Linked List Interview Questions

A list of top frequently asked **Linked List Interview Questions** and answers are given below.

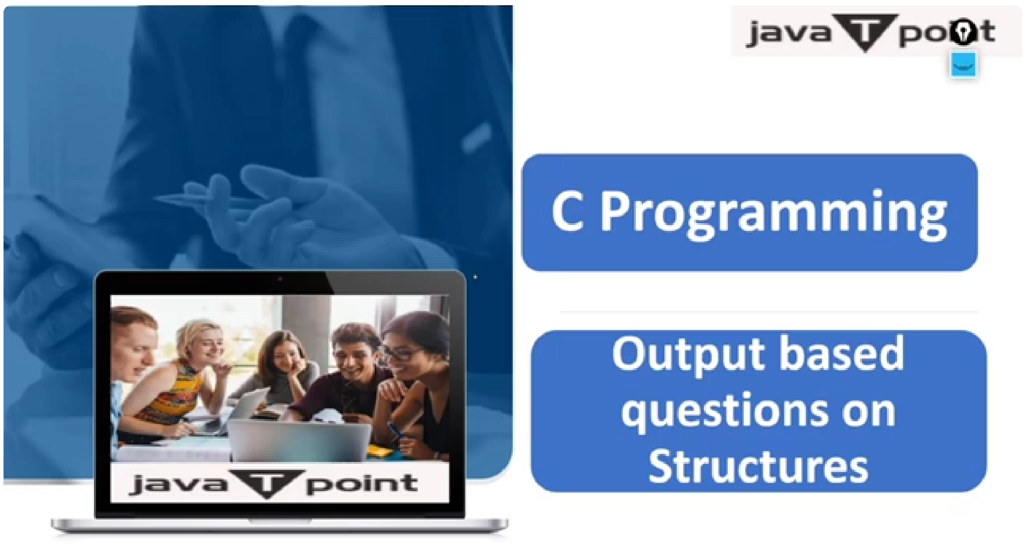
1) Explain Linked List in short.

A linked list may be defined as a linear data structure which can store a collection of items. In another way, the linked list can be utilized to store various objects of similar types. Each element or unit of the list is indicated as a node. Each node contains its data and the address of the next node. It is similar to a chain. Linked lists are used to create graphs and trees.

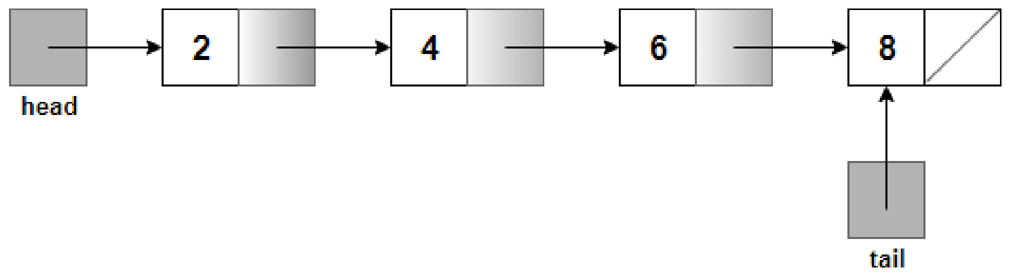
2) How will you represent a linked list in a graphical view?

A linked list may be defined as a data structure in which each element is a separate object. Linked list elements are not kept at the contiguous location. The pointers are used to link the elements of the Linked List.

Each node available in a list is made up of two items- **the data itself** and **a reference (also known as a link) to the next node** in the sequence. The last node includes a reference to null. The starting point into a linked list is known as the head of the list. It should be noted that the head is a reference to the first node, not a separate node. The head is considered as a null reference if the list is empty.



⇧ SCROLL TO TOP



Linked List Representation

3) How many types of Linked List exist?

There are multiple types of Linked Lists available:

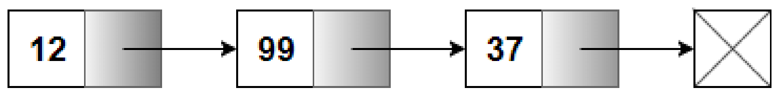
- Singly Linked List
- Doubly Linked List
- Multiply Linked List
- Circular Linked List

4) Explain Singly Linked List in short.

The singly linked list includes nodes which contain a data field and next field. The next field further points to the next node in the line of nodes.

In other words, the nodes in singly linked lists contain a pointer to the next node in the list. We can perform operations like insertion, deletion, and traversal in singly linked lists.

