of Linked List, Memory allocation & de allocation of Linked list, Singly Linked list, Doubly Linked list, Circular linked list, Operations on linked list, Linked representation of stack, Linked representation of Queue, Application of linked list.

UNIT : 4

LINKED LISTS

A linked list is a collection of data element called nodes in which the linear representation is given by links from one node to the next node.

We have studied that an array is a linear collection of data elements in which the element ~~in~~ ~~which~~ the are stored in consecutive memory locations. while declaring arrays, we have to specify the size of array, which will restrict the number of elements that the array can store. for example, if we declare an array as int mark[10], then the array can store a maximum of 10 data elements but not more than that. But what if we are not sure of the number of elements in advance? So, there must be a data structure that removes the restrictions on the maximum number of elements and the storage condition to write efficient programs.

Linked list is a data structure that is free from the aforementioned restrictions. A linked list does not store its elements in consecutive memory locations and the user can add any number of elements to it. However unlike an array, a linked list does not allow random access of data. Elements in a linked list can be accessed only in a

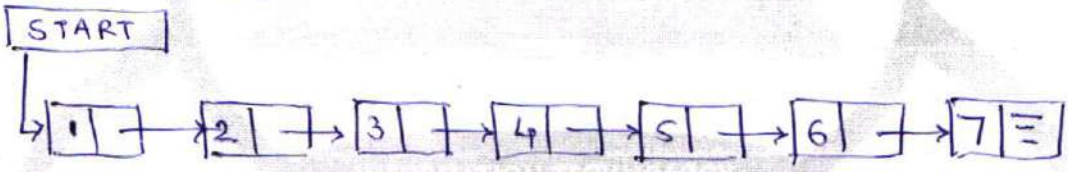
Parshvanath Charitable Trust's
A. P. SHAH INSTITUTE OF TECHNOLOGY
(Approved by AICTE New Delhi & Govt. of Maharashtra, Affiliated to University of Mumbai)
(Religious Jain Minority)

Semester: 3 Subject: DSA Academic Year: 2017-2018

sequential manner But like an array, inser-tions and deletions can be done at any point in the list in a constant time.

Basic Concept of Linked List

A linked list, in simple terms, is a linear collection of data elements. These data elements are called nodes. Linked List is a data structure which in turn can be used to implement other data structures. Thus it acts as a building block to implement data structure such as stacks, queues and their variations. A linked list can be perceived as a train or a sequence of nodes in which each node contains one or more data fields and a pointer to the next node.



Simple linked list

we can see a simple linked list in which every node contains ~~data~~ two parts, an integer and a pointer to the next node. The left part of the node which contains data may include a simple data type, an array or a structure. The right part of the node contains a pointer to the next node (or address of the next node in sequence). The last node will have no next node connected to it so it will store a special value called NULL.

Hence, a NULL pointer denotes the end of the list.

Parshvanath Charitable Trust's
A. P. SHAH INSTITUTE OF TECHNOLOGY
(Approved by AICTE New Delhi & Govt. of Maharashtra, Affiliated to University of Mumbai)
(Religious Jain Minority)

Semester: 3 Subject: DSA Academic Year: 2017-2018

Linked lists contain a pointer variable START that stores the address of the first node in the list. We can traverse the entire list using START which contains the address of the first node; the next part of the first node in turn stores the address of the first node; the next part of the first node in turn stores the address of its succeeding node. Using this technique the individual node of the list will form a chain of nodes. If START = NULL, then the linked list is empty and contains no nodes.

In C, we can implement a linked list using the following code:

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

Linked Lists verses Arrays

Both arrays and linked lists are a linear collection of data elements. But unlike an array, a linked list does not store its nodes in consecutive memory location. Another point of difference between an array and a linked list is that a linked list does not allow random access of data. Nodes in a linked list can be accessed only in a sequential manner. But like an array insertions and deletions can be done at any point in the list in a constant time.

Parshvanath Charitable Trust's
A. P. SHAH INSTITUTE OF TECHNOLOGY
(Approved by AICTE New Delhi & Govt. of Maharashtra, Affiliated to University of Mumbai)
(Religious Jain Minority)

Semester: 3 Subject: DSA Academic Year: 2017-2018

Another advantage of linked list over an array is that we can add any number of elements in the list. This is not possible in case of an array.

Thus linked lists provide an efficient way of storing related data and performing basic operations such as insertion, deletion and updation of information at the cost of extra space required for storing the address of next nodes.

Memory Allocation and De-allocation for a Linked List

We have seen how a linked list is represented in the memory. If we want to add a node to an already existing linked list in the memory, we first find free space in the memory and then use it to store the information for example, Consider the linked list shown in following figure. The linked list contains the roll number of students, marks obtained by them in Biology and finally a next field which stores the address of the next node in sequence Now, if a new student joins the class and is asked to appear for the same test that the other students had taken, then the new student's marks should also be recorded in the linked list. For this purpose, we find a free space and store the information there. In following fig. the grey shaded portion shows free space, and thus we have 4 memory

Parshvanath Charitable Trust's
A. P. SHAH INSTITUTE OF TECHNOLOGY
(Approved by AICTE New Delhi & Govt. of Maharashtra, Affiliated to University of Mumbai)
(Religious Jain Minority)

locations available. We can use any one of them to store our data. This is illustrated in following fig. We can use any one of them to store our data.

Now, the question is which part of the memory is available and which part is occupied? When we delete a node from a linked list, then who changes the status of the memory occupied by it from occupied to available? The answer is the operating system

START → 1 → BIOLOGY

	Roll No	Marks	Next
1			
2	S02	84	3
3	S03	45	5
4			
5	S04	98	7
6			
7	S05	55	8
8	S06	34	10
9			
10	S07	90	11
11	S08	87	12
12	S09	86	13
13	S10	67	15
14			
15	S11	56	-1

(a)

START → 1 → BIOLOGY

	Roll No	Marks	Next
1	S01	78	2
2	S02	84	3
3	S03	45	5
4	S12	75	-1
5	S04	98	7
6			
7	S05	55	8
8	S06	34	10
9			
10	S07	90	11
11	S08	87	12
12	S09	86	13
13	S10	67	15
14			
15	S11	56	4

(b)

(a) Student's linked list and (b) linked list after the insertion of new student's record.

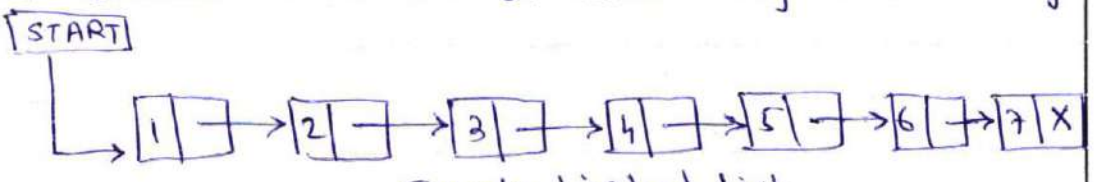
Parshvanath Charitable Trust's
A. P. SHAH INSTITUTE OF TECHNOLOGY
(Approved by AICTE New Delhi & Govt. of Maharashtra, Affiliated to University of Mumbai)
(Religious Jain Minority)

When we delete a particular node from an existing linked list or delete the entire linked list, the space occupied by it must be given back to the free pool so that the memory can be reused by some other program that needs memory space.

The operating system does this task of adding freed memory to the free pool. The operating system will perform this operation whenever it finds the CPU idle or whenever the programs are falling short of memory space. The operating system scans through all the memory cells and marks those cells that are being used by some program. This process is called garbage collection.

SINGLY LINKED LISTS

A singly linked list is the simplest type of linked list in which every node contains some data and a pointer to the next node of the same data type. By saying that the node contains a pointer to the next node, we mean that the node stores the address of the next node sequence. A singly linked list allows traversal of data only in one way



Singly Linked list

Parshvanath Charitable Trust's
A. P. SHAH INSTITUTE OF TECHNOLOGY
(Approved by AICTE New Delhi & Govt. of Maharashtra, Affiliated to University of Mumbai)
(Religious Jain Minority)

Semester: 3 Subject: DSA Academic Year: 2017 - 2018

Traversing a Linked List

Traversing a linked list means accessing the nodes of the list in order to perform some processing on them. Remember a linked list always contains a pointer variable START which stores the address of the first node of the list. End of the list is marked by storing NULL or -1 in the NEXT field of the last node. For traversing the linked list, we also make use of another pointer variable PTR which points to the node that is currently being accessed. The algorithm to traverse a linked list is shown in fig.

Step 1 : [INITIALIZE] SET PTR = START
Step 2 : Repeat Steps 3 and 4 while PTR != NULL
~~Step 3 : SET PTR = PTR → NEXT~~
~~[END OF LOOP]~~
~~Step 4~~
Step 3 : Apply Process to PTR → DATA
Step 4 : SET PTR = PTR → NEXT
[END OF LOOP]
Step 5 : EXIT

Searching for a Value in a Linked List

Searching a linked list means to find a particular element in the linked list. As already discussed, a linked list ~~conta~~ consists of nodes which are divided into two parts, the information part and the next part. So searching means finding

Parshvanath Charitable Trust's
A. P. SHAH INSTITUTE OF TECHNOLOGY
(Approved by AICTE New Delhi & Govt. of Maharashtra, Affiliated to University of Mumbai)
(Religious Jain Minority)

whether a given value is present in the information part ~~and the next part. So searching means finding whether~~ of the node or not. If it is present, the algorithm returns the address of the node that contains the value. The following fig shows the algorithm to search a linked list

```
Step 1 : [INITIALIZE] SET PTR = START
Step 2 : Repeat Step 3 while PTR != NULL
Step 3 : IF VAL = PTR → DATA
              SET POS = PTR
              Go to Step 5
         ELSE
              SET PTR = PTR → NEXT
         [END OF IF]
    [END OF LOOP]
Step 4 : SET POS = NULL
Step 5 : EXIT
```

Inserting a New Node in a Linked List

In this section, we will see how a new node is added into an already existing linked list. We will take four cases and then see how insertion is done in each case.

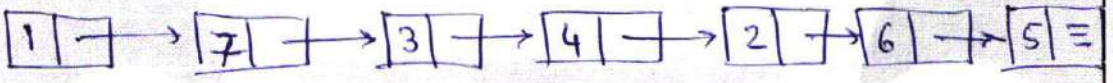
Case 1 : The new node is inserted at the beginning
Case 2 : The new node is inserted at the end
Case 3 : The new node is inserted after a given node
Case 4 : The new node is inserted before a given node.

