DS UNIT 2

[4-marks]

1 Define stack. Explain its basic operations.

Definition of Stack: A stack is a linear data structure in which all insertions and deletions are performed at one end called the *top*. It follows the *Last In First Out (LIFO)* principle, where the element inserted last is removed first.

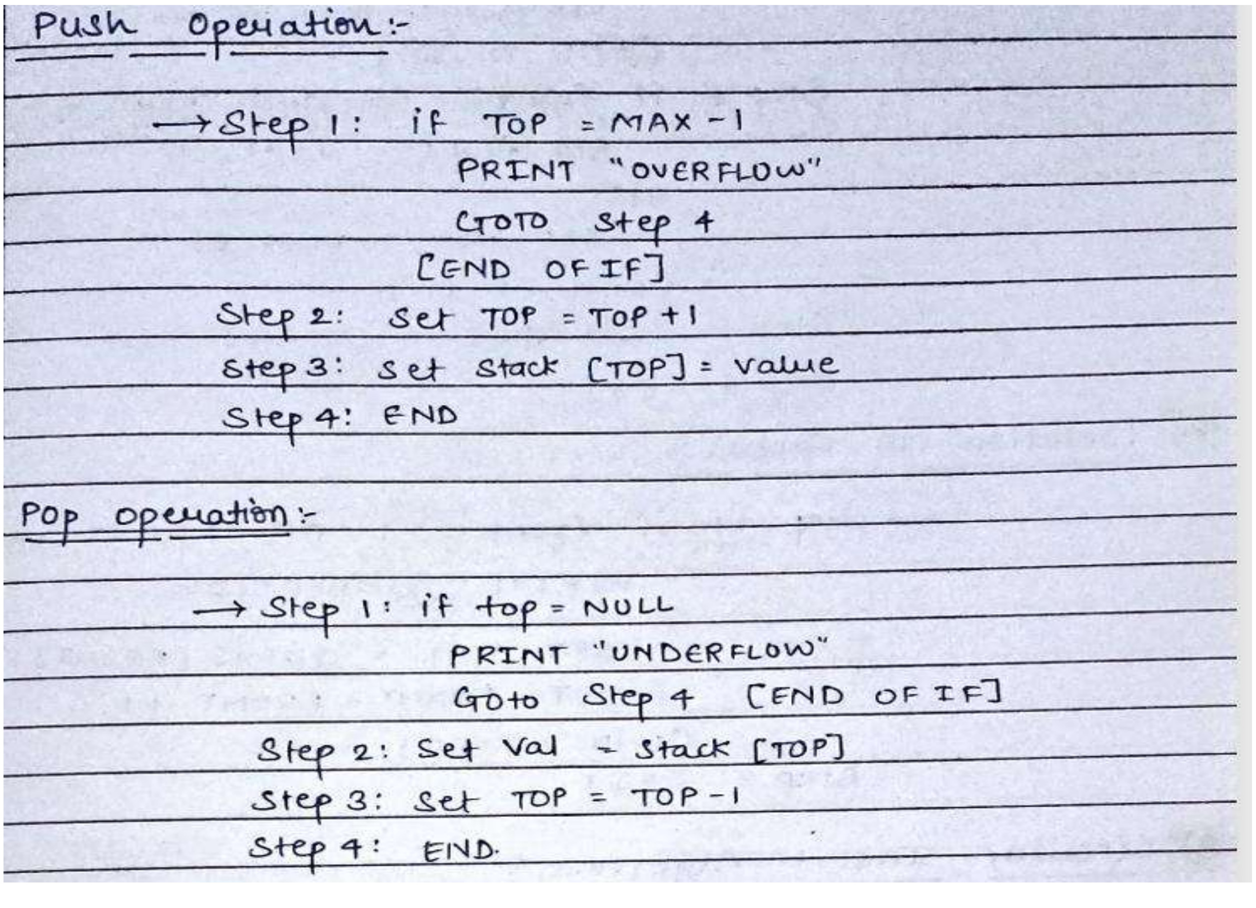
Push Operation: The push operation is used to add a new element to the top of the stack. Before inserting the element, the system checks whether the stack is full to prevent an overflow condition.

Pop Operation: The pop operation removes the top element from the stack. Before removing the element, it checks whether the stack is empty to avoid an underflow condition.

Peek (Top) Operation: The peek operation is used to view the top element of the stack without removing it. This operation helps in accessing the most recently added element safely.

isEmpty and isFull Operations: The isEmpty operation checks whether the stack contains any elements or not. The isFull operation checks whether the stack has reached its maximum capacity and cannot accept more elements.

2 Write push and pop algorithms for stack.



Push Operation:-

```
→ Step 1:  if TOP = MAX - 1
               PRINT "OVERFLOW"
               GOTO Step 4
           [END OF IF]
  Step 2:  Set TOP = TOP + 1
  Step 3:  Set Stack [TOP] = value
  Step 4:  END
```

Pop operation:-

```
→ Step 1:  if top = NULL
               PRINT "UNDERFLOW"
               Goto Step 4   [END OF IF]
  Step 2:  Set Val = Stack [TOP]
  Step 3:  Set TOP = TOP - 1
  Step 4:  END.
```

3 What is stack overflow and stack underflow?

Stack Overflow:
Stack overflow is a condition that occurs when an attempt is made to push an element onto a stack that is already full. This usually happens in array-based stack implementations where the stack has a fixed size and no additional memory space is available to store new elements. When stack overflow occurs, the push operation cannot be completed, which may lead to program errors or abnormal termination if not handled properly.

Stack Underflow:
Stack underflow is a condition that occurs when an attempt is made to pop an element from an empty stack. This situation arises when no elements are present in the stack, but a delete or pop operation is still performed. Stack underflow can cause runtime errors and incorrect program behavior, so programs usually check whether the stack is empty before performing a pop operation.

4 List applications of stack

Function Calls and Method Execution:
Stacks are widely used in programming languages to manage function calls. They store return addresses, parameters, and local variables so that the program can resume execution correctly after a function completes.

Recursion Handling:
Stacks are used to store information about each recursive function call. Each call is pushed onto the stack and removed once the base condition is reached and the function returns.

