E- Lecture
Data Structures and Algorithms

By
H R Choudhary (Asstt. Professor)
Department of CSE
Engineering College Ajmer

Topics to be covered
Stack

What do you mean by Stack ?

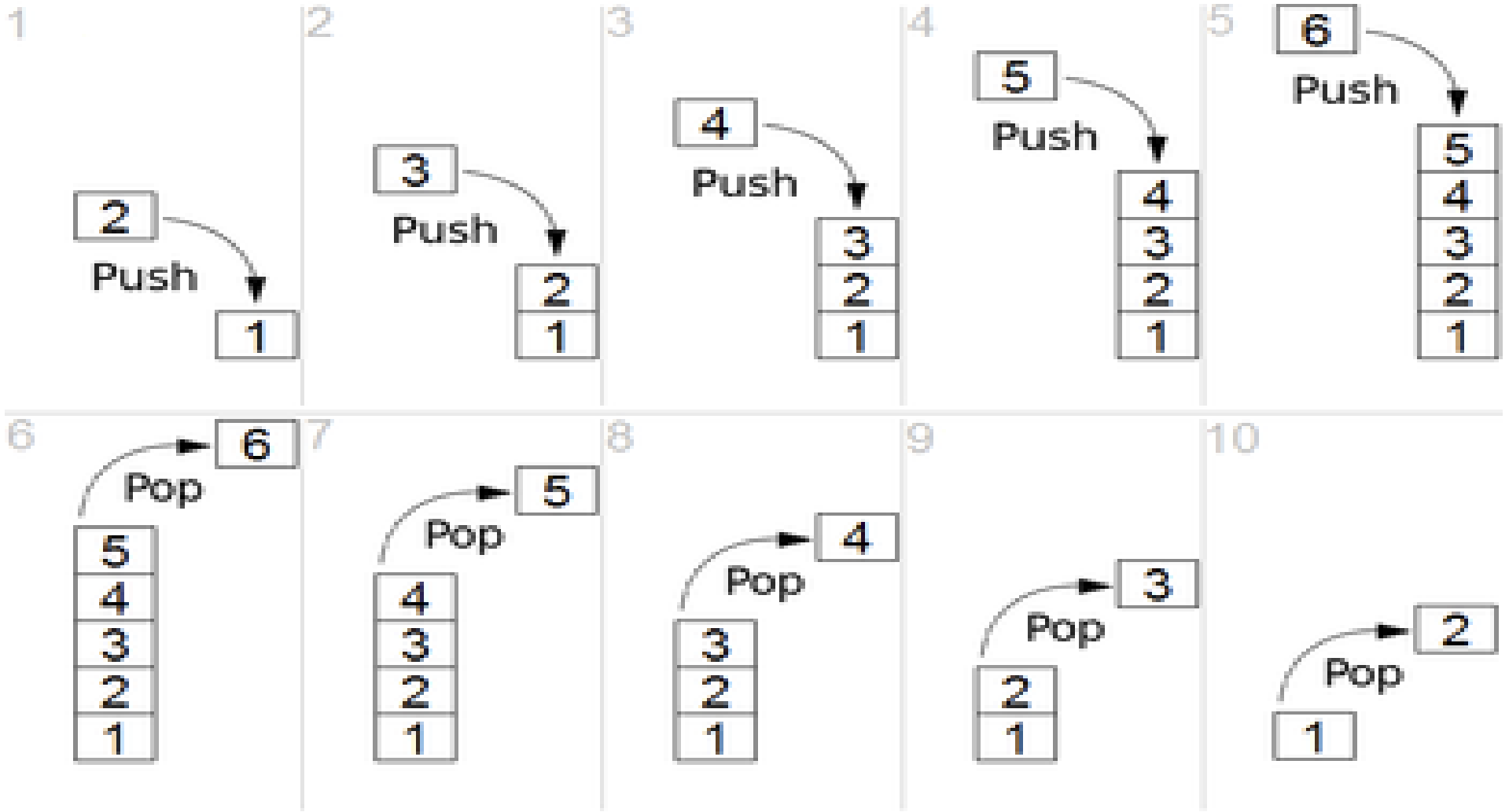
Stack Opertaion.

