UNIT-I

Abstract Data Type (ADT)

When an application requires a special kind of data which is not available as built in data type then it is the programmer responsibility to implement his own kind of data. Here the programmer has to specify how to store a value for that data, what are the operations that can meaningfully manipulate variables of that kind of data, amount of memory required to store a value. The programmer has to decide all these things and accordingly implement them. Programers own datatype is called as Abstract Data Type.

In computer science, an abstract data type (ADT) is a mathematical model for a certain class of data structures that have similar behavior. (or)Abstract Data Type(ADT) is a data type, where only behavior is defined but not implementation.

Advantages of ADT:-

1.ADT is reusable and ensures robust data structure.
2.It reduces coding efforts.
3.Encapsulation ensures that data cannot be correpted.
4.ADT is based on principles of object oriented programming and software engineering.
5.It specifies error conditions associated with operations.

Data type:- A data type is a term which refers to the kind of data that may appear in a computation.Ex: Int, Char, String etc.,

There are different types of data types in programming like built-in, user-defined and abstract etc.,

Concept of Data Structures:- Before going to Data structures we should know the essential tasks for data
=>Storage representation
=>Retrieval of stored data
=>Transformation of user data

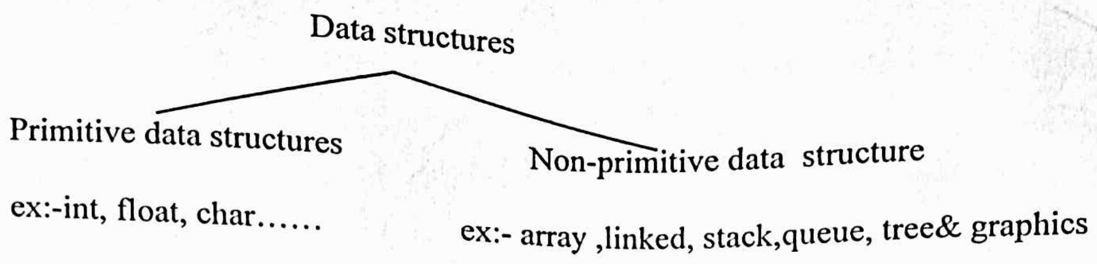
For any kind of given user data, its structure implies the following.

1. **Domain**:- The range of values that the data may have.
2. **Function**:- The set of operations which may legally applied to elements of the data object.
3. **Axioms**:- The set of rules with which the different operations belongs to function. Definition for a data structure defined as, a data structure D is a triplet, that is combination of Domain, Function and axioms. DS = (D,F,A).

Overview of Data Structures:-In computer science, a **data structure** is a particular way of organizing data in a computer so that it can be used efficiently.

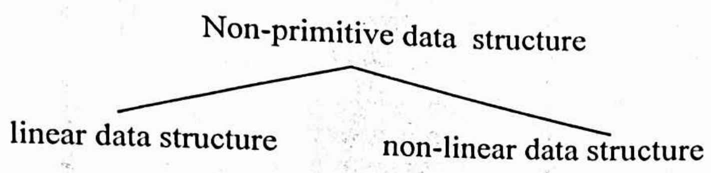
Data structures is a individual concept and also to develop various types of structures by using programming language.,In this, data means combination of numbers and non-numbers,Structure means to represent the data in neat format.Totally the data structures is arrange the data in specific order.

DataStructures are mainly divided into two types. Like primitive data structures and non-primitive data structures.


Data structures
Primitive data structures
Non-primitive data structure
ex:-int, float, char……
ex:- array ,linked, stack,queue, tree& graphics

Primitive data structures :- primitive data structures are access the features directly from the system. **ex**: Int, float, char…… **(With explanation)**

Non-primitive data structure:Non-primitive data structure are access the features indirectly from the system. These are more complex data structures. These data structures are derived from the primitive data structures.
ex: array, linked, stack, queue, tree& graphics **(With explanation)**



linear data structure :In linear data structure,we arrange the elements in sequential manner (or) to store the elements in sequential memory location.
ex:- array,linked list,queue,stacks **(With Definition only)**

Non-linear data structure :-In non-linear data structure, we arrange the elements in non-sequential {or} hierarchical manner.
EX:- tree and graphics **(With Definition only)**

Data structure provides mainly 7 types of operation for solve the problems.

1. Creation
2. Traversing
3. Insertion
4. Deletion
5. Searching
6. Sorting
7. Merging

Creation:- create refers, to create an array and also to read the elements into the list.

Traversing:- Travers refers, to display{or}to visit each and every element from the list.

Insertion:-insertion refers, to insert a new element at particular position into the list.

Deletion:-delete refers, to delete a particular element from the list.

Searching:- Search refers, to Search {or} to find out given element is there {or} not.

Sorting:-sort refers, to arrange the elements in specific order either ascending order {or}descending order.

Merging:- merge refers, to combine two sorted list into a single list.

Explain Storage Structure:-

Main memory – only large storage media that the CPU can access directly. **(Explain RAM and ROM)**

Secondary storage – extension of main memory that provides large nonvolatile storage capacity.**(Explain secondary storage devices)**

Magnetic disks – rigid metal or glass platters covered with magnetic recording material

Write about File Structures:-

File structure was suitable when files were stored only on large reels of magnetic tape and skipping around to access random data was not possible.

Banking transactions (deposits and withdrawals), for example, might be sorted in the same order as the accounts file, so that as each transaction is read the system need only scan ahead (never backward) to find the accounts record to which it applies.

(With explanation of Sequential and Random Access files)

Linear Lists:-

The elements are ordered within the **linear list** in a **linear** sequence. **Linear lists** are usually simply denoted as **lists**. Unlike an array, a **list** is a data structure allowing insertion and deletion of elements at an arbitrary position of the sequence.

The linear list ADT

An example :-we could define an ADT for lists of integers

structure :-

* a finite sequence of nodes
* since a sequence, order is implied
* a node is an undefined term — some collection of data

possible operations :-

* start a new, empty list
* insert a new item into the list
* delete an item with a given value
* search for an item in the list
* print the list etc.

• *To implement such an ADT*, we could use

Aarrays

Nodes

Explain Arrays:-

·**Array :-** "An array is a group of contiguous (continuous) related data items that share a common name."

(or)

"An array is a collection of same type of elements that share a common name."

Arrays are divided into following types

1. one dimensional array
2. Two dimensional array

1.One Dimensional Array:-When an array is declared with only one index (subscript) value then it is called as one dimensional array.