Expression Evaluation:
Stacks are used to evaluate postfix and prefix expressions efficiently. They help in maintaining operands and applying operators in the correct order.

Syntax Parsing and Parenthesis Checking:
Stacks are used to check the correctness of syntax in expressions and programs. They help in verifying balanced parentheses, brackets, and braces.

Reversal of Data:
Stacks are used to reverse strings, arrays, or lists. Elements are pushed onto the stack and popped out in reverse order.

Memory Management:
In some systems, stacks are used for dynamic memory allocation. They help manage temporary memory during program execution.

5 Define recursion. Explain how stack is used in recursion

- **Definition of Recursion:** Recursion is a programming method in which a function calls itself to solve a problem. The problem is broken into smaller and similar subproblems until a base condition stops the recursion.
- **Role of Stack in Recursion:** Each time a recursive function is called, its information is stored in the stack as a stack frame. This frame holds the function's parameters, local variables, and the address to return after execution.
- **Storage of Function Calls:** Recursive function calls are placed in the stack one after another. This helps the program keep track of which function call was made first and which one should return last.
- **Returning from Recursive Calls:** When the base condition is reached, the function begins to return. The stack removes function calls one by one following the LIFO rule, allowing the program to continue correctly.
- **Importance of Stack in Recursion:** The stack is essential for handling recursion because it manages multiple function calls at the same time. Without the stack, recursion would not work properly and results would be incorrect.

6 What is postfix expression? Give one example.

Postfix Expression:
A postfix expression is an arithmetic expression in which the operator is placed **after** the operands. This type of expression is also called **Reverse Polish Notation (RPN)** and is commonly used in computer systems for expression evaluation.

No Need for Parentheses:
In postfix expressions, the order of operations is clearly defined by the position of operators and operands. Because of this clear order, parentheses are not required to control precedence.

Easy Evaluation Using Stack:
Postfix expressions are very easy for computers to evaluate using a stack. Operands are pushed onto the stack, and when an operator appears, operands are popped, evaluated, and the result is pushed back.

Example of Postfix Expression:
For example, the infix expression **A + B** is written as **AB+** in postfix form. Similarly, the infix expression **(A + B) × C** becomes **AB+C*** in postfix notation.

7 Differentiate between infix and postfix expressions.

Position of Operator:
In an infix expression, the operator is placed **between** the operands. In a postfix expression, the operator is placed **after** the operands.

Use of Parentheses:
Infix expressions often require parentheses to clearly specify the order of operations. Postfix expressions do not need parentheses because the order of evaluation is already defined.

Order of Evaluation:
In infix expressions, operator precedence and associativity rules must be followed. In postfix expressions, evaluation is straightforward and follows the sequence of the expression.

Ease of Evaluation:
Infix expressions are easy for humans to read and understand. Postfix expressions are easier for computers to evaluate using a stack.

Example:
An example of an infix expression is **(A + B) × C**. The corresponding postfix expression is **AB+C***.

[5- marks]

1 Explain stack representation using array.

In array representation, a stack is implemented using a one-dimensional linear array. A variable called TOP is used to keep track of the position of the top element in the stack at any time**.**

Initialization of Stack:
At the beginning, the stack is empty, so the value of TOP is set to –1. The maximum size of the stack is fixed and defined at the time of array creation.

Push Operation:
When an element is inserted into the stack, the TOP value is first increased by one. The new element is then stored at the index indicated by TOP, after checking that the stack is not full to avoid overflow.

Pop Operation:
During the pop operation, the element at the TOP position is removed. After removal, the TOP value is decreased by one, and underflow is checked to ensure the stack is not empty.

Advantages and Limitations:
Array-based stack is easy to implement and provides fast access. However, its fixed size may cause overflow if the number of elements exceeds the allocated memory.

2 Explain stack representation using linked list.

In linked list representation, a stack is implemented using nodes where each node contains data and a link to the next node. The top of the stack is represented by the head node of the linked list.

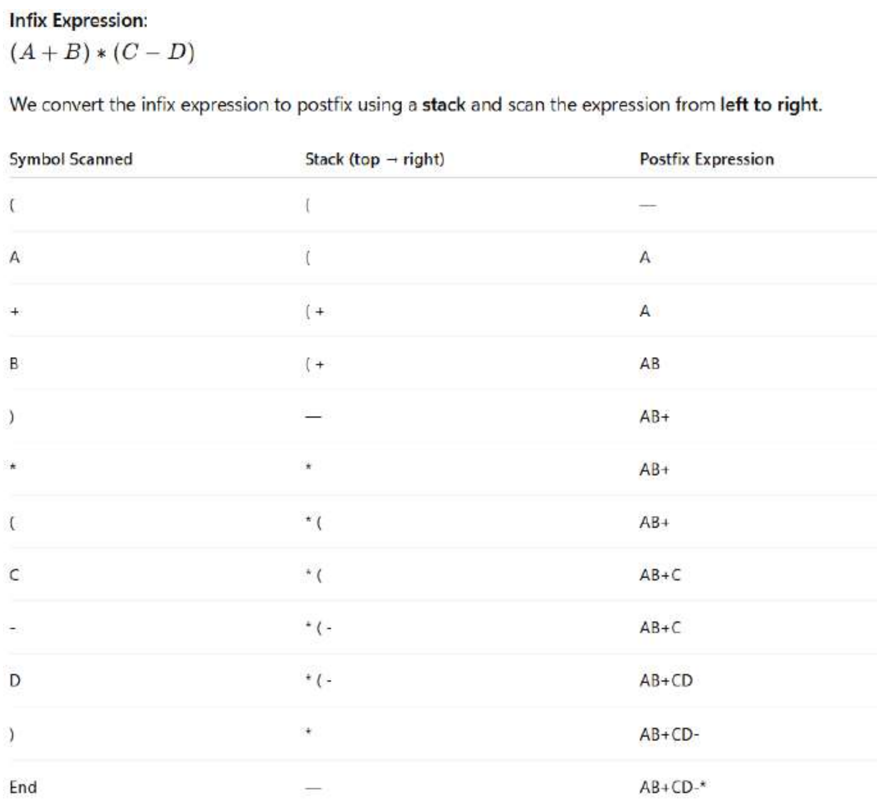
Initialization of Stack: Initially, the top pointer is set to NULL, which indicates that the stack is empty. Memory is allocated dynamically whenever a new element is added.