Parshvanath Charitable Trust's
A. P. SHAH INSTITUTE OF TECHNOLOGY
(Approved by AICTE New Delhi & Govt. of Maharashtra, Affiliated to University of Mumbai)
(Religious Jain Minority)

Semester: 3 Subject: DSA Academic Year: 17-18

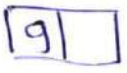
Inserting a Node at the Beginning of a linked List

Consider the linked list shown as below Suppose we want to add a new node with data 9 and add it as the first node of the list. Then the following changes will be done in the linked list.

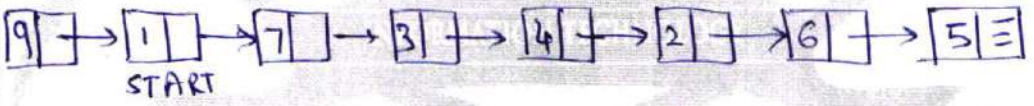


START

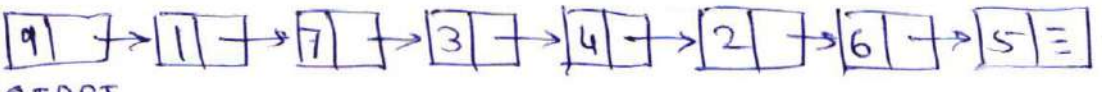
Allocate memory for the new node and initialize its DATA part to 9



Add the new node as the first node of the list by making the next part of the new node contain the address of START



Now make START to point to the first node of the list



Inserting an element at the beginning of a linked List.

Parshvanath Charitable Trust's
A. P. SHAH INSTITUTE OF TECHNOLOGY
(Approved by AICTE New Delhi & Govt. of Maharashtra, Affiliated to University of Mumbai)
(Religious Jain Minority)


```
step 1 : IF AVAIL = NULL
         Write OVERFLOW
         Go to Step 7
     [ END OF IF ]
step 2 : SET NEW_NODE = AVAIL
step 3 : SET NEW_NODE = AVAIL → NEXT
step 4 : SET NEW_NODE → DATA = VAL
step 5 : SET NEW_NODE → NEXT = START
step 6 : SET START = NEW_NODE
step 7 : EXIT
```

In step 1 we first check whether memory is available for the new node. If the free memory has exhausted, then an OVERFLOW message is printed.

Inserting a Node at the End of Linked List

```
Step 1 : IF AVAIL = NULL
         Write OVERFLOW
         Goto to step 10
     [ END OF IF ]
Step 2 : SET NEW_NODE = AVAIL
step 3 : SET AVAIL = AVAIL → NEXT
step 4 : SET NEW_NODE → DATA = VAL
Step 5 : SET NEW_NODE → NEXT = NULL
step 6 : SET PTR = START
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
```

Parshvanath Charitable Trust's
A. P. SHAH INSTITUTE OF TECHNOLOGY
(Approved by AICTE New Delhi & Govt. of Maharashtra, Affiliated to University of Mumbai)
(Religious Jain Minority)

Semester: 3 Subject: DSA Academic Year: 17-18

Inserting a Node After a Given Node in a Linked List

```
Step 1 : IF AVAIL = NULL
           write OVERFLOW
           GO to step 12
         [END OF IF]
Step 2 : SET NEW_NODE = AVAIL
Step 3 : SET AVAIL = AVAL → NEXT
Step 4 : SET NEW_NODE → DATA = VAL
Step 5 : SET PTR = START
Step 6 : SET PREPTR = PTR
Step 7 : Repeat Step 8 and 9 while PREPTR→
           Data
           ! = NUM
Step 8 : SET PREPTR = PTR
Step 9 : SET PTR = PTR→ NEXT
         [END OF LOOP]
Step 10 : PREPTR → NEXT = NEW_NODE
Step 11 : SET NEW_NODE → NEXT = PTR
Step 12 : EXIT
```

Parshvanath Charitable Trust's
A. P. SHAH INSTITUTE OF TECHNOLOGY
(Approved by AICTE New Delhi & Govt. of Maharashtra, Affiliated to University of Mumbai)
(Religious Jain Minority)

Inserting a Node Before a Given Node in Linked List

```
step 1 : IF AVAIL = NULL
           Write OVERFLOW
           Goto step 12
         [END OF IF]
Step 2 :  SET NEW_NODE = AVAIL
Step 3 :  SET AVAIL = AVAIL → NEXT
Step 4 :  SET NEW_NODE → DATA = VAL
Step 5 :  SET PTR = START
Step 6 :  SET PREPTR = PTR
Step 7 :  Repeat step 8 and 9 while
            PTR → DATA != NUM
Step 8 :  SET PREPTR = PTR
Step 9 :  SET PTR = PTR → NEXT
          [END OF LOOP]
Step 10 : PREPTR → NEXT = NEW_NODE
Step 11 : SET NEW_NODE → NEXT = PTR
Step 12 : EXIT
```

[Algorithm to insert a new node before a node that has NUM.]

Parshvanath Charitable Trust's
A. P. SHAH INSTITUTE OF TECHNOLOGY
(Approved by AICTE New Delhi & Govt. of Maharashtra, Affiliated to University of Mumbai)
(Religious Jain Minority)

Semester: 3 Subject: DSA Academic Year: 17-18

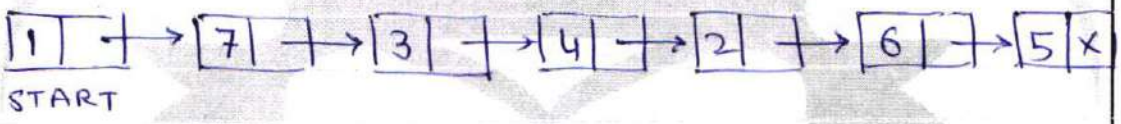
Deleting a Node from a Linked List

In this section, we will discuss how a node is deleted from an already existing linked list. We will consider three cases and then see how deletion is done in each case

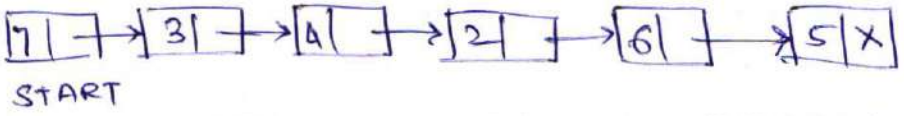
Case 1 : The first node is deleted
Case 2 : The last node is deleted
Case 3 : The node after a given node is deleted

Deleting the first Node from a Linked List

When we want to delete a node from the beginning of the list, then the following changes will be done in the linked list.



Make START to point to the next node in sequence



Deleting the first node of a linked list

```
step 1 : IF START = NULL
         Write UNDERFLOW
         Goto step 5
         [END OF IF]
step 2 : SET PTR = START
step 3 : SET START = START → NEXT
step 4 : FREE PTR
step 5 : EXIT
```

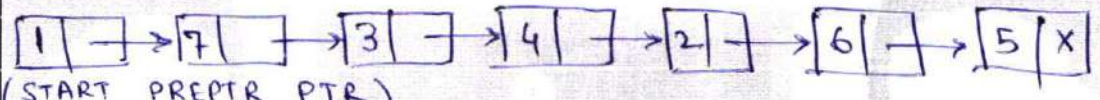
Parshvanath Charitable Trust's
A. P. SHAH INSTITUTE OF TECHNOLOGY
(Approved by AICTE New Delhi & Govt. of Maharashtra, Affiliated to University of Mumbai)
(Religious Jain Minority)

Semester: 3 Subject: DSA Academic Year: 17-18

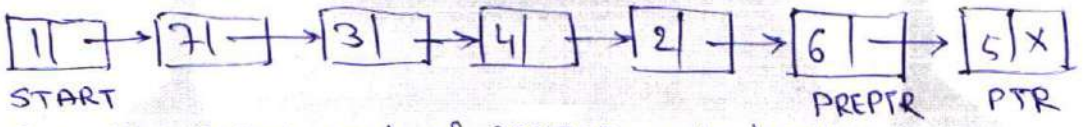
Deleting the last Node from a Linked List

Suppose we want to delete the last node from Linked List, then the following changes will be done in the linked list.

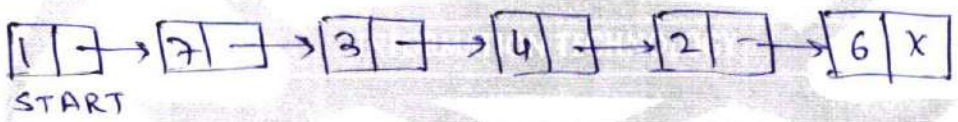
Take a pointer variables PTR & PREPTR which initially point to START



Move PTR and PREPTR such that NEXT Part Of PTR = NULL. PREPTR always points to the node just before the node pointed by PTR



Set the NEXT part of PREPTR node to NULL



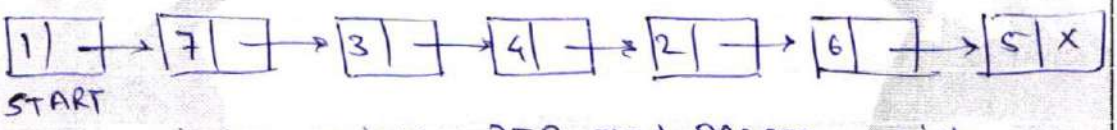

```
Step 1 : IF START = NULL
         Write UNDERFLOW
         GOTO step 8
         [END OF IF]
Step 2 : SET PTR = START
Step 3 : Repeat Step 4 & 5 while PTR→NEXT != NULL
Step 4 :   SET PREPTR = PTR
Step 5 :   SET PTR = PTR → NEXT
         [END OF LOOP]
Step 6 : SET PREPTR → NEXT = NULL
Step 7 : FREE PTR
Step 8 : EXIT
```

Parshvanath Charitable Trust's
A. P. SHAH INSTITUTE OF TECHNOLOGY
(Approved by AICTE New Delhi & Govt. of Maharashtra, Affiliated to University of Mumbai)
(Religious Jain Minority)

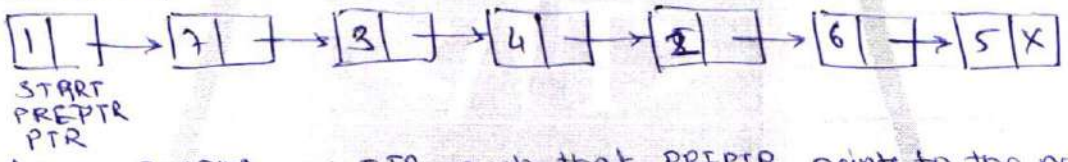
Semester: 3 Subject: DSA Academic Year: 17-18