Creating an array:-

Like any other variables, arrays must be declared and created in computer memory before they are used. Creating an array involves 3 types.

1. Declaring an array.
2. Creating memory.
3. Putting values into memory.

Declaraing the array:- Arrays in java may be declared in two forms:

```
Form 1:-     type arrayName[];
             Eg:- int a[];
                  float b[];
Form 2:-     type[] arrayname;
             Eg:-int[] a;
                 float[] b;
```

Creation of arrays:-

After declaring an array we need to allocate memory locations for the array in the memory. Java allows to create the array using **'new'** operator.

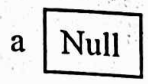
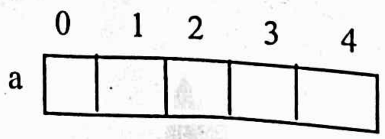
syntax:- arrayName=new type[size];
Eg:- a=new int[5];
Here 'a' refers to an array of 5 integers.

It is also possible to combine declaration and creation into one step as follows:

type arrayName[ ]=new type[size];

eg:- int a[ ]=new int[5];

the following diagram shows creation of an arryay in the memory.

Statement	**Result**
int a [ ]	
a=new int[5];	

Putting values into arrays (or) Initialization of arrays:-

The final step is to put the values in to the created array. This process is known as array initialization. This is done by using the array subscript as shown below.

Syntax: arrayname[subscript]=value;

Eg:-
```
a[0]=10;
a[1]=20;
a[2]=30;
a[3]= -40;
a[4]= -50;
```

java creates array subscripts from'0' to size -1. We can also initialize arrays automatically in the same way as the ordinary variables as shown below:

Syntax:- type arrayname[ ]={list of values};
```
int a[ ]={10, 20, 30, -40, -50};
```

The array initialization is a list of values seperated by commas and enclosed by curly braces. It is also possible to assign an array object to another array object of same type.

```
int[ ] a={1,2,3,4};
int[ ] b ;
b=a; // is valid statement
```

Array length:-

In java, we can calculate array size by using a variable name **length** . We can obtain the length of the arrayname. It returns size of the array as **Integer.**

Eg:- int size = a.length;

This information will be useful in the manipulation of arrays, when their sizes are not known.

```
/* program to calculate the sum and average of array elements*/
class ArraySum
{
public static void main(String args[])
{
int[] a={45,10,22,33,14};
int sum=0;
float avg;
for(int i=0;i<a.length;i++)
sum=sum+a[i];
avg=(float)sum/a.length;
System.out.println("sum="+sum);
System.out.println("average="+avg);
}
}
```

Two dimensional arrays:-

When an array declared with 2 subscripts then that array is known as two-dimensional array. It can be viewed as table of elements which contains rows and columns. For creating two-dimensional arrays, we must follow the same steps as that of single dimensional array.

Creating an array involves 3 types.

1. Declaration of two dimensional array.
2. Creating memory to the two dimensional array .
3. initialization or Putting values into memory.

1. Declaration of two dimensional arrays:- Two dimensional arrays are declared in two forms as follows.

```
Form 1:-     datatype arrayName[ ][ ];
                         int a[ ][ ];
Form 2:-     datatype[ ][ ] arrayName;
                         float [ ][ ] a;
```

2. Creatingmemory to the two dimensional array:- After declaring the array we need to create memory locations by using new operator as follows.

```
Syntax:-     arrayname= new datatype[ rows][columns ];
                  a= new int[3][3];
```

Here creates a table that can store 9 integer values i.e 3rows and 3 columns as shown below.

row0	a[0,0]	a[0,1]	a[0,2]
row1	a[1,0]	a[1,1]	a[1,2]
row2	a[2,0]	a[2,1]	a[2,2]

3. Initialization or putting values into two dimensional arrays:- The final step is to put the values into the created array. This process is known as array initialization.

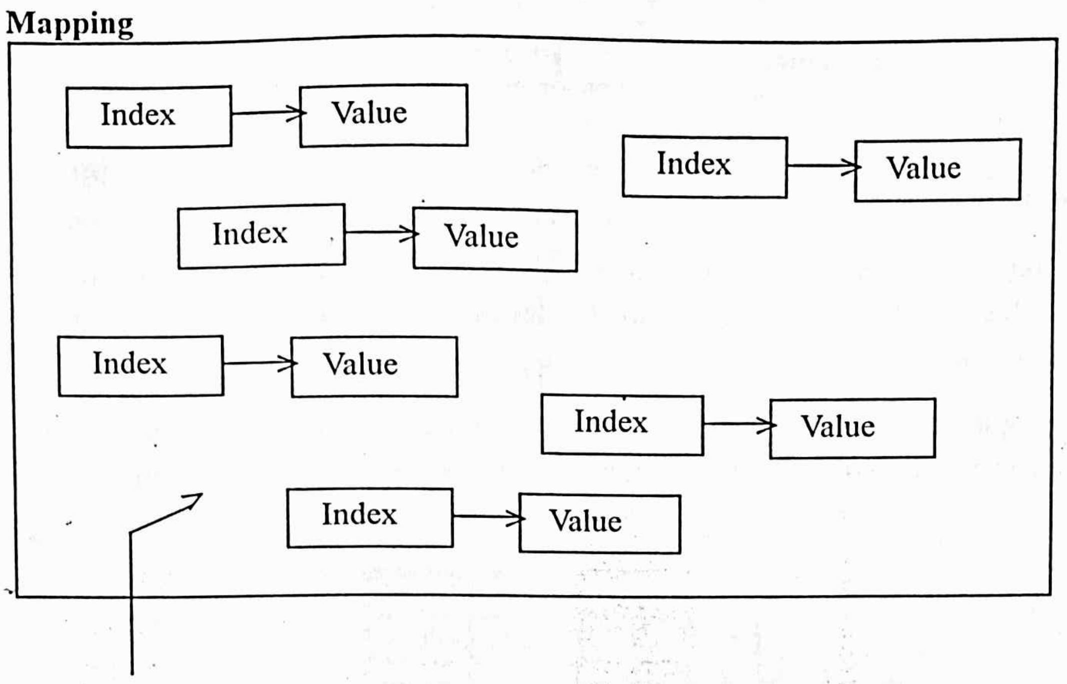
```
arrayName[row][column]=value;
a[0][0]=1000;
```

While declaring an array we can initialize it with some values. The general format to initialize a two-dimensional array is as follows:-

```
Syntax:-     type  arrayName[ ][ ]={ {row 1  values},{row2 values},...};
             int a[ ][ ]={ {90,90,90},{80,70,69} };
```

Wrate short note of Mappings:-

we can imagine that a mapping look like this



Lookup Function

Each index-value pair is floating around freely inside the mapping. There is exactly one value for each index. We also have a (magical) lookup function. This lookup function can find any index in the mapping very quickly. Now, if the mapping is called m and we index it like this: m [i] the lookup function will quickly find the index i in the mapping and return the corresponding value. If the index is not found, zero is returned instead. Writing a constant mapping is easy:

Explain Sparse Matrices:-

"Sparse matrix is a matrix which contains very few non-zero elements". In computer programming, a matrix can be defined with a 2-dimensional array. Any array with 'm' columns and 'n' rows represents a mXn matrix. There may be a situation in which a matrix contains more number of ZERO values than NON-ZERO values. Such matrix is known as sparse matrix.

