Data Structures

Stack

Rutvij H. Jhaveri
Computer Science & Engineering

Outline

- What is Stack ?
- Basic Operations
- Applications
- Expression conversion
  - Infix to Postfix
  - Infix to Prefix
- Postfix expression evaluation
- Recursion

What is a Stack?

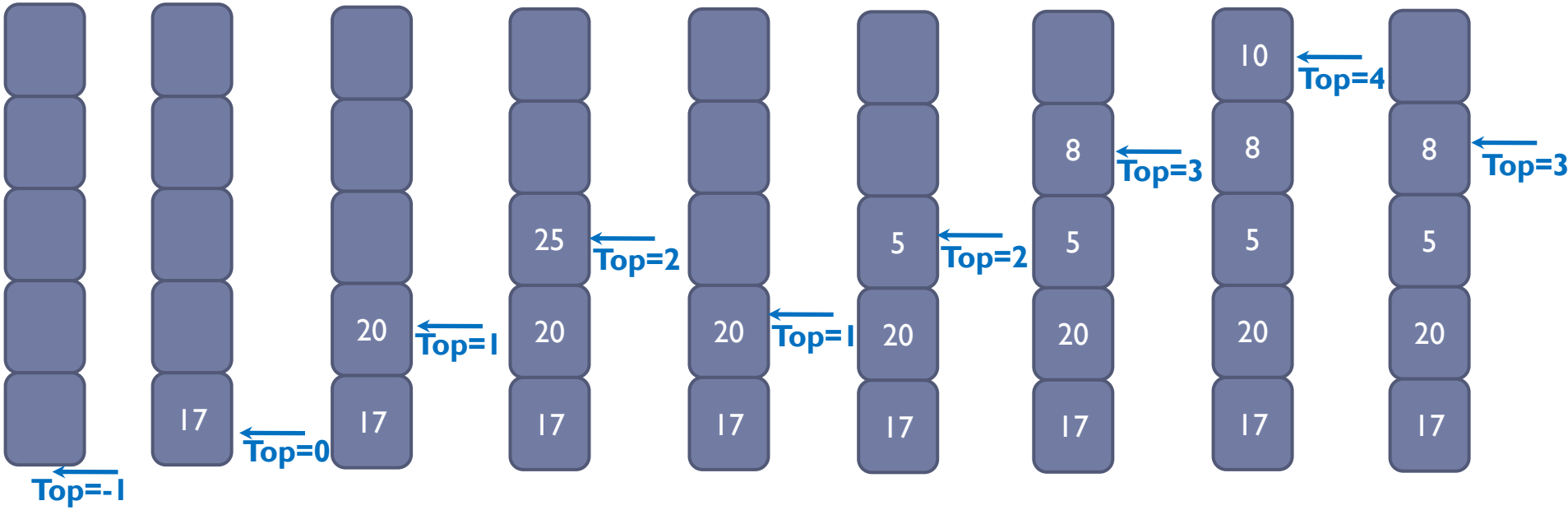
- A linear data structure containing ordered elements
- Elements are stacked on top of each other
- Insertion and deletion of elements are done at one end only
- Most recently inserted item is removed first
- Follows Last In First Out (LIFO) principle

Stack of plates in canteen

Basic Stack Operations

- PUSH: Insert an element
- POP: Remove an element
- PEEK: Display the element at the top of the stack
- isEmpty: Check whether stack is empty
- isFull: Check whether stack is full
- Traverse: Print all the elements of the stack from top to bottom.

PUSH, POP, isEmpty, isFull Operations

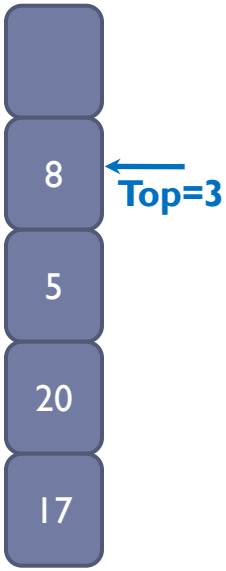


Conditions: isEmpty() -> Top=-1 isFull() -> Top=n-1

Stack Underflow: Trying to POP in an empty stack

Stack Overflow: Trying to PUSH a new element in a full stack

PEEK and TRAVERSE Operations



PEEK: Display 8

TRAVERSE: Display 8, 5, 20, 17

Perform basic stack operations

stack_op.c

Applications

- Reversing string
- Expression conversion: Infix to postfix (or reverse polish) and Infix to prefix (or polish)
- Evaluating expressions
- Verifying validity of an expression
- Recursion
- Solving backtracking problems
- “Undo" mechanism in text editors
- ….