A singly linked list is shown below whose nodes contain two fields: an integer value and a pointer value (a link to the next node).

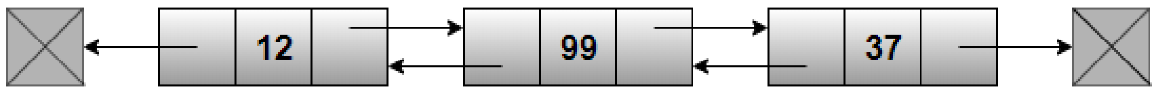


Singly Linked List

5) What do you understand by Doubly Linked List?

The doubly linked list includes a pointer (link) to the next node as well as to the previous node in the list. The two links between the nodes may be called "forward" and "backward, "or "next" and "prev (previous)." A doubly linked list is shown below whose nodes consist of three fields: an integer

⇧ SCROLL TO TOP ts to the next node, and a link that points to the previous node.



Doubly Linked List

A technique (known as **XOR-linking**) is used to allow a doubly-linked list to be implemented with the help of a single link field in each node. However, this technique needs more ability to perform some operations on addresses, and therefore may not be available for some high-level languages.

Most of the modern operating systems use doubly linked lists to maintain references to active threads, processes, and other dynamic objects.

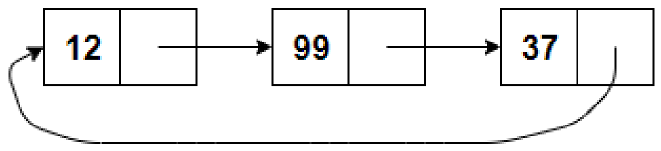
6) Explain Multiply Linked List in short.

In a multiply linked list, each node consists of two or more link fields. Each field is used to join the same set of records in a different order of the same set, e.g. "by name, by date of birth, by the department, etc.".

7) How will you explain Circular Linked List?

In the last node of a linked list, the link field often contains a null reference. Instead of including a null pointer at the end of the list, the last node in circular linked lists includes a pointer pointing to the first node. In such cases, the list is said to be ?circularly linked? otherwise it is said to be 'open' or 'linear.' A circular linked list is that type of list where the last pointer points or contains the address of the first node.

⇧ SCROLL TO TOP



Circular Linked List

In case of a circular doubly linked list, the first node also points to the last node of the list.

8) How many pointers are necessary to implement a simple Linked List?

There are generally three types of pointers required to implement a simple linked list:

- A 'head' pointer which is used for pointing to the start of the record in a list.
- A 'tail' pointer which is used for pointing to the last node. The key factor in the last node is that its subsequent pointer points to nothing (NULL).
- A pointer in every node which is used for pointing to the next node element.

9) How can you represent a linked list node?

The simplest method to represent a linked list node is wrapping the data and the link using **typedef structure**. Then further assigning the structure as a Node pointer that points to the next node.

An example of representation in C can be defined as:

```
/*ll stands for linked list*/
typedefstructll
{
int data;
structll *next;
} Node
```

10) Which type of memory allocation is referred for Linked List?

Dynamic memory allocation is referred for Linked List.

⇧ SCROLL TO TOP

know about traversal in linked lists?

The term 'traversal' refers to the operation of processing each element present in the list.

12) What are the main differences between the Linked List and Linear Array?

Linked List	**Linear Array**
Deletion and insertion are easy.	Deletion and insertion are tough.
The linked list doesn't require movement of nodes while performing deletion and insertion.	Linear Array requires movement of nodes while performing deletion and insertion.
In the Linked List, space is not wasted.	In Linear Array, space is wasted.
Linked List is not expensive.	Linear Array is a bit expensive.
It has an option to be extended or reduced as per the requirements.	It cannot be reduced or extended.
To avail each element in Linked List, a different amount of time is required.	To avail each element in Linear Array, the same amount of time is required.
Elements in the linked list may or may not be stored in consecutive memory locations.	In consecutive memory locations, elements are stored.
To reach a particular node, we need to go through all the nodes that come before that particular node.	We can reach any particular element directly.

13) What does the dummy header in the linked list contain?

In a linked list, the dummy header consists of the first record of the actual data.

14) Mention a few applications of Linked Lists?

Few of the main applications of Linked Lists are:

- Linked Lists let us implement queues, stacks, graphs, etc.

⇧ SCROLL TO TOP us insert elements at the beginning and end of the list.

15) How can you insert a node to the beginning of a singly linked list?