When a sparse matrix is represented with 2-dimensional array, we waste lot of space to represent that matrix. For example, consider a matrix of size 100 X 100 containing only 10 non-zero elements. In this matrix, only 10 spaces are filled with non-zero values and remaining spaces of matrix are filled with zero. That means, totally we allocate 100 X 100 X 2 = 20000 bytes of space to store this integer matrix. And to access these 10 non-zero elements we have to make scanning for 10000 times.

Sparse Matrix Representations

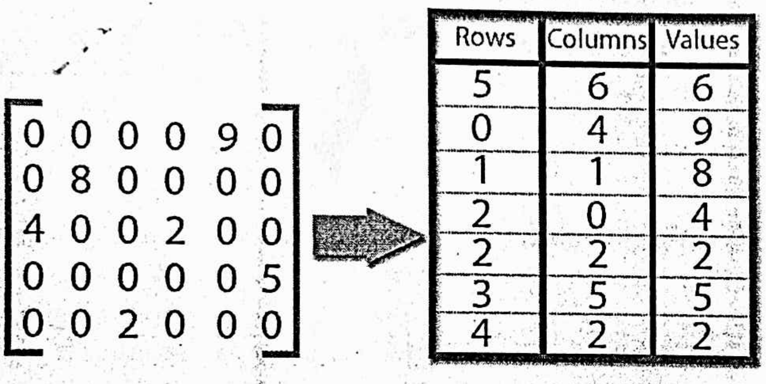
A sparse matrix can be represented by using TWO representations, those are as follows...

1. Triplet Representation
2. Linked Representation

Triplet Representation

In this representation, we consider only non-zero values along with their row and column index values. In this representation, the 0th row stores total rows, total columns and total non-zero values in the matrix.

For example, consider a matrix of size 5 X 6 containing 6 number of non-zero values. This matrix can be represented as shown in the image...



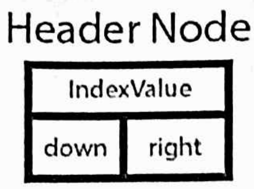
$$\begin{bmatrix} 0 & 0 & 0 & 0 & 9 & 0 \\ 0 & 8 & 0 & 0 & 0 & 0 \\ 4 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 5 \\ 0 & 0 & 2 & 0 & 0 & 0 \end{bmatrix} \Rightarrow$$

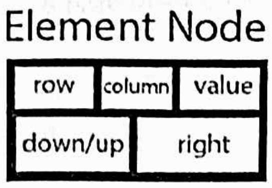
Rows	Columns	Values
5	6	6
0	4	9
1	1	8
2	0	4
2	2	2
3	5	5
4	2	2

In above example matrix, there are only 6 non-zero elements (those are 9, 8, 4, 2, 5 & 2) and matrix size is 5 X 6. We represent this matrix as shown in the above image. Here the first row in the right side table is filled with values 5, 6 & 6 which indicates that it is a sparse matrix with 5 rows, 6 columns & 6 non-zero values. Second row is filled with 0, 4, & 9 which indicates the value in the matrix at 0th row, 4th column is 9. In the same way the remaining non-zero values also follows the similar pattern.

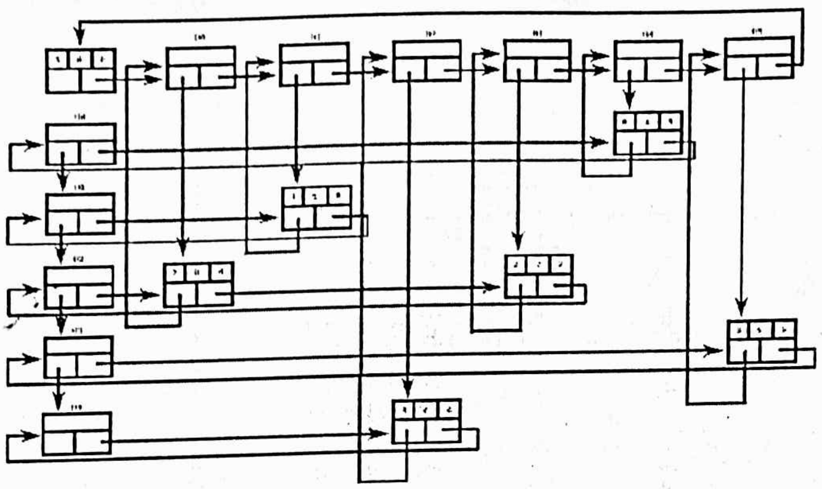
Linked Representation

In linked representation, we use linked list data structure to represent a sparse matrix. In this linked list, we use two different nodes namely **header node** and **element node**. Header node consists of three fields and element node consists of five fields as shown in the image...





Consider the above same sparse matrix used in the Triplet representation. This sparse matrix can be represented using linked representation as shown in the below image...



In above representation, H0, H1,...,H5 indicates the header nodes which are used to represent indexes. Remaining nodes are used to represent non-zero elements in the matrix, except the very first node which is used to represent abstract information of the sparse matrix (i.e., It is a matrix of 5 X 6 with 6 non-zero elements).

write a short note of Sets :

Abstract Data Types (ADTs):- We introduced the idea of an *abstract data type*: a set of operations, properties, and behaviors (axioms) that together define a particular kind of data. We saw two examples of ADTs: Sets (collections of elements with no duplicates and no ordering), and Bags (collections of elements that allow duplicates but are not ordered). We showed that Java interfaces capture the operations, but the properties and behaviors do not have a direct representation in Java (much as types had no direct representation in Racket).