Reverse a string using stack

stack_rev_string.c

Expressions

- Three ways to write an expression:
  - **Infix:** <operand><operator><operand>
  - **Prefix (Polish):** <operator><operand><operand>
  - **Postfix (Reverse Polish):** <operand><operand><operator>
- Infix expression: easy to understand and evaluate for human beings, but:
  - **require extra information** to make the order of evaluation of the operators clear viz. rules about operator precedence, associativity and parentheses ( )
  - processing infix notation is costly and difficult
- Prefix and Postfix expressions are parenthesis-free expressions (i.e. non-ambiguous) and efficient to process.
- Therefore, infix expression is first converted into either postfix or prefix notations and then computed.

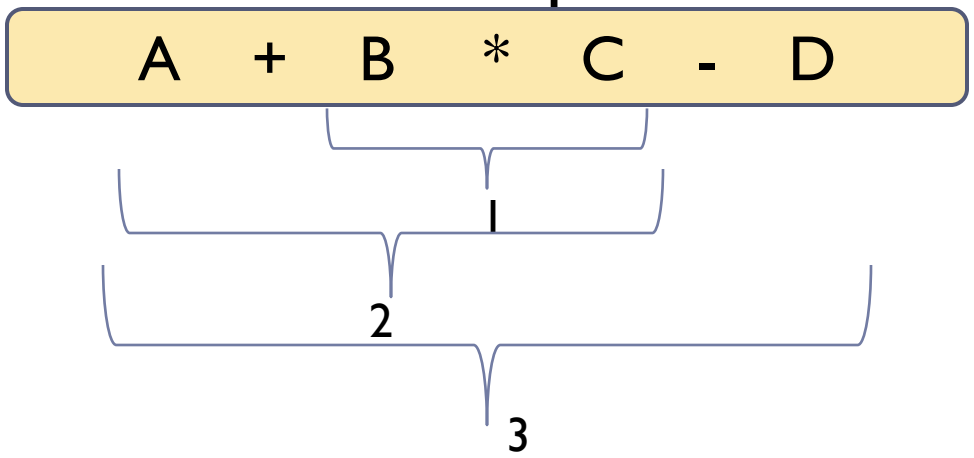
…Expressions

- Operator Precedence and Associativity:

Operator	Precedence	Associativity
Exponentiation (power)	Highest	Right
Multiplication, Division	Next Highest	Left
Addition, subtraction	Lowest	Left

https://en.wikipedia.org/wiki/Order_of_operations

- Evaluation of an infix expression:



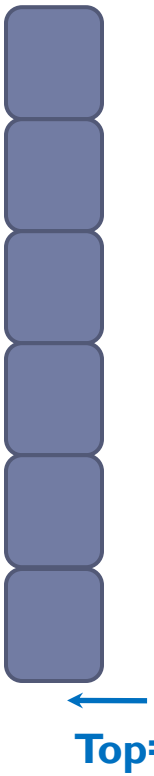
Expression Conversion using Stack

- Infix to Postfix Algorithm

1. Scan the Infix expression from left to right.
2. If the character is an operand, append it to the final output string.
3. If the character is an operator:
    1. If the stack is empty or contains a left parenthesis on top, push it onto the stack.
    2. If it has higher precedence (or same precedence for $ operator) than the top of the stack, push it on the stack.
    3. If it has lower precedence (or same precedence for operators except $) than the top of the stack, pop the operator and append the popped operator to the output string. Repeat this step until an operator with lower precedence is found or stack becomes empty. Push the scanned operator now.
4. If the character is a left parenthesis, push it on the stack.
5. If the character is a right parenthesis, pop the stack and append the operators to output string until a left parenthesis is found. Discard the pair of parentheses. Repeat from Step 1 until whole expression is scanned.
6. Now pop all operators from the stack and append them to the output string.
7. Print the output string.

