GIET INSTITUTIONS
ANDHRA PRADESH, INDIA

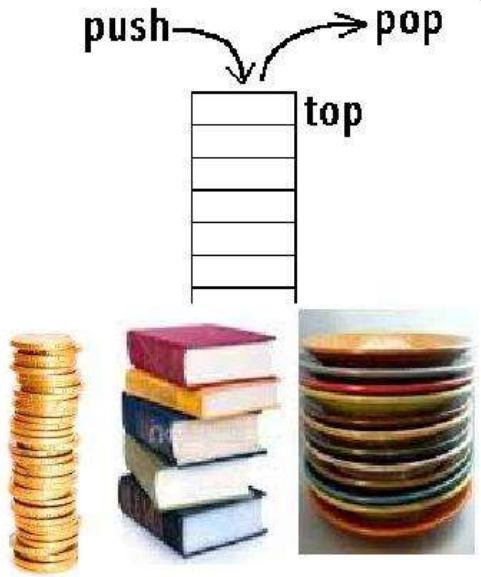
Stacks

Lecture Details:

Stacks

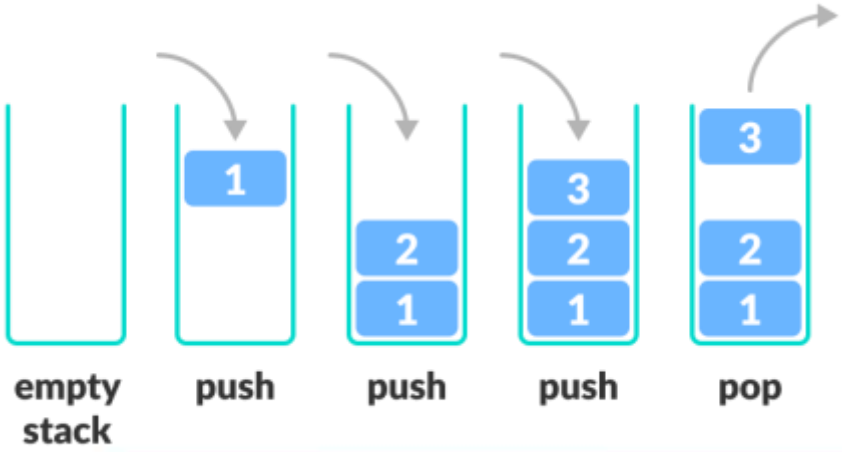
Stack

- A stack is a an ordered list in which elements are added or deleted from only one end of the list termed ***top of the stack***.
- Stack is the linear data structure having collection of elements which follows **LIFO (Last In First Out**) pattern i.e. element which comes last is served first.
- *Examples*: Stack of plates, Stack of coins, pile of bread slices etc.



LIFO Principle of Stack

- In programming terms, putting an item on top of the stack is called **push** and removing an item is called **pop**.