Push Operation: To push an element, a new node is created and its data field is filled. The new node is linked to the current top, and then the top pointer is updated to point to the new node.

Pop Operation: In the pop operation, the node pointed to by the top is removed. The top pointer is updated to point to the next node, and the memory of the removed node is released.

Advantages and Limitations: Linked list implementation allows the stack to grow or shrink dynamically without overflow issues. However, it requires extra memory for storing pointers and is slightly more complex than array implementation.

3 Convert the following infix expression into postfix: $(A+B)*(C-D)$.



Infix Expression:

$(A + B) * (C - D)$

We convert the infix expression to postfix using a **stack** and scan the expression from **left to right**.

Symbol Scanned	Stack (top → right)	Postfix Expression
(	(	—
A	(	A
+	(+	A
B	(+	AB
)	—	AB+
*	*	AB+
(	* (	AB+
C	* (	AB+C
-	* (-	AB+C
D	* (-	AB+CD
)	*	AB+CD-
End	—	AB+CD-*

Final Answer = AB+CD-*

4 Explain evaluation of postfix expression with example.

Evaluation of a postfix expression is done using a stack. The expression is scanned from left to right, and operands and operators are processed according to specific rules.

Rule for Operands:

When an operand (number or variable) is encountered, it is pushed onto the stack. No calculation is done at this stage.

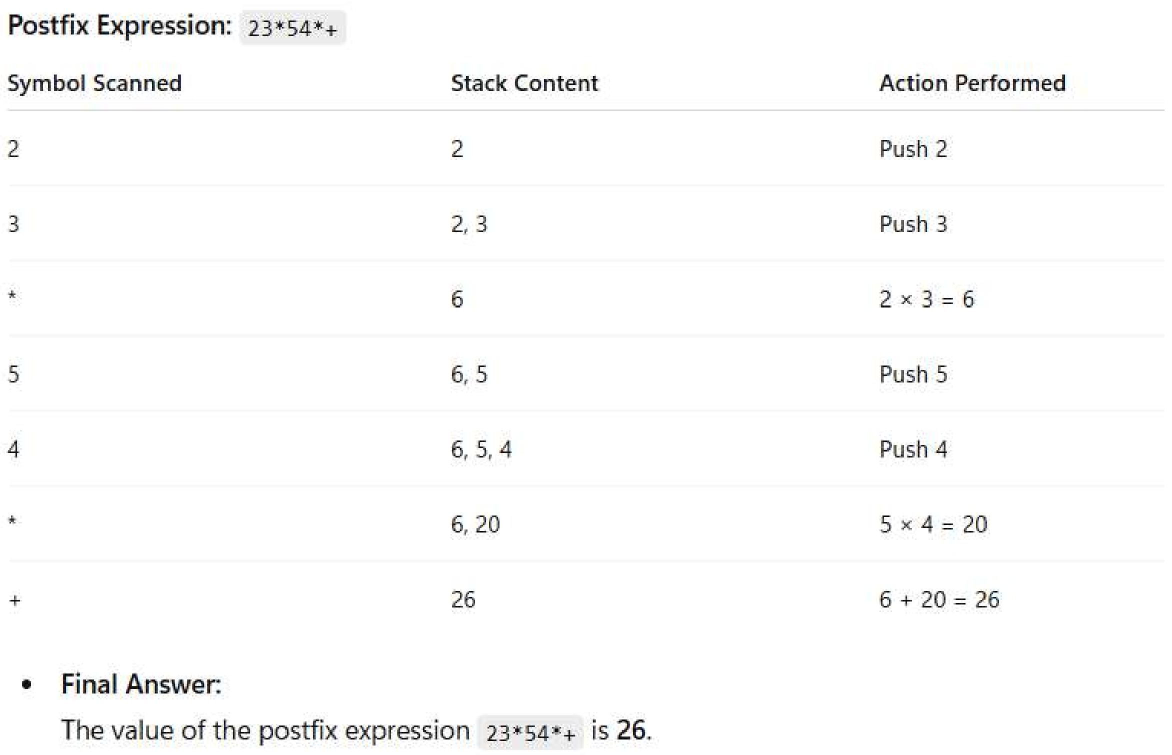
Rule for Operators:

When an operator is encountered, the top two operands are popped from the stack. The operator is applied to these operands, and the result is pushed back onto the stack.

Final Result:

After the entire postfix expression is scanned, the final value left in the stack is the result of the expression. This method avoids the need for parentheses and precedence rules.

(no need to write for 5 marks , depend on you , it’s just to know how it works)



Postfix Expression: `23*54*+`

Symbol Scanned	Stack Content	Action Performed
2	2	Push 2
3	2, 3	Push 3
*	6	2 × 3 = 6
5	6, 5	Push 5
4	6, 5, 4	Push 4
*	6, 20	5 × 4 = 20
+	26	6 + 20 = 26

- **Final Answer:**
  The value of the postfix expression `23*54*+` is **26**.

