**Linked List**

1. **Delete node**
2. *public class* LinkedList {  
    Node head;  
    Node last;  
    *static int size*=0;  
    *public static int* size(){  
    *return size*;  
    }  
     
    */\* Linked list Node\*/  
    class* Node{  
    *int* data;  
    Node next;  
    *public* Node(*int* d){  
    *this*.data=d;  
    next=*null*;  
    *size*++;  
    }  
    }  
     
     
    */\*traversing a linked list\*/  
    public void* traversal(){  
    Node temp = head;  
    *for* (*int* i=0;i<LinkedList.*size*();i++){  
    *if*(temp.next==*null*){  
    System.***out***.println(temp.data);  
    *return* ;  
    }  
    System.***out***.print(temp.data+"-> ");  
    temp=temp.next;  
    }  
    *return*;  
    }  
     
     
    */\*deleting a node from its index\*/  
    void* deleteNode(*int* value){  
    Node temp= head,prev=*null*;  
     
    *if*(temp!=*null* && temp.data==value){  
    head=temp.next;  
    *return*;  
    }  
     
    *while* (temp!=*null* && temp.data!=value){  
    prev=temp;  
    temp=temp.next;  
    }  
     
    *if*(temp==*null*){  
    *return*;  
    }  
     
    prev.next=temp.next;  
    }  
     
    *public void* deleteByKey(*int* key){  
    Node temp= head,prev=*null*;  
    *if*(key==0){  
    head=*null*;  
    }  
    *if*(key>=LinkedList.*size*()){  
    *return*;  
    }  
    *for* (*int* i=0;i<key;i++){  
    prev=temp;  
    temp=temp.next;  
    }  
    prev.next=temp.next;  
    }  
     
    */\*Insert a new Node at Front of the list. \*/  
    public void* push(*int* data){  
    Node new\_node= *new* Node(data);  
    new\_node.next=head;  
    head=new\_node;  
    }  
     
    *public void* pushLast(*int* data){  
     
    Node new\_node = *new* Node(data);  
    *if*(head==*null*){  
    head=new\_node;  
    last=new\_node;  
    }  
    last.next=new\_node;  
    last=new\_node;  
     
    }  
     
    *public static void* main(String[] args) {  
    LinkedList llist= *new* LinkedList();  
     
    llist.pushLast(3);  
    llist.pushLast(4);  
    llist.pushLast(5);  
    llist.pushLast(2);  
    llist.pushLast(6);  
    llist.pushLast(1);  
    llist.pushLast(9);  
     
    llist.traversal();  
    llist.deleteByKey(3);  
    llist.traversal();  
    }  
   }

**2 . Middle of linked list**

*/\*Middle of linked list\*/  
public void* midOfLinkedList(Node head){  
 Node slow=head,fast=head;  
 *while*(fast!=*null*&&fast.next!=*null*){  
 fast=fast.next.next;  
 slow=slow.next;  
 }  
 System.***out***.println(slow.data);  
}

1. **Binary to integer**
2. */\*Binary to integer\*/  
   public void* binaryTointeger(Node head,*int* size){  
    Node temp=head;  
    *int* sum=0;  
    *for* (*int* i=0;i<size;i++){  
    *if*(temp.data==1){  
    sum+=power(2,size-1-i);  
    }  
    temp=temp.next;  
    }  
    System.***out***.println(sum);  
   }  
     
   *private int* power(*int* b, *int* p) {  
    *int* sum=1;  
    *for* (*int* i=0;i<p;i++){  
    sum\*=2;  
    }  
    *return* sum;  
   }

**4.Remove duplicate from unsorted linked list**

*/\*remove Duplicates from Unsorted Linked List\*/  
public* Node removeDuplicate(Node head){  
 Node temp=head,prev=head;  
 *int* i=0;  
 HashMap<Integer,Integer> hmap = *new* HashMap<>();  
 *while*(temp!=*null*){  
 *if*(hmap.containsValue(temp.data)){  
 temp=temp.next;  
 prev.next=temp;  
 }*else* {  
 hmap.put(i++, temp.data);  
 prev=temp;  
 temp=temp.next;  
 }  
  
