K.R. MANGALAM UNIVERSITY
THE COMPLETE WORLD OF EDUCATION

Data Structure

ENCS205

School of Engineering & Technology (SOET)
K.R. MANGALAM University

UNIT-2
Session 26: Recursion & Stack Memory ,STACK APPLICATIONS

Recap

- Type of arithmetic Expression
- Expression Evolutions
- Operator Precedence
- Conversion

Sessions 26 Outlook

- APPLICATIONS
  - Reverse a Data
  - Function Calls
  - Tower of Hanoi Problem
- Conceptualize
- Assignment
- QUESTIONS With Solutions

Application of Stack in Data Structure are :

- Evaluation of Arithmetic Expressions
- Backtracking
- Delimiter Checking
- Reverse a Data
- Processing Function Calls

Reverse a Data

Approach to reverse a string using stack

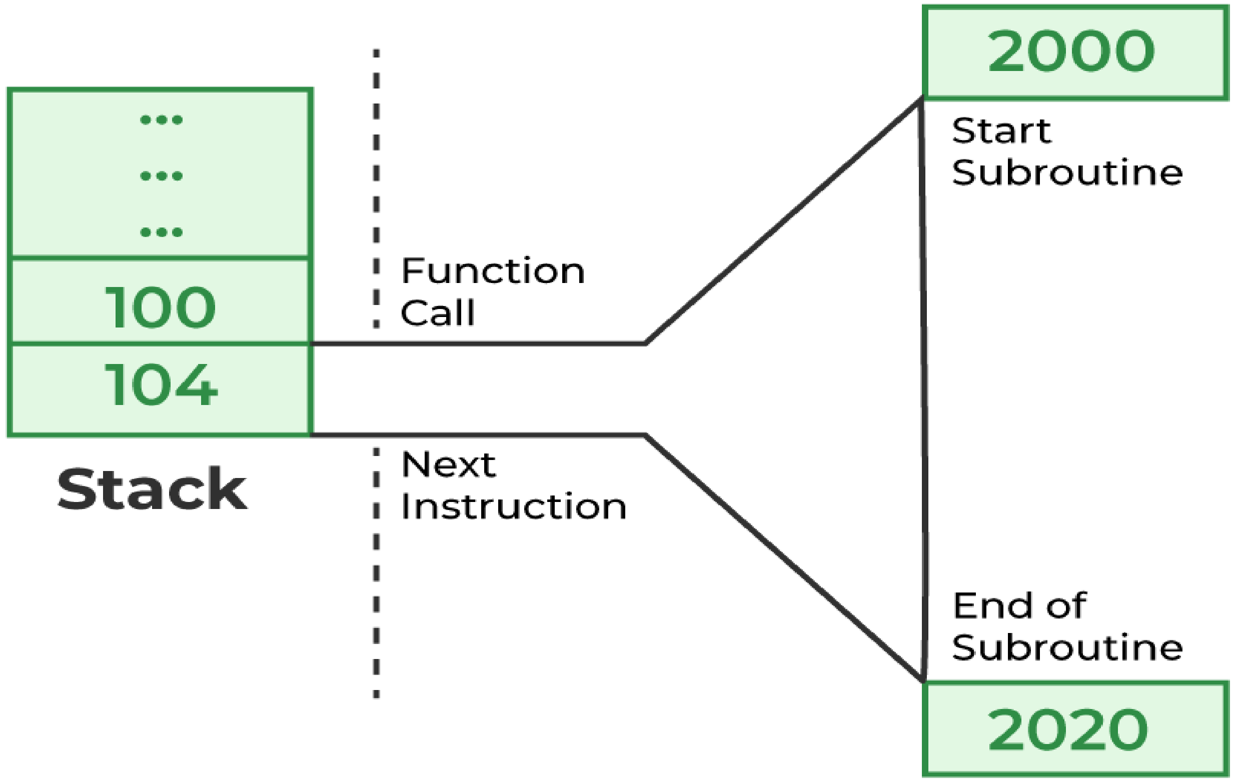
- Push the elements/characters of the string individually into the stack of data type characters.
- Pop the elements/characters individually from the Stack until the stack becomes vacant.
- Add a popped component to the character array.
- Convert character array to string.
- Return turned around the string.

Reverse a Data

Very easy application is to reverse the order of a given array of items.

```
Algorithm reverse(string)
1. Initialize an empty stack.
2. While not the end of string
3.         read a character
4.         push it on to the stack
5. End while
6. While the stack is nonempty
7.         pop a character
8.         print it
9. End while
```