Operations on Sets:

One may define the operations of the algebra of sets:

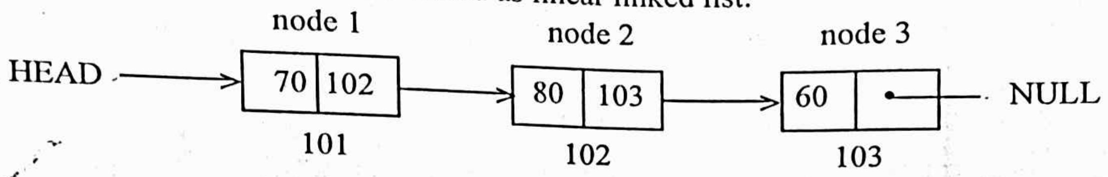
- **union(S,T)**: returns the union of sets S and T.

- **intersection(S,T)**: returns the intersection of sets S and T.
- **difference(S,T)**: returns the difference of sets S and T.
- subset(S,T): a predicate that tests whether the set S is a subset of set T.

LINKED LIST

Linked list is a linear list of elements where we can perform both the insertion and deletion at random. In a linked list the data will be stored in the form of a node, where each node contains two elements namely DATA & LINK. The DATA field contains value and the LINK field contains the address of the next node. There is a special pointer called HEAD, which contains the address of the first node, there is a special constant called NULL which represents the end of the list .

Single Linked list :- A linked list is one in which all nodes are linked together in some sequential manner.Hence it is also called as linear linked list.



operations:-the following operations can perform on a list.

1. Creation
2. Traverse
3. Insertion
4. Deletion

Creation:- in creation check the condition that the HEAD is NULL or not .if HEAD is null then create a new node and assign that address to the HEAD . otherwise create a new node assign that addres to the previous node link.

ALGORITHM:-

```
Creation(num)
Step 1 :     START
STEP 2:      If (head==null)then
             p = new node;
             p.data=num;
             p.link=NULL;
             HEAD=P;
             t = new node;
             t.data=num;
             t.link=NULL;
             p.link=t;
             p = t;
step 3:      STOP
```

Traverse:- In this we display each and every element from the list . in a single linked list traversing can possible only one direction from first element (HEAD) to last element (NULL). Because node contains next node address.

Traverse:

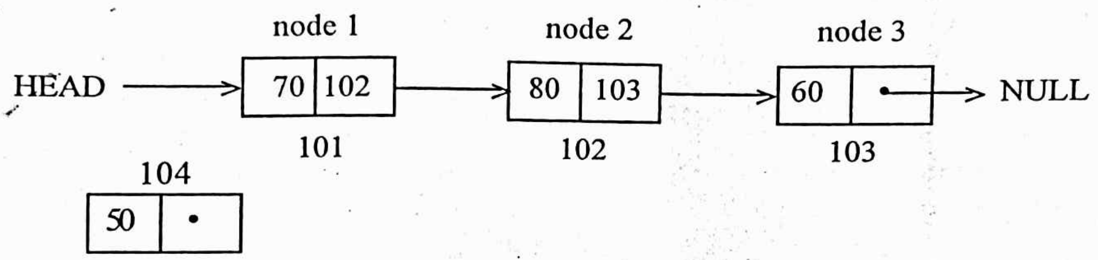
```
ALGORITHM:-
Step1  : START
STEP 2:  P=HEAD;
STEP 3: Repeat the step4&5
        While (p!==NULL)
Step 4:  print p.data;
Step 5: p=p.link;
Step 6: STOP
```

Insertion:- In sing;e linked list insertion can possible in 3ways

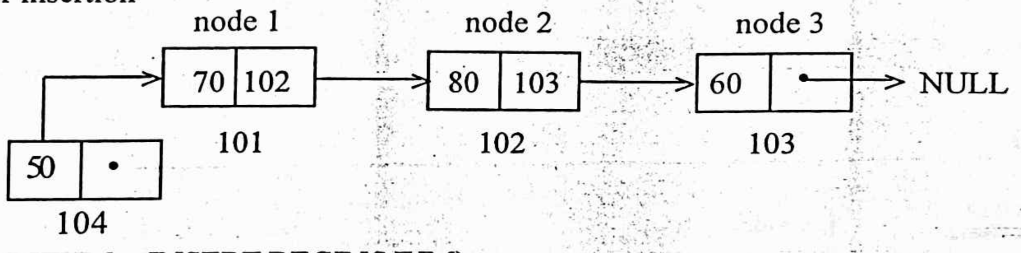
1. Insert the element at the beginning of the list
2. Insert the element at the end of the list.
3. Insert the element after specified node.

Insert the element at the beginning of the list:- suppose to insert the element at beginning of the list , to create a new node and assign that address to the head and also establish link to the next node.

Before insertion



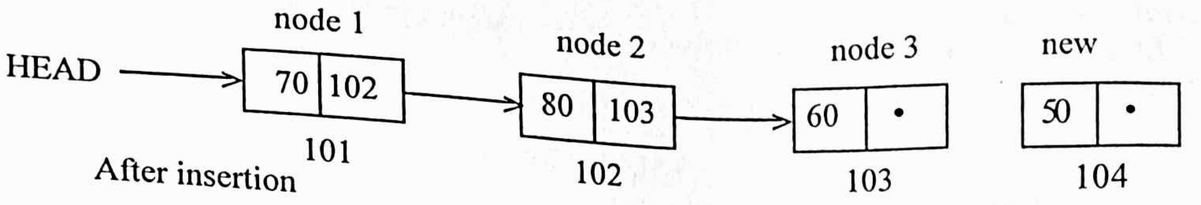
After insertion



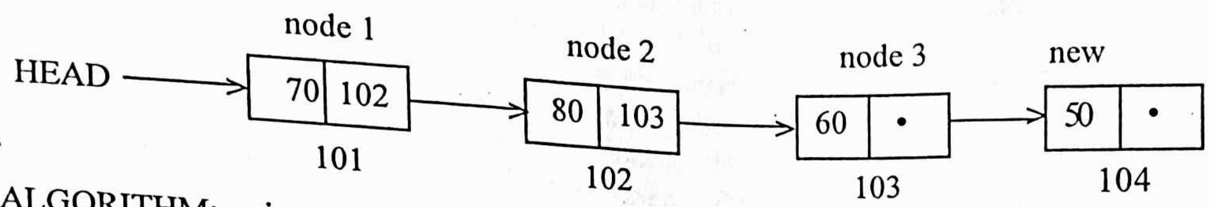
```
ALGORITHM:-  INSERT BEGIN(NUM)
STEP 1:-START
STEP 2; -t=new node;
STEP 3:- t.data=num;
STEP 4:- t.link-head;
STEP 5:- head=t;
STEP 6:- STOP
```

Insert the element at the end of the list:- : suppose to insert the element at END of the list , to create a new node and assign that address to the last node link and store NULL pointer to the link.