Different applications of Stack.

What do you mean by Stack ?

A stack is an Abstract Data Type (ADT), commonly used in most programming languages. It is named stack as it behaves like a real-world stack, for example – a deck of cards or a pile of plates, etc.

Example..


1
2
Push
1
2
3
Push
2
1
3
4
Push
3
2
1
4
5
Push
4
3
2
1
5
6
Push
5
4
3
2
1
6
6
Pop
5
4
3
2
1
7
5
Pop
4
3
2
1
8
4
Pop
3
2
1
9
3
Pop
2
1
10
2
Pop
1

Example..

LIFO (Last In First Out)

Stack..

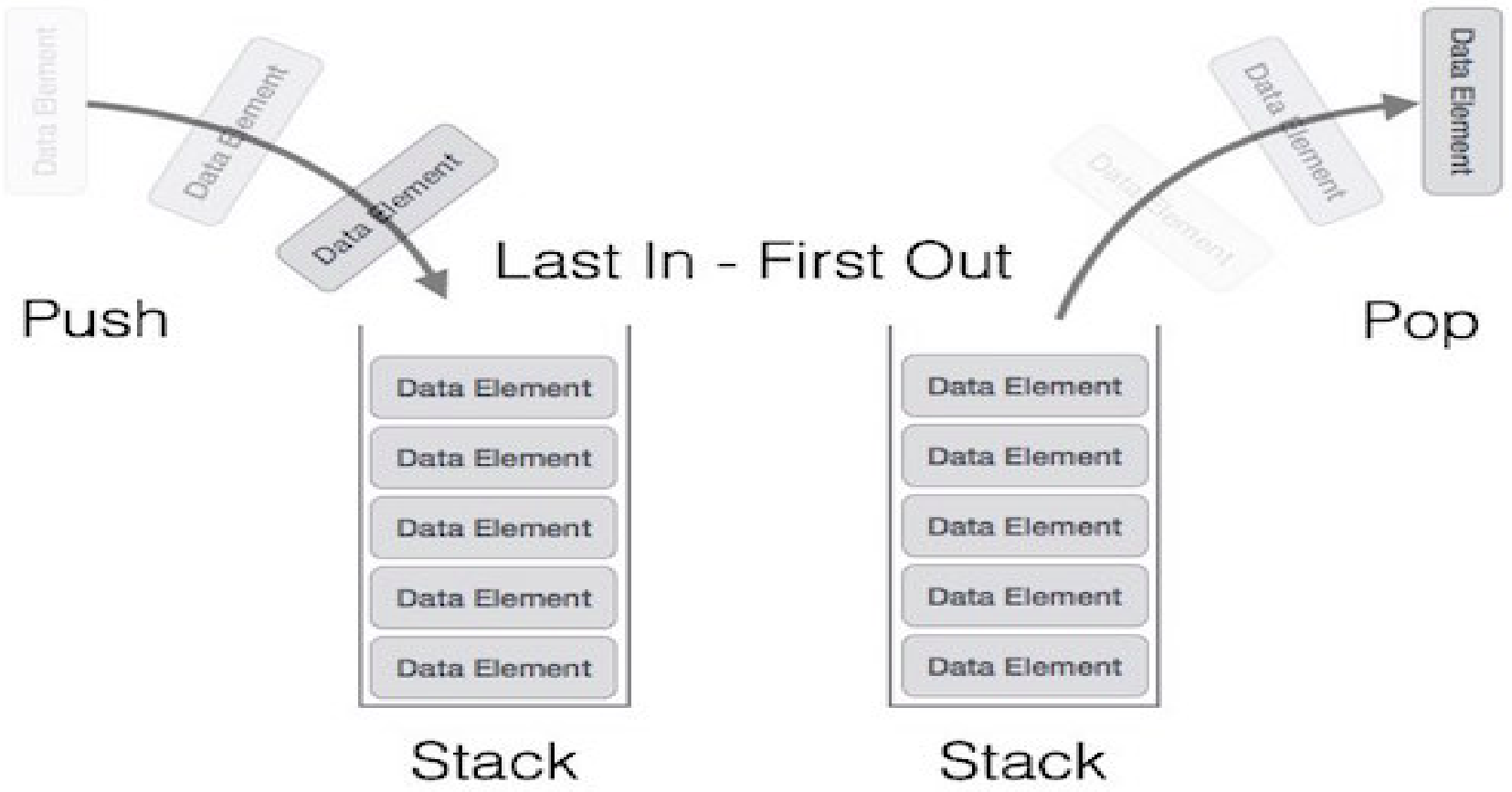
A real-world stack allows operations at one end only. For example, we can place or remove a card or plate from the top of the stack only. Likewise, Stack ADT allows all data operations at one end only. At any given time, we can only access the top element of a stack.