Deleting the Node After a given Node in a Linked List

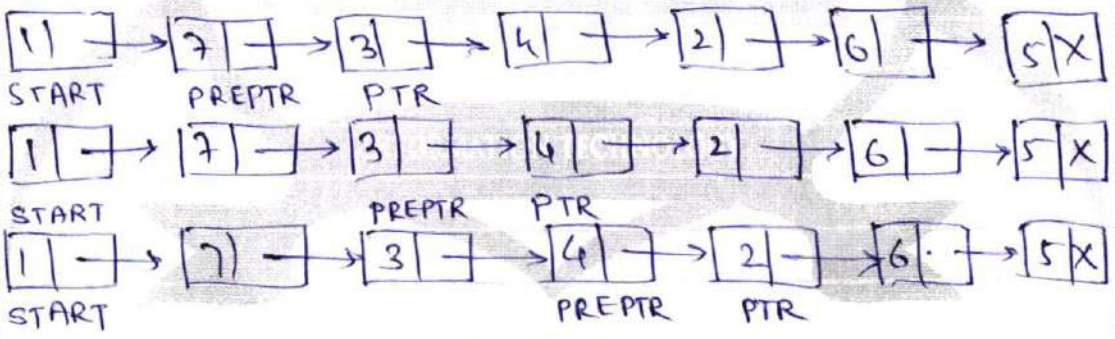
Suppose we want to delete the node that succeeds the node which contains data value 4 then the following changes will be done in the linked list.



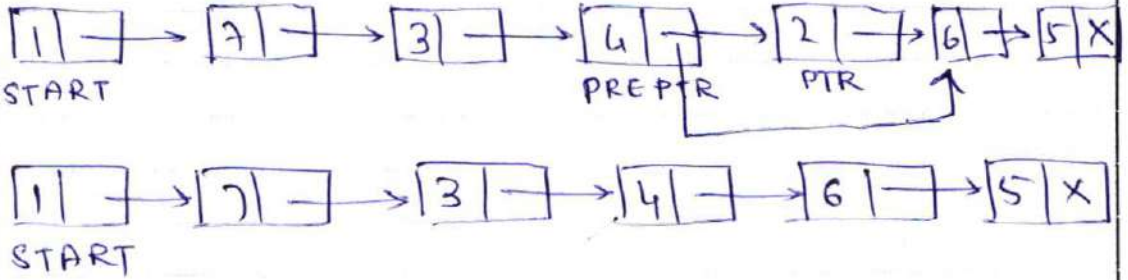
Take pointer variable PTR and PREPTR which initially point to start.



Move PREPTR and PTR such that PREPTR points to the node containing VAL and PTR points to the succeeding



SET THE NEXT part of PREPTR to the NEXT part of PTR



Deleting the node after a given node in linked List

Parshvanath Charitable Trust's
A. P. SHAH INSTITUTE OF TECHNOLOGY
(Approved by AICTE New Delhi & Govt. of Maharashtra, Affiliated to University of Mumbai)
(Religious Jain Minority)

Semester: 3 Subject: DSA Academic Year: 17-18

Step 1 : IF START = NULL
WRITE UNDERFLOW
Go to Step 10
[END OF IF]
Step 2 : SET PTR = START
Step 3 : SET PRPTR = PTR
Step 4 : Repeat Step 5 and 6 while
PREPTR -> DATA != NUM
step 5 : SET PREPTR = PTR
step 6 : SET PTR = PTR -> NEXT
[END OF LOOP]
step 7 : SET TEMP = PTR
step 8 : SET PREPTR -> NEXT = PTR -> NEXT
step 9 : FREE TEMP
Step 10 : EXIT

Alogorithm to delete the node after a given node

//Creating a Singly Linked List inserting a node at the beginning,At end and at specific position.Also deleting a node from begining,from end and from specific position.

```
#include<stdio.h>
#include<stdlib.h>

// A linked list (LL) node to store a data element
struct node
{
 int data;
 struct node* next;
}*start;

//Insert data at the beginning
 void insert_at_beg(int x)
 {
  /*Allocate Node or allocate memory for new node*/
  struct node* newnode=(struct node*)malloc(sizeof(struct node));
  printf("\nEnter the data :");
  scanf("%d",&x);
  if(start==NULL)//if Linked List is empty
  {
   /*Put in the data in the node and set next to NULL*/
    newnode->data=x;
    newnode->next=NULL;
   /*Move start to point to the newnode*/
    start=newnode;
  }
  else
  {
  /*Put in the data in the node*/
   newnode->data=x;
  /*Make next of new node as start*/
   newnode->next=start;
  /*Move the start to point to the new node */
   start=newnode;
  }
 }

//Insert data at the end
 void insert_at_end(int x)
 {
  /*create a pointer to traverse till end of linked list*/
  struct node* ptr;

   /*Allocate Node or allocate memory for new node*/
  struct node* newnode=(struct node*)malloc(sizeof(struct node));
  printf("\nEnter the data :");
  scanf("%d",&x);

  /*Put in the data in the node and set next to NULL*/
  newnode->data=x;
  newnode->next=NULL;

  if(start==NULL)//if Linked List is empty Make next of new node as start
```

```
 {  //newnode->data=x;
    //newnode->next=NULL;

    /*Move the start to point to the new node*/
    start=newnode;
 }
 else //if not empty then
 {
  ptr=start;
  while(ptr->next!=NULL)//traverse through Linked List till last node
    ptr=ptr->next;

  ptr->next=newnode;//linked last node of next to newnode
 }
}

//Insert data at specific position
 void insert_at_specific_position(int x,int pos)
 {
  struct node* ptr;
  ptr=start;
  int i=0;
  struct node*newnode=(struct node*)malloc(sizeof(struct node));
  printf("\nEnter the data :");
  scanf("%d",&x);
  printf("Enter the position :");
  scanf("%d",&pos);
  if(pos==1)
  {
    printf("Please use insert_at_begin() ");
    //break;
   }
   else //if posotion is not 1 then
   {
       newnode->data=x;
       if(start==NULL)//if LL is empty
       {
         start=newnode;
         newnode->next=NULL;
       }
       else
       {
          while(i<(pos-2))//if pos=5 while loop will continue from 0 till i<3
          {
           ptr=ptr->next;//in last iteration ptr=3->next i.e 4
           i++;
          }
          newnode->next=ptr->next;//ptr->next is 4->next i.e 5 that is 5th node shifted to        6th position
          ptr->next=newnode;//4->next is set to newnode i.e newnode is added at 5th position
       }
  }
 }
//Delete data from beginning
 void delete_from_beg()
 {
```

```
 struct node* ptr;
 if(start==NULL)
 {
  printf("\nUNDERFLOW ! ! !");
  //break;
 }
 else
 {
  ptr=start;
  start=start->next;
  printf("\n%d is deleted...",ptr->data);
  free(ptr);
 }
}

//Delete data from end
 void delete_from_end()
 {
  struct node* ptr,*preptr;
  if(start==NULL)
  {
   printf("\nUNDERFLOW ! ! !");
   //break;
  }
  else
  {
   ptr=start;
   if(ptr->next==NULL)//if there is only one element in the LL
   {
    printf("\n%d is deleted...",ptr->data);
    start=NULL;
    free(ptr);
   }
   else
   {
    while(ptr->next!=NULL)
    {
     preptr=ptr;//preptr=1
     ptr=ptr->next;//ptr=ptr->next=1->next=2
    }
    preptr->next=NULL;
    printf("\n%d is deleted...",ptr->data);
    free(ptr);
   }
  }
 }

//Delete data from specific position
 void delete_from_specific_position(int pos)
 {
  struct node* ptr,* temp;
  int i=0;
  if(start==NULL)
  {
   printf("\nUNDERFLOW ! ! !");
```

```
 //break;
 }
 else
 {
  printf("\nEnter the position :");
  scanf("%d",&pos);
  ptr=start;
  if(ptr->next==NULL)//if there is only one node in the LL
  {
   printf("\n%d is deleted...",ptr->data);
   start=NULL;
   free(ptr);
  }
  else
  {
   while(i<(pos-1))//if pos=4 then while loop will continue from 0 to 3
   {
    temp=ptr;//in last iteration temp=4
    ptr=ptr->next;//ptr=4->next=5
    i++;
   }
   temp->next=ptr->next;//4->next=5->next i.e. 4->next=6
   printf("\n%d is deleted...",ptr->data);
   free(ptr);
  }
 }
}
 void display()
 {
  struct node *ptr=start;
  if(ptr==NULL)
  {
   printf("\nSORRY ! NO ELEMENT ! ! !");
  }
  else
  {
   while(ptr!=NULL)
   {
    printf("%d -> ",ptr->data);
    ptr=ptr->next;
   }
  }
 }
void main()
{
 start=NULL;
 //head;
 int ch,x,pos;
 int z=1;

while(z)  //While loop to keep program in loop
{
 printf("\n\n----------------------------------------\n");

 printf("1. Insert at the begining\n");
```

```
printf("2. Insert at the end\n");

printf("3. Insert at a specific position\n");

printf("4. Delete from begining\n");

printf("5. Delete from the end\n");

printf("6. Delete from a specific position\n");

printf("7. Display\n");

printf("8. Exit");

printf("\n-----------------------------------------\n");

printf("ENTER A CHOICE :");
scanf("%d",&ch);
switch(ch)
{
 case 1: insert_at_beg(x);
         break;
 case 2: insert_at_end(x);
         break;
 case 3: insert_at_specific_position(x,pos);
         break;
 case 4: delete_from_beg();
         break;
 case 5: delete_from_end();
         break;
 case 6: delete_from_specific_position(pos);
         break;
 case 7: display();
         break;
 case 8: exit(0);
         break;
 default : printf("\nOOPS ! WRONG CHOICE !");
          break;
 }
 //goto head;
printf(" \n Do you want to cotin....? press 1 or 0 ");
scanf("%d",&x);
}//end of while loop
}//end of main()
```

//Perform Operations on Singly Linked List like copy,concat,count no of nodes, split and reverse.