Before insertion



After insertion

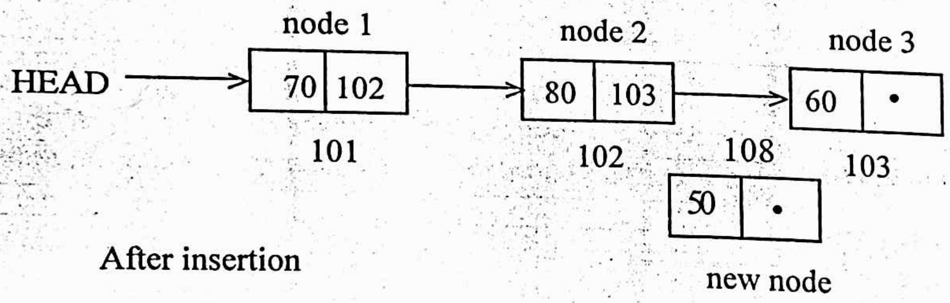


ALGORITHM:- insert End(num)

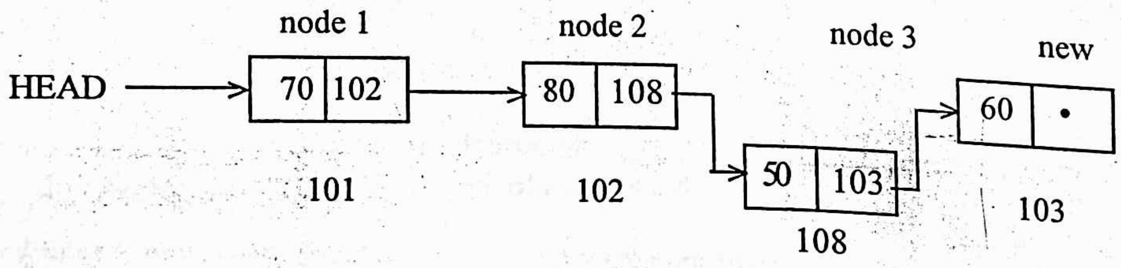
```
STEP 1:-START
STEP 2; -p=head;
STEP 3:- repeat the step 4 while(p.link!= null)
STEP 4:- p=p.link;
STEP 5:- t=new node;
STEP 6:- t.data=num;
STEP 7:- t.link=null;
STEP 8:- p.next=t;
STEP 9:- STOP
```

3. Insert the element after specified node:- suppose if we insert the element after the specified node ,first we check specified node is there or not .If it is found then insert new node otherwise display element not found.

Before insertion



After insertion



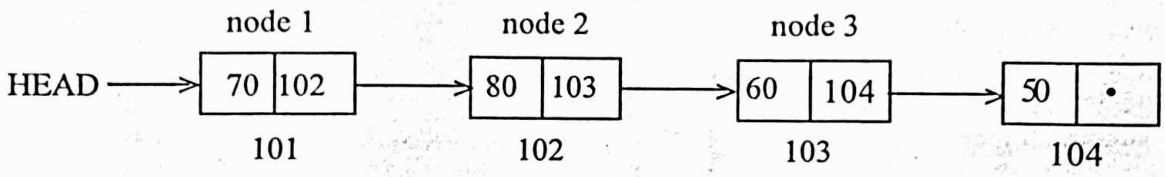
ALGORITHM:-

```
 insert After (ele,num)
STEP 1:-START
STEP 2; - set P=head;
STEP 3:- repeat the steps 4&5
          While(p!=NULL)
STEP 4:- if (ele=p.data)then
           t=new node ;
           t.data=num;
           t.link= p.link;
           p.link=t;
     else
    p=p.link;
STEP 5:- stop
```

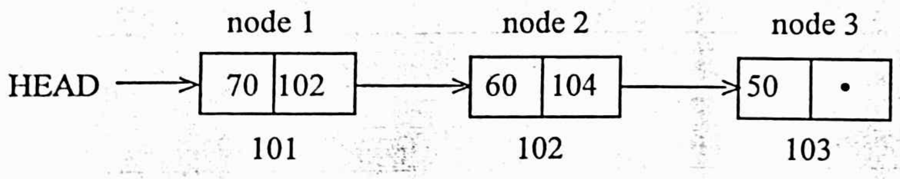
DELETION:- suppose if you want to delete the element from the list then we follow mainly two type of methods.

1. If deleted element is first node then to assign the address of second node to head.
2. If deleted element is middle element then to assign the address of next node to previous node.

Before Deletion



After Deletion (Value =80)



ALGORITHM :-

Delete(num):

```
STEP 1:-    START
STEP 2; -   set P=head;
STEP 3:-    if (P.data==num)then
              Head=P.link and deletd P
            else
            Set q=P and P=P.link;

STEP 4:-    repeat the step 6&7
            While(P!=NULL)
```

```
STEP 5:-     if(P.data==num) then
             q.link = P.link and delete p;

STEP 6 :-    else
             set q=p and p=p.link

STEP 7:-     STOP
```