To insert the node to the beginning of the list, we need to follow these steps:

- Create a new node
- Insert new node by assigning the head pointer to the new node's next pointer
- Update the head pointer to point the new node

```
Node *head;
voidInsertNodeAtFront(int data)
{
/* 1. create the new node*/
Node *temp = new Node;
temp->datadata = data;
/* 2. insert it at the first position*/
temp->next = head;
/* 3. update head to point to this new node*/
head = temp;
}
```

16) Name some applications which use Linked Lists.

Both queues and stacks can be implemented using a linked list. Some of the other applications that use linked list are a binary tree, skip, unrolled linked list, hash table, etc.

17) Differentiate between singly and doubly linked lists?

Doubly linked list nodes consist of three fields: an integer value and two links pointing to two other nodes (one to the last node and another to the next node).

On the other hand, a singly linked list consists of a link which points only to the next node.

18) What is the main advantage of a linked list?

The main advantage of a linked list is that we do not need to specify a fixed size for the list. The more elements we add to the chain, the bigger the chain gets.

⇧ SCROLL TO TOP

neone add an item to the beginning of the list?

Adding an item to the beginning of the list, we need to follow the given steps:

- Create a new item and set up its value.
- Link the newly created item pointing to the head of the list.
- Set up the head of the list as our new item.

If we are using a function to do this operation, we need to alter the head variable.

20) How can someone insert a node at the end of Linked List?

This case is a little bit difficult as it depends upon the type of implementation. If we have a tail pointer, then it is simple. In case we do not have a tail pointer, we will have to traverse the list until we reach to the end (i.e., the next pointer is NULL). Then we need to create a new node and make that last node?s next pointer point to the new node.

```
voidInsertNodeAtEnd(int data)
{
  /* 1. create the new node*/
   Node *temp = new Node;
temp->datadata = data;
temp->next = NULL;
   /* check if the list is empty*/
if (head == NULL)
   {
head = temp;
return;
   }
else
   {
      /* 2. traverse the list till the end */
      Node *traveler = head;
                         xt != NULL)
                          aveler->next;
```

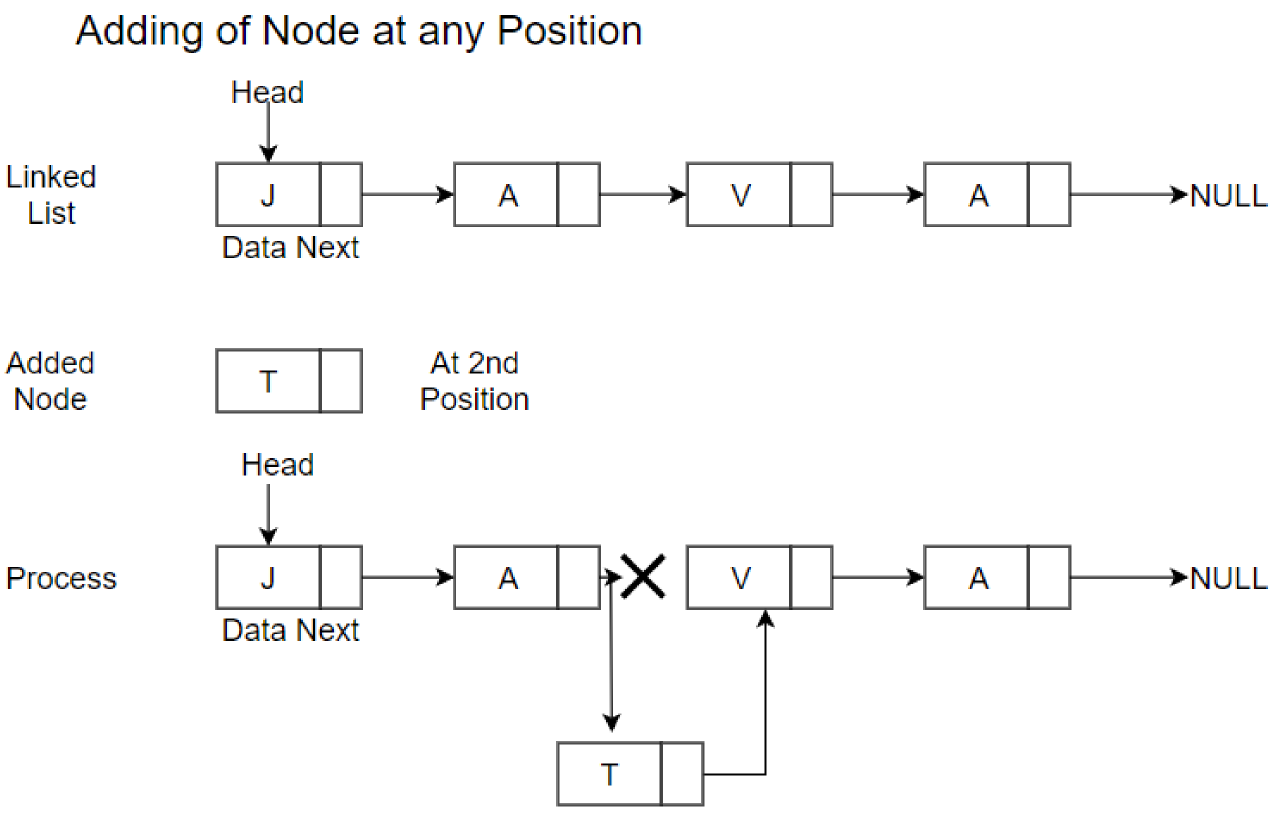
⇧ SCROLL TO TOP

```
        /* 3. Set up the last node to point to this new node*/
traveler->next = temp;
    }
}
```

