

Data Structures and Algorithms

Prepared by Prof. Shereen Taie

IT251
Information Technology Department

Lecture 6



Data Structures and Algorithms

Lecture 6

IT251

Information Technology Department

Prepared by
Prof. Shereen Taie

Learning Objectives:

- Representation of Stack (or) Implementation of stack
- Stack using linked list
- Linked list implementation of stack
- Adding a node to the stack (Push operation)
- Deleting a node from the stack (POP operation)
- Display the nodes (Traversing)
- QUEUE
- Operations on QUEUE

DATA STRUCTURES

Chapter

03

Representation of Stack (or) Implementation of stack

The stack should be represented in two ways:

1. Stack using array
2. Stack using linked list

2. Stack using linked list:

- We can represent a stack as a linked list.
- In a stack push and pop operations are performed at one end called top.
- We can perform similar operations **at one end of list using top pointer.**

The linked stack looks as shown in figure

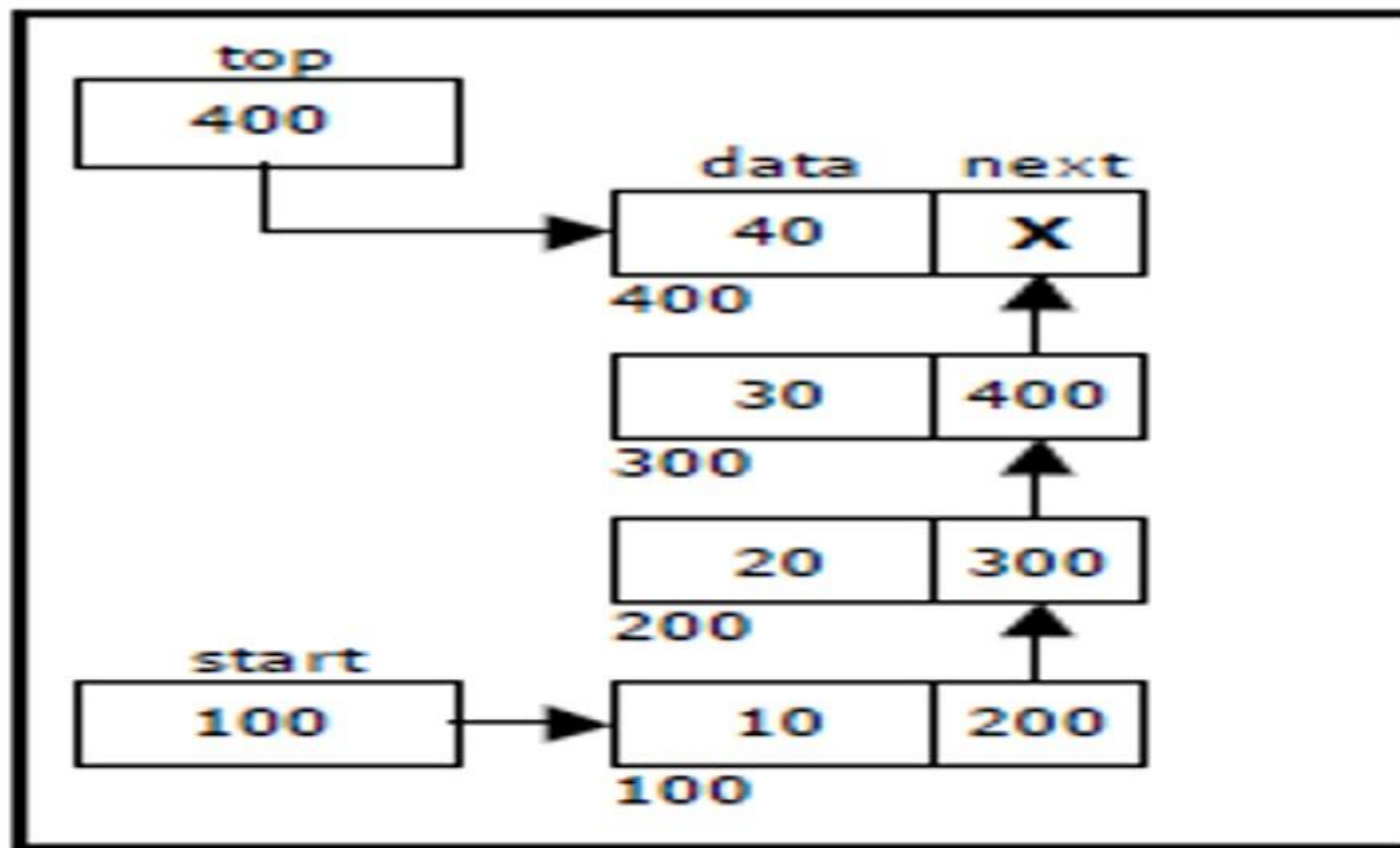


Figure Linked stack representation

Chapter
03

Linked list implementation of stack



Linked list implementation of stack:

- Instead of using array, we can also use linked list to implement stack.
- Linked list **allocates the memory dynamically**.

Linked list implementation of stack:

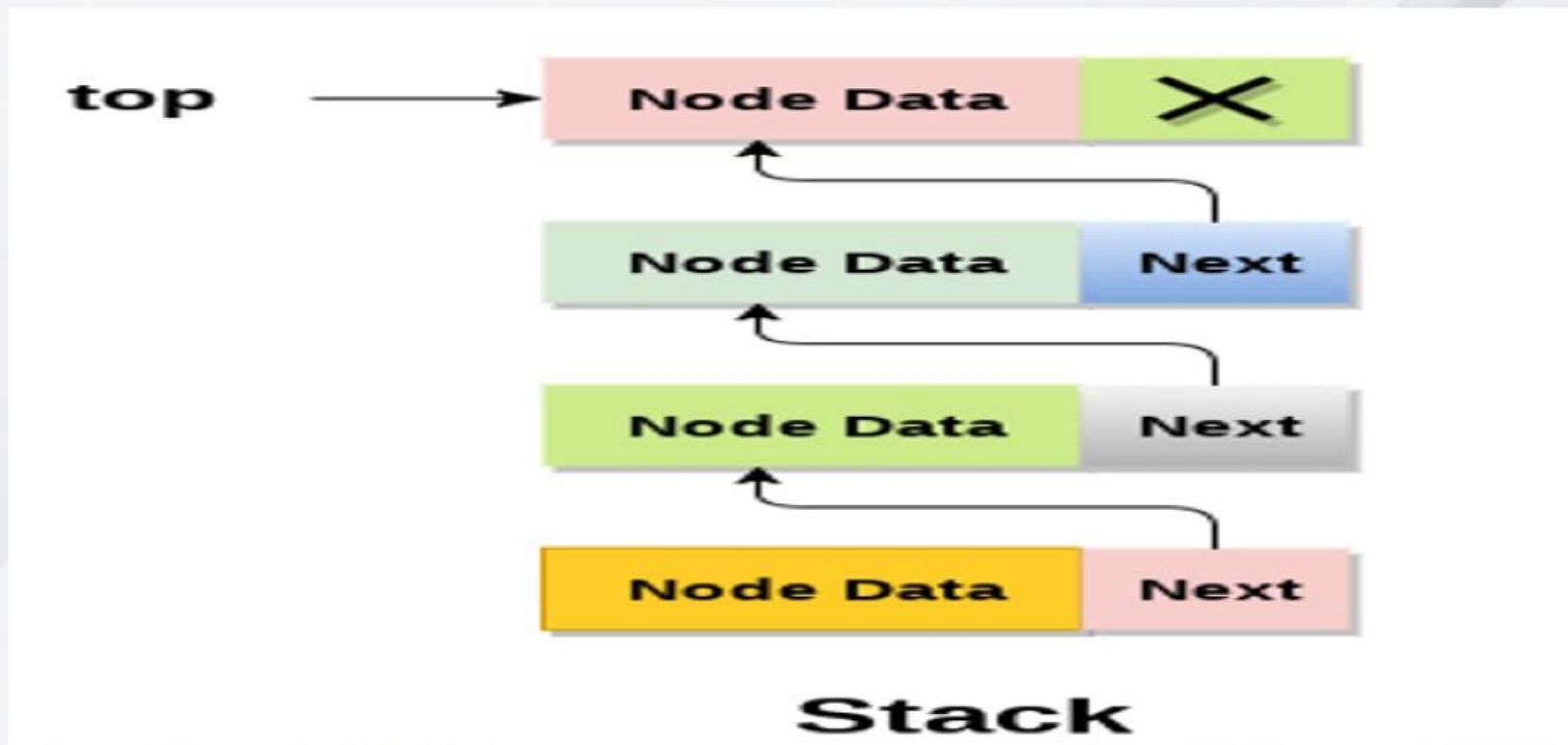
- However, time complexity in both the scenario is same for all the operations i.e. push, pop.
- In linked list implementation of stack, **the nodes are maintained non-contiguously in the memory.**

