Advanced Algorithms & Data Structures

AIM OF THE SESSION

To familiarize students with the basic concept of stack operations using Linked Lists

INSTRUCTIONAL OBJECTIVES

This Session is designed to:

1. Demonstrate how to implement a stack using a linked list.
2. Discuss the advantages of using a linked list for a stack, such as dynamic memory allocation.
3. Describe the fundamental operations of a stack (push, pop, and peek).
4. Analyze the time complexity of stack operations in a linked list implementation..

.

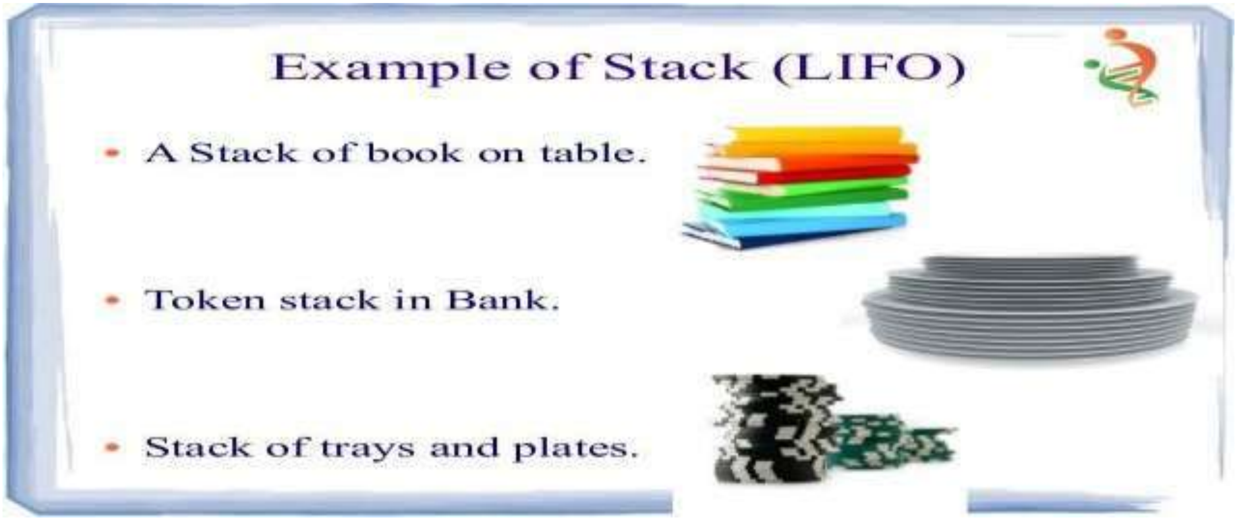
LEARNING OUTCOMES

At the end of this session, you should be able to:

- Evaluate the time complexities of the different Algorithmic solutions for various types of Problems.
- Write the notations for calculating the running time complexity of different algorithms

Session -Introduction

➢Stack is a linear data structure which implements LIFO (Last In First Out) pattern where insertions and deletions are done at only one side i.e., top of the stack.