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

void copy()
{
 struct node *ptr=start,*t=NULL;
 if(start->next==NULL)//only one element in LL
 {
  struct node* temp=(struct node*)malloc(sizeof(struct node));
  temp->data=start->data;
  temp->next=NULL;
  t=temp;
 }
 else//if LL has more elements
 {
  while(ptr!=NULL)
  {
   struct node* temp=(struct node*)malloc(sizeof(struct node));
   temp->data=ptr->data;
   temp->next=NULL;
   if(t==NULL)//assign t to start of copied LL
   {
    t=temp;//if copied LL is empty then assign t to temp
   }
   else// if t is not empty then traverse till the end
   {
    struct node *add=t;//pointer to link list "t"
    while(add->next!=NULL)
    {
     add=add->next;
    }
    add->next=temp;
   }
   ptr=ptr->next;//to copy next node from original LL to copied LL
  }
 }
 printf("\n->Copied list:\n");
 struct node *k=t;//pointer to link list "t"
 while(k!=NULL)//display copied LL using while loop
   {
    printf("%d ",k->data);
    k=k->next;
   }
}
void count()
{
 struct node *ptr=start;
 int k=0;
```

```
 while(ptr->next!=NULL)
 {
  k++;
  ptr=ptr->next;
 }
 k++;//as intially k=0 so incermented by 1 to count the last node
 printf("\n->Length of the list : %d",k);
}
void concatenate()
{
 struct node *head=NULL;
 int m,j,x;
 printf("Create list2:-\n");
 printf("\nEnter the no. of nodes : ");
 scanf("%d",&m);

 for(j=0;j<m;j++)
 {
  struct node* temp=(struct node*)malloc(sizeof(struct node));
  printf("\nEnter the data %d : ",j+1);
  scanf("%d",&x);
  temp->data=x;
  temp->next=NULL;
  if(head==NULL)//for first node of empty LL
  {
   head=temp;
  }
  else //for next nodes
  {
   struct node *add=head;
   while(add->next!=NULL)//traverse till the last node
   {
     add=add->next;
   }
   add->next=temp;
  }
 }//linked list2 created
 struct node *ptr1=start,*ptr2=head;
 while(ptr1->next!=NULL)//traverse till last node of LL1
 {
   ptr1=ptr1->next;
 }

 ptr1->next=ptr2;//set last node ptr of LL1 to first node of LL2

 printf("\n->Concatenated List :-\n");
 struct node *k=start;
 while(k!=NULL)
   {
    printf("%d ",k->data);
    k=k->next;
   }
}
void split()
{
```

```
struct node *st,*ptr1,*ptr=start,*head=NULL,*k=start;
int p=1,pos;
printf("\nEnter the position of split : ");
scanf("%d",&pos);
while(ptr->next!=NULL)
{
 ptr=ptr->next;
 p++;
 if(p==pos)//if pos=3 when we reach to third node we store its next ptr in "st"
            // and set 3->next to NULL
 {
  st=ptr->next;
  ptr->next=NULL;
 }
}
while(st!=NULL)
{
 if(head==NULL)//assign head as starting point of LL2
 {
  struct node* temp=(struct node*)malloc(sizeof(struct node));
  temp->data=st->data;
  temp->next=NULL;
  head=temp;
  st=st->next;
 }
 else //if head is not null then
 {
  struct node* temp=(struct node*)malloc(sizeof(struct node));
  temp->data=st->data;
  temp->next=NULL;
  ptr1=head;
  while(ptr1->next!=NULL)//travese till the end of new LL
  {
   ptr1=ptr1->next;
  }
  ptr1->next=temp;
  st=st->next;
 }
}
printf("\nList 1:-\n");
while(k!=NULL)
{
 printf("%d ",k->data);
 k=k->next;
}
struct node *j=head;
printf("\nList 2:-\n");
while(j!=NULL)
{
 printf("%d ",j->data);
 j=j->next;
}
}
void reverse()
{
```

```
 struct node *head=NULL,*ptr=start;
 struct node* temp=(struct node*)malloc(sizeof(struct node));
 while(ptr!=NULL)
 {
  if(head==NULL)//if LL is empty
  {
   temp->data=ptr->data;
   temp->next=NULL;
   head=temp;
  }
  else //if LL has more nodes
  {
   struct node* temp=(struct node*)malloc(sizeof(struct node));
   temp->data=ptr->data;
   temp->next=head;
   head=temp;
  }
  ptr=ptr->next;
 }
 struct node *k=head;
 while(k!=NULL)
 {
  printf("%d ",k->data);
  k=k->next;
 }
}
void main()
{
 start=NULL;
 int ch,n,x,i;
 printf("Create list1:-\n");
 printf("\nEnter the no. of nodes : ");
 scanf("%d",&n);

 for(i=0;i<n;i++)
 {
  struct node* newnode=(struct node*)malloc(sizeof(struct node));
  printf("\nEnter the data %d : ",i+1);
  scanf("%d",&x);
  newnode->data=x;
  newnode->next=NULL;
  if(start==NULL)//if the LL is empty
  {
   start=newnode;
  }
  else
  {
   struct node *add=start;

   while(add->next!=NULL)//traverse till the end of LL
   {
     add=add->next;
   }
   add->next=newnode;
  }
```

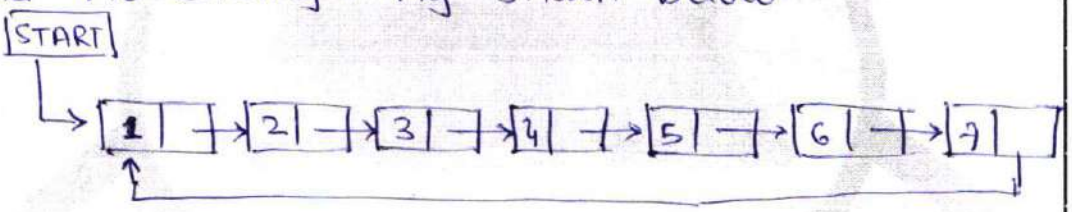
```
}
printf("\nLIST CREATED SUCCESFULLY !\n");

struct node *k=start;
while(k!=NULL)//display elements from LL
{
 printf("%d ",k->data);
 k=k->next;
}
do
{
printf("\n\n--------LINKED LIST OPERATION MENU--------\n");
printf("1. COPY\n2. CONCATENATE\n3. SPLIT\n4. COUNT\n5. REVERSE\n6. EXIT");
printf("\n-----------------------------------\n");
printf("ENTER YOUR CHOICE : ");
scanf("%d",&ch);
switch (ch)
{
 case 1: copy();
          break;
 case 2: concatenate();
          break;
 case 3: split();
          break;
 case 4: count();
          break;
 case 5: reverse();
          break;
 case 6: exit(0);
          break;
 default : printf("OOPS ! WRONG CHOICE !");
           break;
}
 }while(ch!=6);
}
```

Parshvanath Charitable Trust's
A. P. SHAH INSTITUTE OF TECHNOLOGY
(Approved by AICTE New Delhi & Govt. of Maharashtra, Affiliated to University of Mumbai)
(Religious Jain Minority)

Semester: 3 Subject: DSA Academic Year: 17-18

Circular Linked Lists

In a circular linked list, the last node contains a pointer to the first node of the list. We can have a circular singly linked list as well as a circular doubly linked list. While traversing a circular linked list, we can begin at any node and traverse the list in any direction forward or backward until we reach the the same node where we started. Thus a circular linked list has no beginning and no ending. Fig shown below



Inserting a New Node in a Circular Linked list

In this section, we will see how a new node is added into an already existing linked list. We will take two cases and then see how insertion is done in each case.

Case 1 : The new node is inserted at the beginning of the circular linked list

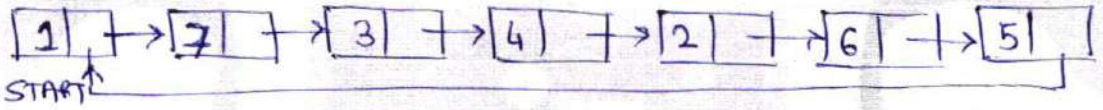
Case 2 : The new node is inserted at the end of the circular linked list.

Parshvanath Charitable Trust's
A. P. SHAH INSTITUTE OF TECHNOLOGY
(Approved by AICTE New Delhi & Govt. of Maharashtra, Affiliated to University of Mumbai)
(Religious Jain Minority)

Semester: 3 Subject: DSA Academic Year: 17-18

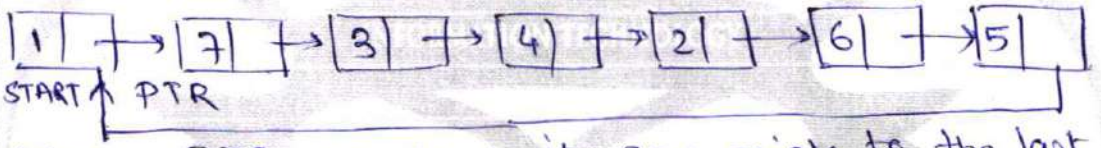
Inserting a Node at the Beginning of a Circular Linked List.

Consider the linked list shown in following fig. Suppose we want to add a new node with data 9 as the first node of list. Then the following changing will be in the linked list.

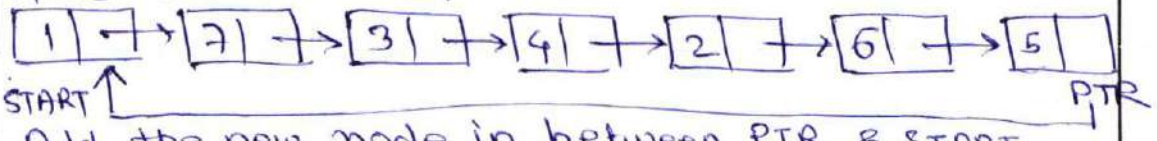


Allocate memory for the new node and initialize its DATA part to 9

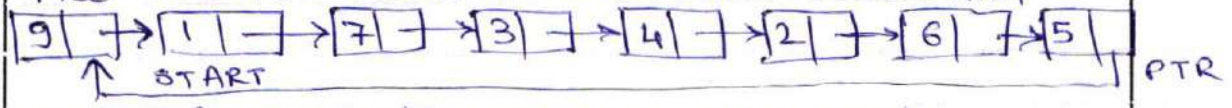
Take a pointer variable PTR that points to the START node of the list.



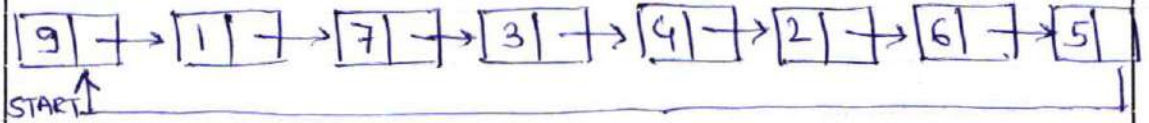
Move PTR so that it now points to the last node of the list.



Add the new node in between PTR & START



Make START point to the new node



Inserting a new node at the beginning of circular linked list.

Parshvanath Charitable Trust's
A. P. SHAH INSTITUTE OF TECHNOLOGY
(Approved by AICTE New Delhi & Govt. of Maharashtra, Affiliated to University of Mumbai)
(Religious Jain Minority)

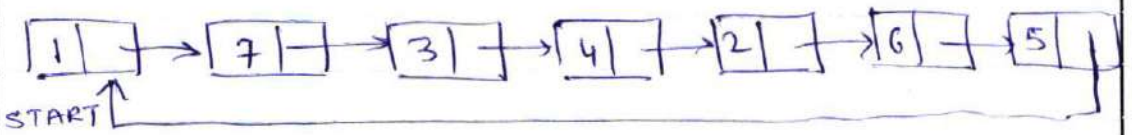
Semester: 3 Subject: DSA Academic Year: 17-18