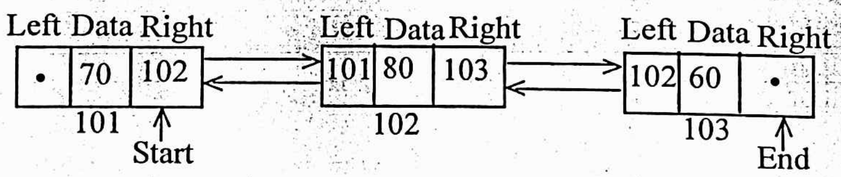
```
import java.io.*;
import java.util.*;
class node
{
int ele;
node link;
}
class list
{
node head=null;
void create(int x)
{
node p;
node t=new node();
t.ele=x;
t.link=null;
if(head==null)
{
head=p=t;
}
else
{
for(p=head;p.link!=null;p=p.link);
p.link=t;
}
}
void display()
{
node p;
if(head==null)
{
System.out.println("cannot display");
}
else
{
for(p=head;p!=null;p=p.link)
System.out.println("nos="+p.ele);
}
}
void addbig(int x)
{
if(head==null)
{
System.out.println("cannot perform");
}
else
{
node t=new node();
t.ele=x;
t.link=head;
head=t;
}
}
void insert(int x,int k)
{
node p,q;
p=head;
while(p!=null&&p.ele!=k)
{
p=p.link;
}
if(p==null)
System.out.println("item not found");
else
{
node t=new node();
t.ele=x;
t.link=null;
q=p.link;
p.link=t;
t.link=q;
}
}
void delete(int x)
{
node p,q=null;
p=head;
if(head==null)
System.out.println("cannot delete");
if(head.ele==x)
{
head=head.link;
}
```

```
else
{
while(p!=null&&p.link.ele!=x)
{
p=p.link;
}
if(p==null)
System.out.println("Item not found");
else
q=p.link;
p.link=q.link;
}
}
}
class linklist
{
public static void main(String arg[])throws IOException
{
int x,ch,k;
list ob=new list();
do
{
System.out.println("1.CREATE");
System.out.println("2.ADD BIG");
System.out.println("3.INSERT");
System.out.println("4.DELETE");
System.out.println("5.DISPLAY");
System.out.println("6.EXIT");
System.out.println("ENTER YOUR CHOICE");
DataInputStream inn=new DataInputStream(System.in);
ch=Integer.parseInt(inn.readLine());
switch(ch)
{
case 1:
System.out.println("Enter Element");
x=Integer.parseInt(inn.readLine());
ob.create(x);
break;
case 2:
System.out.println("ENTER ELEMENT");
x=Integer.parseInt(inn.readLine());
ob.addbig(x);
break;
case 3:
System.out.println("ENTER ELEMENT");
x=Integer.parseInt(inn.readLine());
System.out.println("ENTER POSITION");
k=Integer.parseInt(inn.readLine());
ob.insert(x,k);
break;
case 4:
System.out.println("ENTER ELEMENT");
x=Integer.parseInt(inn.readLine());
ob.delete(x);
break;
case 5:
ob.display();
break;
case 6:
System.exit(0);
default:
System.out.println("Wrong choice");
}
}while(ch!=6);
}
}
```

DOUBLE –LINKED LIST

Double linked list is a linear data structure and also it is dynamic data structure because to insert or delete the elements at rumtime. In this element refers node. It contains namely three fields like left – link(LEFT),DATA and right-link (RIGHT).

LEFT:- It contains previous node address.
DATA:- It contain actual value.
RIGHT:- It contains next node address.



In the above diagram we use mainly two types of special pointers like START and END. On that START is always points to the first node address, END is always points to the last node addresss.

OPERATIONS:- The following operations can be perform on a Double linke list.

1. Creation
2. Insertion
3. Traverse or list
4. Deletion

Creation:- In this we first check the condition START is null or not . If it is null then create a new node and assign that address to the START and END . Otherwise to create a new node and assign that address to the END.

```
ALGORITHM:- Creation(num)
Step 1 : START
Step 2:- If (START=NULL)then
            t=new node;

            t.LEFT=NULL;
            t.RIGHT-NULL;
            START=END=t;
    else
          t=new node;
          t.data=num;
          t.LEFT=NULL;
          t.RIGHT=NULL;
          END.RIGHT=t;
          t.LEFT=END;
          END=t;
 Step 3:- STOP
```

Traverse:- The doubly linked list traverse can possible both directions like.

1. Forward traverse
2. Backward traverse

1.Forward traverse:- In this we visit each and every element from first element to last element (START to END)

```
Algorithm :-  Forward traverse ():-
Step 1:- START
Step 2:- set a=START;
STEP 3:- Repeat the step 4&5
            While (a!=NULL)
            Write (a.data)
             a=a.RIGHT;
step 4:- STOP
```

Backward Travese:- :- In this we visit each and every element from last element to First element (END to START)

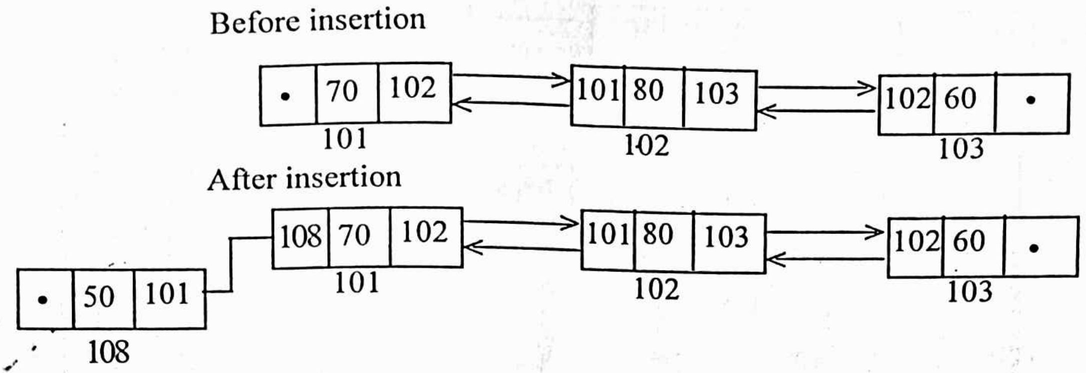
```
Algorithm :-
      Backward traverse()
Step 1:-   START
Step 2:-   set a=END;
STEP 3:-   Repeat the step 4&5
            While (a!=NULL)
           Write (a.data)
            a=a.LEFT;
```

step 4:- STOP

Insertion :- suppose if you want to insert an element into the list called insertion .In this we follow mainly 3 methods like.

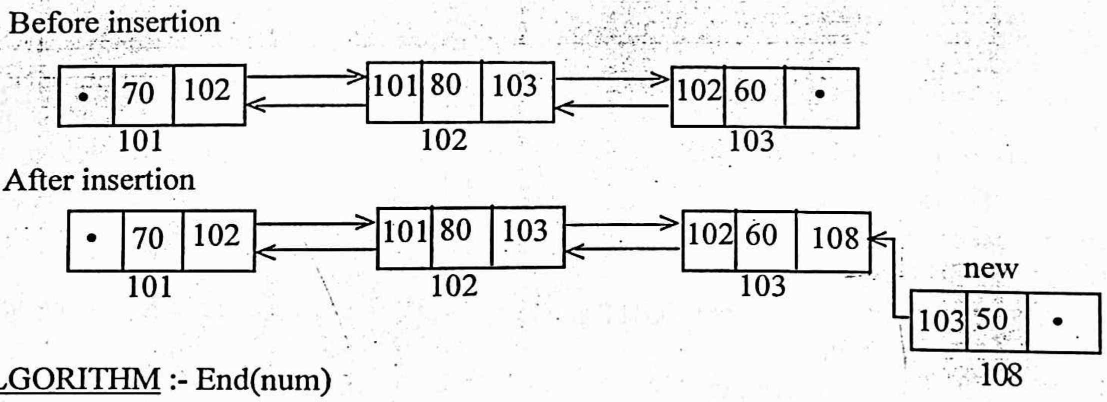
1. Insert the element at the beginning of the list
2. Insert the element at the end of the list
3. Insert the element after specified node.