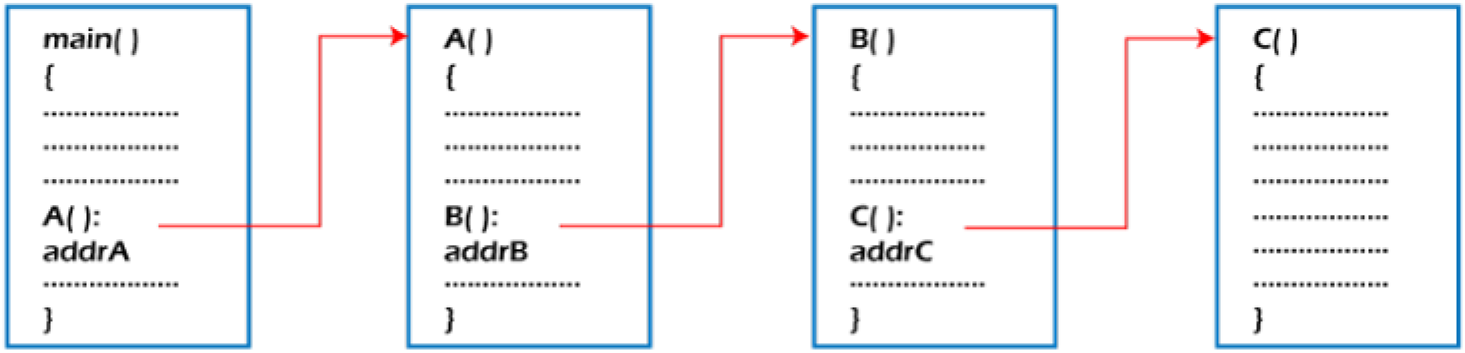
Function Calls



https://www.geeksforgeeks.org/function-call-stack-in-c/

Function Calls

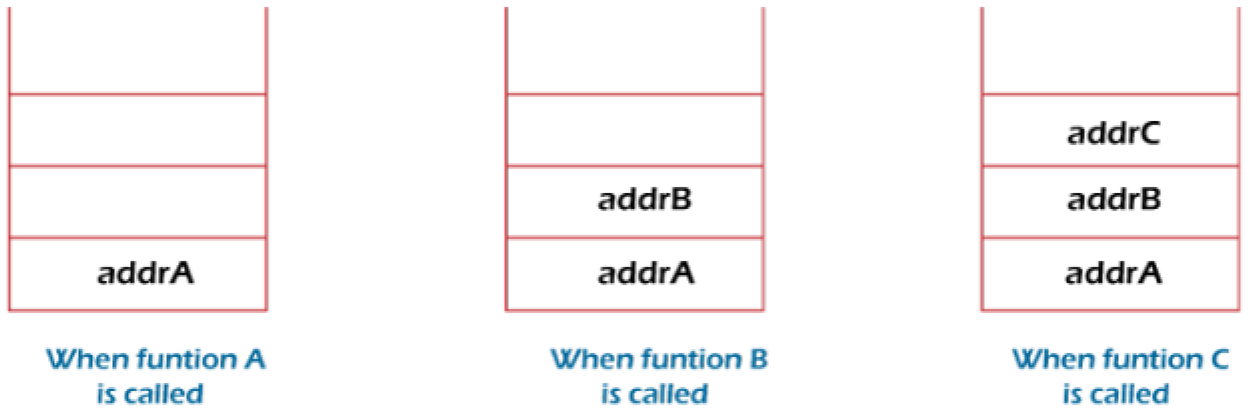
Stack plays an important role in programs that call several functions in succession. Suppose we have a program containing three functions: A, B, and C. function A invokes function B, which invokes the function C.


main( ) { ... A( ): addrA ... }
A( ) { ... B( ): addrB ... }
B( ) { ... C( ): addrC ... }
C( ) { ... }

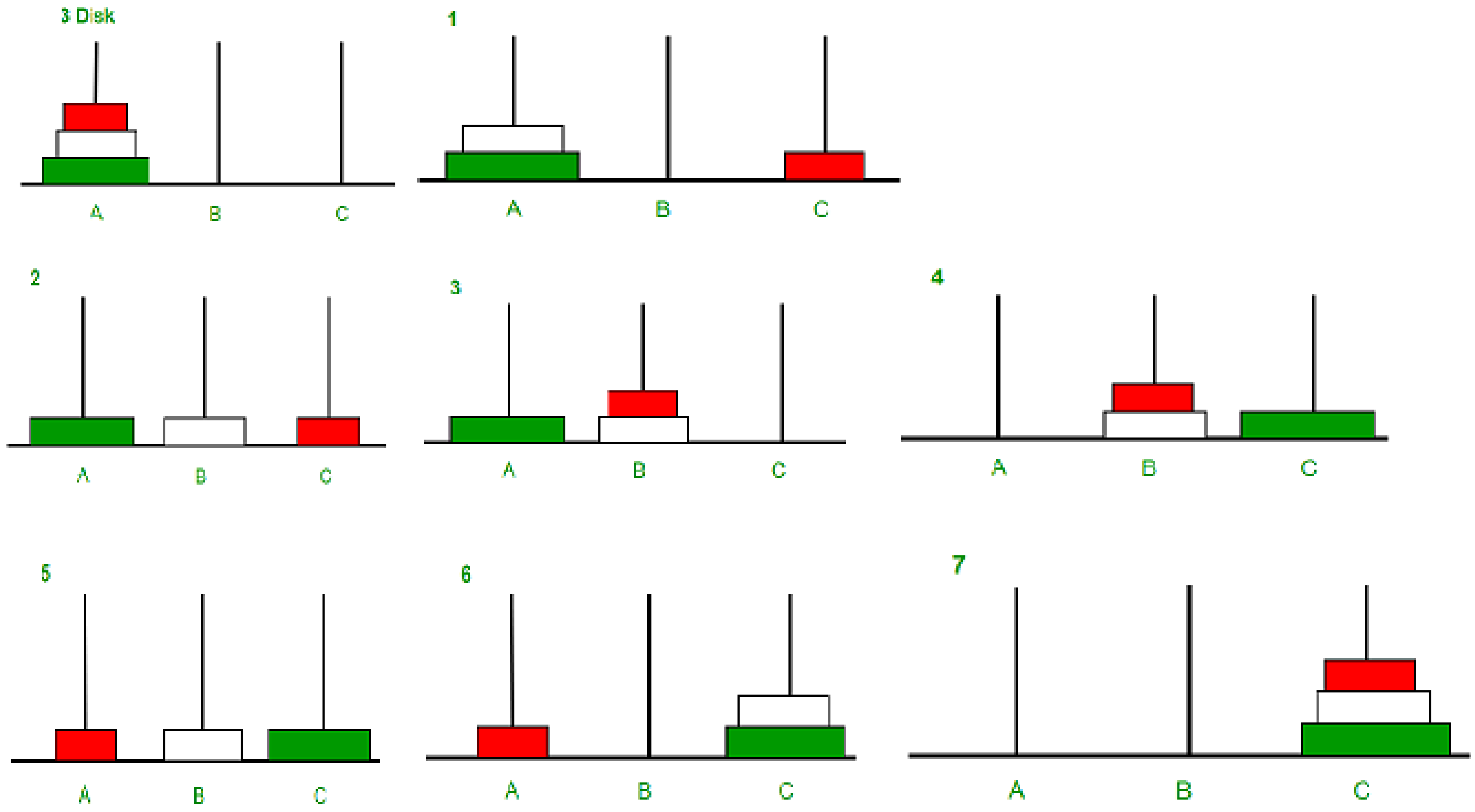
Function call

Function Calls

Consider addrA, addrB, addrC be the addresses of the statements to which control is returned after completing the function A, B, and C, respectively.