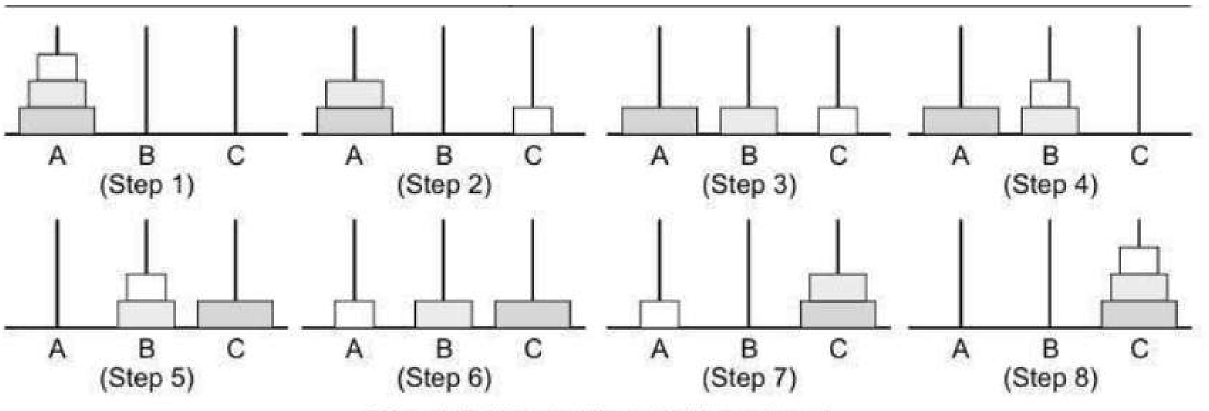
5 Explain Tower of Hanoi problem using stack or recursion.

The Tower of Hanoi is a problem where a number of disks are moved from one rod to another using a third rod. The disks must be moved one at a time, and a bigger disk cannot be placed on a smaller disk.

Basic Rules: Only one disk can be moved at a time. The order of disks must always be maintained, with smaller disks on top.

Use of Recursion: The problem is solved using recursion by breaking it into smaller steps. First, move *n–1* disks to the auxiliary rod, then move the largest disk, and finally move the *n–1* disks to the destination rod.

Use of Stack: Each recursive call is stored in the stack along with its details. The stack helps the program remember the sequence of moves and execute them in the correct order.



(Consider the working with three rings.)

6 Explain prefix notation and its evaluation with example.

Prefix notation is a form of arithmetic expression in which the operator is written before the operands. It is also known as Polish Notation and does not require parentheses to specify the order of operations.

Characteristics of Prefix Notation:

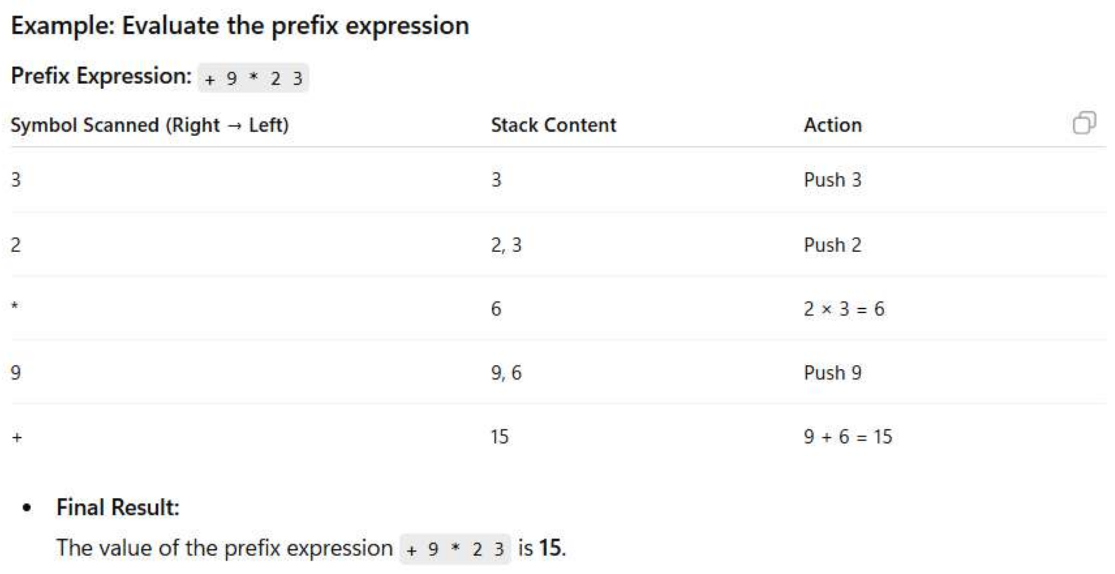
In prefix expressions, the position of the operator decides the order of evaluation. This makes the expression clear and unambiguous for the computer.

Evaluation Method:

Prefix expressions are evaluated using a stack. The expression is scanned from right to left, unlike postfix which is scanned left to right.

Evaluation Rule:

When an operand is encountered, it is pushed onto the stack. When an operator is encountered, two operands are popped, the operation is performed, and the result is pushed back onto the stack.



Example: Evaluate the prefix expression

Prefix Expression: `+ 9 * 2 3`

Symbol Scanned (Right → Left)	Stack Content	Action
3	3	Push 3
2	2, 3	Push 2
*	6	2 × 3 = 6
9	9, 6	Push 9
+	15	9 + 6 = 15

- **Final Result:**
  The value of the prefix expression `+ 9 * 2 3` is **15.**

7 Differentiate between recursion and iteration.

Recursion is a technique in which a function calls itself to solve a problem. Iteration is a technique where a set of instructions is repeated using loops such as *for*, *while*, or *do-while*.

Termination Condition:
In recursion, a base condition is required to stop further function calls. In iteration, the loop runs until the given condition becomes false**.**

Use of Memory:
Recursion uses more memory because each function call is stored in the stack. Iteration uses less memory since it does not create multiple function calls.

Performance:
Recursive solutions may be slower due to function call overhead. Iterative solutions are usually faster and more efficient.

Code Readability:
Recursion makes the code easier to understand for problems like Tower of Hanoi or tree traversal. Iteration is simpler and clearer for problems involving repetition.

Risk of Errors:
Recursion may cause stack overflow if the base condition is not defined properly. Iteration does not have this risk because it does not depend on stack calls.

[10- marks]

1 Explain stack representation using array and linked list. Write algorithms for push and pop operations.

ARRAY : In array representation, a stack is implemented using a one-dimensional linear array. A variable called TOP is used to keep track of the position of the top element in the stack at any time**.**

At the beginning, the stack is empty, so the value of TOP is set to –1. The maximum size of the stack is fixed and defined at the time of array creation.

Push Operation: When an element is inserted into the stack, the TOP value is first increased by one. The new element is then stored at the index indicated by TOP, after checking that the stack is not full to avoid overflow.