This feature makes it LIFO data structure. LIFO stands for Last-in-first-out. Here, the element which is placed (inserted or added) last, is accessed first. In stack terminology, insertion operation is called PUSH operation and removal operation is called POP operation.

Stack..

Stack Representation:

The following diagram depicts a stack and its operations −



Stack..

A stack can be implemented by means of Array, Structure, Pointer, and Linked List. Stack can either be a fixed size one or it may have a sense of dynamic resizing. Here, we are going to implement stack using arrays, which makes it a fixed size stack implementation.

Stack Opertaion

Basic Operations:

Stack operations may involve initializing the stack, using it and then de-initializing it. Apart from these basic stuffs, a stack is used for the following two primary operations −

- push() − Pushing (storing) an element on the stack.
- pop() − Removing (accessing) an element from the stack.

When data is PUSHed onto stack.

To use a stack efficiently, we need to check the status of stack as well. For the same purpose, the following functionality is added to stacks −

- peek() − get the top data element of the stack, without removing it.
- isFull() − check if stack is full.
- isEmpty() − check if stack is empty.

Stack Opertaion..

At all times, we maintain a pointer to the last PUSHed data on the stack. As this pointer always represents the top of the stack, hence named top. The top pointer provides top value of the stack without actually removing it.

First we should learn about procedures to support stack functions −

peek()

Algorithm of peek() function −

```
begin procedure peek
        Return stack[top]
end procedure
```

Stack Opertaion..

Implementation of peek() function in C programming language −

```
int peek() {
return stack[top];
}
```

Stack Opertaion..

- **isfull()**

Algorithm of isfull() function −

```
begin procedure isfull
   if top equals to MAXSIZE
      return true
   else
      return false
   endif
end procedure
```

Stack Opertaion..

Implementation of isfull() function in C programming language −

```
bool isfull() {
     if(top == MAXSIZE)
        return true;
    else
      return false;
}
```

Stack Opertaion..

- **isempty()**

Algorithm of isempty() function −

```
begin procedure isempty
   if top less than 1
      return true
   else
      return false
   endif
end procedure
```

Stack Opertaion..

Implementation of isempty() function in C programming language is slightly different. We initialize top at -1, as the index in array starts from 0. So we check if the top is below zero or -1 to determine if the stack is empty. Here's the code −

```
bool isempty() {
   if(top == -1)
         return true;
    else
      return false;
}
```

Stack Opertaion..

Push Operation:

The process of putting a new data element onto stack is known as a Push Operation. Push operation involves a series of steps −

Step 1 − Checks if the stack is full.