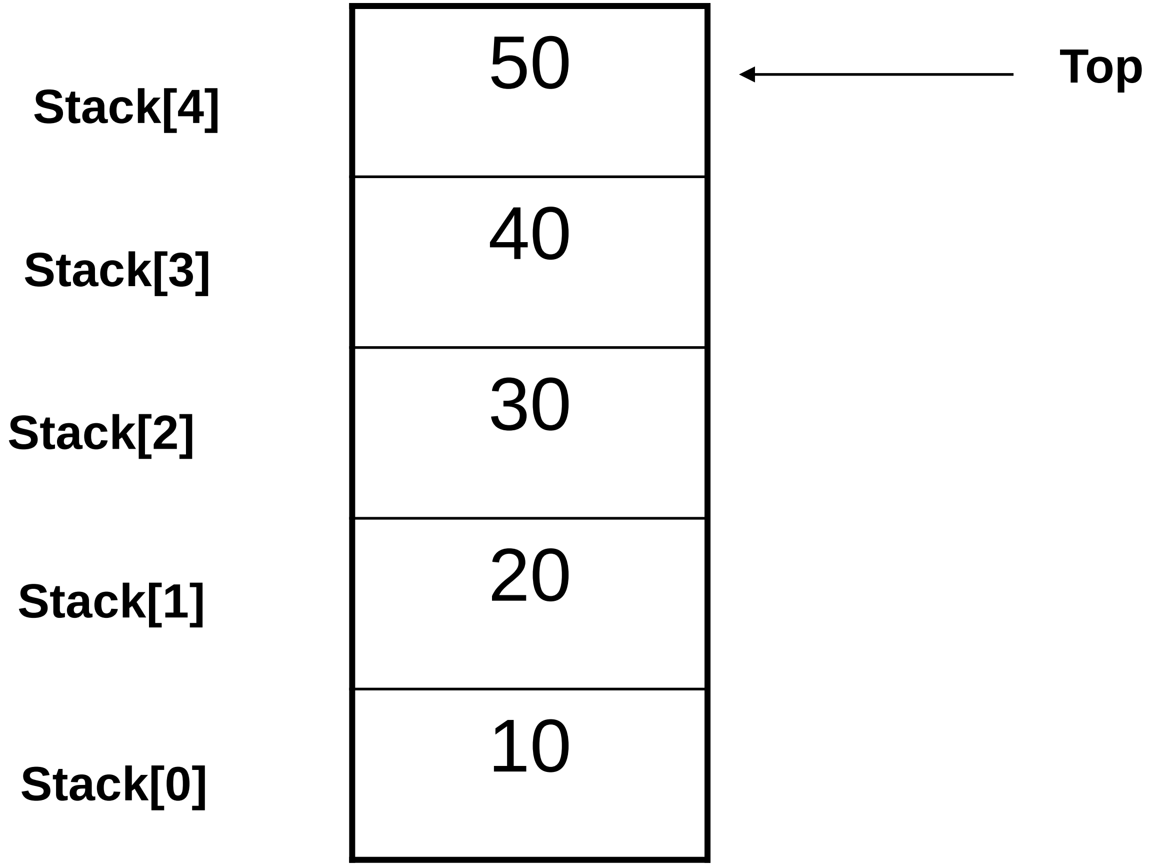
stack

- A ***stack*** is a linear list in which items may be added or removed only at one end, called the top of the stack.
- The last item to be added to a stack is the first item to be removed. Accordingly, stacks are also called ***Last-In-First-Out*** ***(LIFO)*** lists or ***First-In-Last-Out*** ***(FILO)*** lists.

- A stack may be pictorially represented as follows:

10	20	30	40	50
Stack[0]	**Stack[1]**	**Stack[2]**	**Stack[3]**	**Stack[4]**

Top


50
Top
Stack[4]
40
Stack[3]
30
Stack[2]
20
Stack[1]
10
Stack[0]

- The basic operations associated with stacks are:
  - ***"Push"*** is the term used to insert an element into a stack.
  - ***"Pop"*** is the term used to delete an element from a stack.
  - ***"Peek"*** is an operation that returns the value of the topmost element of the stack.

Examples:

1. A stack of dishes
2. A stack of coins
3. A stack of folded towels
4. A stack of bills
5. A railway system for shunting cars

- The stack data structure can be implemented in two different following ways:
  - Stack Implementation using Arrays [Static Implementation]
  - Stack Implementation using Linked List [Dynamic Implementation]

Array representation of a stack:

- Stack may be represented in a computer in various ways, usually by means of a *one-dimensional array*.
- ***Top*** is a variable, which points top most element of the stack.
- Initially when the stack is empty, ***Top*** has a value of ***"–1"*** and when the stack contains a single element, ***Top*** has a value of ***"0"*** and when the stack contains two elements, ***Top*** has a value of ***"1"*** and so on.

- Each time a new element is inserted in the stack, the top is incremented by *“one”* before the element is placed on the stack.

[Top=Top+1;]

- The **Top** is decremented by *“one”* each time a deletion is made from the stack.

[Top = Top – 1;]

- A variable ***MaxSize*,** which gives the maximum number of elements that can be held by the stack.

- Following is the condition is used to check whether the Stack is Full or not:

(Top == MaxSize-1)

- Following is the condition is used to check whether the Stack is Empty or not:

(Top==-1)