 }  
 *return* head;  
}

1. **Merge two sorted linked list**
2. *public static void* main(String[] args) {  
    LinkedList llist1 = *new* LinkedList();  
    LinkedList llist2 = *new* LinkedList();  
     
    llist1.pushLast(1);  
    llist1.pushLast(4);  
    llist1.pushLast(5);  
     
    llist1.*traversal*(llist1.head);  
     
    llist2.pushLast(2);  
    llist2.pushLast(3);  
    llist2.pushLast(5);  
     
    llist2.*traversal*(llist2.head);  
     
    Node head= *mergeLinkedList*(llist1.head,llist2.head);  
    *traversal*(head);  
     
   }

*/\*merging two sorted array\*/  
public static* Node mergeLinkedList(Node head1,Node head2){  
 Node temp1=head1,temp2=head2;  
 Node head,tail,i=temp1,j=temp2;  
 *if*(temp1.data<temp2.data){  
 head=temp1;  
 tail=temp1;  
 i=i.next;  
 }*else* {  
 head=temp2;  
 tail=temp2;  
 j=j.next;  
 }  
  
 *while*(i!=*null*&&j!=*null*){  
 *if*(i.data<=j.data){  
 tail.next=i;  
 tail=i;  
 i=i.next;  
 }*else* {  
 tail.next=j;  
 tail=j;  
 j=j.next;  
 }  
 }  
  
 *if*(i==*null*){  
 tail.next=j;  
 tail=j;  
 *while* (tail.next!=*null*){  
 tail=tail.next;  
 }  
 }*else* {  
 tail.next=i;  
 tail=i;  
 *while*(tail.next!=*null*){  
 tail=tail.next;  
 }  
 }  
  
 *return* head;  
 }

# 6. Cycle Detection in a Singly Linked List

*public static void* main(String[] args) {  
  
 LinkedList llist= *new* LinkedList();  
  
 llist.pushLast(1);  
 llist.pushLast(4);  
 llist.pushLast(0);  
 llist.pushLast(1);  
 llist.pushLast(4);  
 llist.pushLast(1);  
 llist.pushLast(6);  
 llist.pushLast(1);  
*// llist.pushLast(9);* llist.cycleList(9,4);  
  
 *boolean* isCycle= llist.isCyclePresent(llist.head);  
 System.***out***.println(isCycle);}

*public void* cycleList(*int* data,*int* next){  
 Node new\_node= *new* Node(data);  
 last.next=new\_node;  
 last=new\_node;  
 Node temp=head;  
 *for*(*int* i=0;i<next;i++){  
 temp=temp.next;  
 }  
 new\_node.next=temp;  
  
}

*public boolean* isCyclePresent(Node head){  
 Node fast=head,slow=head;  
 *if*(fast.next==*null*&&fast.next.next==head){  
 *return true*;  
 }  
 slow=slow.next;  
 fast=fast.next.next;  
  
 *while* (fast!=slow&&fast.next!=*null*&&fast.next.next!=*null*){  
 slow=slow.next;  
 fast=fast.next.next;  
 }  
 *if*(fast.next==*null*||fast.next.next==*null*){  
 *return false*;  
 }  
  
 *return true*;  
}

1. **Intersection of linked list**

*public static void* main(String[] args) {  
  
 LinkedList llist1 = *new* LinkedList();  
 LinkedList llist2 = *new* LinkedList();  
 LinkedList llist3 = *new* LinkedList();  
  
 llist1.pushLast(1);  
 llist1.pushLast(4);  
 llist1.pushLast(1);  
 llist2.pushLast(1);  
 llist2.pushLast(4);  
 llist2.pushLast(1);  
 llist3.pushLast(6);  
 llist3.pushLast(1);  
 llist3.pushLast(9);  
*// llist.cycleList(9,4);* Node head = *Intersection*(llist1.head, llist2.head, llist3.head);  
 *traversal*(llist1.head);  
 *traversal*(llist2.head);  
  
 *int* point = *intersectionPoint*(llist1.head, llist2.head);  
 System.***out***.println(point);  
  