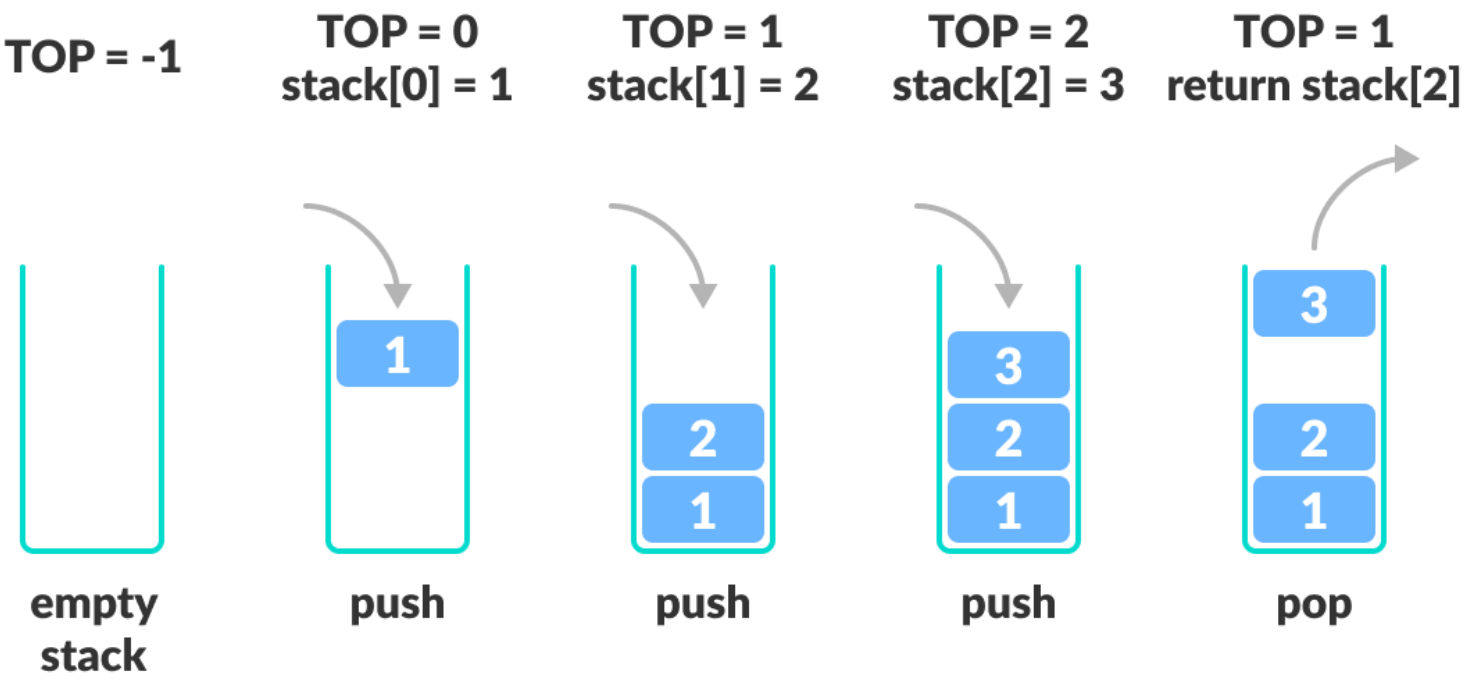
Stack Operations

- **Push**: Add an element to the top of a stack
- **Pop**: Remove an element from the top of a stack
- **IsEmpty**: Check if the stack is empty
- **IsFull**: Check if the stack is full
- **Peek**: Get the value of the top element without removing it

Working of Stack Data Structure

- A pointer called TOP is used to keep track of the top element in the stack.
- When initializing the stack, we set its value to -1 so that we can check if the stack is empty by comparing TOP == -1.
- On pushing an element, we increase the value of TOP and place the new element in the position pointed to by TOP.
- On popping an element, we return the element pointed to by TOP and reduce its value.
- Before pushing, we check if the stack is already full
- Before popping, we check if the stack is already empty

Working of Stack Data Structure



Stack Applications

- ✓ Expression conversion
- ✓ Expression evaluation
- ✓ Function calls use run time stack to store the context and return address.
- ✓ Tree traversal techniques use the concept of stacks.
- ✓ Depth first search of a graph also uses the concepts of stacks.
- ✓ Syntax analysis case of compiler uses stack.
- ✓ Interrupt processing uses stack.
- ✓ Web browser uses stack to implement “BACK” functionality [navigate] in the history.
- ✓ Parsing well formed parenthesis
- ✓ Decimal to binary conversion
- ✓ Reversing a string
- ✓ Storing function calls

Stack Implementation

- ✓ Using Arrays
- ✓ Using Linked List

Stack Implementation using Arrays

- In array implementation, the stack is formed by using the array.
- All the operations regarding the stack are performed using arrays.

Stack Implementation using Arrays

- **Adding an element onto the stack**

  Push operation : Algorithm

  Step 1 : Start

  Step 2 : if top = n-1 then stack is full

  Step 3 : top = top + 1

  Step 4 : stack [top] = item;

  Step 5 : Stop

Stack Implementation using Arrays

- **Deletion of an element from a stack**

Pop operation : Algorithm

Step 1 : Start

Step 2 : if top = -1 then stack empty

Step 3 : item := stack[top];

Step 4 : top = top - 1;

Step 5 : Stop

Stack Implementation using Arrays

- **Visiting top element of the stack**

  Peek operation : Algorithm

  Step 1 : Start

  Step 2 : if top = -1 then stack is empty

  Step 3 : item = stack[top]