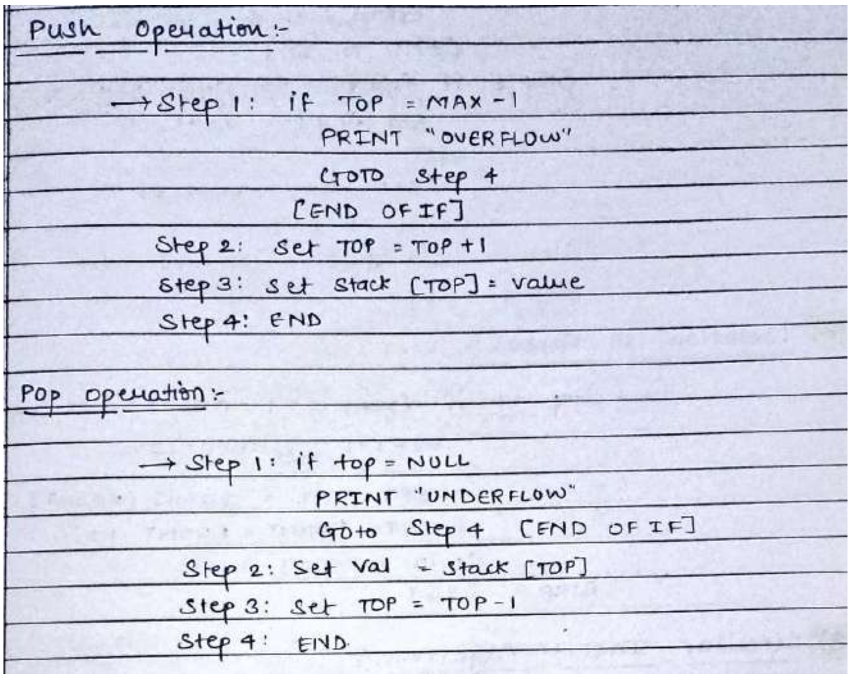
Pop Operation: During the pop operation, the element at the TOP position is removed. After removal, the TOP value is decreased by one, and underflow is checked to ensure the stack is not empty.

LinkedList : In linked list representation, a stack is implemented using nodes where each node contains data and a link to the next node. The top of the stack is represented by the head node of the linked list.

Initially, the top pointer is set to NULL, which indicates that the stack is empty. Memory is allocated dynamically whenever a new element is added.

Push Operation: To push an element, a new node is created and its data field is filled. The new node is linked to the current top, and then the top pointer is updated to point to the new node.

Pop Operation: In the pop operation, the node pointed to by the top is removed. The top pointer is updated to point to the next node, and the memory of the removed node is released.



Push Operation:-

```
→ Step 1: if TOP = MAX - 1
              PRINT "OVERFLOW"
              GOTO Step 4
          [END OF IF]
  Step 2: Set TOP = TOP + 1
  Step 3: Set Stack [TOP] = value
  Step 4: END
```

Pop operation:-

```
→ Step 1: if top = NULL
              PRINT "UNDERFLOW"
              Goto Step 4 [END OF IF]
  Step 2: Set Val = Stack [TOP]
  Step 3: Set TOP = TOP - 1
  Step 4: END
```

2 Explain applications of stack in detail: recursion, Tower of Hanoi, and expression evaluation

1. Stack in Recursion
Recursion is a programming technique where a function calls itself to solve a problem. Every recursive call needs to remember its own data and where to return after execution.

- **Role of Stack:**
  When a recursive function is called, its details such as parameters, local variables, and return address are stored in the stack as a **stack frame**. Each new recursive call is pushed onto the stack.
- **Execution and Return:**
  When the base condition is reached, the function stops calling itself. Then, the stack starts popping function calls one by one in **LIFO order**, allowing the program to return correctly.

2. Stack in Tower of Hanoi
The Tower of Hanoi is a problem where disks are moved from one rod to another using a third rod. It follows strict rules about disk movement.

- **Use of Recursion:**
  The problem is solved recursively by breaking it into smaller steps. To move *n* disks, the function first moves *n–1* disks, then the largest disk, and again *n–1* disks.
- **Role of Stack:**
  Each recursive call made during the solution is stored in the stack. The stack keeps track of which disk to move and between which rods.

3. Stack in Expression Evaluation
Stacks are widely used to evaluate arithmetic expressions, especially **postfix** and **prefix** expressions. These expressions are easier for computers to process.

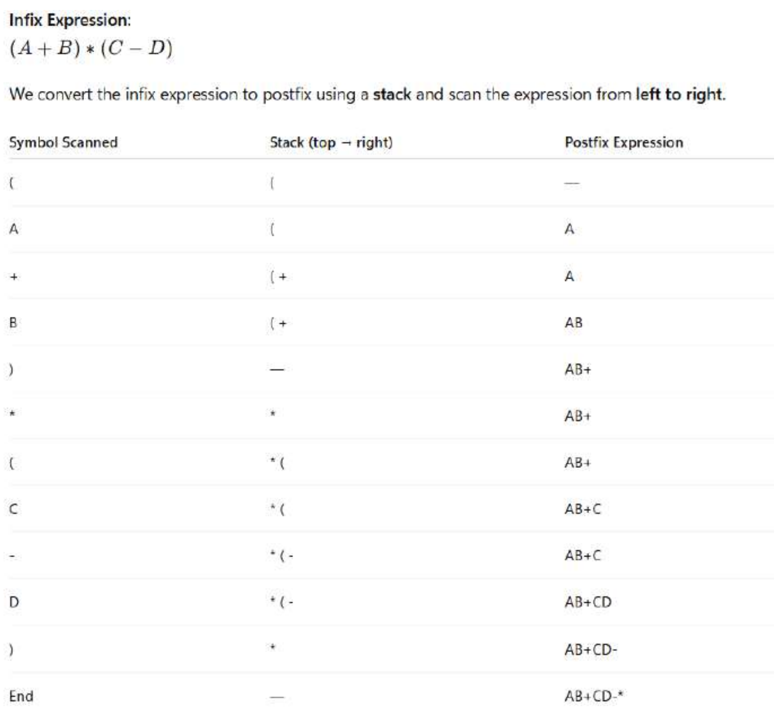
- **Working Method:**
  In postfix evaluation, operands are pushed onto the stack. When an operator is encountered, operands are popped, the operation is performed, and the result is pushed back.
- **Advantages:**
  Stack-based evaluation removes the need for parentheses and operator precedence rules. This makes expression evaluation simple and efficient.