```
Step 1: IF AVAIL = NULL
        Write OVERFLOW
        Go to Step 11
    [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL → NEXT
Step 4: SET NEW_NODE → DATA = VAL
Step 5: SET PTR = START
Step 6: Repeat Step 7 while PTR →
        NEXT != START
Step 7: PTR = PTR → NEXT
    [END OF LOOP]
Step 8: SET NEW_NODE → NEXT =
        START
Step 9: SET PTR → NEXT = NEXT_NODE
Step 10: SET START = NEW_NODE
Step 11: EXIT
```

Algorithm to insert a new node at the beginning

Inserting a Node at the END of a circular Linked List

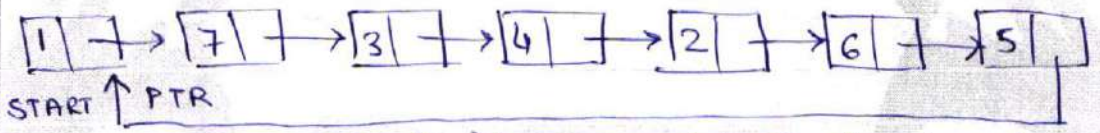
Suppose we want to add a new node with data 9 as the last node of the list. Then the following changes will be done in the linked list



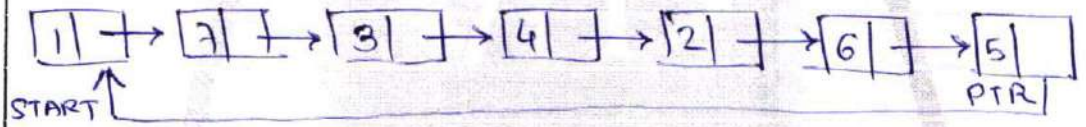
Parshvanath Charitable Trust's
A. P. SHAH INSTITUTE OF TECHNOLOGY
(Approved by AICTE New Delhi & Govt. of Maharashtra, Affiliated to University of Mumbai)
(Religious Jain Minority)

Allocate memory for the new node and initialize its DATA part to 9.

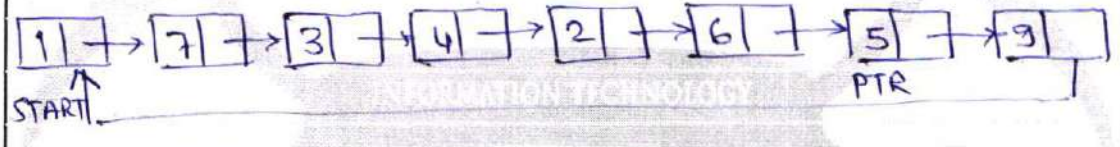
Take a pointer variable PTR which will initially point to START



Move PTR so that it now points to the least node of the list.



Add the new node after the node pointed by PTR.



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

Parshvanath Charitable Trust's
A. P. SHAH INSTITUTE OF TECHNOLOGY
(Approved by AICTE New Delhi & Govt. of Maharashtra, Affiliated to University of Mumbai)
(Religious Jain Minority)

Semester: 3 Subject: DSA Academic Year: 17-18

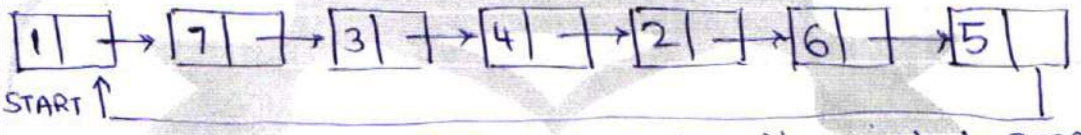
Deleting a Node from a Circular Linked List

In this section, we will discuss how a node is deleted from an already existing circular linked list. We will take two cases and then see how deletion is done in each case. Rest of the cases of deletion are same as that given for singly linked list.

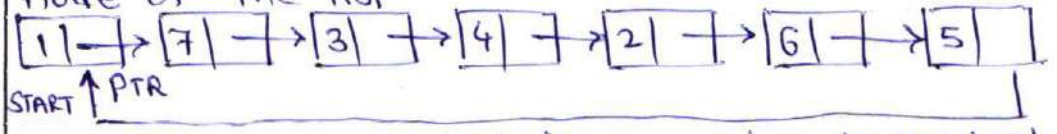
Case 1 : The first node is deleted
Case 2 : The last node is deleted

Deleting the First Node from a Circular Linked List

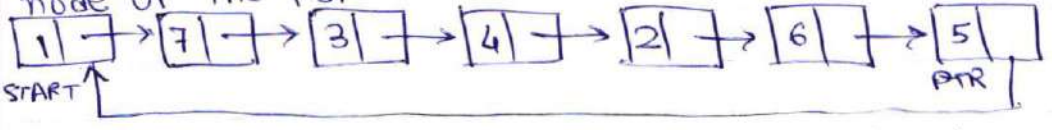
Consider the circular linked list shown in following fig. When we want to delete a node from the beginning of the list, then the following changes will be done in the linked list



Take a variable PTR and make it point to START node of the list

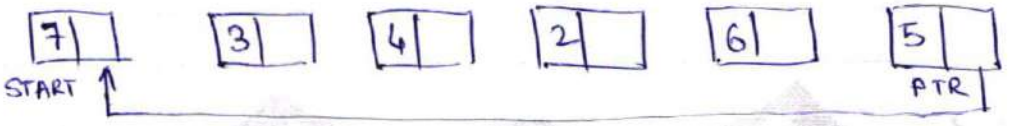


Move PTR further so that it now points to the last node of the list



The NEXT part of PTR is made to point to the second node of the list and the memory of

the first node is freed. The second node becomes the first node of the list.

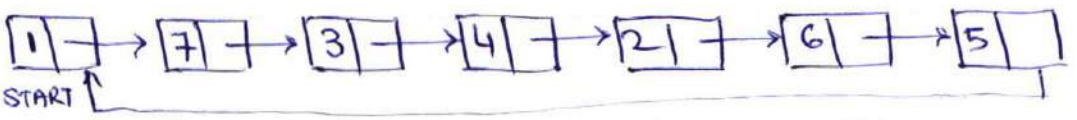



```
Step 1 : IF START = NULL
         Write UNDERFLOW
         Go to Step 8
       [END OF IF]
Step 2 : SET PTR = START
step 3 : Repeat Step 4 while PTR → NEXT
                != START
Step 4 : SET PTR = PTR → NEXT
       [END OF LOOP]
Step 5 : SET PTR → NEXT = START → NEXT
step 6 : FREE START
step 7 : SET START = PTR → NEXT
step 8 : EXIT
```

Algorithm to delete the first node

Deleting the Last Node from a Circular Linked List

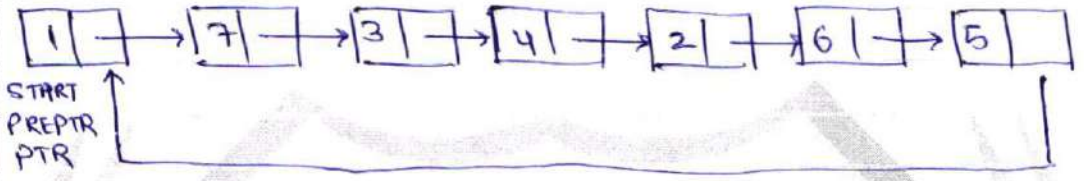
Consider the circular linked list shown in following fig. Suppose we want to delete the last node from the linked list, then the folbowing changes will be done in the linked list.



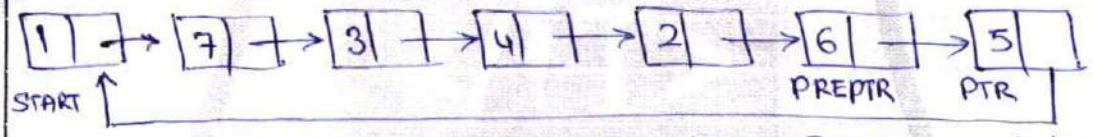
Parshvanath Charitable Trust's
A. P. SHAH INSTITUTE OF TECHNOLOGY
(Approved by AICTE New Delhi & Govt. of Maharashtra, Affiliated to University of Mumbai)
(Religious Jain Minority)

Semester: 3 Subject: DSA Academic Year: 17-18

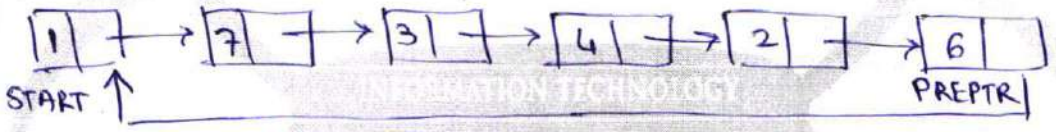
Take two pointer PREPTR and PTR which will initially point to START



Move PTR so that it points to the last node of the list. PREPTR will always point to the node preceding PTR.



Make the PREPTR's next part store START node's address and free the space allocated for PTR Now PREPTR is last node of the list



step 1 : IF START = NULL
Write UNDERFLOW
Go to Step 8
[END OF IF]
Step 2 : SET PTR = START
step 3 : Repeat Step 4 and 5 while
PTR → NEXT ! = START
Step 4 : SET PREPTR = PTR
Step 5 : SET PTR = PTR → NEXT
[END OF LOOP]
Step 6 : SET PREPTR → NEXT = START
Step 7 : FREE PTR
Step 8 : EXIT