…Expression Conversion using Stack

- Example Infix to Postfix: A + (B – C) * D / E



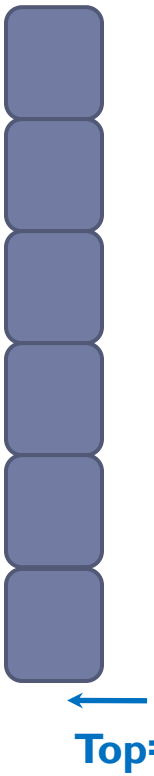
Initially: Stack Empty

- Let's start scanning the expression from Left to Right

Output String:

…Expression Conversion using Stack

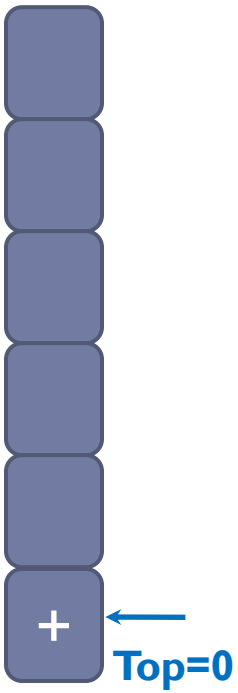
- Example Infix to Postfix: A + (B – C) * D / E
  - Scan A: As A is operand append directly to the Output String.



Output String: A

…Expression Conversion using Stack

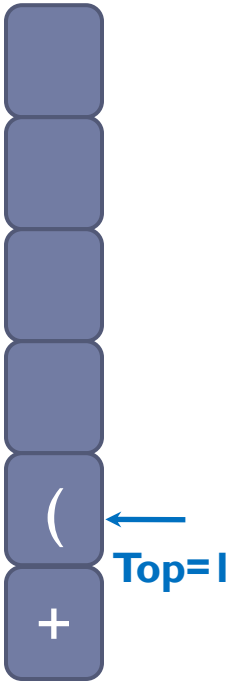
- Example Infix to Postfix: A + (B – C) * D / E
  - Scan +: As + is operator and stack is empty, Push it onto the stack.



Output String: A

…Expression Conversion using Stack

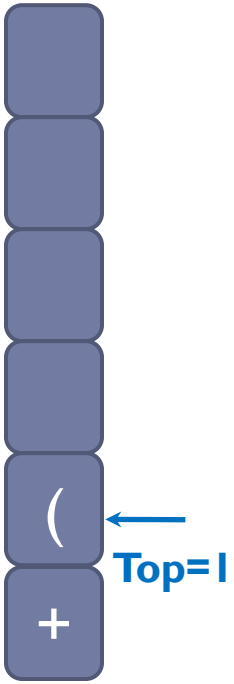
- Example Infix to Postfix: A + (B – C) * D / E
  - Scan (: As (is left paranthesis, Push it onto the stack.



Output String: A

…Expression Conversion using Stack

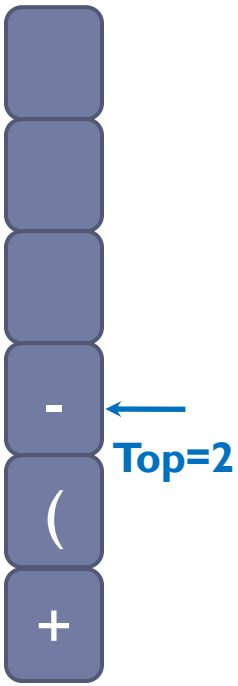
- Example Infix to Postfix: A + (B – C) * D / E
  - Scan B: As B is operand append directly to the Output String.



Output String: AB

…Expression Conversion using Stack

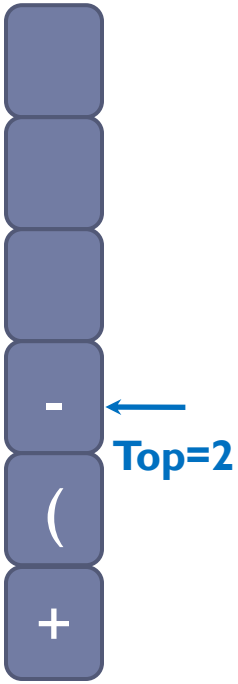
- Example Infix to Postfix: A + (B – C) * D / E
  - Scan –: As – is operator and top of the stack is left paranthesis, Push it onto the stack.