3 Explain conversion of infix expression to postfix expression with algorithm and example.

In an infix expression, operators are written between operands, for example **A + B**. This form is easy for humans to read but difficult for computers to evaluate directly because of operator precedence and parentheses.

Need for Conversion: To simplify evaluation, infix expressions are converted into **postfix expressions**. Postfix expressions do not need parentheses and can be evaluated easily using a stack.

Use of Stack: A stack is used to temporarily store operators and parentheses while scanning the infix expression from left to right. Operands are directly added to the postfix expression.



Infix Expression:
$(A + B) * (C - D)$

We convert the infix expression to postfix using a **stack** and scan the expression from **left to right**.

Symbol Scanned	Stack (top → right)	Postfix Expression
(	(	—
A	(	A
+	(+	A
B	(+	AB
)	—	AB+
*	*	AB+
(	* (	AB+
C	* (	AB+C
-	* (-	AB+C
D	* (-	AB+CD
)	*	AB+CD-
End	—	AB+CD-*

Algorithm - (if yoh have better one write that , this one is from chatgpt)

1. Create an empty **stack** for operators and an empty **postfix** string.
2. Scan the infix expression from **left to right**.
3. If the symbol is an **operand**, add it directly to the postfix expression.
4. If the symbol is **(**, push it onto the stack.
5. If the symbol is an **operator**, pop operators from the stack that have higher or equal precedence and add them to postfix, then push the current operator onto the stack.
6. If the symbol is **)**, pop operators from the stack and add them to postfix until **(** is found, then discard **(**.
7. After scanning the expression, pop all remaining operators from the stack and add them to postfix.

4 Explain evaluation of postfix and prefix expressions with algorithms and examples.

1. Evaluation of Postfix Expression

- A postfix expression is evaluated using a **stack**.
- The expression is scanned from **left to right**.
- Operands are stored in the stack, and operations are performed when an operator is found.

Algorithm (from chatgpt)

1. Create an empty stack.
2. Scan the postfix expression from left to right.
3. If the scanned symbol is an operand, push it onto the stack.
4. If the scanned symbol is an operator, pop the top two operands from the stack.
5. Apply the operator on the operands (second popped operand operator first popped operand).
6. Push the result back onto the stack.
7. Repeat until the expression ends.
8. The value left in the stack is the final result

2. Evaluation of Prefix Expression

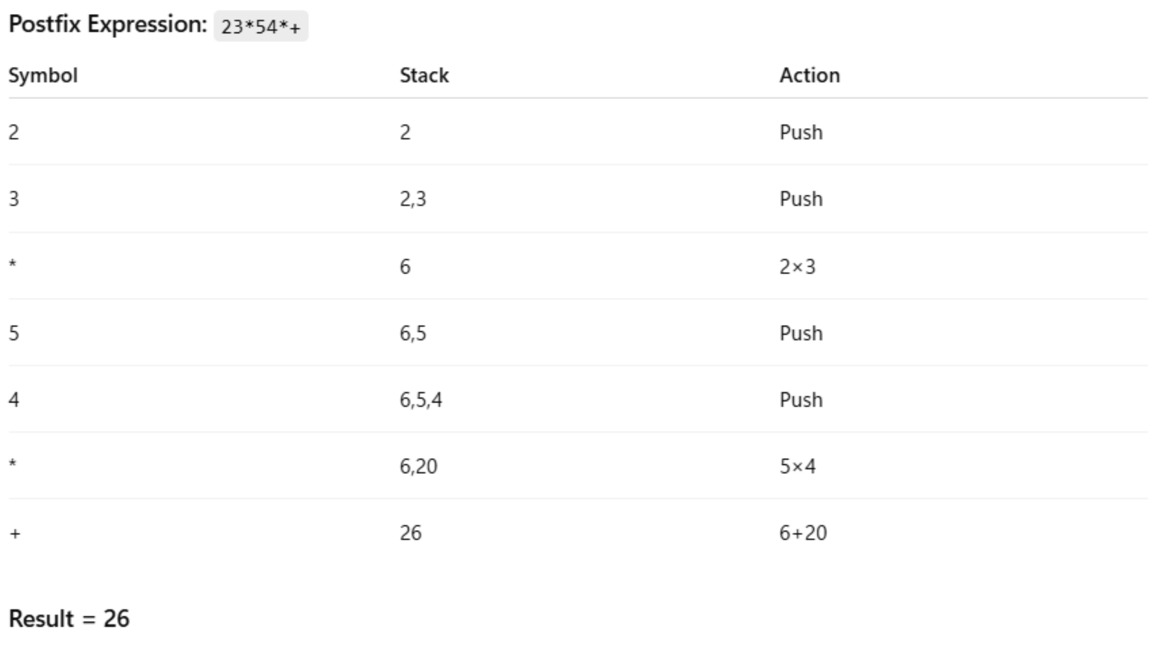
- Prefix expression evaluation also uses a **stack**.
- The expression is scanned from **right to left**.
- Operands are pushed first, and operations are applied when an operator is encountered.

Algorithm (from chatgpt)

1. Create an empty stack.
2. Scan the prefix expression from right to left.
3. If the scanned symbol is an operand, push it onto the stack.
4. If the scanned symbol is an operator, pop the top two operands from the stack.
5. Apply the operator on the operands (first popped operand operator second popped operand).
6. Push the result back onto the stack.
7. Repeat until the expression ends.
8. The value left in the stack is the final result.