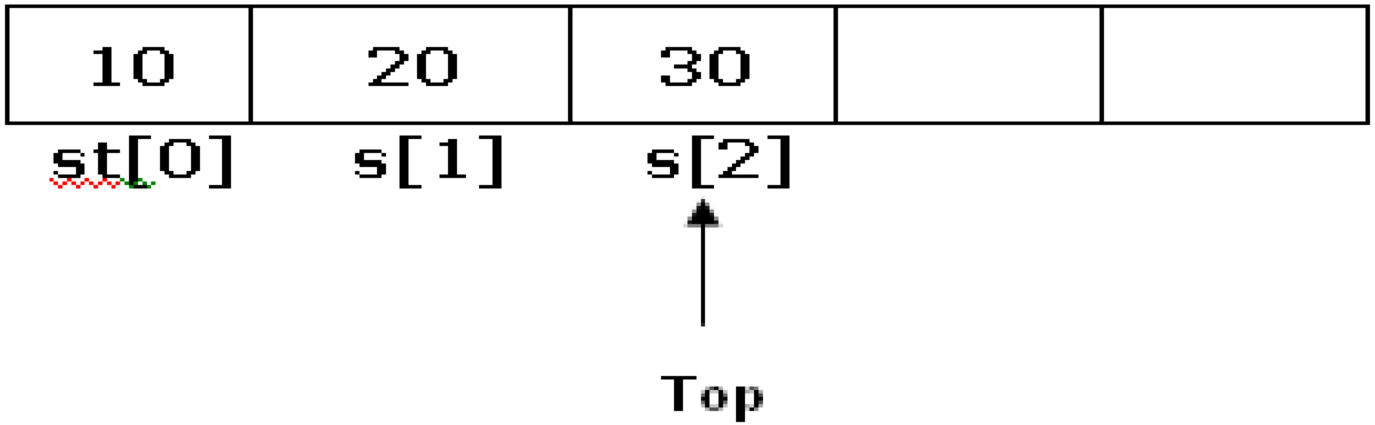


Fig: Array representation of stack.

- The right most occupied element of the stack represents its top element. Since Top=2, the stack has three elements 10, 20, 30 and there in room for two elements in the stack.

Algorithm for push:

```
Step.1:[Check whether the stack is full ]
        If Top >= Max-1 then
      Print 'Stack Overflow'
      Return
Step.2:[Increment pointer Top]
         Top ← Top +1.
Step.3:[Insert ITEM at top of the stack]
          Stack[top] ← ITEM
Step.4: Return.
```

Algorithm for pop:

Step.1:[Checking whether the stack is empty]
If Top = -1 **then**
Print ‘Stack Underflow’
Return

Step.2:[retrieving the topmost element from stack into a temporary variable K]
k ← Stack[Top]

Step.3:[Decrement the pointer Top]
Top ← Top – 1

Step.4: return(k).

Implementation of push operations:

```
Void push()
{
    int ele;
    if(Top == (MaxSize-1))
    {
    printf("Stack Is Overflow \n");
    }
    else
    {
    printf("Enter the Element to be Insert : ");
    scanf("%d",&ele);
    Top=Top+1;
    Stack[Top] = ele;
    }
}
```

Implementation of pop operations:

```
void Deletion()
{
       if(Top==-1)
       {
      printf("Stack Is Underflow \n");
       }
       else
       {
      printf("The Deleted Element is : %d \n", Stack[Top]);
      Top = Top - 1;
       }
}
```

Stack Applications:

1. A stack is useful to convert infix expression into postfix expression.
2. A stack is useful to evaluate the value of postfix expression.
3. It can be used in function calls.
4. Compiler application uses the stack in parenthesis matching.
5. A stack can also be used in string processing to evaluate reverse of a string.
6. A stack is useful in writing recursive calls
7. Redo-undo features at many places like editors, photoshop.
8. Used in many algorithms like Towers of Hanoi, Tree Traversals, Histogram problem.
9. *Backtracking (game playing, finding paths, exhaustive searching.*

Stack using an array - drawback

- If we implement the stack using an array, we need to specify the array size at the beginning(at compile time).
- We can't change the size of an array at runtime. So, it will only work for a fixed number of elements.

Solution:

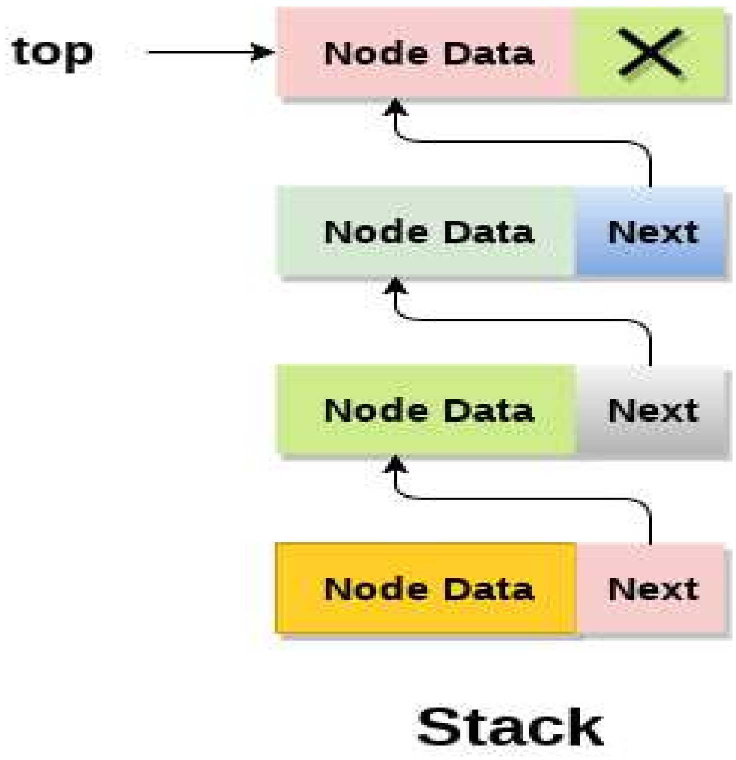
- We can implement the stack using the linked list.
- In the linked list, we can change its size at runtime.

Operations on Stack

- **push():** It inserts an element to the top of the stack. It takes O(1) time, as each node is inserted at the head/top of the linked list.
- **pop():** It removes an element from the top of the stack. It takes O(1) time, as the top always points to the newly inserted node.
- **Display():** It displays the elements of the stack.

Node structure

```
struct node
    {
     int data;
     struct node * next;
     };
     node *top=NULL;
```



PUSH OPERATION ON STACK

ALGORITHM:

Step 1 - Create a **newNode** with given value.

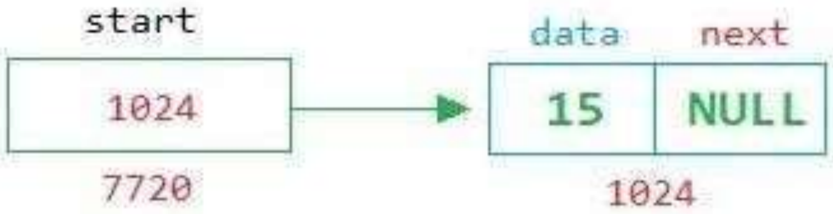
Step 2 - Check whether stack is **Empty** (**top** == **NULL**)

Step 3 - If it is **Empty**, then set **newNode** → **next** = **NULL**.

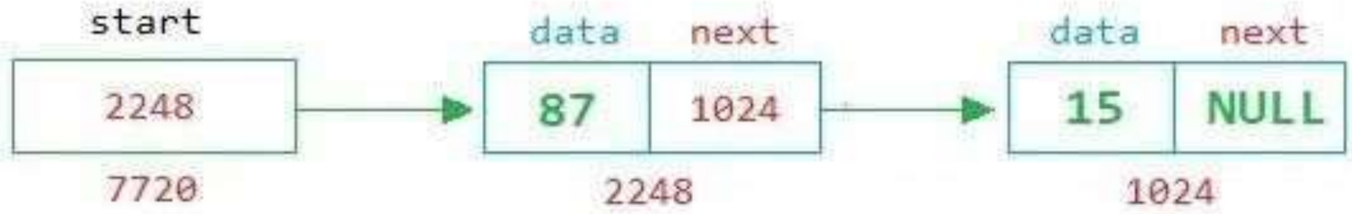
Step 4 - If it is **Not Empty**, then set **newNode** → **next** = **top**.

Step 5 - Finally, set **top** = **newNode**.

Push 15



Push 87



POP OPERATION ON STACK

ALGORITHM:

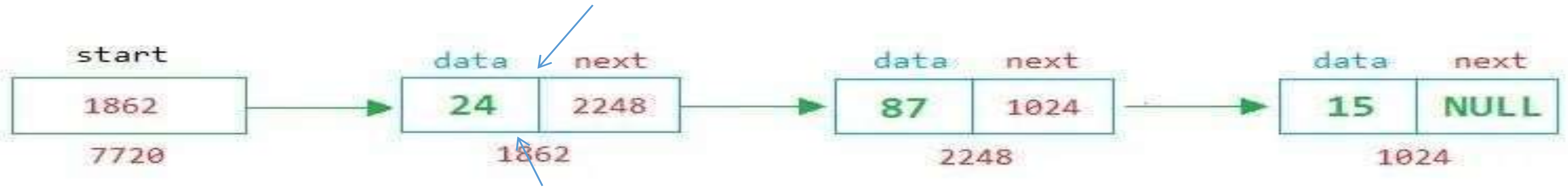
Step 1 - Check whether **stack** is **Empty** (**top == NULL**).