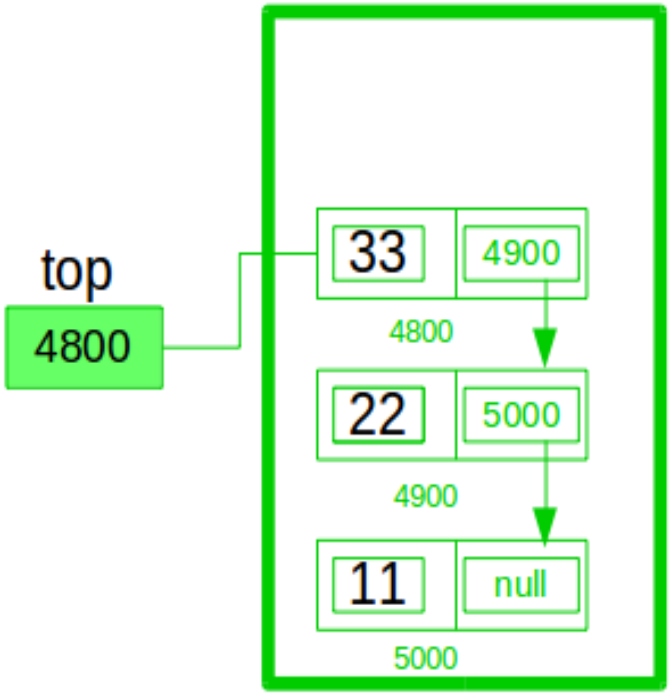
  Step 4 : return item

  Step 5 : Stop

Stack Implementation using Linked List

- In linked list implementation of a stack, every new element is inserted as '**top**' element.
- That means every newly inserted element is pointed by '**top**'. Whenever we want to remove an element from the stack, simply remove the node which is pointed by '**top**' by moving '**top**' to its previous node in the list.
- The **next** field of the first element must be always **NULL**.

Stack Implementation using Linked List



Initial Stack Having Three element
And top have address 4800

Stack Implementation using Linked List

push(value) - Inserting an element into the Stack

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.

Stack Implementation using Linked List

pop() - Deleting an Element from a Stack

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)).

Stack Implementation using Linked List

display() - Displaying stack of elements

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).

Stack Applications

Reversing a List

- To reverse a string stack can be used.
- The simple mechanism is to push all the characters of a string onto the stack and then pop all the characters from the stack and print them.
- If the input string is

- Then push all the characters onto the stack till '\0' is not encountered.

- Now if we pop each character from the stack and print we get, which is a Reverse string.

Stack Applications

Factorial Calculation

- Enter the number for which the factorial is to be computed – say n
- Push the numbers from 1 to n onto the stack.
- Initialize fact = 1
- Perform the multiplication with fact by popping the element each time.
- Store the multiplication in fact variable.
- Repeat the steps 4 and 5 for n elements.
- Finally display the factorial value.

Stack Applications

Convert Decimal to Binary

The idea of reversing a series can be used in solving classical problems such as transforming a decimal number to a binary number

- read (num)
- loop (num > 0)
  - set digit to num modulo 2
  - print (digit)
  - set number to quotient of num/2
- end loop
- It creates the binary number backward. We can solve this problem by using a stack.
- Instead of printing the binary digit as soon as it is produced, we push it into the stack.
- Then, after the number has been completely converted, simply pop the stack and print the results one digit at a time in a line.

Stack Applications

Expression Evaluation

- Stacks place a major role in evaluating an arithmetic expression. Combination of operator and operands.
- An arithmetic expression can be represented in 3 different notations.
- Infix expression : The operator comes in between the operands.
  - (a+b), (a+b)*(c-d)

  Operand1 Operator Operand2
- Postfix expression: The operator comes before the operands
  - ab+, ab+cd-*

  Operand1 Operand2 Operator
- Prefix expression : The operator comes after the operands.
  - +ab, *+ab-cd

  Operator Operand1 Operand2

Stack Applications

- The words In, pre, post specifies position of operator corresponding to the operand.
- The computer evaluates expressions in post fix form. But normally we enter the elements in " Infix" form.
- In order to evaluate the "Infix" expressions first the expression is converted into its equivalent postfix form and then evaluated the expression which is in postfix form.

Conversion from Infix to Postfix

1. Scan the infix expression from left to right.
2. a) If the scanned symbol is left parenthesis, push it onto the stack.
   b) If the scanned symbol is an operand, then place directly in the postfix expression (output).
   c) If the symbol scanned is a right parenthesis, then go on popping all the items from the stack and place them in the postfix expression till we get the matching left parenthesis.
   d) If the scanned symbol is an operator, then go on removing all the operators from the stack and place them in the postfix expression, if and only if the precedence of the operator which is on the top of the stack is greater than (*or greater than or equal) to the precedence of* the scanned operator and push the scanned operator onto the stack otherwise, push the scanned operator onto the stack.

Convert the following infix expression A + B * C – D / E * H into its equivalent postfix expression

SYMBOL	POSTFIX STRING	STACK	REMARKS
A	A		
+	A	+	
B	A B	+	
*	A B	+ *	
C	A B C	+ *	
-	A B C * +	-	
D	A B C * + D	-	
/	A B C * + D	- /	
E	A B C * + D E	- /	
*	A B C * + D E /	- *	
H	A B C * + D E / H	- *	
End of string	A B C * + D E / H * -	The input is now empty. Pop the output symbols from the stack until it is empty.	