Example's

Evaluation of Postfix :

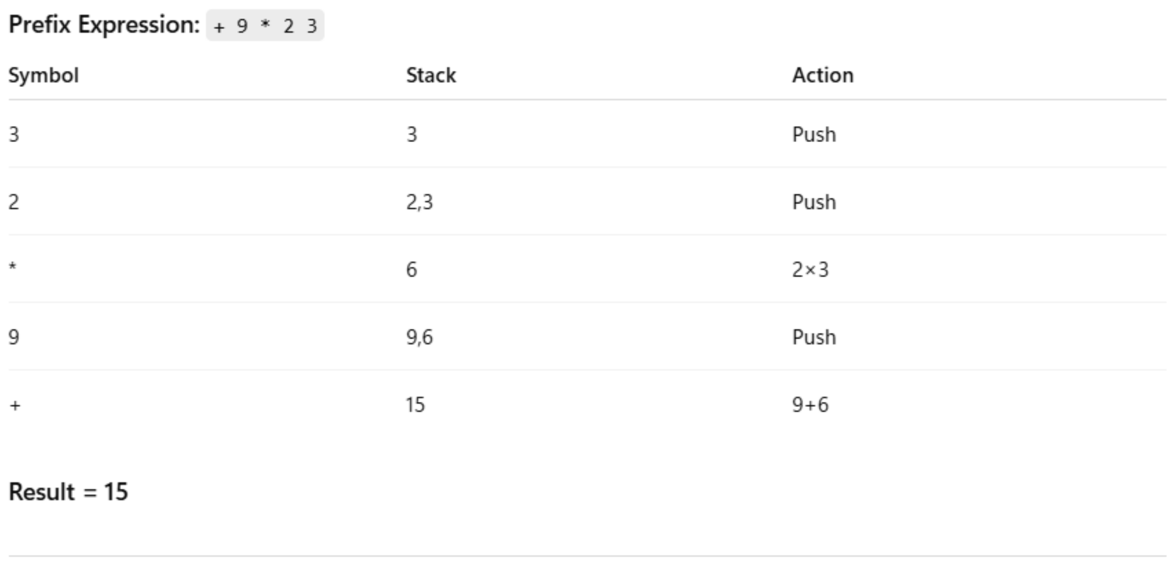


Postfix Expression: `23*54*+`

Symbol	Stack	Action
2	2	Push
3	2,3	Push
*	6	2×3
5	6,5	Push
4	6,5,4	Push
*	6,20	5×4
+	26	6+20

Result = 26

Evaluation of Prefix :



Prefix Expression: `+ 9 * 2 3`

Symbol	Stack	Action
3	3	Push
2	2,3	Push
*	6	2×3
9	9,6	Push
+	15	9+6

Result = 15

5 Explain recursion in detail. Show how stack is used in recursive function calls with suitable example.

Recursion and the Call Stack

- **Definition of Recursion:** Recursion occurs when a function solves a problem by calling a copy of itself to work on a smaller version of the input. To prevent it from running forever, the function must always have a "base case" that stops the calls when the problem becomes simple enough to solve directly.
- **Role of the Stack:** The computer uses a special memory area called the "call stack" to keep track of function calls. When a recursive call happens, the computer "pushes" the current state (variables and return address) onto the stack and pauses execution to run the new function call.
- **Unwinding the Stack:** Once a call hits the base case and returns a value, the computer "pops" the previous stack frame off the top. This allows the paused function to resume exactly where it left off and use the returned value to finish its calculation.
- **Memory Overhead:** Every recursive step adds a new layer to the stack, consuming more memory with each call. If the recursion goes too deep without stopping, the stack runs out of space, causing a crash known as a "stack overflow."

Example: Calculating Factorial of 3 (3!)

Call 1: You call Factorial(3). The function sees 3 > 1, so it pauses and calls 3 * Factorial(2), pushing this incomplete task onto the stack

Call 2: Inside Factorial(2), it sees 2 > 1, so it pauses again and calls 2 * Factorial(1), pushing this second task onto the stack.

Call 3 (Base Case): Inside Factorial(1), the base case is met, so it immediately returns 1. This value is sent back to the waiting Factorial(2) function

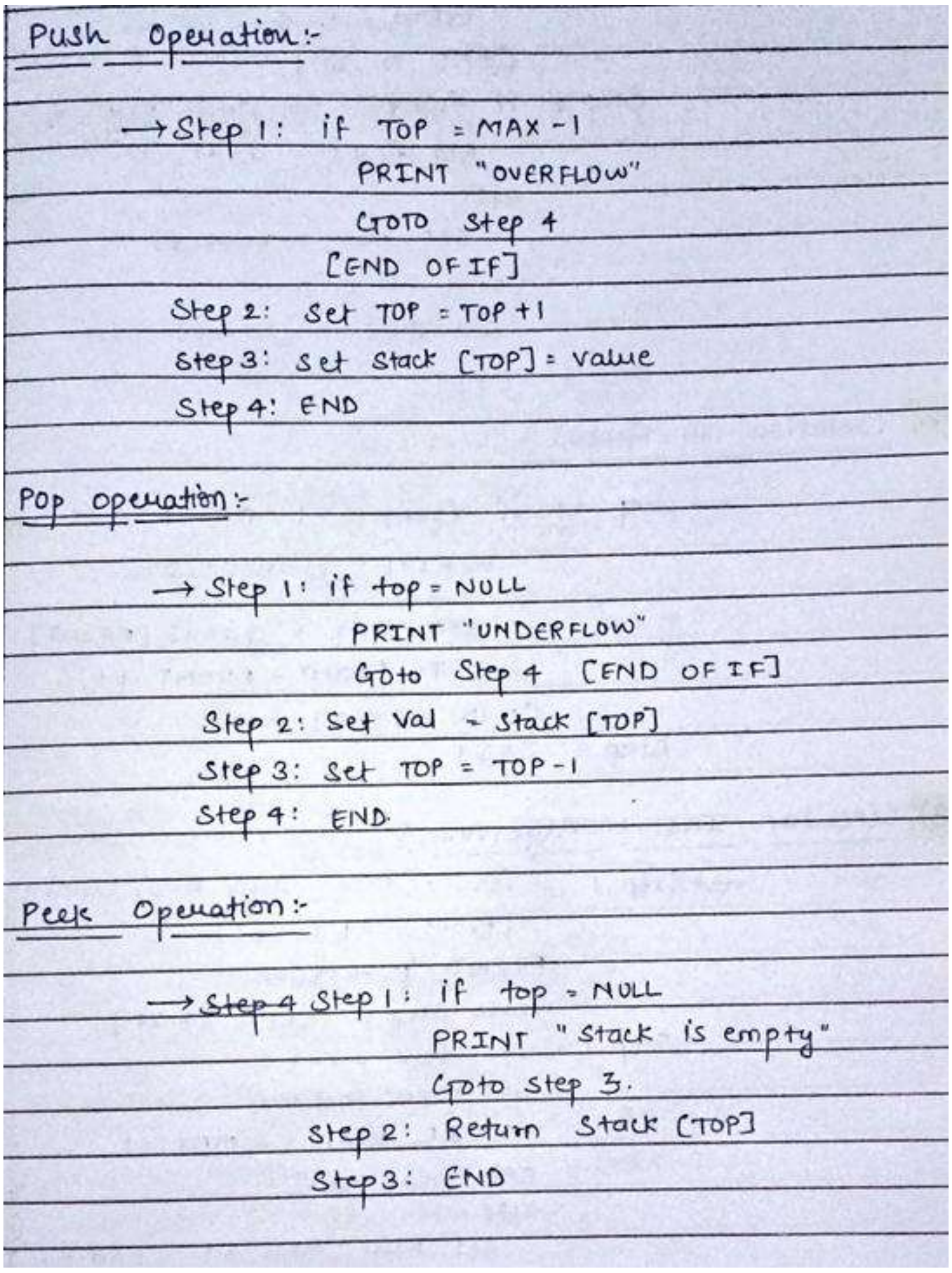
Resolution: Factorial(2) wakes up, calculates 2 * 1 = 2, and returns it. Finally, Factorial(3) wakes up, calculates 3 * 2 = 6, and returns the final answer.