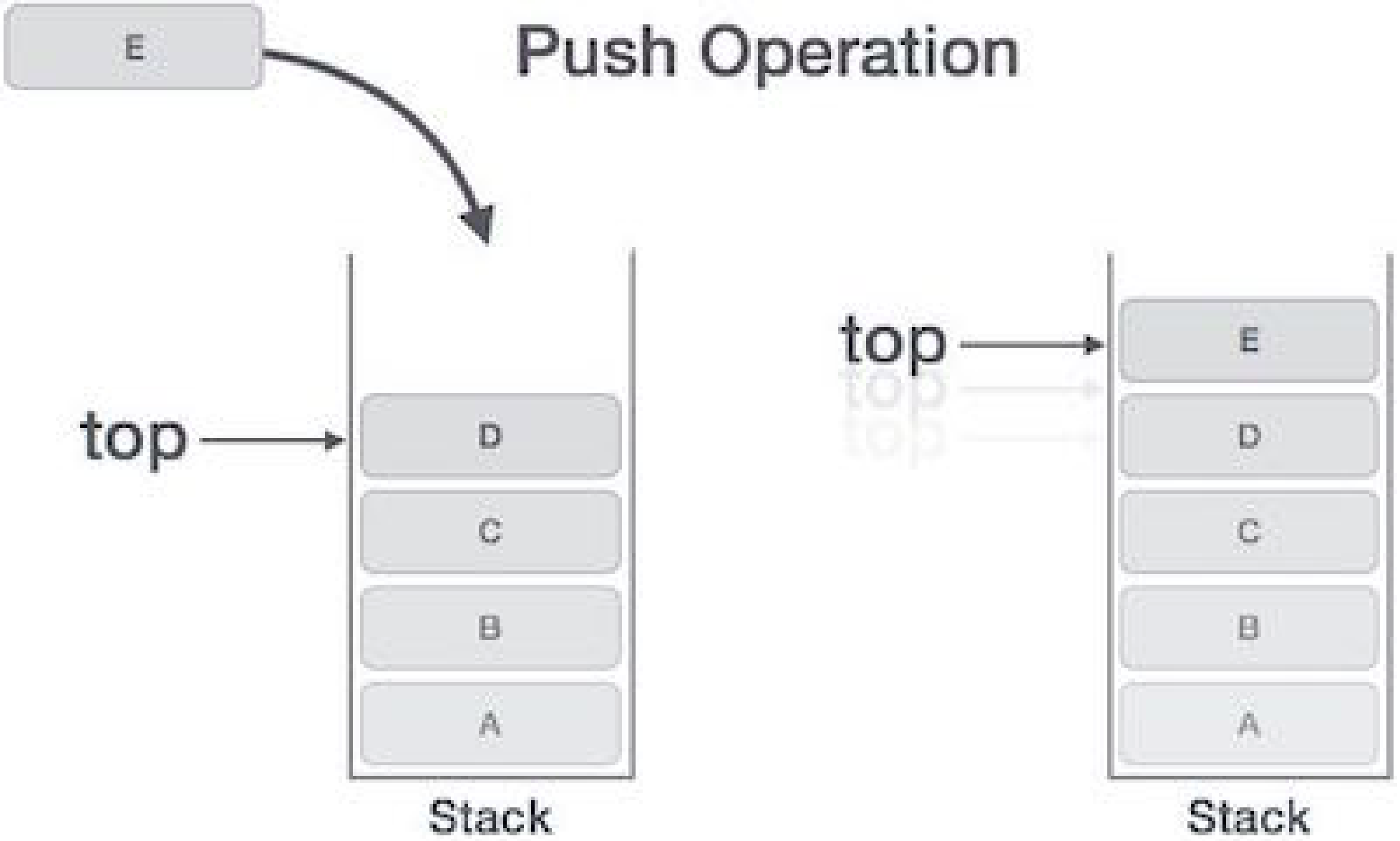
Step 2 − If the stack is full, produces an error and exit.

Step 3 − If the stack is not full, increments top to point next empty space.

Step 4 − Adds data element to the stack location, where top is pointing.

Step 5 − Returns success.

Stack Opertaion..



Stack Opertaion..

Algorithm for PUSH Operation:

```
begin procedure push: stack, data
   if stack is full
      return null
   endif
   top ← top + 1
   stack[top] ← data
end procedure
```

Stack Opertaion..