Insert the element at the beginning of the list :- In this we first create a new node and next to assign that address to the START. Because it is the first node.



```
ALGORITHM :-insert begin(num)
Step 1:-    START
Step 2:-    t=new node;
Step 3:-    t.data=num;
Step 4:-    START.LEFT=t;
Step 5:-    t.RIGHT=START;
Step 6:-    t,LEFT =NULL;
Step 7:-    START=t;
Step 8:-    STOP
```

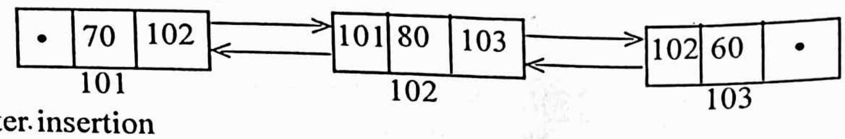
Insert the element at the end of the list:- In this we first create a new node and assign tha address to the END. Because it is the last node.



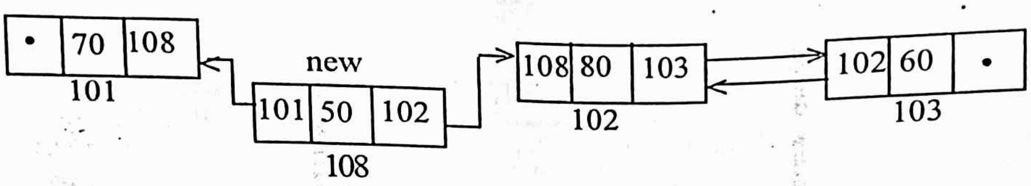
```
ALGORITHM :- End(num)
Step 1:-    START
Step 2:-    t=new node;
Step 3:-    t. data=num;
Step 4:-    t.LEFT=END;
Step 5:-      END.RIGHT=t
Step 6:-      t,RIGHT =NULL;
Step 7:-      END=t;
Step 8:-      STOP
```

Insert the element after specified node:- In this we first search element is there or not . It is found then insert a new node and assign that address to the previous node and next node otherwise to display element not found.

Before insertion



After insertion



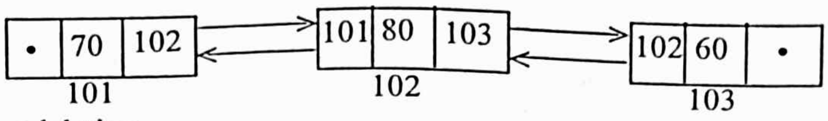
ALGORITHM :-

```
Insert after(ele ,num)
Step 1:-      START
Step 2:-      set tmp =START;
Step 3:-      while (tmp!=NULL)
              if(tmp.data==ele)then
              t=new node;
              tmp.RIGHT=t;
              t.left=tmp;
              t. RIGHT=tmp.RIGHT;
        else
              write(" NOT FOUND");
Step 4:- STOP
```

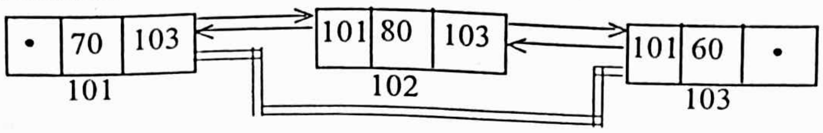
DELETION:- In this you first search deleted elements is there or not . if it is found then to assign the address previous node to next and next node to previous node otherwise to display element not found..

```
 ALGORITHM :-     Delete (num)
Step 1:-    START
Step 2:-    set a= START;
Step 3:-    if (a.data =num)then
                 START=START.RIGHT and delete (a)
            While (a!=NULL)
             if (a.data =num)then
            a.LEFT.RIGHT=a.RIGHT.RIGHT;
            a.RIGHT.LEFT=as.LEFT;
Step 4:-    STOP
```

Before deletion



After deletion



```
import java.io.*;
import java.util.*;
class node
{
int ele;
node link,prev;
}
class list
{
node head=null;
void create(int x)
{
node p;
node t=new node();
t.ele=x;
t.link=null;
t.prev=null;
if(head==null)
{
head=p=t;
}
else
{
for(p=head;p.link!=null;p=p.link);
p.link=t;
t.prev=p;
p=t;
}
}
void display()
{
node p;
if(head==null)
{
System.out.println("cannot display");
}
else
{
for(p=head;p!=null;p=p.link)
System.out.println("nos="+p.ele);
}
}
void addbig(int x)
{
if(head==null)
{
System.out.println("cannot perform");
}
else
{

node t=new node();
t.ele=x;
head.prev=t;
t.link=head;
t.prev=null;
head=t;
}
}
void insert(int x,int k)
{
node p,q;
p=head;
while(p!=null&&p.ele!=k)
{
p=p.link;
}
if(p==null)
System.out.println("item not found");
else
{
node t=new node();
t.ele=x;
q=p.link;
p.link=t;
t.link=q;
q.prev=t;
t.prev=p;
}
}
```

```
void delete(int x)
{
node p,r,q=null;
p=head;
if(head==null)
System.out.println("cannot delete");
if(head.ele==x)
{
head=head.link;
head.prev=null;
}
else
{
while(p!=null&&p.link.ele!=x)
{
p=p.link;
}
if(p==null)
System.out.println("Item not found");
else
{
q=p.link;
r=q.link;
p.link=r;
r.prev=p;
}
}
}
}
class doulinklist
{
public static void main(String arg[])throws IOException
{
int x,ch,k;
list ob=new list();
do
{
System.out.println("1.CREATE");
System.out.println("2.ADD BIG");
System.out.println("3.INSERT");
System.out.println("4.DELETE");
System.out.println("5.DISPLAY");
System.out.println("6.EXIT");
System.out.println("ENTER YOUR CHOICE");
DataInputStream inn=new DataInputStream(System.in);
ch=Integer.parseInt(inn.readLine());
switch(ch)
{
case 1:
System.out.println("Enter Element");
x=Integer.parseInt(inn.readLine());
ob.create(x);
break;
case 2:
System.out.println("ENTER ELEMENT");
x=Integer.parseInt(inn.readLine());
ob.addbig(x);
break;
case 3:
System.out.println("ENTER ELEMENT");
x=Integer.parseInt(inn.readLine());
System.out.println("ENTER POSITION");
k=Integer.parseInt(inn.readLine());
ob.insert(x,k);
break;
case 4:
System.out.println("ENTER ELEMENT");
x=Integer.parseInt(inn.readLine());
ob.delete(x);
break;
case 5:
ob.display();
break;
case 6:
System.exit(0);
default:
System.out.println("Wrong choice");
}
}while(ch!=6);
}
}
```

Advantages of Linked Lists

- They are a dynamic in nature which allocates the memory when required.
- Insertion and deletion operations can be easily implemented.
- Stacks and queues can be easily executed.
- Linked List reduces the access time.