Linked list implementation of stack:

- Each node contains a pointer to its immediate successor node in the stack.
- Stack is said to be overflowed in Linked list implementation if the space left in the memory heap is not enough to create a node.

Linked list implementation of stack:

- The top most node in the stack always contains null in its address field.



The linked stack looks as shown in figure

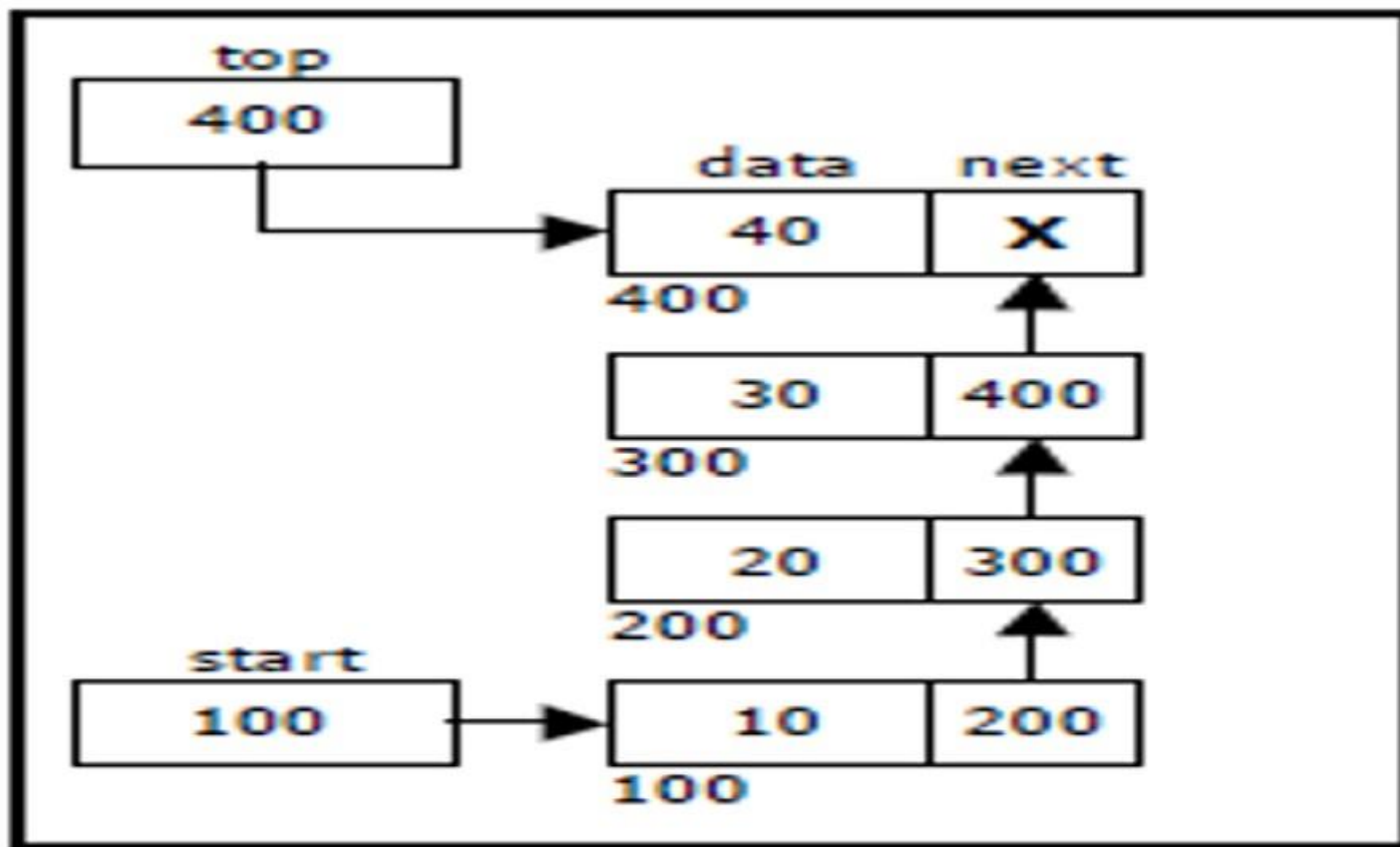


Figure Linked stack representation

Adding a node to the stack (Push operation)

Adding a node to the stack (Push operation):

- Adding a node to the stack is referred to as **push** operation.