 }

*public static* Node Intersection(Node head1,Node head2,Node head3){  
 Node temp1=head1,temp2=head2;  
 *while* (temp1.next!=*null*){  
 temp1=temp1.next;  
 }  
 *while* (temp2.next!=*null*){  
 temp2=temp2.next;  
 }  
  
 *if* (head3!=*null*){  
 temp1.next=head3;  
 temp1=head3;  
 temp2.next=head3;  
 temp2=head3;  
  
 }*else* {  
 *return* head3;  
 }  
 *return* head1;  
}  
  
*public static int* intersectionPoint(Node head1,Node head2){  
 Node temp1=head1,temp2=head2;  
  
 HashMap<Integer,Node> hmap = *new* HashMap<>();  
 *int* i=0,j=0;  
 *while* (temp1.next!=*null*){  
 hmap.put(i++,temp1);  
 temp1= temp1.next;  
 }  
 *while* (temp2.next!=*null*&&!hmap.containsValue(temp2)){  
 temp2=temp2.next;  
 }  
  
 *if*(temp2.next==*null*){  
 *return* -1;  
 }  
  
 *return* temp2.data;  
  
}

1. **Palindrome linked list**

*public static void* main(String[] args) {  
  
 LinkedList llist1 = *new* LinkedList();  
*// LinkedList llist2 = new LinkedList();  
// LinkedList llist3 = new LinkedList();* llist1.pushLast(1);  
 llist1.pushLast(4);  
 llist1.pushLast(1);  
 llist1.pushLast(1);  
 llist1.pushLast(4);  
 llist1.pushLast(1);  
 llist1.pushLast(5);  
*// llist3.pushLast(6);  
// llist3.pushLast(1);  
// llist3.pushLast(9);  
// llist.cycleList(9,4);  
  
 traversal*(llist1.head);  
 *boolean* ans = *isPalindrome*(llist1.head);  
 System.***out***.println(ans);  
  
 }

*public static boolean* isPalindrome(Node head){  
 Node s=head,f=head;  
 ArrayList<Integer> arr = *new* ArrayList<>();  
  
  
 *while* (f.next!=*null*&&f.next.next!=*null*){  
 arr.add(s.data);  
 s=s.next;  
 f=f.next.next;  
 }  
 arr.add(s.data);  
 *int* i= arr.size()-1;  
 *if*(*size*(head)%2==0){  
 *while* (s.next!=*null*){  
 *if*(s.next.data!= arr.get(i)){  
 *return false*;  
 }  
 i--;  
 s=s.next;  
 }  
 }*else* {  
 *while* (s.next!=*null*){  
 *if*(s.next.data!= arr.get(i-1)){  
 *return false*;  
 }  
 i--;  
 s=s.next;  
 }  
 }  
 *return true*;  
}

**9.Remove all Nodes with value K**

*public static void* main(String[] args) {  
  
 LinkedList llist1 = *new* LinkedList();  
*// LinkedList llist2 = new LinkedList();  
// LinkedList llist3 = new LinkedList();* llist1.pushLast(1);  
 llist1.pushLast(4);  
 llist1.pushLast(1);  
 llist1.pushLast(1);  
 llist1.pushLast(4);  
 llist1.pushLast(1);  
 llist1.pushLast(5);  
*// llist3.pushLast(6);  
// llist3.pushLast(1);  
// llist3.pushLast(9);  
// llist.cycleList(9,4);  
  
 traversal*(llist1.head);  
 llist1.deleteAllNode(6);  
 *traversal*(llist1.head);  
  
 }

*/\*deleting all node from its value\*/  
void* deleteAllNode(*int* value){  
 Node temp= head,prev=*null*;  
  
 *if*(temp==*null*){  
 *return*;  
 }  
   
 *while* (temp.data==value){  
 head=temp.next;  
 temp=head;  
 }  
  
 *while* (temp!=*null*){  
 *if*(temp.data==value){  
 prev.next=temp.next;  
 temp=temp.next;  
 }*else* {  
 prev = temp;  
 temp = temp.next;  
 }  
 }  
  
  
  
  
}