```
/*
1.  * C Program to Demonstrate Circular Single Linked List
2.  */



3. #include <stdio.h>
4. #include <stdlib.h>
5.
6. struct node
7. {
8.     int data;
9.     struct node *link;
10. };
11.
12. struct node *head = NULL, *x, *y, *z;
13.
14. void create();
15. void ins_at_beg();
16. void ins_at_pos();
17. void del_at_beg();
18. void del_at_pos();
19. void traverse();
20. void search();
21. void sort();
22. void update();
23. void rev_traverse(struct node *p);
24.
25. void main()
26. {
27.     int ch;
28.
29.     printf("\n 1.Creation \n 2.Insertion at beginning \n 3.Insertion at
remaining");
30.     printf("\n4.Deletion at beginning \n5.Deletion at remaining
\n6.traverse");
31.     printf("\n7.Search\n8.sort\n9.update\n10.Exit\n");
32.     while (1)
33.     {
34.         printf("\n Enter your choice:");
35.         scanf("%d", &ch);
36.         switch(ch)
37.         {
38.         case 1:
39.             create();
40.             break;
41.         case 2:
42.             ins_at_beg();
43.             break;
44.         case 3:
45.             ins_at_pos();
46.             break;
47.         case 4:
48.             del_at_beg();
49.             break;
50.         case 5:
```

```
51.             del_at_pos();
52.             break;
53.         case 6:
54.             traverse();
55.             break;
56.         case 7:
57.             search();
58.             break;
59.         case 8:
60.             sort();
61.             break;
62.         case 9:
63.             update();
64.             break;
65.         case 10:
66.             rev_traverse(head);
67.             break;
68.         default:
69.             exit(0);
70.         }
71.     }
72. }
73.
74. /*Function to create a new circular linked list*/
75. void create()
76. {
77.     int c;
78.
79.     x = (struct node*)malloc(sizeof(struct node));
80.     printf("\n Enter the data:");
81.     scanf("%d", &x->data);
82.     x->link = x;
83.     head = x;
84.     printf("\n If you wish to continue press 1 otherwise 0:");
85.     scanf("%d", &c);
86.     while (c != 0)
87.     {
88.         y = (struct node*)malloc(sizeof(struct node));
89.         printf("\n Enter the data:");
90.         scanf("%d", &y->data);
91.         x->link = y;
92.         y->link = head;
93.         x = y;
94.         printf("\n If you wish to continue press 1 otherwise 0:");
95.         scanf("%d", &c);
96.     }
97. }
98.
99. /*Function to insert an element at the begining of the list*/
100.
101. void ins_at_beg()
102. {
103.     x = head;
104.     y = (struct node*)malloc(sizeof(struct node));
105.     printf("\n Enter the data:");
106.     scanf("%d", &y->data);
```

```
107.     while (x->link != head)
108.     {
109.         x = x->link;
110.     }
111.     x->link = y;
112.     y->link = head;
113.     head = y;
114. }
115.
116. /*Function to insert an element at any position the list*/
117.
118. void ins_at_pos()
119. {
120.     struct node *ptr;
121.     int c = 1, pos, count = 1;
122.
123.     y = (struct node*)malloc(sizeof(struct node));
124.     if (head == NULL)
125.     {
126.         printf("cannot enter an element at this place");
127.     }
128.     printf("\n Enter the data:");
129.     scanf("%d", &y->data);
130.     printf("\n Enter the position to be inserted:");
131.     scanf("%d", &pos);
132.     x = head;
133.     ptr = head;
134.     while (ptr->link != head)
135.     {
136.         count++;
137.         ptr = ptr->link;
138.     }
139.     count++;
140.     if (pos > count)
141.     {
142.         printf("OUT OF BOUND");
143.         return;
144.     }
145.     while (c < pos)
146.     {
147.         z = x;
148.         x = x->link;
149.         c++;
150.     }
151.     y->link = x;
152.     z->link = y;
153. }
154.
155. /*Function to delete an element at any begining of the list*/
156.
157. void del_at_beg()
158. {
159.     if (head == NULL)
160.         printf("\n List is empty");
161.     else
162.     {
```

```
163.         x = head;
164.         y = head;
165.         while (x->link !=  head)
166.         {
167.             x = x->link;
168.         }
169.         head = y->link;
170.         x->link = head;
171.         free(y);
172.     }
173. }
174.
175. /*Function to delete an element at any position the list*/
176.
177. void del_at_pos()
178. {
179.     if (head == NULL)
180.         printf("\n List is empty");
181.     else
182.     {
183.         int c = 1, pos;
184.         printf("\n Enter the position to be deleted:");
185.         scanf("%d", &pos);
186.         x = head;
187.         while (c < pos)
188.         {
189.             y = x;
190.             x = x->link;
191.             c++;
192.         }
193.         y->link = x->link;
194.         free(x);
195.     }
196. }
197.
198. /*Function to display the elements in the list*/
199.
200. void traverse()
201. {
202.     if (head == NULL)
203.         printf("\n List is empty");
204.     else
205.     {
206.         x = head;
207.         while (x->link !=  head)
208.         {
209.             printf("%d->", x->data);
210.             x = x->link;
211.         }
212.         printf("%d", x->data);
213.     }
214. }
215.
216. /*Function to search an element in the list*/
217.
218. void search()
```

```
{
    int search_val, count = 0, flag = 0;
    printf("\nenter the element to search\n");
    scanf("%d", &search_val);
    if (head == NULL)
        printf("\nList is empty nothing to search");
    else
    {
        x = head;
        while (x->link !=  head)
        {
            if (x->data == search_val)
            {
                printf("\nthe element is found at %d", count);
                flag = 1;
                break;
            }
            count++;
            x = x->link;
        }
        if (x->data == search_val)
        {
            printf("element found at postion %d", count);
        }
        if (flag == 0)
        {
            printf("\nelement not found");
        }

    }
}

/*Function to sort the list in ascending order*/

void sort()
{
    struct node *ptr, *nxt;
    int temp;

    if (head == NULL)
    {
        printf("empty linkedlist");
    }
    else
    {
        ptr = head;
        while (ptr->link !=  head)
        {
            nxt = ptr->link;
            while (nxt !=  head)
            {
                if (nxt !=  head)
                {
                    if (ptr->data > nxt->data)
                    {
                        temp = ptr->data;
```

```
                        ptr->data = nxt->data;
                        nxt->data = temp;
                    }
                }
                else
                {
                    break;
                }
                nxt = nxt->link;
            }
            ptr = ptr->link;
        }
    }
}

/*Function to update an element at any position the list*/
void update()
{
    struct node *ptr;
    int search_val;
    int replace_val;
    int flag = 0;

    if (head == NULL)
    {
        printf("\n empty list");
    }
    else
    {
        printf("enter the value to be edited\n");
        scanf("%d", &search_val);
        fflush(stdin);
        printf("enter the value to be replace\n");
        scanf("%d", &replace_val);
        ptr = head;
        while (ptr->link !=  head)
        {
            if (ptr->data == search_val)
            {
                ptr->data = replace_val;
                flag = 1;
                break;
            }
            ptr = ptr->link;
        }
        if (ptr->data == search_val)
        {
            ptr->data = replace_val;
            flag = 1;
        }
        if (flag == 1)
        {
            printf("\nUPdate sucessful");
        }
        else
        {
```

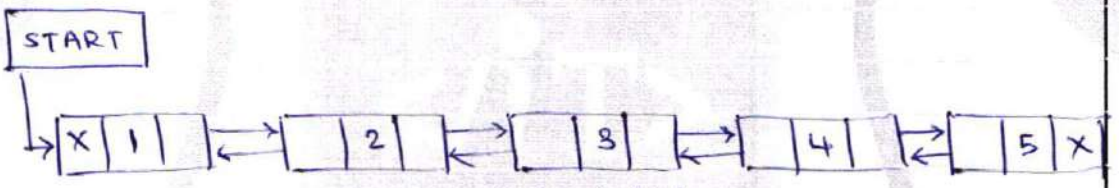
```
331.             printf("\n update not successful");
332.         }
333.     }
334. }
335.
336. /*Function to display the elements of the list in reverse order*/
337.
338. void rev_traverse(struct node *p)
339. {
340.     int i = 0;
341.
342.     if (head == NULL)
343.     {
344.         printf("empty linked list");
345.     }
346.     else
347.     {
348.         if (p->link !=  head)
349.         {
350.             i = p->data;
351.             rev_traverse(p->link);
352.             printf(" %d", i);
353.         }
354.         if (p->link == head)
355.         {
356.             printf(" %d", p->data);
357.         }
358.     }
359. }
```

Parshvanath Charitable Trust's
A. P. SHAH INSTITUTE OF TECHNOLOGY
(Approved by AICTE New Delhi & Govt. of Maharashtra, Affiliated to University of Mumbai)
(Religious Jain Minority)

Semester: 3 Subject: DSA Academic Year: 17-18

DOUBLY LINKED LISTS

A doubly linked list or two-way linked list is a more complex type of linked list which contains a pointer to the next as well as the previous node in the sequence. Therefore it consists of three parts - data, pointer to the next node and a pointer to the previous node as shown in following fig.



Doubly linked list

In C, the structure of a doubly linked list can be given as

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

The PREV field of the first node and the NEXT field of the last node will contain NULL. The PREV field is used to store the address of the proceding node, which enables us to traverse the list in the backward direction.

Parshvanath Charitable Trust's
A. P. SHAH INSTITUTE OF TECHNOLOGY
(Approved by AICTE New Delhi & Govt. of Maharashtra, Affiliated to University of Mumbai)
(Religious Jain Minority)

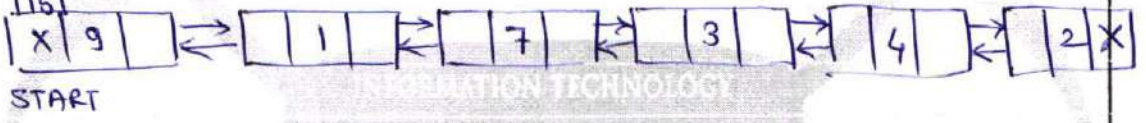
Semester: 3 Subject: DSA Academic Year: 17-18

Inserting a Node at the Beginning of a Doubly Linked List

Consider the doubly linked list shown in following figure. Suppose we want to add a new node with data 9 as the first node of the list Then the following changes will be done in the linked list

Allocate memory for the new node and initialize its DATA part to 9 and PREV field to NULL

Add the new node before the START node. Now the new node becomes the first node of the list.