Tower of Hanoi Problem



https://www.geeksforgeeks.org/c-program-for-tower-of-hanoi//

Tower of Hanoi Problem

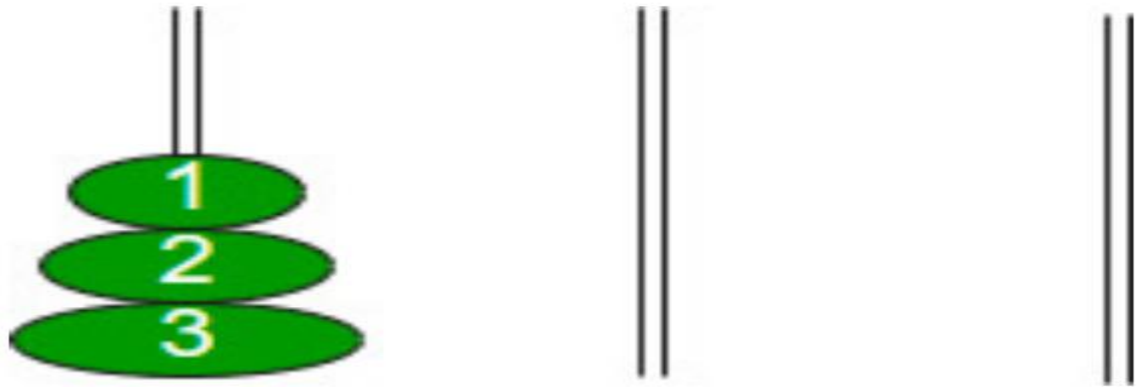
1. Calculate the total number of moves required i.e. "pow(2, n) - 1" here n is number of disks.
2. If number of disks (i.e. n) is even then interchange destination pole and auxiliary pole.
3. for i = 1 to total number of moves:

   if i%3 == 1: legal movement of top disk between source pole and destination pole
   if i%3 == 2: legal movement top disk between source pole and auxiliary pole
   if i%3 == 0: legal movement top disk between auxiliary pole and destination pole

Example:

Let us understand with a simple example with 3 disks:

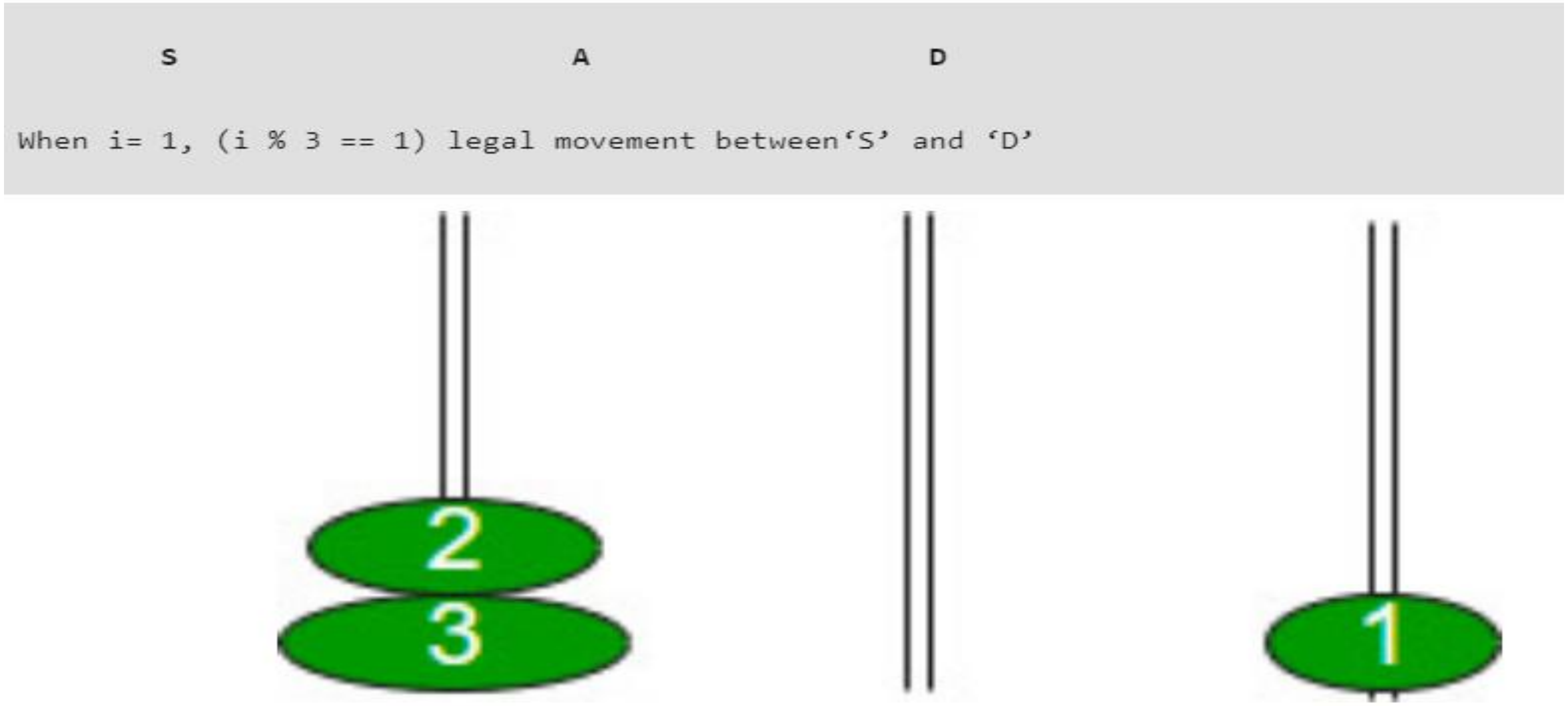
So, total number of moves required = 7



Example:

Let us understand with a simple example with 3 disks:

So, total number of moves required = 7



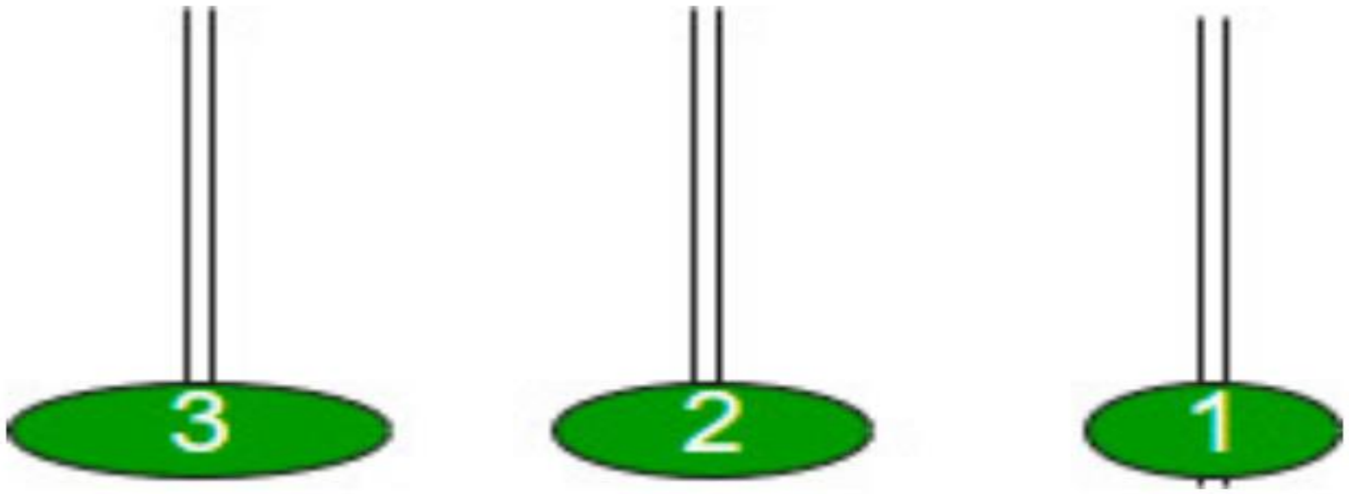
When i= 1, (i % 3 == 1) legal movement between'S' and 'D'

Example:

Let us understand with a simple example with 3 disks:

So, total number of moves required = 7

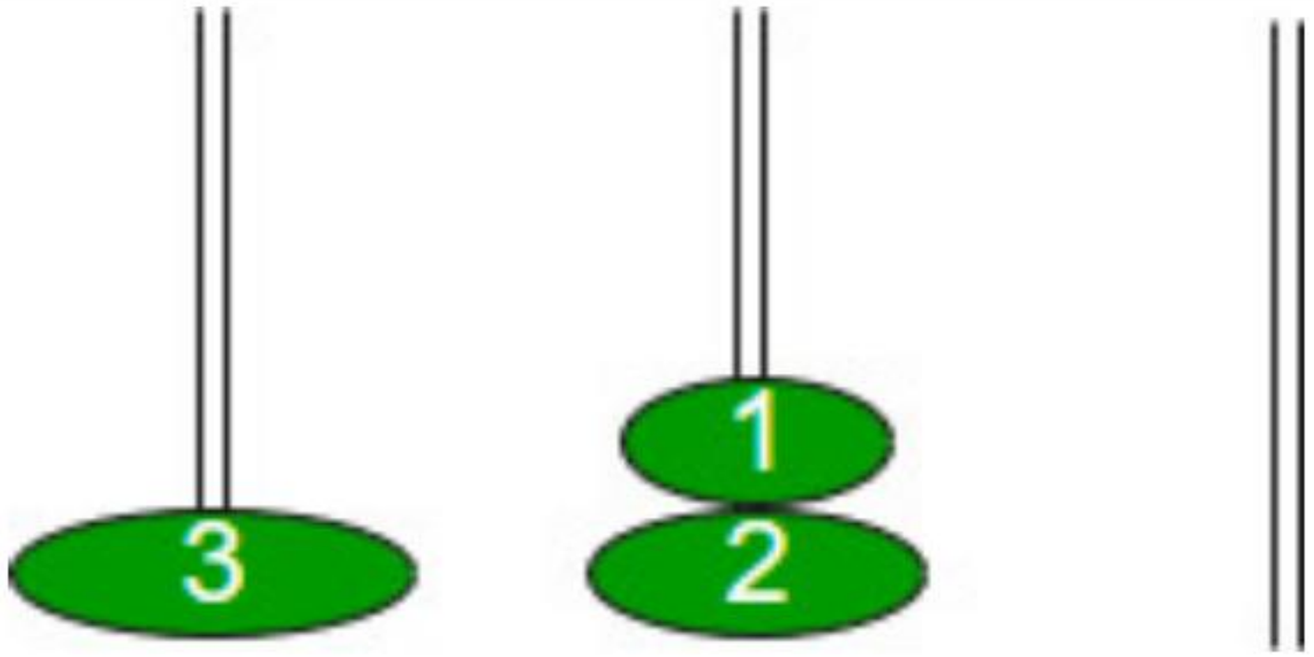
```
When i = 2,  (i % 3 == 2) legal movement between 'S' and 'A'
```



