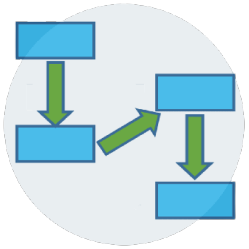
Linked List

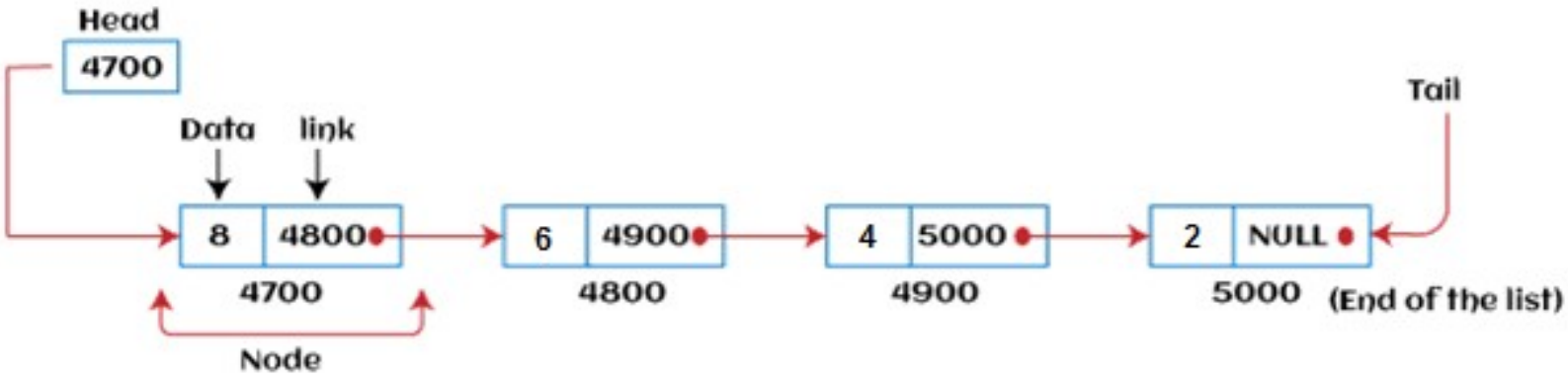
Tushar B. Kute,
http://tusharkute.com



Linked List

- Linked list is a linear data structure that includes a series of connected nodes.
- Linked list can be defined as the nodes that are randomly stored in the memory.
- A node in the linked list contains two parts, i.e., first is the data part and second is the address part.
- The last node of the list contains a pointer to the null.
- After array, linked list is the second most used data structure.
- In a linked list, every link contains a connection to another link.

Linked List


Head
4700
Data
link
8
4800
4700
Node
6
4900
4800
4
5000
4900
Tail
2
NULL
5000
(End of the list)

Linked List: Why?

- Linked list is a data structure that overcomes the limitations of arrays. Let's first see some of the limitations of arrays -
  - The size of the array must be known in advance before using it in the program.
  - Increasing the size of the array is a time taking process. It is almost impossible to expand the size of the array at run time.
  - All the elements in the array need to be contiguously stored in the memory. Inserting an element in the array needs shifting of all its predecessors.

Linked List

- Linked list is useful because -
  - It allocates the memory dynamically. All the nodes of the linked list are non-contiguously stored in the memory and linked together with the help of pointers.
  - In linked list, size is no longer a problem since we do not need to define its size at the time of declaration.
  - List grows as per the program's demand and limited to the available memory space.

Linked List

- It is simple to declare an array, as it is of single type, while the declaration of linked list is a bit more typical than array. Linked list contains two parts, and both are of different types, i.e., one is the simple variable, while another is the pointer variable.
- We can declare the linked list by using the user-defined data type structure.
- The declaration of linked list is given as follows -

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

Linked List: Advantages

- Dynamic data structure - The size of the linked list may vary according to the requirements. Linked list does not have a fixed size.
- Insertion and deletion - Unlike arrays, insertion, and deletion in linked list is easier. Array elements are stored in the consecutive location, whereas the elements in the linked list are stored at a random location.
- To insert or delete an element in an array, we have to shift the elements for creating the space. Whereas, in linked list, instead of shifting, we just have to update the address of the pointer of the node.

Linked List: Advantages

- Memory efficient - The size of a linked list can grow or shrink according to the requirements, so memory consumption in linked list is efficient.
- Implementation - We can implement both stacks and queues using linked list.

Linked List: Disadvantages

- Memory usage - In linked list, node occupies more memory than array. Each node of the linked list occupies two types of variables, i.e., one is a simple variable, and another one is the pointer variable.
- Traversal - Traversal is not easy in the linked list. If we have to access an element in the linked list, we cannot access it randomly, while in case of array we can randomly access it by index. For example, if we want to access the 3rd node, then we need to traverse all the nodes before it. So, the time required to access a particular node is large.
- Reverse traversing - Backtracking or reverse traversing is difficult in a linked list. In a doubly-linked list, it is easier but requires more memory to store the back pointer.