Implementation of this algorithm in C, is very easy. See the following code −

```
void push(int data) {
if(!isFull()) {
      top = top + 1;
      stack[top] = data;
   } else {
      printf("Could not insert data, Stack is full.\n");
   }
}
```

Stack Opertaion..

Pop Operation:

Accessing the content while removing it from the stack, is known as a Pop Operation. In an array implementation of pop() operation, the data element is not actually removed, instead top is decremented to a lower position in the stack to point to the next value. But in linked-list implementation, pop() actually removes data element and deallocates memory space.

Stack Opertaion..

A Pop operation may involve the following steps −

Step 1 − Checks if the stack is empty.

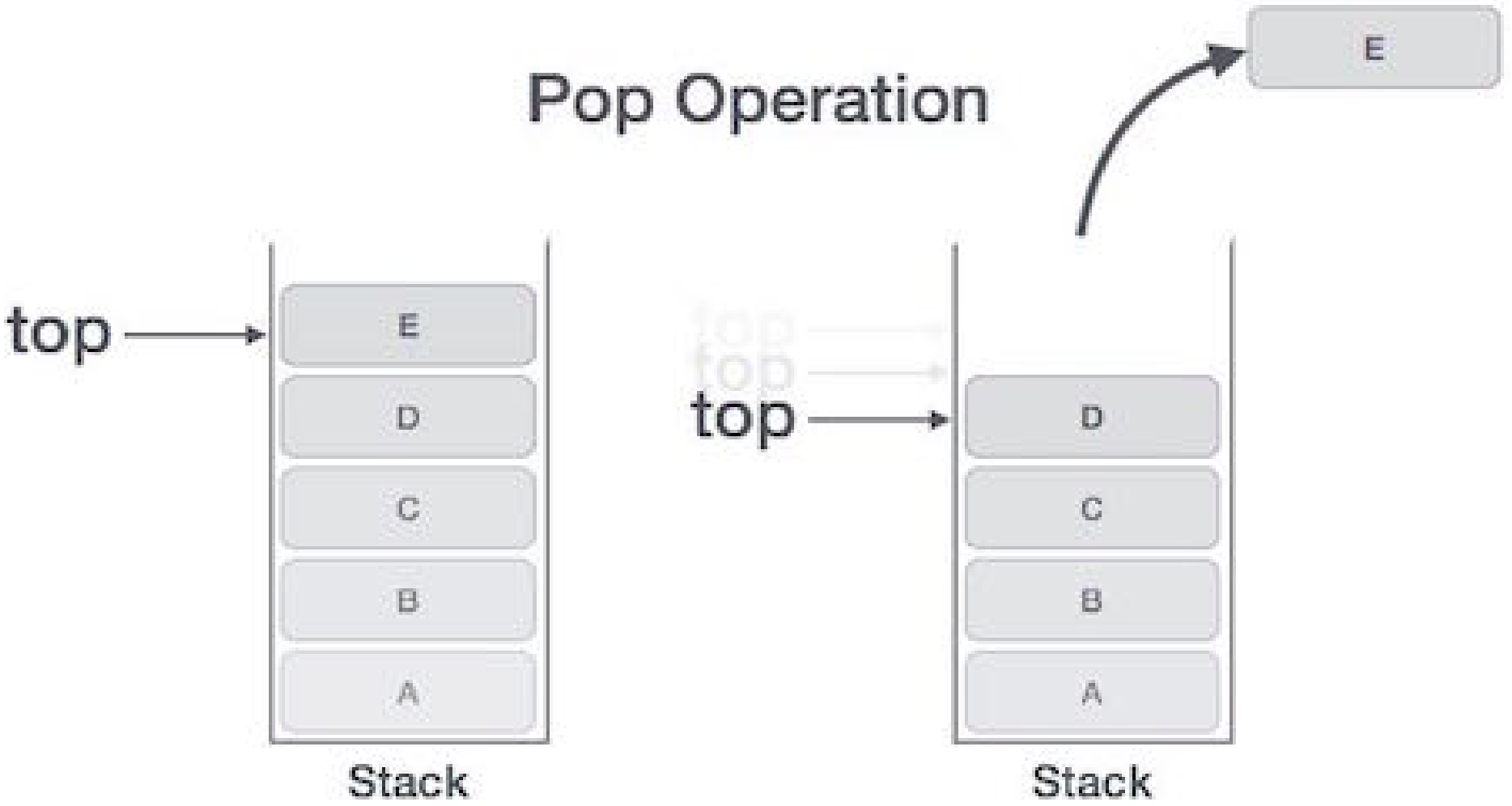
Step 2 − If the stack is empty, produces an error and exit.