Example:

Let us understand with a simple example with 3 disks:
So, total number of moves required = 7

```
When i = 3, (i % 3 == 0) legal movement between 'A' and 'D' '
```

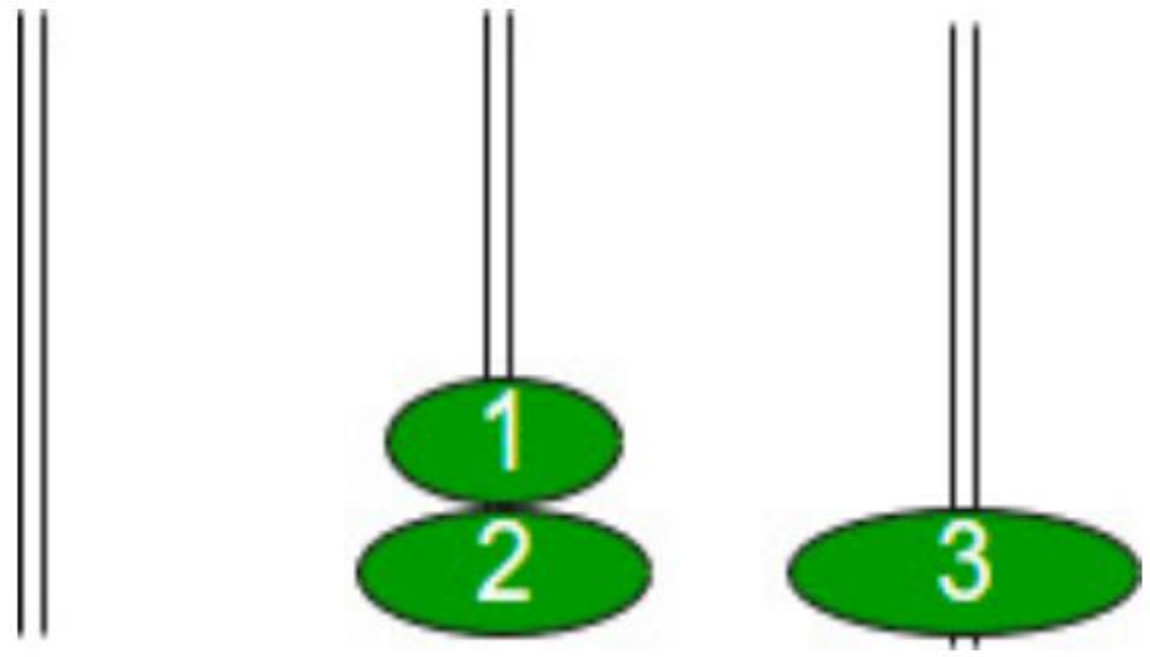


Example:

Let us understand with a simple example with 3 disks:

So, total number of moves required = 7

```
When i = 4, (i % 3 == 1) legal movement between 'S' and 'D'
```

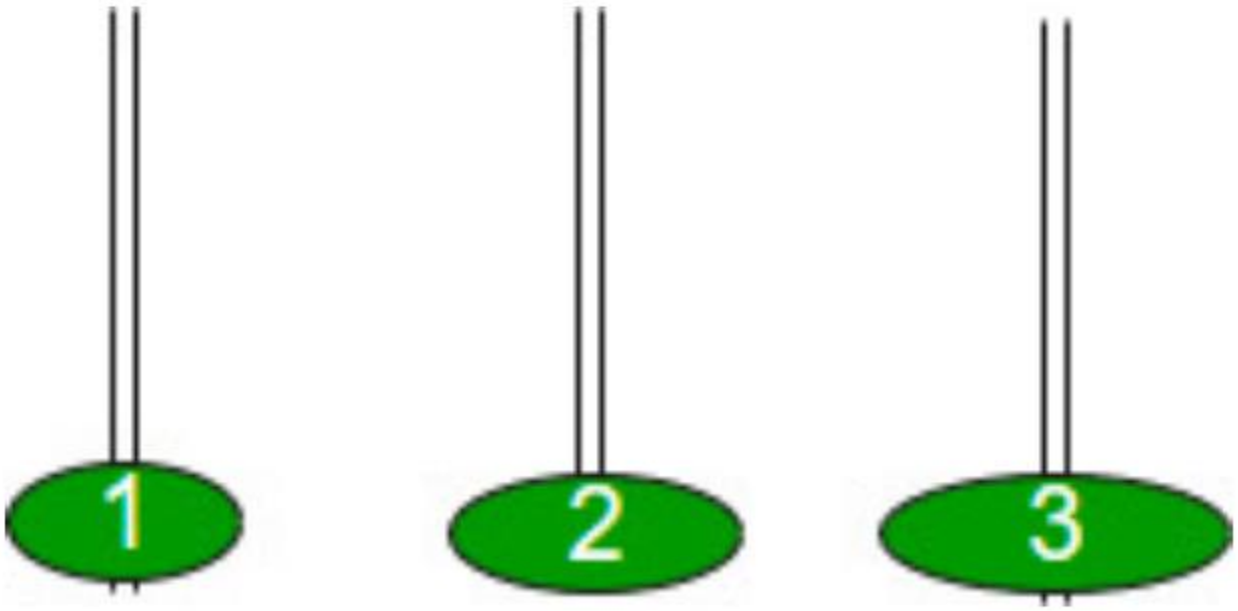


Example:

Let us understand with a simple example with 3 disks:

So, total number of moves required = 7

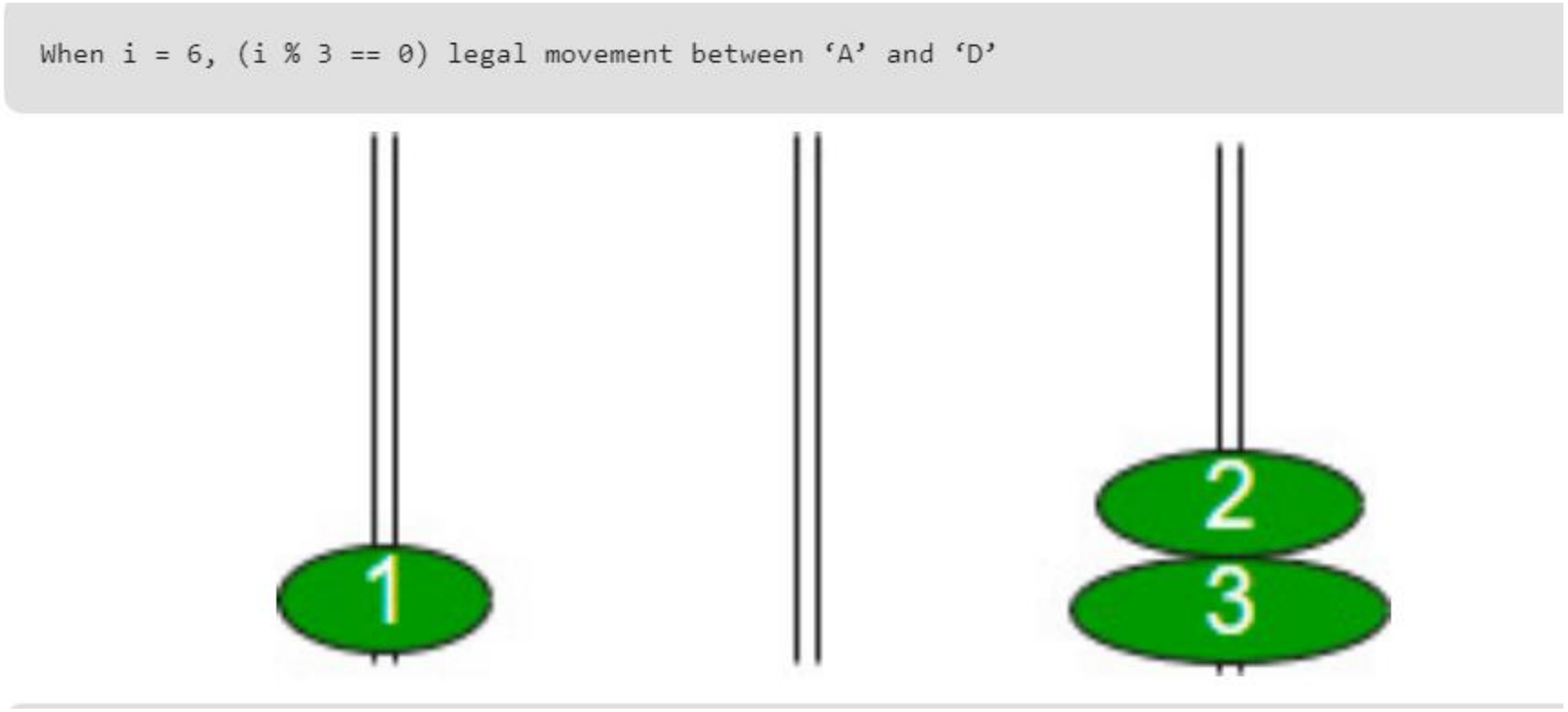
```
When i = 5, (i % 3 == 2) legal movement between 'S' and 'A'
```



Example:

Let us understand with a simple example with 3 disks:

So, total number of moves required = 7



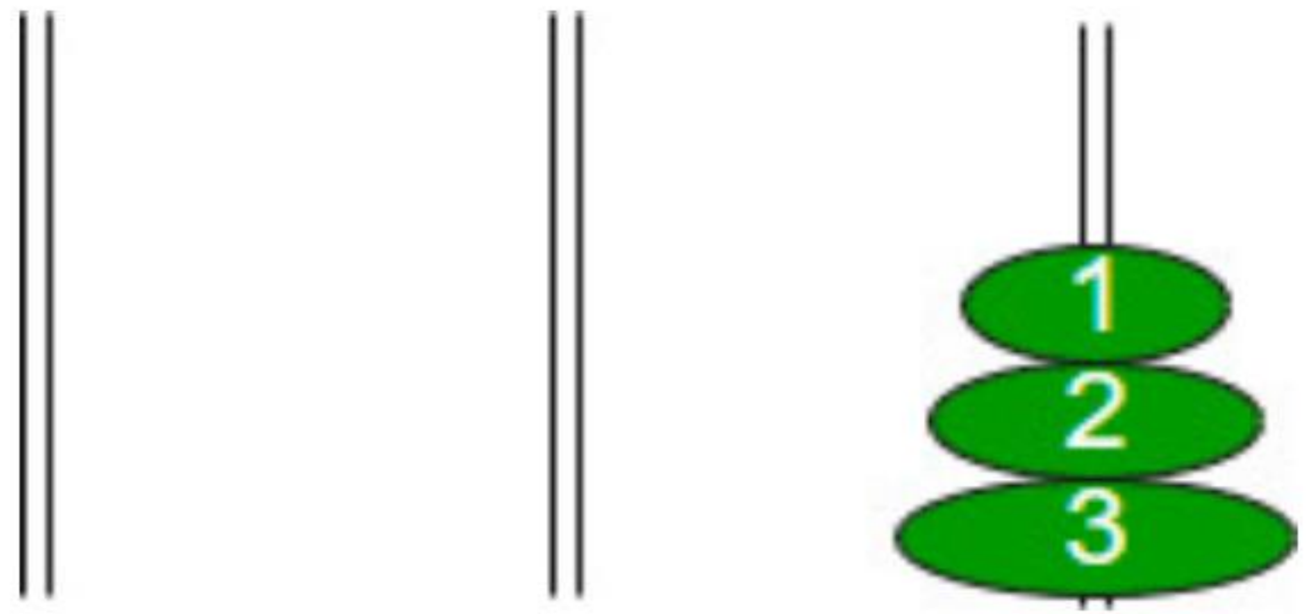
```
When i = 6, (i % 3 == 0) legal movement between 'A' and 'D'
```