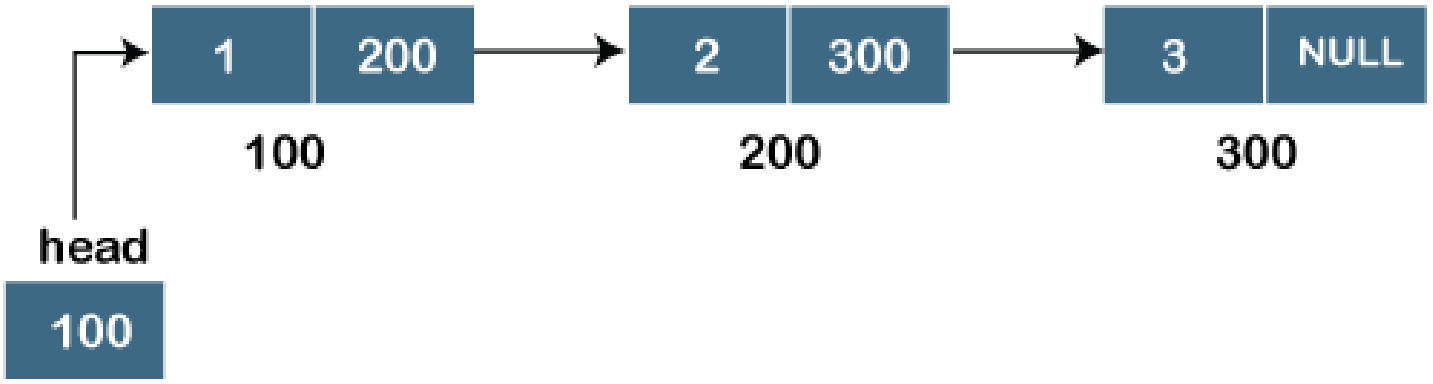
Linked List: Types

- Singly Linked list
- Doubly Linked list
- Circular Linked list
- Doubly Circular Linked list

Singly Linked List

- It is the commonly used linked list in programs. If we are talking about the linked list, it means it is a singly linked list.
- The singly linked list is a data structure that contains two parts, i.e., one is the data part, and the other one is the address part, which contains the address of the next or the successor node.
- The address part in a node is also known as a pointer.

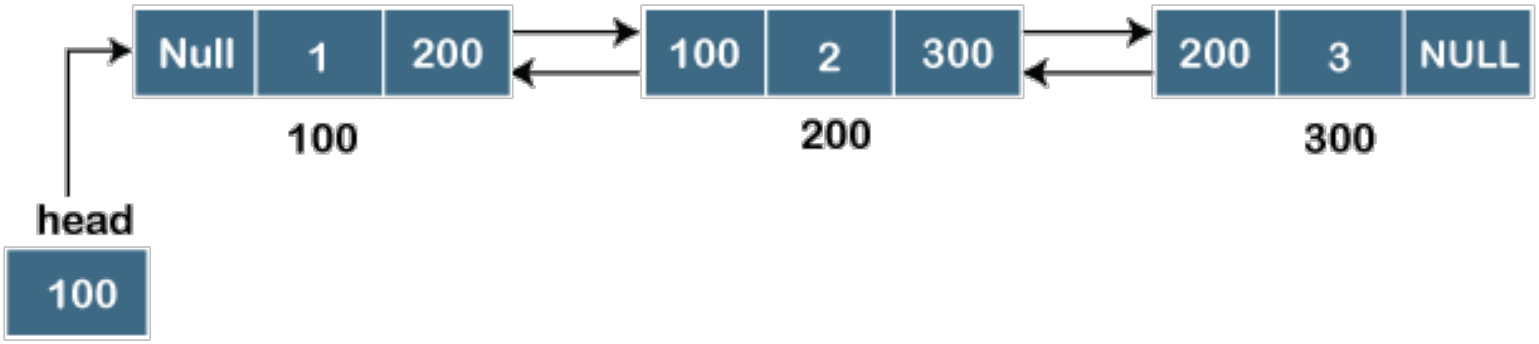
Singly Linked List



Doubly Linked List

- As the name suggests, the doubly linked list contains two pointers.
- We can define the doubly linked list as a linear data structure with three parts: the data part and the other two address part.
- In other words, a doubly linked list is a list that has three parts in a single node, includes one data part, a pointer to its previous node, and a pointer to the next node.

Doubly Linked List



Doubly Linked List

- the node in a doubly-linked list has two address parts; one part stores the address of the next while the other part of the node stores the previous node's address.
- The initial node in the doubly linked list has the NULL value in the address part, which provides the address of the previous node.
- Representation of the node in a doubly linked list

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

Circular Linked List

- A circular linked list is a variation of a singly linked list. The only difference between the singly linked list and a circular linked list is that the last node does not point to any node in a singly linked list, so its link part contains a NULL value.
- On the other hand, the circular linked list is a list in which the last node connects to the first node, so the link part of the last node holds the first node's address.
- The circular linked list has no starting and ending node.