21) How can someone insert a node in a random location of the Linked List?

If we want to insert a node in the first position or an empty list, we can insert the node easily. Otherwise, we need to traverse the list until we reach the specified position or at the end of the list. Then we can insert a new node. Inserting a node in the middle position is a little bit difficult as we have to make sure that we perform the pointer assignment in the correct order. To insert a new node in the middle, follow the steps:

- First, we need to set the new node's next pointer to that node which is present before it.
- Then we are required to assign a previous node's next pointer to the starting position of the new node.



Check the example below:

```
voidInsertNode(int data, int position)
{
```

⇧ SCROLL TO TOP   node */

```
    Node *temp = new Node;
temp->datadata = data;
temp->next = NULL;
    /* confirm if the position to insert is first or the list is empty */
if ((position == 1) || (head == NULL))
    {
        // set up the new node pointing to head
        // because list may not be empty
temp->next = head;
        // Now point head to the first node
head = temp;
return;
    }
else
    {
   /* 2. traverse to the required position */
    Node *t = head;
intcurrPos = 2;
while ((currPos< position) && (t->next != NULL))
        {
            tt = t->next;
currPos++;
        }
 /* 3. now we are at the required location */
        /* 4 first set up the pointer for the new node */
temp->next = t->next;
        /* 5 now set up the previous node pointer */
t->next = temp;
    }
}
```

22) How can we delete the first node from the singly linked list?

To delete the first node from the singly linked list, we need to follow below steps:

- Copy the first node address to some temporary variable.
- Cl ad to the second node of the linked list.

⇧ SCROLL TO TOP

nnection of the first node to the second node.

- Delete temp and free up the memory occupied by the first node.

23) How can we delete any specific node from the linked list?

If a node that we want to delete is a root node, we can delete it easily. To delete a middle node, we must have a pointer pointing to that node which is present before the node that we want to delete. So if the position is non-zero, then we run a position loop and get a pointer to the previous node.

The steps given below are used to delete the node from the list at the specified position:

- Set up the head to point to that node which the head is pointing.
- Traverse the list to the desired position or till the end of the list (whichever comes first)
- We need to point the previous node to the next node.

24) How can we reverse a singly linked list?

To reverse the singly linked list, we are required to iterate through the list. We need to reverse the link at each step like after the first iteration, the head will point to null, and the next element will point to the head. At the end of traversal when we reach the tail of the linked list, the tail will start pointing to the second last element and will become a new head. It is because we can further traverse through all the elements from that particular node.

The steps given below can be used to reverse a singly linked list:

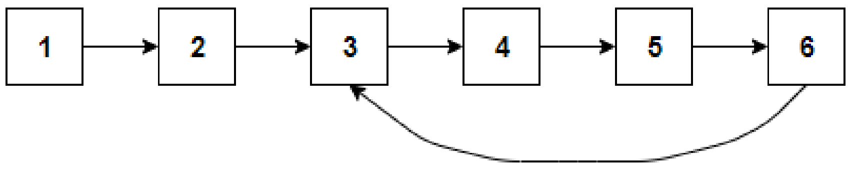
- First, we need to set a pointer (*current) pointing to the first node (i.e., current=head).
- Move ahead until current! =null (till the end of the list).
- Set another pointer (*next) pointing to the next node (i.e. next=current->next>next=result
- Keep reference of *next in a temporary variable (*result) i.e. current->next=result
- Exchange the result value with current, i.e., result=current
- And then swap the current value with next, I.e., current=next.
- Return result and repeat from step 2.