- Pushing an element to a stack in linked list implementation **is different from that of an array implementation.**
- In order to push an element onto the stack, the following steps are involved.

Adding a node to the stack (Push operation):

1. Create a node first and allocate memory to it.
2. If the list is empty then the item is to be pushed as the start node of the list.

This includes:

- assigning value to the **data part** of the node, and
- assign null to the **address part** of the node.

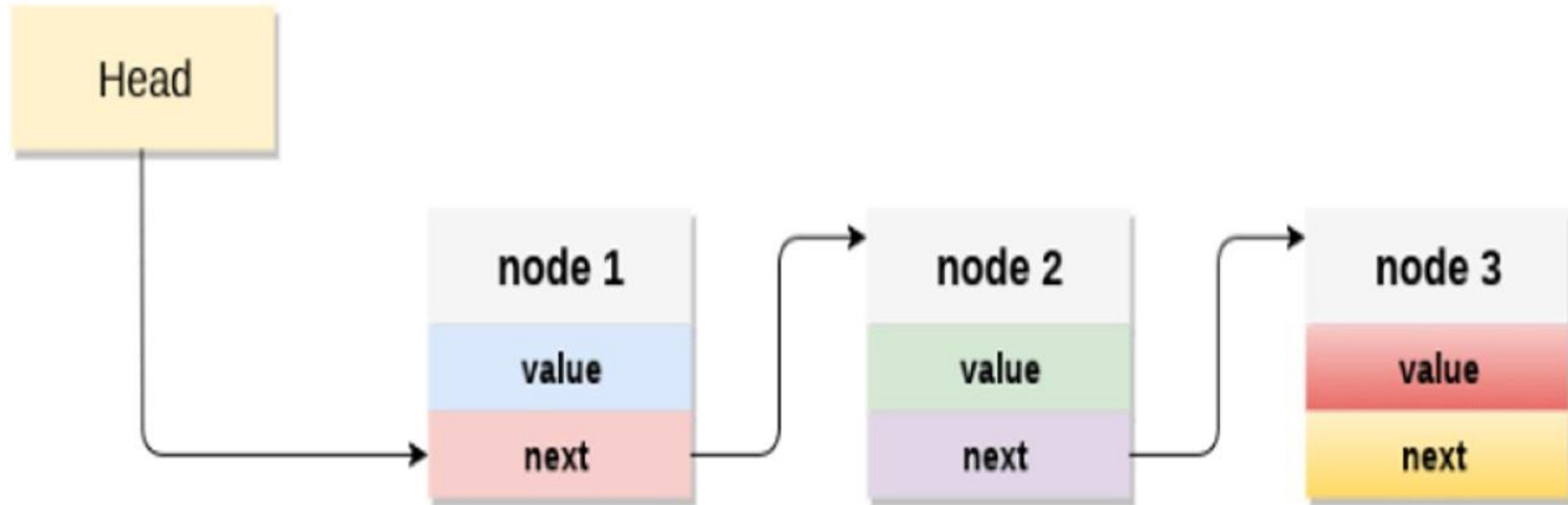
Adding a node to the stack (Push operation):

3. If there are some nodes in the list already, then we have to **add the new element in the beginning of the list** (to not violate the property of the stack).