6 Define and explain the working of the stack data structure. Give algorithm for the push, pop, peek, isFull, isEmpty functions of stack.

A stack is a linear data structure that follows the LIFO (Last-In, First-Out) principle. This means the last item you placed on the pile is the first one you can remove, similar to a stack of dinner plates.

Working Principle: All insertion and deletion operations happen at only one end of the structure, commonly referred to as the "Top." You cannot access elements in the middle or bottom without first removing the elements sitting above them.

Algorithms for Stack Operations :



Push Operation:-

```
→Step 1:  if  TOP = MAX - 1
                PRINT "OVERFLOW"
                GOTO Step 4
              [END OF IF]
 Step 2:  Set TOP = TOP + 1
 Step 3:  Set Stack [TOP] = value
 Step 4:  END
```

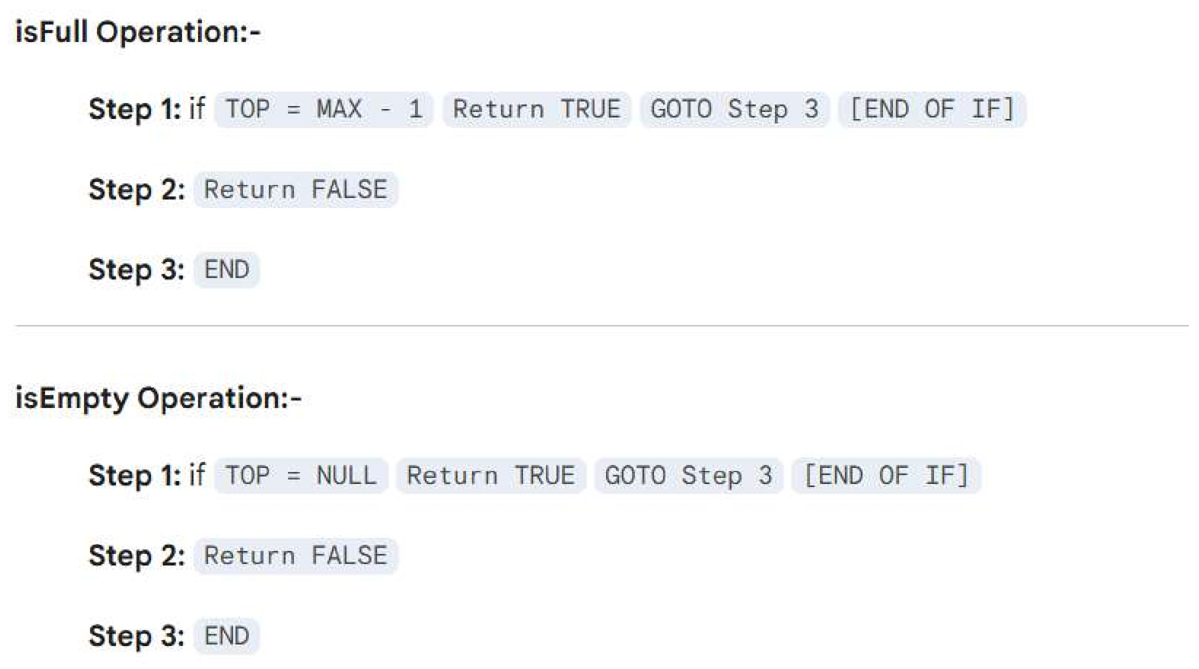
Pop operation:-

```
→ Step 1: if top = NULL
              PRINT "UNDERFLOW"
              GOto Step 4  [END OF IF]
  Step 2: Set Val ← Stack [TOP]
  Step 3: Set TOP = TOP - 1
  Step 4: END
```

Peek Operation:-

```
→ ~~Step 4~~ Step 1: if top = NULL
                       PRINT "Stack is empty"
                       Goto step 3.
             Step 2: Return Stack [TOP]
             Step 3: END
```

IsFull and IsEmpty :



isFull Operation:-

```
Step 1: if TOP = MAX - 1 Return TRUE GOTO Step 3 [END OF IF]
Step 2: Return FALSE
Step 3: END
```

---

isEmpty Operation:-

```
Step 1: if TOP = NULL Return TRUE GOTO Step 3 [END OF IF]
Step 2: Return FALSE
Step 3: END
```