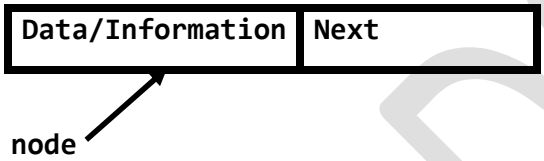
Linked Lists:-

It is a list or collection of data elements, where each element is point to the next element in the list.

Each element of the Linked List represents by a node and each node contains two parts.

1. Data/Information.
2. Next.

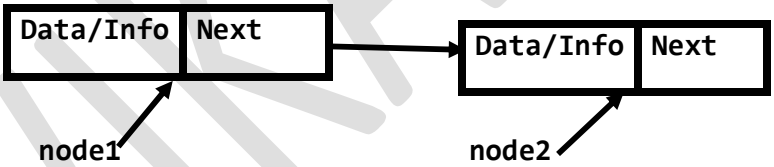
Example:-



Data/Infromation:-This parts represent data or information which we will store in Linked List.

Next:-This part refers to or holds the next node.

Example:-



Advantage of Linked List:-

1. Dynamic memory allocation, based on our requirement we can add and remove element.
2. To performed insert and delete operation is very easy in linked list compare to array.
3. Very efficient memory utilization in linked list because linked list only reserved that amount of memory, which is required for storing for element.

Disadvantage of Linked List:-

1. Accessing the element is very difficult compare to array because there is no index identifier associate with list element.
2. Linked list elements required more memory space because it has to also reference of next element in the list.

Types of Linked Lists:-

1. Singly Linked Lists.
2. Double Linked Lists.
3. Circular Singly Linked Lists.
4. Circular Double Linked Lists.

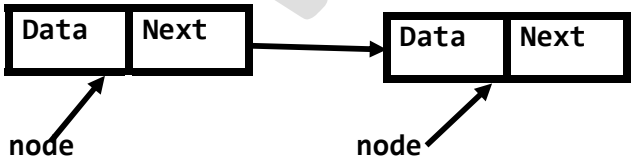
Singly Linked List:-In this type of the linked list each node point to the node, so the list can be traversed in the forward direction only.

In this, each node contains two parts:-

Data/Infromation:-This parts represent data or information which we will store in Linked List.

Next:-This part refers to or holds the next node

Example:-



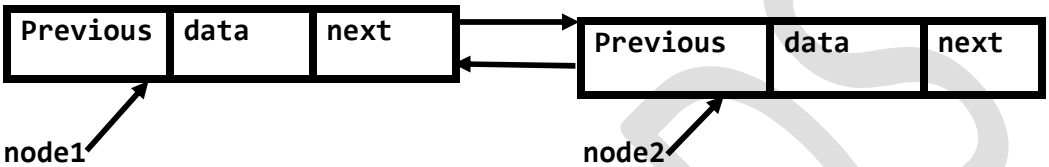
Double Linked List:-In this types of linked list each node point to previous and next nodes, so the list can be traversed in both directions forward and backward.

In this each node contains three parts.

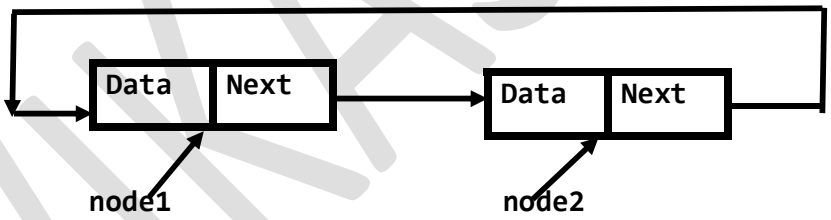
Data/Infromation:-This parts represent data or information which we will store in Linked List.

Next:-This part refers to or holds the next node.

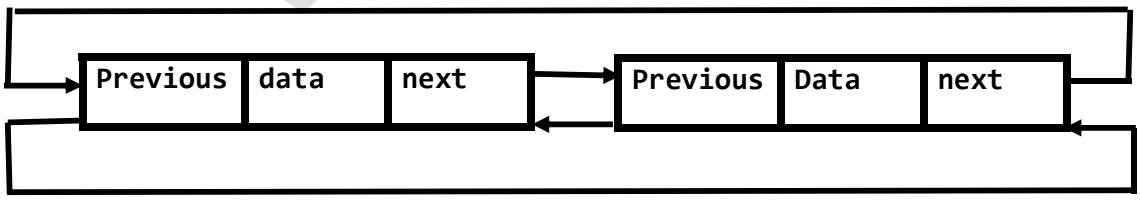
Previous:-This part refer to or hold the previous node.



Circular Singly Linked List:-In this type of linked list last node connected with first node. So it's make circular list formation.



Circular Double Linked List:-In this type of linked list last node and first node logically connected with each other.