Output String: AB

…Expression Conversion using Stack

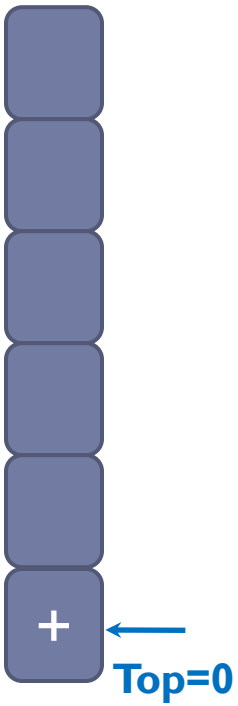
- Example Infix to Postfix: A + (B – C) * D / E
  - Scan C: As C is operand append directly to the Output String.



Output String: ABC

…Expression Conversion using Stack

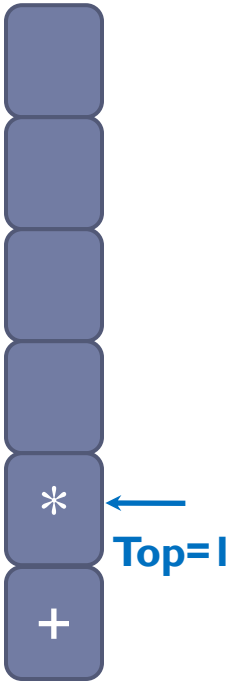
- Example Infix to Postfix: A + (B – C) * D / E
  - Scan): As) is right paranthesis, Pop all the operators until left paranthesis is found and, discard the parantheses pair.



Output String: ABC –

…Expression Conversion using Stack

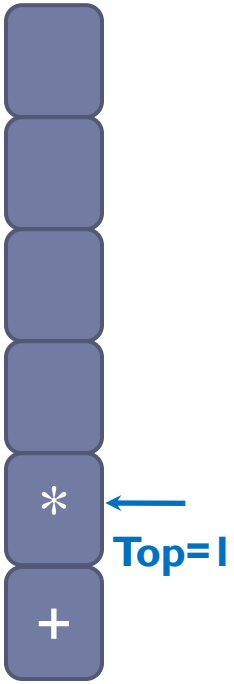
- Example Infix to Postfix: A + (B – C) * D / E
  - Scan *: As * is operator and has higher precedence than + at the top of the stack, Push it onto the stack.



Output String: ABC –

…Expression Conversion using Stack

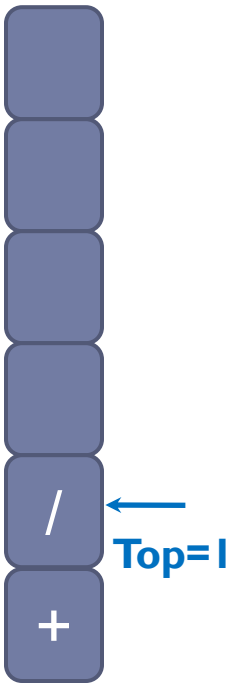
- Example Infix to Postfix: A + (B – C) * D / E
  - Scan D: As D is operand append directly to the Output String.



Output String: ABC –D

…Expression Conversion using Stack

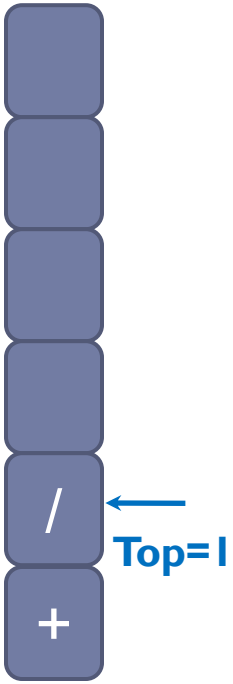
- Example Infix to Postfix: A + (B – C) * D / E
  - Scan /: As / is operator and has same precedence as * at the top of the stack, and as it is left associative, Pop the stack until a lower precedence operator is found (or stack is empty). Push it onto the stack.



Output String: ABC –D*

…Expression Conversion using Stack

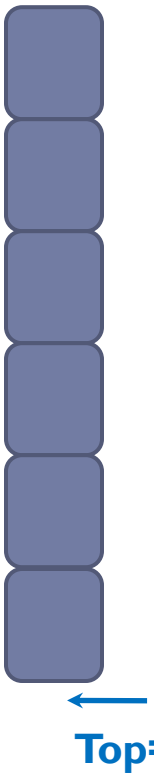
- Example Infix to Postfix: A + (B – C) * D / E
  - Scan E: As E is operand, append directly to the Output String.