A linked list can also be reversed by using recursion which eliminates the use of a temporary variable.

25) How can we remove loops in a linked list? What are the functions of fast and slow pointers?

⇧ SCROLL TO TOP

The main concept to detect and remove a loop in a linked list is to use two pointers (one slow pointer and a fast pointer). The slow pointer traverses single node at a time, whereas the fast pointer traverses twice as fast as the first one. If the linked list contains loop in it, the fast and slow pointer will be at the same node. On the other hand, if the list doesn't consist loop in it, obviously the fast pointer will reach the end before the slow pointer does. A loop is detected if these two pointers ever met. If the loop is detected, the start of the loop can help us remove the detected loop in the linked list. It is called **Floyd's Cycle-Finding Algorithm**. The given diagram shows how a loop looks like in a linked list:



26) What will you prefer for traversing through a list of elements between singly and doubly linked lists?

Double linked lists need more space for each node in comparison to the singly linked list. In a doubly linked list, the elementary operations such as insertion and deletion are more expensive, but they are easier to manipulate because they provide quick and easy sequential access to the list in both the directions. But, doubly linked lists cannot be used as persistent data structures. Hence, the doubly linked list would be a better choice for traversing through a list of the node.

27) Where will be the free node available while inserting a new node in a linked list?

If a new node is inserted in a linked list, the free node is found in **Avail List**.

28) For which header list, the last node contains the null pointer?

For grounded header list, you will find the last node containing the null pointer.

29) How can you traverse a linked list in Java?

There are several ways to traverse a linked list in Java, e.g., we can use traditional for, while, or do-while loops and go through the linked list until we reach at the end of the linked list. Alternatively, we can use enhanced for loop of Java 1.5 or iterator to traverse through a linked list in Java. From JDK 8 onwards, we have an option to use **java.util.stream.Stream** for traversing a linked list.

⇧ SCROLL TO TOP

30) How to calculate the length of a singly linked list in Java?

We can iterate over the linked list and keep a count of nodes until we reach the end of the linked list where the next node will be null. The value of the counter is considered as the length of the linked list.

31) How can someone display singly linked list from first to last?

One must follow these steps to display singly linked list from first to last:

- Create a linked list using create().
- The address stored inside the global variable "start" cannot be changed, so one must declare one temporary variable "temp" of type node.
- To traverse from start to end, one should assign the address of starting node in the pointer variable, i.e., temp.

```
struct node *temp;  //Declare temp variable
temp = start;       //Assign Starting Address to temp
```

If the temp is NULL, then it means that the last node is accessed.

```
while(temp!=NULL)
{
printf("%d",temp->data);
temptemp=temp->next;
}
```

32) Mention some drawbacks of the linked list.

Some of the important drawbacks of the linked list are given below:

- Random access is not allowed. We need to access elements sequentially starting from the first node. So we cannot perform a binary search with linked lists.
- More memory space for a pointer is required with each element of the list.
- Slow O(n) index access, since accessing linked list by index means we have to loop over the list recursively.
- Poor locality, the memory used for the linked list is scattered around in a mess.

⇧ SCROLL TO TOP

ackage that is used for Linked List class in Java.

The package that is used for Linked list in Java is **java.util**.

34) Mention some interfaces implemented by Linked List in Java.

Some of the main interfaces implemented by Java Linked Lists are:

- Serializable
- Queue
- List
- Cloneable
- Collection
- Deque
- Iterable

35) How can we find the sum of two linked lists using Stack in Java?

To calculate the sum of the linked lists, we calculate the sum of values held at nodes in the same position. **For example**, we add values at first node on both the linked list to find the value of the first node of the resultant linked list. If the length of both linked lists is not equal, then we only add elements from the shorter linked list and copy values for remaining nodes from the long list.

Interview Tips	Job/HR Interview Questions
JavaScript Interview Questions	jQuery Interview Questions
Java Basics Interview Questions	Java OOPs Interview Questions
Servlet Interview Questions	JSP Interview Questions
Spring Interview Questions	Hibernate Interview Questions
PL/SQL Interview Questions	SQL Interview Questions

⇧ SCROLL TO TOP