Single Linked List implementations:-

Example Single Linked List:-1 In this example we performed add element first, traversed list(print list), and size of the list.

```
package com.vikas.ds;
public class SingleLinkedListDemo1
{
Node head;
int size;
class Node{
int data;
Node next;
Node(int data){
this.data = data;
this.next = null;
size++;
}
Node(int data,Node temp){
this.data = data;
this.next = temp;
size++;
}
}
int getSize(){
return this.size;
}
void printList(){
if(head==null){
System.out.println("list is empty");
}
else
{

Node currNode = head;
while(currNode!=null)
{
System.out.print(currNode.data+" => ");
```

```
currNode = currNode.next;
}
System.out.println("null");
}
}
void addFirst(int data){
Node newNode = new Node(data);
if(head==null)
{
head = newNode;
return;
}
else
{
newNode.next = head;
head = newNode;
}

}
public static void main(String[] args)
{
SingleLinkedListDemo1 slld=new SingleLinkedListDemo1();
slld.addFirst(444);
slld.addFirst(333);
slld.addFirst(222);
slld.addFirst(111);
slld.printList();
System.out.println("Size= "+slld.getSize());
}
}
```

Result:- 111 => 222 => 333 => 444 => null

Size= 4

Example Single Linked List:-2 In this example we performed add element at last, traversed list(print list), and size of the list.

```
package com.vikas.ds;
public class SingleLinkedListDemo1
{
Node head;
int size;
class Node{
int data;
Node next;
Node(int data){
this.data = data;
this.next = null;
size++;
}
Node(int data,Node temp){
this.data = data;
this.next = temp;
size++;
}
}
int getSize(){
return this.size;
}
void printList(){
if(head==null){
System.out.println("list is empty");
}
else
{
Node currNode = head;
while(currNode!=null)
{
```

```
System.out.print(currNode.data+" => ");
currNode = currNode.next;
}
System.out.println("null");
}
}
void addLast(int data)
{
Node newNode = new Node(data);
if(head==null)
{
head = newNode;
return;
}
else
{
Node currNode = head;
while(currNode.next!=null)
{
currNode = currNode.next;
}
currNode.next = newNode;
}
}
public static void main(String[] args)
{
SingleLinkedListDemo1 slld=new SingleLinkedListDemo1();
slld.addLast(121);
slld.addLast(122);
slld.addLast(125);
slld.addLast(126);
slld.printList();
```

```
System.out.println("Size= "+slld.getSize());
}
}
```

Result:-

```
123 => 124 => 125 => 126 => null
Size= 4
```

Example Single Linked List:-3 In this example we performed Inserting the data at a particular position.

```
package com.vikas.ds;
public class SingleLinkedListDemo1
{
Node head;
int size;
class Node{
int data;
Node next;
Node(int data){
this.data = data;
this.next = null;
size++;
}
Node(int data,Node temp){
this.data = data;
this.next = temp;
size++;
}
}
int getSize(){
return this.size;
}
void printList(){
if(head==null){
```

```
System.out.println("list is empty");
}
else
{
Node currNode = head;
while(currNode!=null)
{
System.out.print(currNode.data+" => ");
currNode = currNode.next;
}
System.out.println("null");
}
}
void addLast(int data)
{
Node newNode = new Node(data);
if(head==null)
{
head = newNode;
return;
}
else
{
Node currNode = head;
while(currNode.next!=null)
{
currNode = currNode.next;
}
currNode.next = newNode;
}
}
void addPos(int data,int pos){
int i=0;
```

```
Node newNode = new Node(data);
if(head==null)
{
head = newNode;
return;
}
else
{
if(pos!=0){
Node currNode = head;
Node prevNode = null;
while(currNode.next!=null && i<pos)
{
prevNode = currNode;
currNode = currNode.next;
i++;
}
prevNode.next = newNode;
newNode.next = currNode;
}
else{
newNode.next = head;
head = newNode;
}
}
}
public static void main(String[] args)
{
SingleLinkedListDemo1 slld=new SingleLinkedListDemo1();
slld.addLast(121);
slld.addLast(122);
slld.addLast(125);
slld.addLast(126);
```

```
slld.printList();
System.out.println("Size= "+slld.getSize());
slld.addPos(123,2);
slld.printList();
System.out.println("Size= "+slld.getSize());
}
}
```

Result:-

```
121 => 122 => 125 => 126 => null
Size= 4
121 => 122 => 123 => 125 => 126 => null
Size= 5
```

Example Single Linked List:-4 In this example we performed delete operations like:-

Deleting from first element.

Deleting from the last element.

Delete from particular position.

Deleting Element first match.