Output String: ABC –D*E

…Expression Conversion using Stack

- Example Infix to Postfix: A + (B – C) * D / E
  - Expression ends: Pop all the operators from the stack and append them to the Output String.



Final Output String: ABC –D*E / +

Infix to Postfix using Stack

infix_postfix.c

…Expression Conversion using Stack

- Infix to Prefix Rules:

1. Reverse the infix expression.
2. Convert the expression into Postfix expression (with aforementioned algorithm). Notes: (1) Push the operator with same precedence. (2) Push right parenthesis (instead of left parenthesis) and when left parenthesis is scanned Pop till right parenthesis.
3. Reverse the output string.

- Example: A + B * C – D

1. Reverse: D – C * B + A
2. Convert into Postfix (considering above notes): DCB*A+-
3. Reverse: -+A*BCD ⇐ Prefix Expression

…Expression Conversion using Stack

- Class work:

Infix	Prefix	Postfix
A+B-C	?	?
A+B*C-D	?	?
(A+B) * (C+D)	?	?
A*BCD	?	?

Right associative

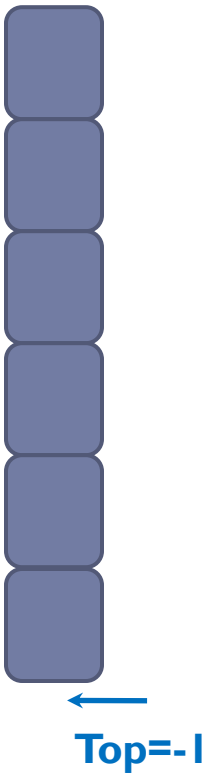
Evaluating Postfix Expression using Stack

Algorithm:

1. Scan the postfix expression from left to right.
2. If the char is a number, push it into the stack
3. If the char is an operator, pop two operands for the operator from stack. Evaluate the operands with the operator and push the result back onto the stack. Repeat from step 1.
4. When the expression ends, pop the only number remaining in the stack which is the final answer to be displayed.

…Evaluating Postfix Expression using Stack

- Example: 8 4 * 3 +

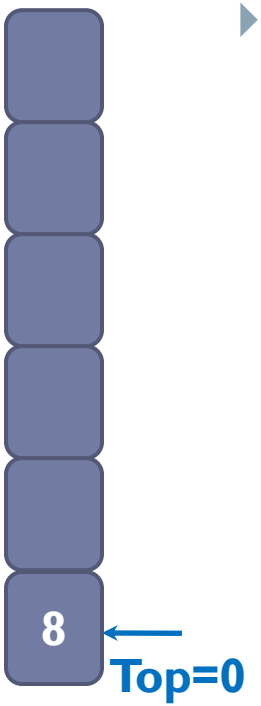


Initially: Stack Empty

- Let's start scanning the expression from Left to Right

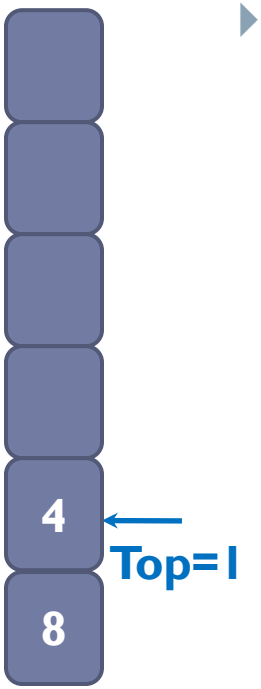
…Evaluating Postfix Expression using Stack

- Example: 8 4 * 3 +
  - Scan 8: As 8 is operand, push it on the stack



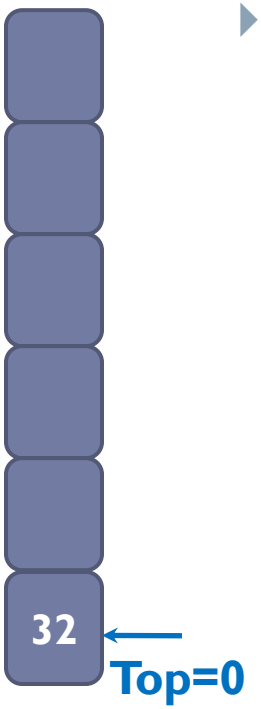
…Evaluating Postfix Expression using Stack