```
Step 1 : IF AVAIL = NULL
           Write OVERFLOW
           Goto Step 9
         [END OF IF]
Step 2 : SET NEW_NODE = AVAIL
step 3 : SET AVAIL = AVAIL → 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 START → PREV = NEW_NODE
Step 8 : SET START = NEW_NODE
Step 9 : EXIT
```

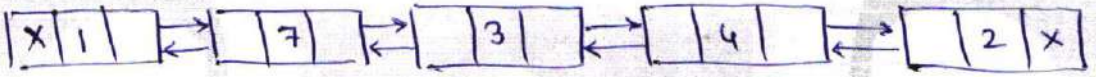
Algorithm to insert a new node at the beginning

Parshvanath Charitable Trust's
A. P. SHAH INSTITUTE OF TECHNOLOGY
(Approved by AICTE New Delhi & Govt. of Maharashtra, Affiliated to University of Mumbai)
(Religious Jain Minority)

Semester: 3 Subject: DSA Academic Year: 17-18

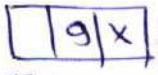
Inserting a Node at the END end of a Doubly Linked List

Consider the doubly linked list shown in figure Suppose we want to add a new node with data 9 as the last node of the list. Then the following changes will be done in the linked list.

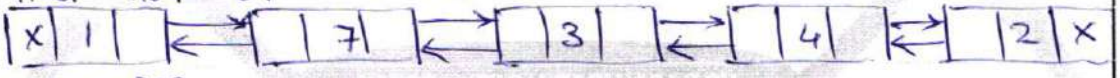


START

Allocate memory for the new node and initialize its DATA part to 9 and its NEXT field to NULL



Take a pointer variable PTR and make it point to the first node of the list



START, PTR

Move PTR so that it point to the last node of the list Add the new node after the node pointed by PTR

START

Inserting a new node at the end of a doubly linked list

```
Step 1 : IF AVAIL = NULL
         Write OVERFLOW
         Goto Step 11
         [END OF IF]
Step 2 : SET NEW_NODE = AVAIL
```

Step 3 : SET AVAIL = AVAIL → NEXT
Step 4 : SET ~~AVAIL~~ NEW_NODE ~~AVAIL~~ DATA = VAL
Step 5 : SET NEW_NODE → NEXT = NULL
Step 6 : SET PTR = START
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 : SET NEW_NODE → PREV = PTR
Step 11 : EXIT

Inserting a Node After a Given Node in a Doubly Linked List

Consider the doubly linked list shown in figure following. Suppose we want to add a new node with value 9 after the node containing 3. Before discussing the changes that will be done in the linked list, let us first look at the algorithm

Step 1 : IF AVAIL = NULL
Write OVERFLOW
Goto step 12
[END OF IF]
Step 2 : SET NEW_NODE = AVAIL
Step 3 : SET AVAIL = AVAIL → NEXT
Step 4 : SET NEW_NODE → DATA = VAL
Step 5 : SET PTR = START
Step 6 : Repeat Step 7 while PTR → DATA != NUM

Parshvanath Charitable Trust's
A. P. SHAH INSTITUTE OF TECHNOLOGY
(Approved by AICTE New Delhi & Govt. of Maharashtra, Affiliated to University of Mumbai)
(Religious Jain Minority)

Semester: 3 Subject: DSA Academic Year: 17-18

Step 7 : SET PTR = PTR → NEXT
[END OF LOOP]
Step 8 : SET NEW_NODE → NEXT = PTR → NEXT
Step 9 : SET NEW_NODE → PREV = PTR
Step 10 : SET PTR → NEXT = NEW_NODE
Step 11 : SET PTR → NEXT → PREV = NEW_NODE
step 12 : EXIT

Inserting a Node Before a Given Node in Doubly Linked list

step 1 : IF AVAIL = NULL
Write OVERFLOW
Go to Step 12
[END OF IF]
Step 2 : SET NEW_NODE = AVAL
step 3 : SET AVAIL = AVAIL → NEXT
step 4 : SET NEW_NODE → DATA = VAL
Step 5 : SET PTR = START
step 6 : Repeat Step 7 while PTR → Data ! = NUM
Step 7 : SET PTR = PTR → NEXT
[END OF LOOP]
Step 8 : SET NEW_NODE → NEXT = PTR
Step 9 : SET NEW_NODE → PREV = PTR - PREV
Step 10 : SET PTR → PREV = NEW_NODE
Step 11 : SET PTR → PREV → NEXT = NEW_NODE
Step 12 : EXIT

Algorithm in insert a new node befor. a given node.

Parshvanath Charitable Trust's
A. P. SHAH INSTITUTE OF TECHNOLOGY
(Approved by AICTE New Delhi & Govt. of Maharashtra, Affiliated to University of Mumbai)
(Religious Jain Minority)

Semester: 3 Subject: DSA Academic Year: 17-18

Deleting a Node from a Doubly Linked list

Deleting the first Node from a Doubly Linked list

Step 1 : IF START = NULL
Write UNDERFLOW
Go to Step 6
[END OF IF]
Step 2 : SET PTR = START
Step 3 : SET START = START → NEXT
Step 4 : SET START → PREV = NULL
Step 5 : FREE PTR
Step 6 : EXIT

Deleting the last Node from a Doubly Linked list

Step 1 : IF START = NULL
Write UNDERFLOW
Go to Step 7
[END OF IF]
Step 2 : SET PTR = START
Step 3 : Repeat Step 4 while PTR → NEXT != NULL
Step 4 : SET PTR = PTR → NEXT
[END OF LOOP]
Step 5 : SET PTR → PREV → NEXT = NULL
Step 6 : FREE PTR
Step 7 : EXIT

Algorithm to delete last node

Deleting the Node After a Given Node in a Doubly Linked List

```
Step 1 : IF START = NULL
           Write UNDERFLOW
           Go to Step 9
         [END OF IF]
Step 2 : SET PTR = START
Step 3 : Repeat Step 4 while PTR →
           DATA != NUM
Step 4 : SET PTR = PTR → NEXT
       [END OF LOOP]
Step 5 : SET TEMP = PTR → NEXT
Step 6 : SET PTR → NEXT = TEMP → NEXT
Step 7 : SET TEMP → NEXT → PREV = PTR
Step 8 : FREE TEMP
Step 9 : EXIT
```

Algorithm to delete a node after a given node

Deleting the Node Before a Given Node in a Doubly Linked List

```
Step 1 : IF START = NULL
           Write UNDERFLOW
           Go to Step 9
         [END OF IF]
Step 2 : SET PTR = START
Step 3 : Repeat Step 4 while PTR → DATA
                                != NUM
```

Parshvanath Charitable Trust's
A. P. SHAH INSTITUTE OF TECHNOLOGY
(Approved by AICTE New Delhi & Govt. of Maharashtra, Affiliated to University of Mumbai)
(Religious Jain Minority)

Semester: 3 Subject: DSA Academic Year: 2017-18

Step 4 : SET PTR = PTR → NEXT
[END OF LOOP]
Step 5 : SET TEMP = PTR → PREV
Step 6 : SET TEMP → PREV → NEXT = PTR
Step 7 : SET PTR → PREV = TEMP → PREV
Step 8 : FREE TEMP
Step 9 : EXIT

// Implementation of Doubly Linked List

```
#include<stdio.h>
#include<stdlib.h>

struct Node  {
        int data;
        struct Node* next;
        struct Node* prev;
};

struct Node* head; // global variable - pointer to head node.

//Creates a new Node and returns pointer to it.
struct Node* GetNewNode(int x) {
        struct Node* newNode
                = (struct Node*)malloc(sizeof(struct Node));
        newNode->data = x;
        newNode->prev = NULL;
        newNode->next = NULL;
        return newNode;
}

//Inserts a Node at head of doubly linked list
void InsertAtHead(int x) {
        struct Node* newNode = GetNewNode(x);
        if(head == NULL) {
                head = newNode;
                return;
        }
        head->prev = newNode;
        newNode->next = head;
        head = newNode;
}

//Inserts a Node at tail of Doubly linked list
void InsertAtTail(int x) {
        struct Node* temp = head;
        struct Node* newNode = GetNewNode(x);
        if(head == NULL) {
                head = newNode;
                return;
        }
        while(temp->next != NULL) temp = temp->next; // Go To last Node
        temp->next = newNode;
        newNode->prev = temp;
}

//Prints all the elements in linked list in forward traversal order
void Print() {
        struct Node* temp = head;
        printf("Forward: ");
        while(temp != NULL) {
                printf("%d ",temp->data);
                temp = temp->next;
```

```
        }
        printf("\n");
}

//Prints all elements in linked list in reverse traversal order.
void ReversePrint() {
        struct Node* temp = head;
        if(temp == NULL) return; // empty list, exit
        // Going to last Node
        while(temp->next != NULL) {
                temp = temp->next;
        }
        // Traversing backward using prev pointer
        printf("Reverse: ");
        while(temp != NULL) {
                printf("%d ",temp->data);
                temp = temp->prev;
        }
        printf("\n");
}

int main() {

        /*Driver code to test the implementation*/
        head = NULL; // empty list. set head as NULL.