Disadvantages of Linked Lists

- The memory is wasted as pointers require extra memory for storage.

- No element can be accessed randomly; it has to access each node sequentially.
- Reverse Traversing is difficult in linked list.

Applications of Linked Lists

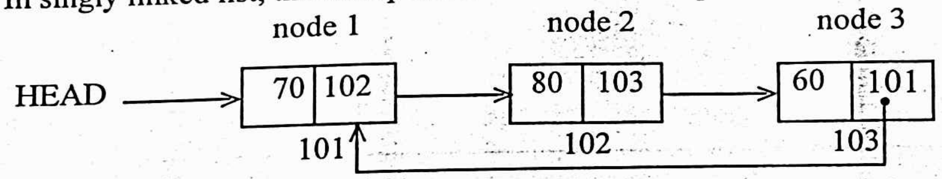
- Linked lists are used to implement stacks, queues, graphs, etc.
- Linked lists let you insert elements at the beginning and end of the list.
- In Linked Lists we don't need to know the size in advance.
- Linked list are used to represent & manuplate polynomial expression
- Represent very large numbers and operations of the large number such as addition,multiplication,division
- Implement the symbol table in compiler construction.

CIRCULAR LINKED LIST:-

Circular Linked List is a variation of Linked list in which the first element points to the last element and the last element points to the first element. Both Singly Linked List and Doubly Linked List can be made into a circular linked list.

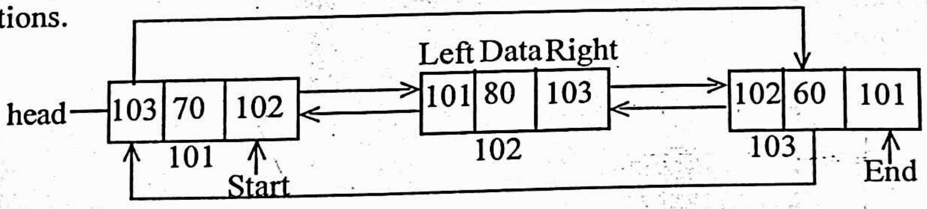
Singly Linked List as Circular

In singly linked list, the next pointer of the last node points to the first node.



Doubly Linked List as Circular

In doubly linked list, the next pointer of the last node points to the first node and the previous pointer of the first node points to the last node making the circular in both directions.



As per the above illustration, following are the important points to be considered.

- The last link's next points to the first link of the list in both cases of singly as well as doubly linked list.
- The first link's previous points to the last of the list in case of doubly linked list.

Basic Operations

Following are the important operations supported by a circular list.

- **insert** " Inserts an element at the start of the list.
- **delete** " Deletes an element from the start of the list.
- **display** " Displays the list.

Insertion Operation

Following code demonstrates the insertion operation in a circular linked list based on single linked list.

Example

```
   public void insertAtStart(int val)
 {
   Node nptr = new Node(val,null);
   nptr.setLink(start);
   if(start == null)
   {
      start = nptr;
      nptr.setLink(start);
      end = start;
   }
   else
   {
      end.setLink(nptr);
      start = nptr;
   }
   size++ ;
 }
 public void insertAtEnd(int val)
 {
   Node nptr = new Node(val,null);
   nptr.setLink(start);
   if(start == null)
   {
     start = nptr;
     nptr.setLink(start);
     end = start;
   }
   else
   {
     end.setLink(nptr);
     end = nptr;
   }
```

```
      size++ ;
    }
    public void insertAtPos(int val , int pos)
    {
      Node nptr = new Node(val,null);
      Node ptr = start;
      pos = pos - 1 ;
      for (int i = 1; i < size - 1; i++)
      {
        if (i == pos)
        {
          Node tmp = ptr.getLink() ;
          ptr.setLink( nptr );
          nptr.setLink(tmp);
          break;
        }
        ptr = ptr.getLink();
      }
      size++ ;
    }
```

Deletion Operation

Following code demonstrates the deletion operation in a circular linked list based on single linked list.

```
    public void deleteAtPos(int pos)
    {
      if (size == 1 && pos == 1)
      {
        start = null;
        end = null;
        size = 0;
        return ;
      }
      if (pos == 1)
      {
        start = start.getLink();
        end.setLink(start);
        size—;
        return ;
      }
      if (pos == size)
      {
        Node s = start;
        Node t = start;
        while (s != end)
        {
          t = s;
```

```
            s = s.getLink();
        }
        end = t;
        end.setLink(start);
        size —;
        return;
    }
    Node ptr = start;
    pos = pos - 1 ;
    for (int i = 1; i < size - 1; i++)
    {
        if (i == pos)
        {
            Node tmp = ptr.getLink();
            tmp = tmp.getLink();
            ptr.setLink(tmp);
            break;
        }
        ptr = ptr.getLink();
    }
    size— ;
}
```

Display List Operation

Following code demonstrates the display list operation in a circular linked list.

```
public void display()
{
    System.out.print("\nCircular Singly Linked List = ");
    Node ptr = start;
    if (size == 0)
    {
        System.out.print("empty\n");
        return;
    }
    if (start.getLink() == start)
    {
        System.out.print(start.getData()+ "->"+ptr.getData()+ "\n");
        return;
    }
    System.out.print(start.getData()+ "->");
    ptr = start.getLink();
    while (ptr.getLink() != start)
    {
        System.out.print(ptr.getData()+ "->");
        ptr = ptr.getLink();
    }
```

```
        System.out.print(ptr.getData()+ "->");
        ptr = ptr.getLink();
        System.out.print(ptr.getData()+ "\n");
    }
}
```