Example:

Let us understand with a simple example with 3 disks:

So, total number of moves required = 7

```
When i = 7, (i % 3 == 1) legal movement between 'S' and 'D'
```



Code for Tower of Hanoi

```
void towerOfHanoi(int n, char from_rod, char to_rod,
                  char aux_rod)
{
    if (n == 0) {
        return;
    }
    towerOfHanoi(n - 1, from_rod, aux_rod, to_rod);
    cout << "Move disk " << n << " from rod " << from_rod
         << " to rod " << to_rod << endl;
    towerOfHanoi(n - 1, aux_rod, to_rod, from_rod);
}
```

Complete code here

Conceptualize

1. What does LIFO stand for in the context of stack data structure?
   a) Last In, First Out
   b) Last In, Last Out
   c) First In, First Out
   d) First In, Last Out

2. In an array-based implementation of a stack, which end is used for push and pop operations?
   a) Front
   b) Middle
   c) Top
   d) Bottom

Conceptualize

3. What is the significance of infix, postfix, and prefix notations in stack-based evaluations?
   a) They represent different types of data structures.
   b) They are used to convert expressions to binary trees.
   c) They determine the order of operations in arithmetic expressions.
   d) They are used for heap data structure operations.

Conceptualize

Q4 Which of the following is an example of a postfix expression?

a) A + B * C
b) (A + B) * C
c) A * B + C
d) A + B + C

Q5 Which problem is often solved using the Tower of Hanoi problem?

a) Sorting
b) Searching
c) Recursion
d) Graph Traversal

Conceptualize

Answers:

1.a) Last In, First Out

2.c) Top

3.c) They determine the order of operations in arithmetic expressions.

4.c) A * B + C

5.c) Recursion

Try yourself

1. What is the equivalent infix expression of the following postfix expression?

   M, N, O, +, *, P, /, Q, R, S, T, /, +, *, –

   a) N*(M+Q)/Q-P*(S+R/T)
   b) (((M*(N+O))/P)-(Q*(R+(S/T))))
   c) O * (M + N)/P – Q * (R + S/T)
   d) M * (N + O)/Q – P * (R/S + T)

2. What is the postfix representation of the following infix expression?

(A + B) * C – D * E / F

a) A B + C * D E * F – /
b) A B * C + D E * F / –
c) A B + C – D E * F / *
d) A B + C * D E * F / –

-

3. The result evaluating the postfix expression 10 5 + 60 6 / * 8 – is

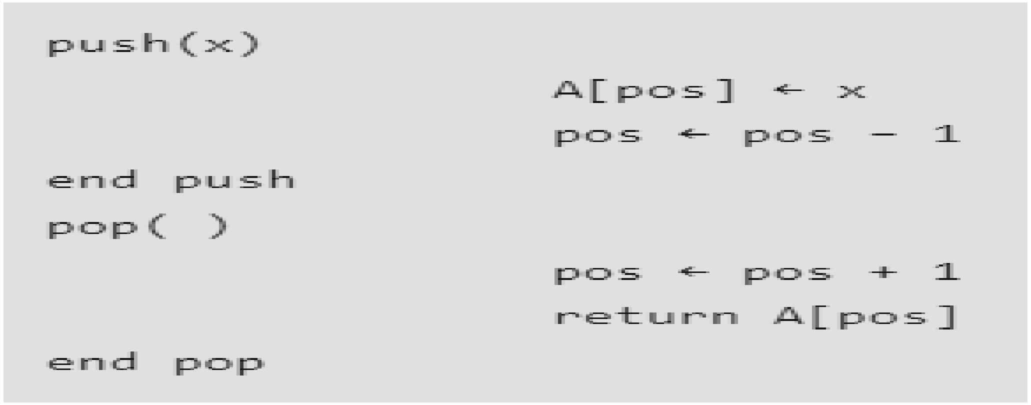
a) 284
b) 213
c) 142
d) 71

-

4. The five items P,Q,R,S and T are pushed in a stack, one after the other starting from P. The stack is popped four times and each element is inserted in a queue. Then two elements are deleted from the queue and pushed back on the stack. now one item is popped from the stack. The popped item is:

a) P
b) R
c) Q
d) S

5.A stack is implemented with an array of ‘A [0..N – 1]’ and a variable ‘pos’. The push and pop operations are defined by the following code.



```
push(x)
                A[pos] ← x
                pos ← pos − 1
end push
pop( )
                pos ← pos + 1
                return A[pos]
end pop
```

Which of the following will initialize an empty stack with capacity N for the above implementation?

pos ← -1

pos ← 0

pos ← 1

pos ← N – 1

6. Consider the following postfix expression with single digit operands:

6 2 3 * / 4 2 * + 6 8 * –

The top two elements of the stack after second * is evaluated, are:

a) 6, 3
b) 8, 1
c) 8, 2
d) 6, 2

7. What is the outcome of the prefix expression +, -, *, 3, 2, /, 8, 4, 1?

a) **12**
b) **5**
c) **11**
d) **4**

8. Which of the following applications may use a stack?

(a) Parenthesis balancing program

(b) Process scheduling operating system

(c) Conversion of infix arithmetic expression to postfix form

a) (a) and (b)
b) (b) and (c)
c) (a) and (c)
d) (a), (b) and (c)