For this purpose,

- assign the address of the starting element to the address field of the new node, and
- make the new node, the starting node of the list.

Time Complexity : $O(1)$



Deleting a node from the stack (POP operation)

Deleting a node from the stack (POP operation):

- Deleting a node from the top of stack is referred to as **pop** operation.
- Deleting a node from the linked list implementation of stack is different from that in the array implementation.

Deleting a node from the stack (POP operation):

In order to pop an element from the stack, we need to follow the following steps :

- Check for the underflow condition
- Adjust the head pointer accordingly
- **Time Complexity : $O(1)$**

Deleting a node from the stack (POP operation):

Check for the underflow condition:

- The underflow condition occurs when we try to pop from an already empty stack. The stack will be empty if the head pointer of the list points to null.

Deleting a node from the stack (POP operation):

Adjust the head pointer accordingly:

- In stack, the elements are popped only from one end, therefore, the value stored in the head pointer must be deleted and the node must be freed. The next node of the head node now becomes the head node.

Display the nodes (Traversing)

Display the nodes (Traversing):

Displaying all the nodes of a stack needs traversing all the nodes of the linked list organized in the form of stack.

Traversing عبور: It is used to access each data item exactly once so that it can be processed.

Display the nodes (Traversing):

To Display the nodes follow the following steps:

1. Copy the head pointer into a temporary pointer.
2. Move the temporary pointer through all the nodes of the list and print the value field attached to every node.

- **Time Complexity : $O(n)$**

Chapter
03

QUEUE



QUEUE:

- A queue is linear data structure and collection of elements.
- A queue is another special kind of list, where items are inserted at one end called the **rear** and **deleted at the other end called the front**.
- **The principle of queue** is a **“FIFO”** or **“First-in-first-out”**.

QUEUE:

- Queue is an abstract data structure.
- A queue is a useful data structure in programming.
- **It is similar to the ticket queue outside a cinema hall, where the first person entering the queue is the first person who gets the ticket.**

Real-world examples of queue

- A real-world example of queue can be a single-lane one-way road, where the vehicle enters first, exits first.



- More real-world examples can be seen as queues at the ticket windows and bus-stops and our college library.



Chapter

03

Operations on QUEUE



QUEUE:

- The operations for a queue are analogues to those for a stack; the difference is that the insertions go at the end of the list, rather than the beginning.
- A queue is an object or more specifically an abstract data structure (ADT) that allows the following operations:
 - **Enqueue or insertion**
 - **Deque or deletion**

QUEUE:

Operations on QUEUE:

A queue is an object or more specifically an abstract data structure (ADT) that allows the following operations:

- **Enqueue or insertion:**
which inserts an element at the end of the queue.
- **Dequeue or deletion:**
which deletes an element at the start of the queue.

QUEUE:

Queue operations work as follows:

1. Two pointers called FRONT and REAR are used to keep track of the first and last elements in the queue.
2. When initializing the queue, we set the value of FRONT and REAR to 0.

QUEUE:

Queue operations work as follows:

3. On enqueueing an element, we increase the value of REAR index and place the new element in the position pointed to by REAR.
4. On dequeueing an element, we return the value pointed to by FRONT and increase the FRONT index.

QUEUE:

Queue operations work as follows:

5. Before enqueueing, we check if queue is already full.
6. Before dequeuing, we check if queue is already empty.

QUEUE:

Queue operations work as follows:

7. When enqueueing the first element, we set the value of FRONT to 1.
8. When dequeuing the last element, we reset the values of FRONT and REAR to 0.

Thank You