Deleting Elements all match.

```
package com.vikas.ds;
public class SingleLinkedListDemo1
{
Node head;
int size;
class Node{
int data;
Node next;
Node(int data){
this.data = data;
this.next = null;
size++;
}
```

```
Node(int data,Node temp){
this.data = data;
this.next = temp;
size++;
}
}
int getSize(){
return this.size;
}
void printList(){
if(head==null){
System.out.println("list is empty");
}
else
{
Node currNode = head;
while(currNode!=null)
{
System.out.print(currNode.data+" => ");
currNode = currNode.next;
}
System.out.println("null");
}
}
void addLast(int data)
{
Node newNode = new Node(data);
if(head==null)
{
head = newNode;
return;
}
else
```

```
{
Node currNode = head;
while(currNode.next!=null)
{
currNode = currNode.next;
}
currNode.next = newNode;
}
}
void deleteFirst(){
if(head==null)
{
System.out.println("List is empty");
return;
}
else
{
size--;
head=head.next;
}
}
void deleteLast()
{
if(head==null){
System.out.println("list is empty");
return;
}
else if(head.next==null)
{
head=null;
return;
}
else
```

```
{
size--;
Node temp1=head,temp2=head.next;
while(temp2.next!=null)
{
temp2 = temp2.next;
temp1 = temp1.next;
}
temp1.next = null;
}
}
//by using this method we delete first match element.
void deleteElement(int data)
{
Node temp = head;
if(temp==null){
System.out.println("empty");
return;
}
else if(temp.data == data){
head = head.next;
size--;
return;
}
while(temp.next!=null)
{
if(temp.next.data == data){
temp.next = temp.next.next;
size--;
return;
}
else
{
```

```
temp = temp.next;
}
}
}
//by using this method we all mathch element.
void deleteElements(int data)
{
Node temp = head;
if(temp==null)
{
System.out.println("empty");
return;
}
else if(temp.data == data)
{
head = head.next;
size--;
}
else
{
while(temp.next!=null){
if(temp.next.data == data)
{
temp.next = temp.next.next;
size--;
}
else if(temp.next!=null)
{
temp = temp.next;
}
}
}
}
```

```
//delete element at particular position
void deleteElementAtPos(int pos){
Node temp = head;
int i=0;
if(temp==null){
System.out.println("empty");
return;
}
else if(pos==0){
head = head.next;
size--;
return;
}
else
{
while(temp.next!=null && i<pos){
if(i==pos-1)
{
temp.next = temp.next.next;
size--;
return;
}
i++;
temp = temp.next;
}
}
}
public static void main(String[] args)
{
SingleLinkedListDemo1 slld=new SingleLinkedListDemo1();
slld.addLast(121);
slld.addLast(122);
slld.addLast(123);
```

```
slld.addLast(125);
slld.addLast(125);
slld.addLast(125);
slld.addLast(126);
slld.printList();
System.out.println("Size= "+slld.getSize());
slld.deleteFirst();
slld.printList();
System.out.println("Size= "+slld.getSize());
slld.deleteLast();
slld.printList();
System.out.println("Size= "+slld.getSize());
slld.deleteElement(125);
slld.printList();
System.out.println("Size= "+slld.getSize());
slld.deleteElements(125);
slld.printList();
System.out.println("Size= "+slld.getSize());
slld.deleteElementAtPos(1);
slld.printList();
System.out.println("Size= "+slld.getSize());
}
}
```

Result:-

```
121 => 122 => 123 => 125 => 125 => 125 => 126 => null
Size= 7
122 => 123 => 125 => 125 => 125 => 126 => null
Size= 6
122 => 123 => 125 => 125 => 125 => null
Size= 5
122 => 123 => 125 => 125 => null
Size= 4
122 => 123 => null
```

```
Size= 2
122 => null
Size= 1

Example Single Linked List:-5 In this example we remove all
duplicate elements.
package com.vikas.ds;
public class SingleLinkedListDemo1
{
Node head;
int size;
class Node{
int data;
Node next;
Node(int data){
this.data = data;
this.next = null;
size++;
}
Node(int data,Node temp){
this.data = data;
this.next = temp;
size++;
}
}
int getSize(){
return this.size;
}
void printList(){
if(head==null){
System.out.println("list is empty");
}
else
```

```
{

Node currNode = head;
while(currNode!=null)
{
System.out.print(currNode.data+" => ");
currNode = currNode.next;
}
System.out.println("null");
}
}
void addLast(int data)
{
Node newNode = new Node(data);
if(head==null)
{
head = newNode;
return;
}
else
{
Node currNode = head;
while(currNode.next!=null)
{
currNode = currNode.next;
}
currNode.next = newNode;
}
}
void removeDuplicates()
{
Node currNode = head;
while(currNode!=null){
```

```
if(currNode.next!=null && currNode.data == currNode.next.data)
{
currNode.next = currNode.next.next;
}
else
{
currNode = currNode.next;
}
}
}

public static void main(String[] args)
{
SingleLinkedListDemo1 slld=new SingleLinkedListDemo1();
slld.addLast(121);
slld.addLast(122);
slld.addLast(123);
slld.addLast(125);
slld.addLast(125);
slld.addLast(125);
slld.addLast(126);
slld.printList();
System.out.println("Size= "+slld.getSize());
slld.removeDuplicates();
slld.printList();
System.out.println("Size= "+slld.getSize());
}
}
```