Conversion from Infix to Prefix

1. The precedence rules for converting an expression from infix to prefix are identical.
2. The only change from postfix conversion is that traverse the expression from right to left and the operator is placed before the operands rather than after them.
3. The prefix form of a complex expression is not the mirror image of the postfix form.

Convert the following infix expression A + B - C into its equivalent prefix expression

SYMBOL	PREFIX STRING	STACK	REMARKS
C	C		
-	C	-	
B	B C	-	
+	B C	- +	
A	A B C	- +	
End of string	- + A B C	The input is now empty. Pop the output symbols from the stack until it is empty.	

Evaluation of Postfix Expression

1. The postfix expression is evaluated easily by the use of a stack.
2. When a number is seen, it is pushed onto the stack
3. when an operator is seen, the operator is applied to the two numbers that are popped from the stack and the result is pushed onto the stack.
4. When an expression is given in postfix notation, there is no need to know any precedence rules

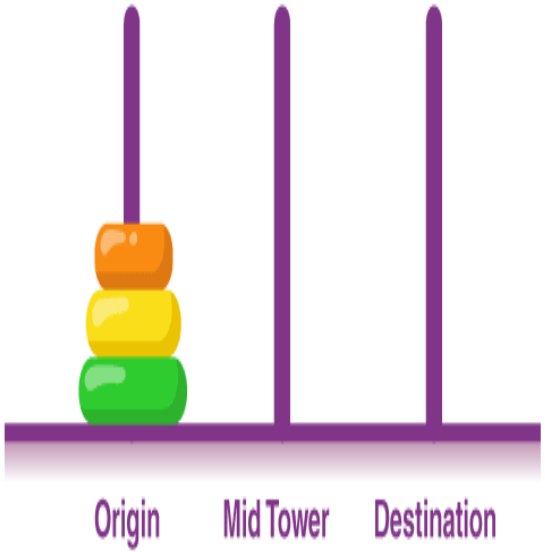
Evaluate the following postfix expression

6 2 3 + - 3 8 2 / + * 2 ↑ 3 +

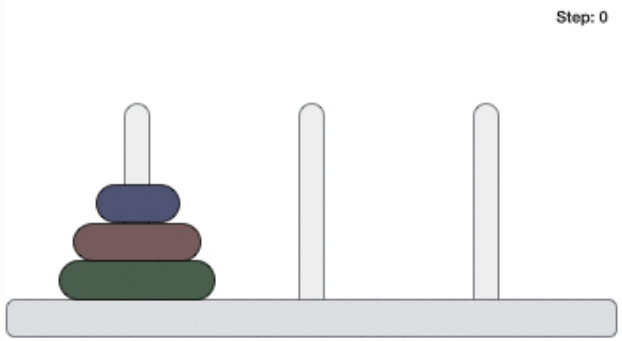
SYMBOL	OPERAND 1	OPERAND 2	VALUE	STACK
6				6
2				6, 2
3				6, 2, 3
+	2	3	5	6, 5
-	6	5	1	1
3	6	5	1	1, 3
8	6	5	1	1, 3, 8
2	6	5	1	1, 3, 8, 2
/	8	2	4	1, 3, 4
+	3	4	7	1, 7
*	1	7	7	7
2	1	7	7	7, 2
↑	7	2	49	49
3	7	2	49	49, 3
+	49	3	52	52

Tower of Hanoi

- Tower of Hanoi is a mathematical puzzle where we have three towers and **N** disks.
- Initially, all the disks are stacked in decreasing value of diameter i.e., the smallest disk is placed on the top.
- The objective of the puzzle is to move the entire stack to another tower.
- Tower of Hanoi puzzle with n disks can be solved in minimum $\mathbf{2^n - 1}$ steps



Tower of Hanoi - Rules



1. Only one disk can be moved among the towers at any given time.
2. Only the "top" disk can be removed.
3. No large disk can sit over a small disk.

Tower of Hanoi - Algorithm

```
START
Procedure Hanoi(disk, source, dest, aux)
IF disk == 1, THEN
   move disk from source to dest
ELSE
   Hanoi(disk - 1, source, aux, dest)
   move disk from source to dest
   Hanoi(disk - 1, aux, dest, source)
END IF
END Procedure
STOP
```

GIET
INSTITUTIONS
ANDHRA PRADESH, INDIA

Thank You