- Example: 8 4 * 3 +
  - Scan 4: As 4 is operand, push it on the stack



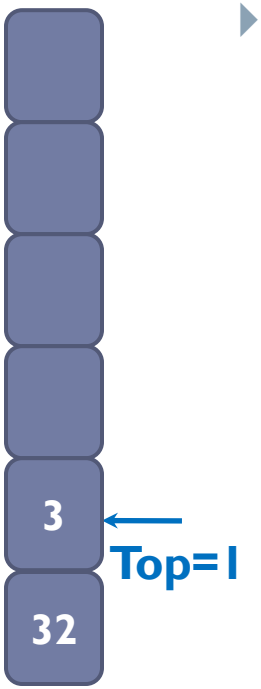
…Evaluating Postfix Expression using Stack

- Example: 8 4 * 3 +
  - Scan *: As * is operator, pop 4 and 8 from the stack. Evaluating 8 * 4=32. Push 32 back onto the stack.



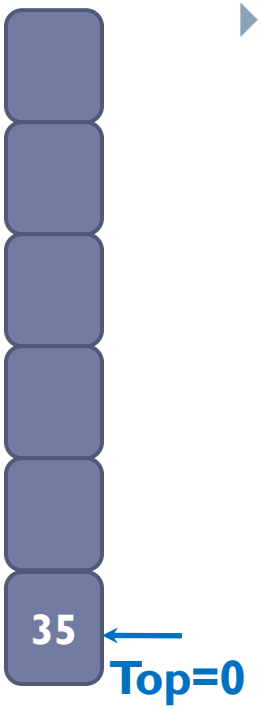
…Evaluating Postfix Expression using Stack

- Example: 8 4 * 3 +
  - Scan 3: As 3 is operand, push it on the stack



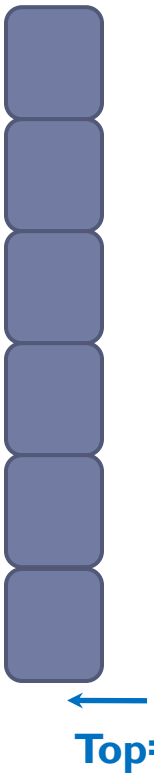
…Evaluating Postfix Expression using Stack

- Example: 8 4 * 3 +
  - Scan +: As + is operator, pop 3 and 32 from the stack. Evaluating 32 + 3=35. Push 35 back onto the stack.



…Evaluating Postfix Expression using Stack

- Example: 8 4 * 3 +
  - Expression ends: Pop the only remaining number 35 and display.



Final Result: 35

Evaluation of Postfix expression using Stack

postfix_eval.c

Recursion

- Recursion is a method of solving a problem where the solution depends on solutions to smaller instances of the same problem.
- A recursive procedure contains a procedure call to itself.
- A recursive procedure is divided into two parts:
  - Base case
  - Recursive case

Factorial

- Iteration:

```
fact=1
For i=n to 1:
    fact=fact*i
```

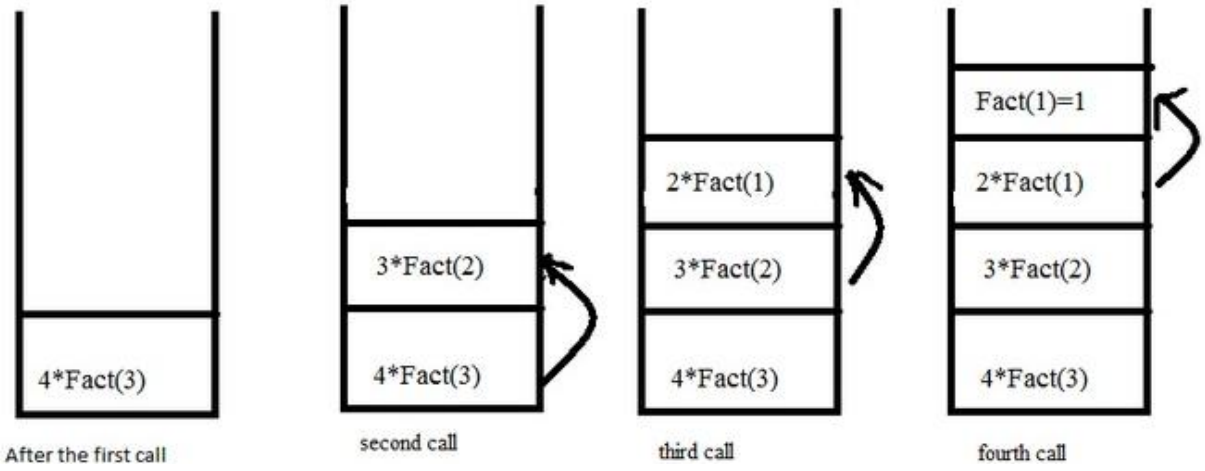
- Recursion:

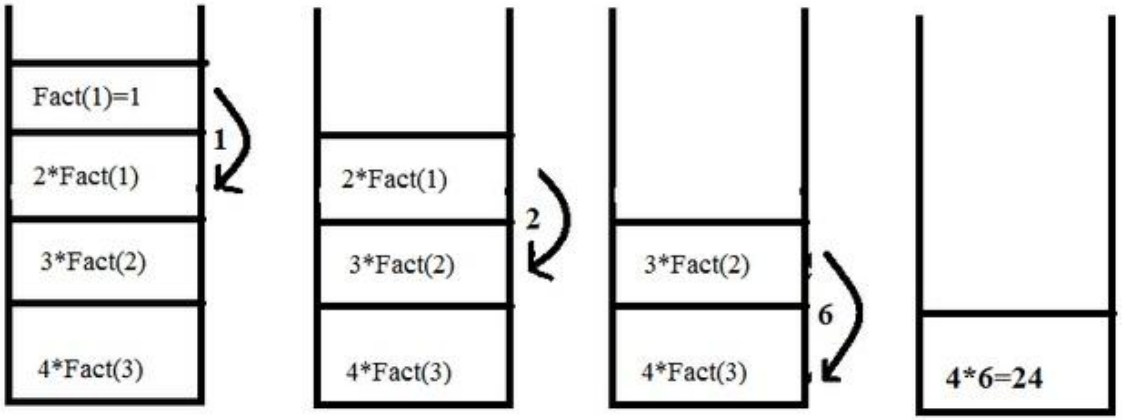
```
procedure fact (n)
        IF n==0 OR n==1:
                return 1
        ELSE:
                return n*fact (n-1)
```

Stack for Recursive Factorial Procedure

- Fact(4):

During function call, previous variables get stored onto the stack





Returning values from base case to the caller function

Factorial

rec_factorial.c

When Recursion?

- Iterative functions are typically faster than their recursive counterparts. So, if speed is an issue, you would normally use iteration.
- If the stack limit is too constraining, prefer iteration over recursion.
- Some procedures are very naturally programmed recursively, but unmanageable iteratively. Choosing recursion in this case is obvious.