Result:-

```
121 => 122 => 123 => 125 => 125 => 125 => 126 => null
Size= 7
121 => 122 => 123 => 125 => 126 => null
Size= 7
```

Example Single Linked List:-6 In this example we insert the records in ascending order.

```
package com.vikas.ds;
public class SingleLinkedListDemo1
{
Node head;
int size;
class Node{
int data;
Node next;
Node(int data){
this.data = data;
this.next = null;
size++;
}
Node(int data,Node temp){
this.data = data;
this.next = temp;
size++;
}
}
int getSize(){
return this.size;
}
void printList(){
if(head==null){
System.out.println("list is empty");
}
else
{
Node currNode = head;
while(currNode!=null)
```

```
{
System.out.print(currNode.data+" => ");
currNode = currNode.next;
}
System.out.println("null");
}
}
void sortedInsertAsc(int data){
Node newNode = new Node(data);
Node currNode = head;
if(currNode==null||currNode.data>data){
newNode.next = head;
head = newNode;
return;
}
while(currNode.next!=null && currNode.next.data<data){
currNode = currNode.next;
}
newNode.next = currNode.next;
currNode.next = newNode;
}
public static void main(String[] args)
{
SingleLinkedListDemo1 slld=new SingleLinkedListDemo1();
slld.sortedInsertAsc(4444);
slld.sortedInsertAsc(5555);
slld.sortedInsertAsc(2222);
slld.sortedInsertAsc(6666);
slld.sortedInsertAsc(3333);
slld.sortedInsertAsc(1111);
slld.printList();
System.out.println("Size= "+slld.getSize());
}
```

```
}
```

Result:-

```
1111 => 2222 => 3333 => 4444 => 5555 => 6666 => null
Size= 6
```

Example Single Linked List:-7 In this example we insert the records in descending order.

```
package com.vikas.ds;
public class SingleLinkedListDemo1
{
Node head;
int size;
class Node{
int data;
Node next;
Node(int data){
this.data = data;
this.next = null;
size++;
}
Node(int data,Node temp){
this.data = data;
this.next = temp;
size++;
}
}
int getSize(){
return this.size;
}
void printList(){
if(head==null){
System.out.println("list is empty");
}
```

```
else
{
Node currNode = head;
while(currNode!=null)
{
System.out.print(currNode.data+" => ");
currNode = currNode.next;
}
System.out.println("null");
}
}
void sortedInsertDesc(int data){
Node newNode = new Node(data);
Node currNode = head;
if(currNode==null||currNode.data<data){
newNode.next = head;
head = newNode;
return;
}
while(currNode.next!=null && currNode.next.data>data){
currNode = currNode.next;
}
newNode.next = currNode.next;
currNode.next = newNode;
}

public static void main(String[] args)
{
SingleLinkedListDemo1 slld=new SingleLinkedListDemo1();
slld.sortedInsertDesc(4444);
slld.sortedInsertDesc(5555);
slld.sortedInsertDesc(2222);
slld.sortedInsertDesc(6666);
```

```
slld.sortedInsertDesc(3333);
slld.sortedInsertDesc(1111);
slld.printList();
System.out.println("Size= "+slld.getSize());
}
}
```

Result:-

```
6666 => 5555 => 4444 => 3333 => 2222 => 1111 => null
Size= 6
```

Example Single Linked List:-9 In this example we search an element.

```
package com.vikas.ds;
public class SingleLinkedListDemo1
{
Node head;
int size;
class Node{
int data;
Node next;
Node(int data){
this.data = data;
this.next = null;
size++;
}
Node(int data,Node temp){
this.data = data;
this.next = temp;
size++;
}
}
void addLast(int data)
{
Node newNode = new Node(data);
```

```
if(head==null)
{
head = newNode;
return;
}
else
{
Node currNode = head;
while(currNode.next!=null)
{
currNode = currNode.next;
}
currNode.next = newNode;
}
}
void printList(){
if(head==null){
System.out.println("list is empty");
}
else
{

Node currNode = head;
while(currNode!=null)
{
System.out.print(currNode.data+" => ");
currNode = currNode.next;
}
System.out.println("null");
}
}

boolean search(int data){
```

```
Node currNode = head;
while(currNode!=null){
if(currNode.data == data)
{
return true;
}
else
{
currNode = currNode.next;
}
}
return false;
}
public static void main(String[] args)
{
SingleLinkedListDemo1 slld=new SingleLinkedListDemo1();
slld.addLast(777);
slld.addLast(333);
slld.addLast(111);
slld.addLast(444);
slld.printList();
System.out.println(slld.search(333));
}
}
```