9. A stack can be implemented using queue, but then we need to use at least:

a) 3 queues
b) 2 queues
c) only one queue is sufficient
d) none of the options

Answers

1. *Answer: Option 2 : (((M * (N + O)) / P) – (Q * (R + (S / T))))*
2. *Answer: Option 4 : A B + C * D E * F / –*
3. *Answer: Option 3 : 142*
4. *Answer: Option 4 : S*
5. *Answer: Option 4 : pos ← N – 1*
6. *Answer: Option 2 : 8, 1*
7. *Answer: Option 2 : 5*
8. *Answer: Option 3 : (a) and (c)*
9. *Answer: Option 2 : 2 queues*

Assignment

- Let S be a stack size of 4>=1 and it is initially empty. Suppose we push the numbers 1,2,3,4 and then perform 4 pop operations. Let one push operation takes 5ns; one pop operation takes 5ns; the time between the end of one such stack operation and start of next operation is 2ns.

  The stack life of a particular element p>=1 is defined as time elapsed from end of push (P) to the start of operation that removes (P) from the stack.The average stack life of an element of this stack is......................(in ns).

- Assume stack A has the entries p, q and r (with p on top and r on bottom). Initially stack B is empty. An entry popped out of stack A can be printed immediately or pushed to stack B. A entry popped out of stack B can only be printed. The least number of stack permutations of input sequence that start with a particular letter

- Consider the following infix expression which is to be converted to postfix expression using stack.

$$(((P+Q)*(R+S))/T)+(A*(B+C))$$

What is the size of stack?

- ➢ Choose the equivalent prefix form of the following expression

(a+(b-c))*((d-e)/(f+g-h))

- ➢ A stack of size (1 to N) and the initial position of top pointer is 0. Get (i, S) is a routine to get ith element from stack 'S' with respect to top. Then, what is the underflow condition on stack to perform get() operation.

https://gateoverflow.in/115691/stack-underflow

Questions And Answers (Stack)

What is the time efficiency of the push() and pop() operations of stacks?

The push() and pop() methods implemented in stacks take constant time to execute. i.e. the time to execute is very quick and is not dependent on the number of items on the stack. In Big O notation this is represented as O(1).

What is a stack and how does it work?

Answer: A stack is a linear data structure that follows the Last In, First Out (LIFO) principle. It means the last element added to the stack is the first one to be removed. Operations on a stack are performed at one end, typically called the top. Common operations include push (to add an element), pop (to remove the top element), and peek (to view the top element without removing it).

How do you implement a stack using an array?

Answer: To implement a stack using an array, we typically use an array with a fixed size and a variable to keep track of the top element's index. When pushing an element, we increment the top index and insert the element at that position. When popping an element, we retrieve the element at the top index and decrement the top index.

What are infix, postfix, and prefix notations? Can you provide examples of each?

Answer: Infix notation: It is the standard way of writing arithmetic expressions, where operators are placed between operands. Example: (2 + 3) * 4

Postfix notation: Also known as Reverse Polish Notation (RPN), operators are placed after their operands. Example: 2 3 + 4 *

Prefix notation: Also known as Polish Notation, operators are placed before their operands. Example: * + 2 3 4

What factors determine the time complexity of an algorithm?

Answer: Time complexity depends on the number of basic operations performed by the algorithm as the size of the input increases. Common factors influencing time complexity include loop iterations, recursion depth, and the number of comparisons or assignments.

How can stacks be used to implement function calls in programming languages?

Answer: Stacks are used to manage function calls and their local variables in programming languages. When a function is called, its parameters and return address are pushed onto the stack. Local variables are allocated space on the stack, and when the function completes execution, its frame is popped off the stack.

How are recursion and stacks related?

Answer: Recursion involves a function calling itself, which leads to a chain of function calls. Each function call creates a new activation record (or stack frame), which is stored on the call stack. As recursion progresses, the stack grows with each function call and shrinks as functions return.

What factors influence the space complexity of a stack-based algorithm?

Answer: Space complexity depends on the amount of memory required by the algorithm, including variables, data structures, and recursion depth. For stack-based algorithms, space complexity is often influenced by the size of the call stack, especially in recursive algorithms.

How would you analyze the space complexity of an algorithm for converting infix to postfix expressions?

Answer: The space complexity of such an algorithm depends on the usage of additional data structures, such as stacks, to store operands, operators, and intermediate results. If the algorithm uses only a stack to convert the expression, the space complexity would be O(n), where n is the length of the input expression. However, if additional data structures are used, such as arrays or linked lists, the space complexity may vary accordingly

How do you handle stack underflow in an array-based implementation?

Answer: Stack underflow occurs when popping from an empty stack. To handle this, we can check if the stack is empty before performing a pop operation. If the stack is empty and a pop operation is attempted, it should raise an error or return a special value indicating underflow.

References:

https://practicepaper.in/gate-cse/stack
https://www.geeksforgeeks.org/introduction-to-stack-data-structure-and-algorithm-tutorials/?ref=lbp
https://www.javatpoint.com/data-structure-stack
https://www.fullstack.cafe/interview-questions/stacks
https://www.interviewkickstart.com/learn/stack-data-structure
https://www.youtube.com/watch?v=HRDHgKrYHgU
https://www.youtube.com/watch?v=Ci9jZSRn3os
https://www.udemy.com/topic/data-structures/

Review

- Understand the LIFO principle.
- Practice implementing stack operations.
- Explore various applications and understand their use cases.
- Analyze the time complexity for each operation.
- Experiment with different implementations (arrays vs. linked lists).

THANK
YOU