Step 2 - If it is **Empty**, then display **"Stack is Empty!!! Deletion is not possible!!!"** and terminate the function

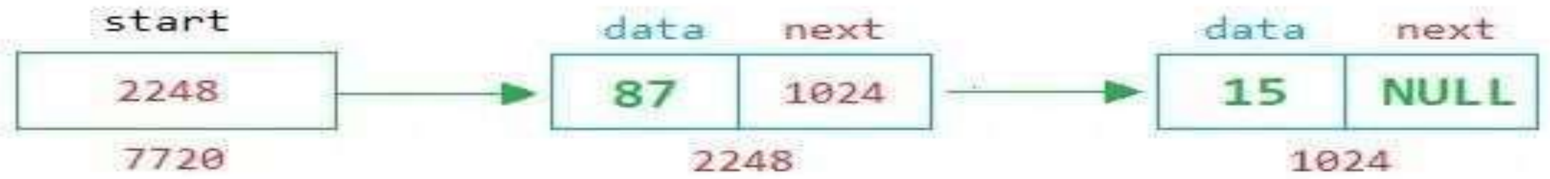
Step 3 - If it is **Not Empty**, then define a **Node** pointer '**temp**' and set it to '**top**'.

Step 4 - Then set '**top = top → next**'.

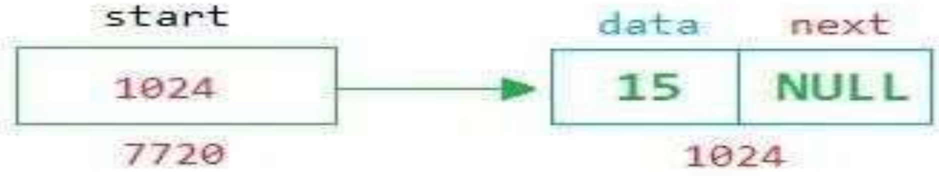
Step 5 - Finally, delete '**temp**'. (**free(temp)**).



Pop (Remove 24)



Pop (Remove 87)



Display

Algorithm:

Step 1 - Check whether stack is **Empty** (**top** == **NULL**).

Step 2 - If it is **Empty**, then display **'Stack is Empty!!!'** and terminate the function.

Step 3 - If it is **Not Empty**, then define a Node pointer **'temp'** and initialize with **top**.

Step 4 - Display '**temp → data** --->' and move it to the next node. Repeat the same until **temp** reaches to the first node in the stack. (**temp → next** != **NULL**).

Step 5 - Finally! Display '**temp → data** ---> **NULL**'.

SUMMARY

stack Implementation, a stack contains a top pointer. which is the “head” of the stack where pushing and popping items happens at the head of the list. The first node has a null in the link field and second node-link has the first node address in the link field and so on and the last node address is in the “top” pointer.

The main advantage of using a linked list over arrays is that it is possible to implement a stack that can shrink or grow as much as needed. Using an array will put a restriction on the maximum capacity of the array which can lead to stack overflow. Here each new node will be dynamically allocated. so overflow is not possible

SELF-ASSESSMENT QUESTIONS

1. What is the best case time complexity of deleting a node in a Singly Linked list?

a) O (n)
b) O (n^2)
c) O (nlogn)
d) O (1)

2 What does ‘stack overflow’ refer to?

a) accessing item from an undefined stack

b) adding items to a full stack

c) removing items from an empty stack

d) index out of bounds exception

TERMINAL QUESTIONS

1. Compare Array vs Linked list stack implementations.
2. Implement push operation on stack using linked list.
3. Implement pop operation on stack using linked list.
4. Analyze the time complexity of push and pop operation on stack using linked list.

Reference Books:

1. Mark Allen Weiss, Data Structures and Algorithm Analysis in C, 2010 , Second Edition, PearsonEducation.
2. 2. Ellis Horowitz, Fundamentals of Data Structures in C: Second Edition, 2015
3. A.V.Aho, J. E. Hopcroft, and J. D. Ullman, “Data Structures And Algorithms”, Pearson Education, First Edition Reprint2003.

Sites and Web links:

1. https://nptel.ac.in/courses/106102064
2. https://in.udacity.com/course/intro-to-algorithms--cs215
3. https://www.coursera.org/learn/data-structures?action=enroll

THANK YOU