Result:-

```
777 => 333 => 111 => 444 => null
true
```

Example Single Linked List:-10 In this example we perform reverse operation.

```
package com.vikas.ds;
public class SingleLinkedListDemo1
{
Node head;
```

```
int size;
class Node{
int data;
Node next;
Node(int data){
this.data = data;
this.next = null;
size++;
}
Node(int data,Node temp){
this.data = data;
this.next = temp;
size++;
}
}
void printList(){
if(head==null){
System.out.println("list is empty");
}
else
{
Node currNode = head;
while(currNode!=null)
{
System.out.print(currNode.data+" => ");
currNode = currNode.next;
}
System.out.println("null");
}
}
void addLast(int data)
{
Node newNode = new Node(data);
```

```
if(head==null)
{
head = newNode;
return;
}
else
{
Node currNode = head;
while(currNode.next!=null)
{
currNode = currNode.next;
}
currNode.next = newNode;
}
}
void reverse(){
Node curr = head, prev=null,next=null;
while(curr!=null){
next = curr.next;
curr.next = prev;
prev = curr;
curr = next;
}
head = prev;
}
public static void main(String[] args)
{
SingleLinkedListDemo1 slld=new SingleLinkedListDemo1();
slld.addLast(121);
slld.addLast(122);
slld.addLast(125);
slld.addLast(126);
slld.printList();
```

```
slld.reverse();
slld.printList();
}
}
```

Result:-

```
121 => 122 => 125 => 126 => null
126 => 125 => 122 => 121 => null
```

Double Linked List:-

Double Linked List implementations:-

Example Double Linked List:-1 In this example we performed add element first, traversed list(print list), and size of the list.

```
package com.vikas.ds;
public class DoubleLinkedListDemo
{
Node head;
int size = 0;
class Node{
int data;
Node next,prev;
Node(int data,Node next,Node prev){
this.data = data;
this.next = next;
this.prev = prev;
size++;
}
}

void addFirst(int data)
{
Node newNode = new Node(data,null,null);
if(head==null)
{
```

```
head = newNode;
}
else
{
head.prev = newNode;
newNode.next = head;
head = newNode;
}
}
void traverse() {
if(head==null) {
System.out.println("List is Empty");
return;
}
else
{
Node currNode = head;
while(currNode!=null)
{
System.out.print(currNode.data+" => ");
currNode = currNode.next;
}
System.out.println("NULL");
}
}
public static void main(String[] args)
{
DoubleLinkedListDemo dlld=new DoubleLinkedListDemo();
dlld.addFirst(555);
dlld.addFirst(444);
dlld.addFirst(333);
dlld.addFirst(222);
dlld.traverse();
```

```
}
}
```

Result:-

```
222 => 333 => 444 => 555 => NULL
```

Example Double Linked List:-2 In this example we performed add element last, traversed list(print list), and size of the list.

```
package com.vikas.ds;
public class DoubleLinkedListDemo
{
Node head;
int size = 0;
class Node{
int data;
Node next,prev;
Node(int data,Node next,Node prev){
this.data = data;
this.next = next;
this.prev = prev;
size++;
}
}
void addLast(int data) {
Node newNode = new Node(data,null,null);
if(head==null)
{
head = newNode;
}
else
{
Node currNode = head;
while(currNode.next != null)
{
```

```
currNode = currNode.next;
}
currNode.next = newNode;
newNode.prev = currNode;
}
}
void traverse() {
if(head==null) {
System.out.println("List is Empty");
return;
}
else
{
Node currNode = head;
while(currNode!=null)
{
System.out.print(currNode.data+" => ");
currNode = currNode.next;
}
System.out.println("NULL");
}
}
public static void main(String[] args)
{
DoubleLinkedListDemo dlld=new DoubleLinkedListDemo();
dlld.addLast(555);
dlld.addLast(444);
dlld.addLast(666);
dlld.addLast(111);
dlld.addLast(888);
dlld.traverse();
}
}
```