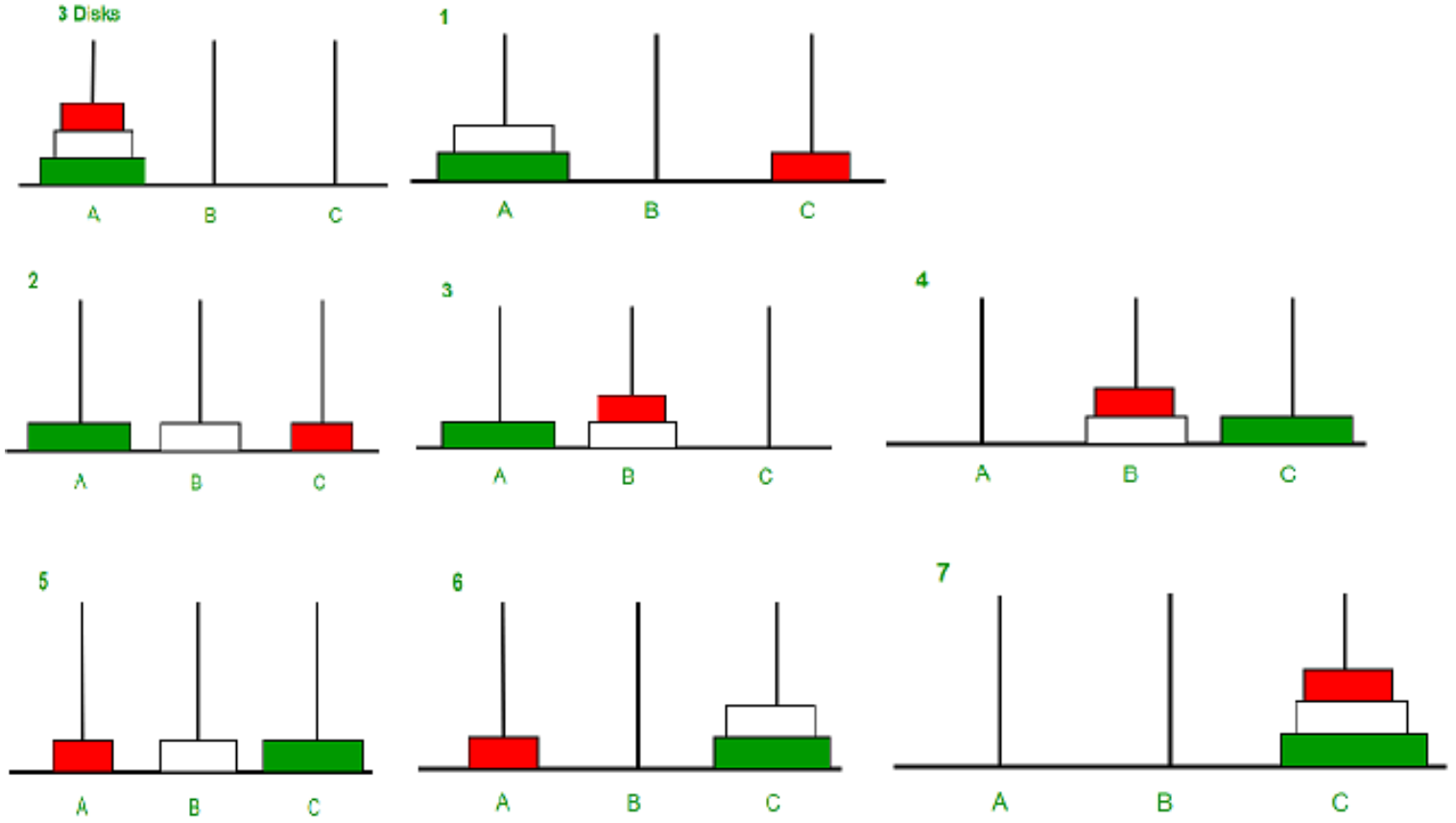
Tower of Hanoi

- Classic ancient problem:
  - n rings in increasing size and 3 poles.
  - Rings stacked on pole 1.
  - Goal: To move rings so that they are stacked on pole 3 ... BUT !
    - Can only move one ring at a time.
    - Can't put larger ring on top of smaller.

…Tower of Hanoi

Considering n=3 disks to be moved

3 Poles: Source A, Destination C and Auxiliary B



…Tower of Hanoi

```
Procedure Hanoi(n, source, dest, aux)
    IF n == 1:
        move the disk from source to dest
    ELSE:
        // Step 1 Move n-1 disks from source to aux
        Hanoi(n - 1, source, aux, dest)
        // Step 2 Move nth disk from source to dest
        move the disk from source to dest
        // Step 3 Move n-1 disks from aux to dest
        Hanoi(n - 1, aux, dest, source)
```

…Tower of Hanoi

Explanation for n=3

1. Hanoi (2, A, B, C)
    1. Hanoi(1,A,C,B)
        1. Moving the disk 1 from A to C
    2. Move 2nd disk from A to B
    3. Hanoi(1,C,B,A)
        1. Moving the disk 1 from C to B
2. Move 3rd disk from A to C
3. Hanoi(2,B,C,A)
    1. Hanoi(1,B,A,C)
        1. Move the disk 1 from B to A
    2. Move 2nd disk from B to C
    3. Hanoi(1,A,C,B)
        1. Move the disk 1 from A to C

Summary

- What is stack and how operations are performed
- Applications of stack
- Conversion between different types of operations
- Evaluation of Postfix expression
- Recursion vs Iteration
- Recursion to solve various problems

Assignments

- Pusedocode to check a palindrome string with stack.
- Psuedocode to convert infix expression into prefix.
- Convert the following expressions into prefix and postfix using stack: (1) a*(b-c*d)+e (2) a+((b-c)*d)/e
- Psuedocode to evaluate a postfix expression.
- Evaluate the following expressions using stack: (1) 34+86-* (2) 222$$3*2+2*
- Psuedocode for Fibonacci series with recursion.