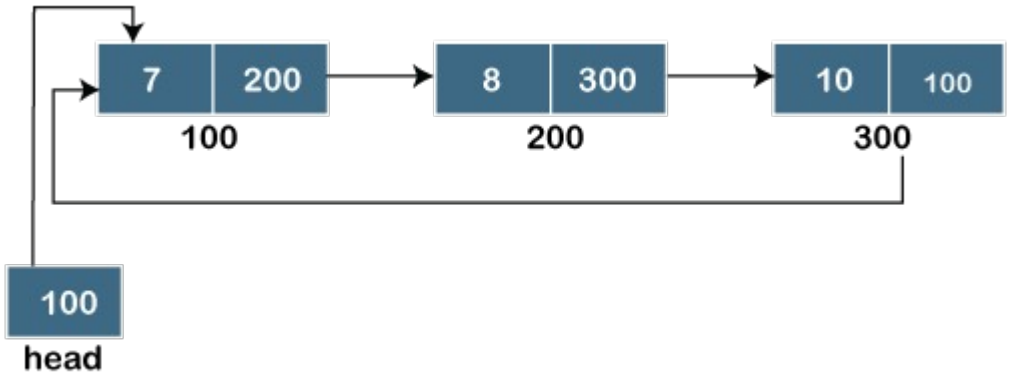
Circular Linked List

- We can traverse in any direction, i.e., either backward or forward. The diagrammatic representation of the circular linked list is shown below:

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

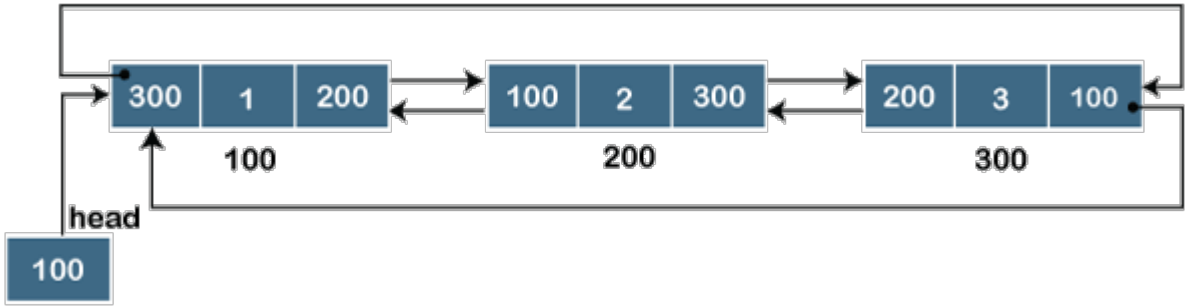
Circular Linked List

- A circular linked list is a sequence of elements in which each node has a link to the next node, and the last node is having a link to the first node.
- The representation of the circular linked list will be similar to the singly linked list, as shown below:



Circular Doubly Linked List

- The doubly circular linked list has the features of both the circular linked list and doubly linked list.



Circular Doubly Linked List

- The representation of the doubly circular linked list in which the last node is attached to the first node and thus creates a circle.
- It is a doubly linked list also because each node holds the address of the previous node also.
- The main difference between the doubly linked list and doubly circular linked list is that the doubly circular linked list does not contain the NULL value in the previous field of the node.
- As the doubly circular linked contains three parts, i.e., two address parts and one data part so its representation is similar to the doubly linked list.

Circular Doubly Linked List

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

Stack using Linked List

- Create a Node class with two fields: data and next.
- Create a Stack class with two fields: head and top.
- The head pointer points to the top of the stack, and the top pointer points to the next element in the stack.
- To push an element onto the stack, we create a new node with the given data and set the next pointer of the new node to the current head of the stack. Then, we set the head of the stack to the new node.
- To pop an element from the stack, we return the data of the current head node and set the head of the stack to the next node in the stack.
- To check if the stack is empty, we check if the head pointer is null.

Queue using Linked List

- To implement a queue using a linked list, you can use the following steps:
  - Create a linked list with two pointers, head and tail.
  - The head pointer will point to the first element in the queue, and the tail pointer will point to the last element in the queue.
  - To enqueue an element, add it to the end of the linked list and update the tail pointer.
  - To dequeue an element, remove it from the front of the linked list and update the head pointer.

Thank you

@mitu_skillologies

@mITuSkillologies

@mitu_group

@mitu-skillologies

@MITUSkillologies

kaggle

@mituskillologies

Web Resources
https://mitu.co.in
http://tusharkute.com

@mituskillologies

contact@mitu.co.in

tushar@tusharkute.com