Result:-

```
555 => 444 => 666 => 111 => 888 => NULL
```

Example Double Linked List:-3 In this example we performed Inserting the data at a particular position.

```
package com.vikas.ds;
public class DoubleLinkedListDemo
{
Node head;
int size = 0;
class Node{
int data;
Node next,prev;
Node(int data,Node next,Node prev){
this.data = data;
this.next = next;
this.prev = prev;
size++;
}
}
void addPos(int data,int pos) {
int i=0;
if(pos<0 || pos>=size)
{
System.out.println("out of range");
return;
}
Node newNode = new Node(data,null,null);
if (head==null) {
head = newNode;
return;
}
if(pos!=0)
```

```
{
Node currNode = head, temp = null;
while(currNode.next!=null && i<pos)
{
temp=currNode;
currNode = currNode.next;
i++;
}
temp.next = newNode;
newNode.prev = temp;
newNode.next = currNode;
currNode.prev = newNode;
}
else {
newNode.next = head;
head.prev = newNode;
head = newNode;
}
}
void addLast(int data) {
Node newNode = new Node(data,null,null);
if(head==null)
{
head = newNode;
}
else
{
Node currNode = head;
while(currNode.next != null)
{
currNode = currNode.next;
}
currNode.next = newNode;
```

```
newNode.prev = currNode;
}
}
void traverse() {
if(head==null) {
System.out.println("List is Empty");
return;
}
else
{
Node currNode = head;
while(currNode!=null)
{
System.out.print(currNode.data+" => ");
currNode = currNode.next;
}
System.out.println("NULL");
}
}
public static void main(String[] args)
{
DoubleLinkedListDemo dlld=new DoubleLinkedListDemo();
dlld.addLast(555);
dlld.addLast(444);
dlld.addLast(666);
dlld.addLast(111);
dlld.addLast(888);
dlld.traverse();
dlld.addPos(333,3);
dlld.traverse();
}}
```

Result:- 555 => 444 => 666 => 111 => 888 => NULL

555 => 444 => 666 => 333 => 111 => 888 => NULL

Example Double Linked List:-4 In this example we performed delete operations like:-

Deleting from first element

Deleting from the last element

Delete from particular position

Deleting Element first match

Deleting Elements all match

```
package com.vikas.ds;
public class DoubleLinkedListDemo
{
Node head;
int size = 0;
class Node{
int data;
Node next,prev;
Node(int data,Node next,Node prev){
this.data = data;
this.next = next;
this.prev = prev;
size++;
}
}
void deleteFirst() {
if(head==null) {
System.out.println("DLL is empty");
return;
}
size--;
head = head.next;
if(head!=null)
head.prev = null;
}
void deleteLast() {
```

```
if(head==null) {
System.out.println("DLL is empty");
return;
}
if(head.next == null) {
head = null;
size--;
return;
}
size--;
Node temp1 = head, temp2 = head.next;
while(temp2.next!=null) {
temp2 = temp2.next;
temp1 = temp1.next;
}
temp1.next = null;
}
void deleteElementAtPos(int pos) {
Node temp1 = head, temp2;
int i=0;
if(temp1==null) {
System.out.println("DLL is empty");
return;
}
if(pos<0 || pos>=size) {
System.out.println("out of range");
return;
}
if(pos==0) {
head = head.next;
if(head!=null)
head.prev = null;
size--;
```

```
return;
}
while(temp1.next!=null && i<pos) {
if(i==pos-1) {
temp1.next = temp1.next.next;
temp2 = temp1.next;
if(temp2!=null)
temp2.prev = temp1;
size--;
return;
}
i++;
temp1 = temp1.next;
temp2 = temp1.next;
}
}
//by using this method we delete first match element.
void deleteElement(int data) {
Node temp1 = head, temp2;
if(temp1==null) {
System.out.println("DLL empty");
return;
}
if(temp1.data == data) {
head = head.next;
if(head!=null)
head.prev = null;
size--;
return;
}
while(temp1.next!=null)
{
if(temp1.next.data == data) {
```

```
temp1.next = temp1.next.next;
temp2 = temp1.next;
if(temp2!=null)
temp2.prev = temp1;
size--;
return;
}
temp1 = temp1.next;
}
}
//by using this method we all mathch element.
void deleteElements(int data) {
Node temp1 = head, temp2;
if(temp1==null) {
System.out.println("DLL empty");
return;
}
if(temp1.data == data) {
head = head.next;
if(head!=null)
head.prev = null;
size--;
}
while(temp1.next!=null)
{
if(temp1.next.data == data) {
temp1.next = temp1.next.next;
temp2 = temp1.next;
if(temp2!=null)
temp2.prev = temp1;
size--;
}
if(temp1.next!=null)
```

```
temp1 = temp1.next;
}
}
void addLast(int data) {
Node newNode = new Node(data,null,null);
if(head==null)
{
head = newNode;
}
else
{
Node currNode = head;
while(currNode.next != null)
{
currNode = currNode.next;
}
currNode.next = newNode;
newNode.prev = currNode;
}
}
void traverse() {
if(head==null) {
System.out.println("List is Empty");
return;
}
else
{
Node currNode = head;
while(currNode!=null)
{
System.out.print(currNode.data+" => ");
currNode = currNode.next;
}
```

```
System.out.println("NULL");
}
}
public static void main(String[] args)
{
DoubleLinkedListDemo dlld=new DoubleLinkedListDemo();
dlld.addLast(111);
dlld.addLast(555);
dlld.addLast(444);
dlld.addLast(111);
dlld.addLast(666);
dlld.addLast(111);
dlld.addLast(888);
dlld.addLast(111);
dlld.addLast(111);
dlld.addLast(333);
dlld.addLast(777);
dlld.traverse();
dlld.deleteFirst();
dlld.traverse();
dlld.deleteLast();
dlld.traverse();
dlld.deleteElementAtPos(2);
dlld.traverse();
dlld.deleteElement(111);
dlld.traverse();
dlld.deleteElements(111);
dlld.traverse();
}
}
```