        // Calling an Insert and printing list both in forward as well as reverse direction.
        InsertAtTail(2); Print(); ReversePrint();
        InsertAtTail(4); Print(); ReversePrint();
        InsertAtHead(6); Print(); ReversePrint();
        InsertAtTail(8); Print(); ReversePrint();

}
```

Parshvanath Charitable Trust's
A. P. SHAH INSTITUTE OF TECHNOLOGY
(Approved by AICTE New Delhi & Govt. of Maharashtra, Affiliated to University of Mumbai)
(Religious Jain Minority)

Semester: 3 Subject: D&A Academic Year: 2017-18

Application of Linked List

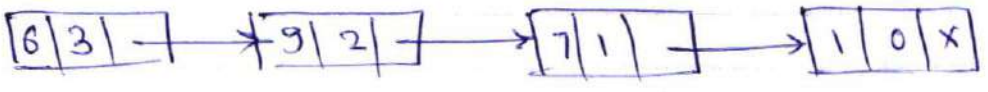
Linked lists can be used to represent polynomials and the different operation that can be performed on them. In this section, we will see how polynomials are represented in the memory using linked list

Polynomial Representation

Let us see how a polynomials ~~and the different operations~~ in the memory using a linked list. Consider a polynomial

$$6x^3 + 9x^2 + 7x + 1.$$

Every individual term in a polynomial consists of two parts a coefficient and a power. Here 6, 9, 7 and 1 are the coefficients of the terms that have 3, 2, 1 and 0 as their powers respectively.
Every term of a polynomial can be represented as a node of the linked list.
Following is the linked representation of the term of the above polynomial.



Linked representation of polynomial

Linked List Application

- Linked Lists can be used to implement Stacks , Queues.
- Linked Lists can also be used to implement Graphs. (Adjacency list representation of Graph).
- Implementing Hash Tables :- Each Bucket of the hash table can itself be a linked list. (Open chain hashing).
- Undo functionality in Photoshop or Word . Linked list of states.
- A polynomial can be represented in an array or in a linked list by simply storing the coefficient and exponent of each term.
- However, for any polynomial operation , such as addition or multiplication of polynomials , linked list representation is more easier to deal with.
- Linked lists are useful for dynamic memory allocation.
- The real life application where the circular linked list is used is our Personal Computers, where multiple applications are running.
- All the running applications are kept in a circular linked list and the OS gives a fixed time slot to all for running. The Operating System keeps on iterating over the linked list until all the applications are completed

Parshvanath Charitable Trust's
A. P. SHAH INSTITUTE OF TECHNOLOGY
(Approved by AICTE New Delhi & Govt. of Maharashtra, Affiliated to University of Mumbai)
(Religious Jain Minority)

Semester: 3 Subject: DSA Academic Year: 2017-18

Linked Representation of Stacks

We have seen how a stack is created using an array. This technique of creating a stack is easy but the drawback is that the array must be declared to have some fixed size. In case stake is a very small one or its maximum size is known in advance, then the array implementation of the stack gives an efficient implementation. But if the array size cannot be determined in advance then the other alternative ie linked representation, is used.

The storage requirement of linked representation of the stack with n element is O(n) and the typical time requirement for the operation is O(1)

In a linked stack, every node has two parts - one that stores data & another that stores the address of next node. The start pointer of the linked list is used as TOP. All insertions and deletions are done at the node pointed by TOP. If TOP = NULL, then it indicates that the stack is empty

Linked Stack

Parshvanath Charitable Trust's
A. P. SHAH INSTITUTE OF TECHNOLOGY
(Approved by AICTE New Delhi & Govt. of Maharashtra, Affiliated to University of Mumbai)
(Religious Jain Minority)

Operation on a linked STACK

The PUSH operation is used to insert an element into the stack. The new element is added at the topmost position of the Stack

```
Step 1 : Allocate memory for the new
         node and name it as NEW_NODE
Step 2 : SET NEW_NODE -> DATA = VAL
Step 3 : IF TOP = NULL
             SET NEW_NODE -> NEXT = NULL
             SET TOP = NEW_NODE
         ELSE
             SET NEW_NODE -> NEXT = TOP
             SET TOP = NEW_NODE
         [END OF IF]
Step 4 : END
```

POP Operation

The pop operation is used to delete the topmost element from a stack. However before deleting the value we must first check if TOP = NULL because if this is the case, then it means that the stack is empty and no more deletions can be done. If an attempt is made to delete a value from a stack that is already empty, an underflow message is printed.

```
Step 1 : IF TOP = NULL
         PRINT " UNDEFLOW "
         Goto Step 5
       [END OF IF]
Step 2 : SET PTR = TOP
Step 3 : SET TOP = TOP → NEXT
Step 4 : FREE PTR
Step 5 : END
```

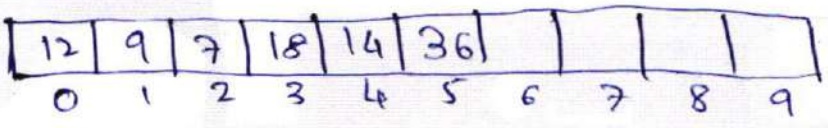
```c
#include<stdio.h>
#include<conio.h>

struct Node
{
   int data;
   struct Node *next;
}*top = NULL;

void push(int);
void pop();
void display();

void main()
{
   int choice, value;
   clrscr();
   printf("\n:: Stack using Linked List ::\n");
   while(1){
      printf("\n****** MENU ******\n");
      printf("1. Push\n2. Pop\n3. Display\n4. Exit\n");
      printf("Enter your choice: ");
      scanf("%d",&choice);
      switch(choice){
         case 1: printf("Enter the value to be insert: ");
                 scanf("%d", &value);
                 push(value);
                 break;
         case 2: pop(); break;
         case 3: display(); break;
         case 4: exit(0);
         default: printf("\nWrong selection!!! Please try again!!!\n");
      }
   }
}
void push(int value)
{
   struct Node *newNode;
   newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = value;
   if(top == NULL)
      newNode->next = NULL;
   else
      newNode->next = top;
   top = newNode;
   printf("\nInsertion is Success!!!\n");
}
void pop()
{
   if(top == NULL)
      printf("\nStack is Empty!!!\n");
   else{
      struct Node *temp = top;
      printf("\nDeleted element: %d", temp->data);
      top = temp->next;
      free(temp);
   }
}
void display()
{
   if(top == NULL)
      printf("\nStack is Empty!!!\n");
   else{
      struct Node *temp = top;
      while(temp->next != NULL){
         printf("%d--->",temp->data);
         temp = temp -> next;
      }
      printf("%d--->NULL",temp->data);
   }
```

Parshvanath Charitable Trust's
A. P. SHAH INSTITUTE OF TECHNOLOGY
(Approved by AICTE New Delhi & Govt. of Maharashtra, Affiliated to University of Mumbai)
(Religious Jain Minority)

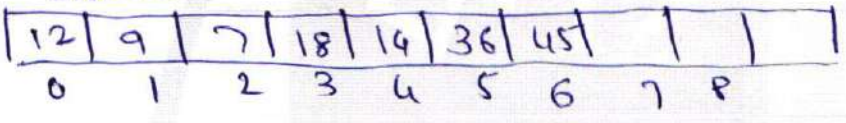
Semester: 3 Subject: DSA Academic Year: 2017-18

ARRAY REPRESENTATION OF QUEUES

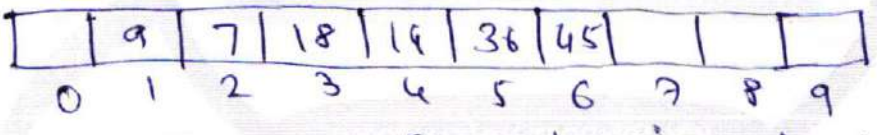
Queues can be easily represented using linear arrays. As stated earlier, every queue has FRONT and REAR variable that point to the position from where deletions and insertions can be done, respectively The array representation of a queue is



Queue



Queue after insertion of a new element



Queue after deletion of element

Queue : Operations on Queues

In above fig FRONT = 0 and REAR = 5 Suppose we want to add another element with value 45 then REAR would be incremented by 1 and the value would be stored at the position pointed by REAR. The queue after addition would be as shown in second Here FRONT = 0 and REAR = 6 Every time a new element has to

Parshvanath Charitable Trust's
A. P. SHAH INSTITUTE OF TECHNOLOGY
(Approved by AICTE New Delhi & Govt. of Maharashtra, Affiliated to University of Mumbai)
(Religious Jain Minority)

Semester: 3 Subject: DSA Academic Year: 2017-18

be added, we repeat the same procedure.

If we want to delete an element from the queue, then the value of FRONT will be ~~increase~~ incremented Deletion are done from only this end of queue. The queue after deletion will be shown in fig 3.

However, before using insertion of an element in queue, we must check for overflow condition. An overflow will occur when we try to insert an element into a queue that is already full. When REAR = MAX-1, where MAX is the size of queue, we have an OVERFLOW condition. Note that we have written MAX-1 because the index starts from 0.

```
Step 1 : IF REAR = MAX-1
           Write OVERFLOW
           Go to Step 4
         [END OF IF]
Step 2 : IF FRONT = -1 and
           REAR = -1
           SET FRONT = REAR = 0
         ELSE
           SET REAR = REAR + 1
         [END OF IF]
Step 3 : SET QUEUE[REAR] = NUM
Step 4 : EXIT
```

Algorithm to insert an element in a Queue

Parshvanath Charitable Trust's
A. P. SHAH INSTITUTE OF TECHNOLOGY
(Approved by AICTE New Delhi & Govt. of Maharashtra, Affiliated to University of Mumbai)
(Religious Jain Minority)

Similarly, before deleting an element from a queue, we must check for underflow conditions. An underflow condition occurs when we try to delete an element from a queue that is already empty. IF FRONT = -1 and REAR = -1 it means there is no element in the queue.

```
Step 1: IF FRONT = -1 OR FRONT > REAR
          Write UNDERFLOW
        ELSE
          SET VAL = QUEUE[FRONT]+1
        [END OF IF]
Step 2: EXIT
```

Algorithm to delete an element from a queue

```c
#include<stdio.h>
#include<conio.h>

struct Node
{
   int data;
   struct Node *next;
}*front = NULL,*rear = NULL;

void insert(int);
void delete();
void display();

void main()
{
   int choice, value;
   clrscr();
   printf("\n:: Queue Implementation using Linked List ::\n");
   while(1){
      printf("\n****** MENU ******\n");
      printf("1. Insert\n2. Delete\n3. Display\n4. Exit\n");
      printf("Enter your choice: ");
      scanf("%d",&choice);
      switch(choice){
         case 1: printf("Enter the value to be insert: ");
                 scanf("%d", &value);
                 insert(value);
                 break;
         case 2: delete(); break;
         case 3: display(); break;
         case 4: exit(0);
         default: printf("\nWrong selection!!! Please try again!!!\n");
      }
   }
}
void insert(int value)
{
   struct Node *newNode;
   newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = value;
   newNode -> next = NULL;
   if(front == NULL)
      front = rear = newNode;
   else{
      rear -> next = newNode;
      rear = newNode;
   }
   printf("\nInsertion is Success!!!\n");
}
void delete()
{
   if(front == NULL)
      printf("\nQueue is Empty!!!\n");
   else{
      struct Node *temp = front;
      front = front -> next;
      printf("\nDeleted element: %d\n", temp->data);
      free(temp);
   }
}
void display()
{
   if(front == NULL)
      printf("\nQueue is Empty!!!\n");
   else{
      struct Node *temp = front;
      while(temp->next != NULL){
         printf("%d--->",temp->data);
         temp = temp -> next;
      }
      printf("%d--->NULL\n",temp->data);
   }
```

```
}
```