Result:-

```
111 => 555 => 444 => 111 => 666 => 111 => 888 => 111 => 111 => 333
=> 777 => NULL
555 => 444 => 111 => 666 => 111 => 888 => 111 => 111 => 333 => 777
=> NULL
555 => 444 => 111 => 666 => 111 => 888 => 111 => 111 => 333 => NULL
555 => 444 => 666 => 111 => 888 => 111 => 111 => 333 => NULL
555 => 444 => 666 => 888 => 111 => 111 => 333 => NULL
555 => 444 => 666 => 888 => 111 => 333 => NULL
```

Circular Single Linked List:-

Example Circular Single Linked List:-1

```
package com.vikas.ds;
public class CircularSingleLinkedList
{
Node tail;
int size = 0;
class Node{
int value;
Node next;
Node(int value,Node next){
this.value = value;
this.next = next;
}
}
void print() {
if(size==0) {
System.out.println("CSLL is empty");
return;
}
Node temp = tail.next;
while(temp!=tail) {
System.out.print(temp.value+" => ");
```

```
temp=temp.next;
}
System.out.println(temp.value);
}
void addHead(int value) {
Node temp = new Node(value,null);
if(size==0) {
tail = temp;
temp.next = temp;
}
else {
temp.next = tail.next;
tail.next = temp;
}
size++;
}
void addTail(int value) {
Node temp = new Node(value,null);
if(size==0) {
tail = temp;
temp.next = temp;
}
else {
temp.next = tail.next;
tail.next = temp;
tail = temp;
}
size++;
}
void addPos(int pos,int value) {
Node newNode = new Node(value,null);
if(size==0) {
tail = newNode;
```

```
newNode.next = newNode;
}
else {
if(pos==0) {
Node temp = tail.next;
newNode.next = temp;
tail.next = newNode;
return;
}
Node temp = tail.next;
int i=0;
while(temp.next!=tail && i<pos-1) {
temp=temp.next;
i++;
}
newNode.next = temp.next;
temp.next = newNode;
}
size++;
}
void removeHead() {
if(size==0) {
System.out.println("CSLL is empty");
return;
}
if(tail==tail.next)
tail = null;
else
tail.next = tail.next.next;
size--;
}
void removeTail() {
if(size==0) {
```

```
System.out.println("CSLL is empty");
return;
}
if(tail==tail.next)
tail = null;
else
{
Node temp = tail.next;
while(temp.next!=tail) {
temp = temp.next;
}
temp.next = tail.next;
tail = temp;
}
size--;
}
void deleteElement(int value) {
if(size==0) {
System.out.println("CSLL is empty");
return;
}
Node prev=tail,currNode=tail.next,head=tail.next;
if(currNode.value==value) {
if(currNode==currNode.next)
tail = null;
else
tail.next = tail.next.next;
return;
}
prev = currNode;
currNode = currNode.next;
while(currNode!=head) {
if(currNode.value == value) {
```

```
if(currNode==tail)
tail = prev;
prev.next = currNode.next;
return;
}
prev = currNode;
currNode = currNode.next;
}
return;
}
boolean search(int value) {
Node temp = tail;
for(int i=0;i<size;i++) {
if(temp.value==value)
return true;
temp = temp.next;
}
return false;
}

public static void main(String[] args)
{
CircularSingleLinkedList csll=new CircularSingleLinkedList();
csll.addHead(222);
csll.addHead(444);
csll.addHead(666);
csll.addHead(666);
csll.addHead(777);
csll.addHead(999);
csll.addHead(111);
csll.print();
csll.addTail(888);
csll.print();
```

```
csll.addPos(3,555);
csll.print();
csll.removeHead();
csll.print();
csll.removeTail();
csll.print();
csll.deleteElement(666);
csll.print();
System.out.println(csll.search(999));
}
}
```

```
Result:-
111 => 999 => 777 => 666 => 666 => 444 => 222
111 => 999 => 777 => 666 => 666 => 444 => 222 => 888
111 => 999 => 777 => 555 => 666 => 666 => 444 => 222 => 888
999 => 777 => 555 => 666 => 666 => 444 => 222 => 888
999 => 777 => 555 => 666 => 666 => 444 => 222
999 => 777 => 555 => 666 => 444 => 222
true
```

Circular Double Linked List:-

Example Circular Double Linked List:-1

```
package com.vikas.ds;
public class CircularDoubleLinkedList
{
Node head = null;
Node tail = null;
int size = 0;
class Node{
int value;
Node next,prev;
Node(int value,Node next,Node prev){
```

```
this.value = value;
this.next = next;
this.prev = prev;
}
}
void print() {
if(size==0) {
System.out.println("CDLL is empty");
return;
}
Node temp = tail.next;
while(temp!=tail) {
System.out.print(temp.value+" ==> ");
temp = temp.next;
}
System.out.println(temp.value);
}
void addHead(int value) {
Node newNode = new Node(value,null,null);
if(size==0) {
tail = head = newNode;
newNode.next = newNode;
newNode.prev = newNode;
}
else {
newNode.next = head;
newNode.prev = head.prev;
head.prev = newNode;
newNode.prev.next = newNode;
head = newNode;
}
size++;
}
```

```
void addTail(int value) {
Node newNode = new Node(value,null,null);
if(size==0) {
head = tail = newNode;
newNode.next = newNode;
newNode.prev = newNode;
}
else {
newNode.next = tail.next;
newNode.prev = tail;
tail.next = newNode;
newNode.next.prev = newNode;
tail = newNode;
}
size++;
}
void removeHead() {
if(size==0) {
System.out.println("CDLL is empty");
return;
}
size--;
if(size==0) {
head = null;
tail = null;
return;
}
Node temp = head.next;
temp.prev = tail;
tail.next = temp;
head = temp;
}
void removeTail() {
```

```
if(size==0) {
System.out.println("CDLL is empty");
return;
}
size--;
if(size==0) {
head=null;
tail=null;
return;
}
Node temp = tail.prev;
temp.next = head;
head.prev = temp;
tail = temp;
}
public static void main(String[] args)
{
CircularDoubleLinkedList cdll=new CircularDoubleLinkedList();
cdll.addHead(444);
cdll.addHead(222);
cdll.addHead(777);
cdll.addHead(555);
cdll.addHead(999);
cdll.print();
cdll.addTail(666);
cdll.print();
cdll.removeHead();
cdll.print();
cdll.removeTail();
cdll.print();
}
}
```

Result:-

999 ==> 555 ==> 777 ==> 222 ==> 444

999 ==> 555 ==> 777 ==> 222 ==> 444 ==> 666

555 ==> 777 ==> 222 ==> 444 ==> 666

555 ==> 777 ==> 222 ==> 444

Polynomial Representation and Addition:-

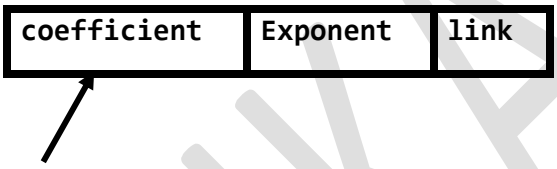
Polynomial:- **It is a mathematical expression, that consisting coefficient(constant), variable and exponent(power).**

$2x^7$

Where 2 is coefficient, x is variable and 7 is exponent.

We perform Polynomial addition by using Linked List:- We store each polynomial as Singly Linked List.

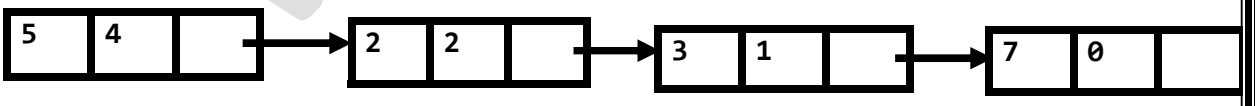
Where each node stores the coefficient, exponent, and reference to the next node like:-



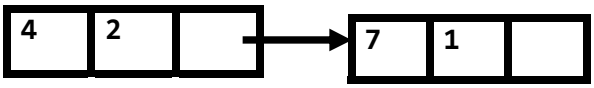
node

Example:-1

Polynomial Expression:-1 $5x^4+2x^2+3x+7$

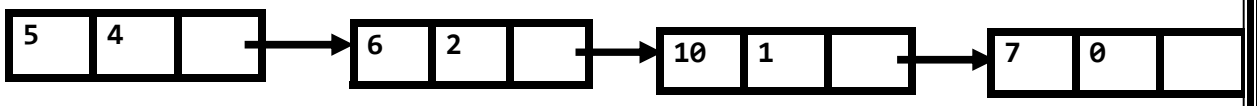


Polynomial Expression:-2 $4x^2+7x$



Perform addition with above polynomial expression:-

$5x^4+6x^2+10x+7$



Generalized Linked List:- A Generalization linked list is a finite sequence of n element where n>=0, l1, l2, l3, l4,.....ln. Where li are the item or list of item or both thus L(l1,l2,l3....ln) Here n represent number of nodes.

Represent Generalized Linked List:-

Flag	Data	Down Reference	Next reference

Flag value can 0 or 1.

0 value represent next node reference.

1 value represent down node reference.

Example:- (a,(b,c)d)