Step 3 − If the stack is not empty, accesses the data element at which top is pointing.

Step 4 − Decreases the value of top by 1.

Step 5 − Returns success.

Stack Opertaion..



Stack Opertaion..

Algorithm for Pop Operation:

```
begin procedure pop: stack
   if stack is empty
      return null
   endif

   data ← stack[top]
   top ← top - 1
   return data

end procedure
```

Stack Opertaion..

Implementation of this algorithm in C, is as follows −

```
int pop(int data) {

   if(!isempty()) {
      data = stack[top];
      top = top - 1;
      return data;
   } else {
      printf("Could not retrieve data, Stack is empty.\n");
   }
}
```

Stack Application

The way to write arithmetic expression is known as a notation. An arithmetic expression can be written in three different but equivalent notations, i.e., without changing the essence or output of an expression. These notations are −

Infix Notation

Prefix (Polish) Notation

Postfix (Reverse-Polish) Notation

Stack Application..

Infix Notation:

We write expression in infix notation, e.g. a - b + c, where operators are used in-between operands.

Prefix Notation:

In this notation, operator is prefixed to operands, i.e. operator is written ahead of operands. For example, +ab. This is equivalent to its infix notation a + b. Prefix notation is also known as Polish Notation.

Postfix Notation:

This notation style is known as Reversed Polish Notation. In this notation style, the operator is postfixed to the operands i.e., the operator is written after the operands. For example, ab+. This is equivalent to its infix notation a + b.

Stack Application..

Sr.No.	Infix Notation	Prefix Notation	Postfix Notation
1	a + b	+ a b	a b +
2	(a + b) * c	* + a b c	a b + c *
3	a * (b + c)	* a + b c	a b c + *
4	a / b + c / d	+ / a b / c d	a b / c d / +
5	(a + b) * (c + d)	* + a b + c d	a b + c d + *
6	((a + b) * c) - d	- * + a b c d	a b + c * d -

Stack Application..

Parsing Expressions:

It is not a very efficient way to design an algorithm or program to parse infix notations. Instead, these infix notations are first converted into either postfix or prefix notations and then computed.

To parse any arithmetic expression, we need to take care of operator precedence and associativity also.

Precedence:

When an operand is in between two different operators, which operator will take the operand first, is decided by the precedence of an operator over others. For example −

Operator Precendence

As multiplication operation has precedence over addition, b * c will be evaluated first.

Stack Application..

Associativity:

Associativity describes the rule where operators with the same precedence appear in an expression. For example, in expression a + b − c, both + and – have the same precedence, then which part of the expression will be evaluated first, is determined by associativity of those operators. Here, both + and − are left associative, so the expression will be evaluated as (a + b) − c.

Stack Application..

Precedence and associativity determines the order of evaluation of an expression. Following is an operator precedence and associativity table (highest to lowest) −

Sr.No	Operator	Precedence	Associativity
1	Exponentiation ^	Highest	Right Associative
2	Multiplication (∗) & Division (/)	Second Highest	Left Associative
3	Addition (+) & Subtraction (−)	Lowest	Left Associative

Stack Application..

The above table shows the default behavior of operators. At any point of time in expression evaluation, the order can be altered by using parenthesis. For example −

In a + b*c, the expression part b*c will be evaluated first, with multiplication as precedence over addition. We here use parenthesis for a + b to be evaluated first, like (a + b)*c.

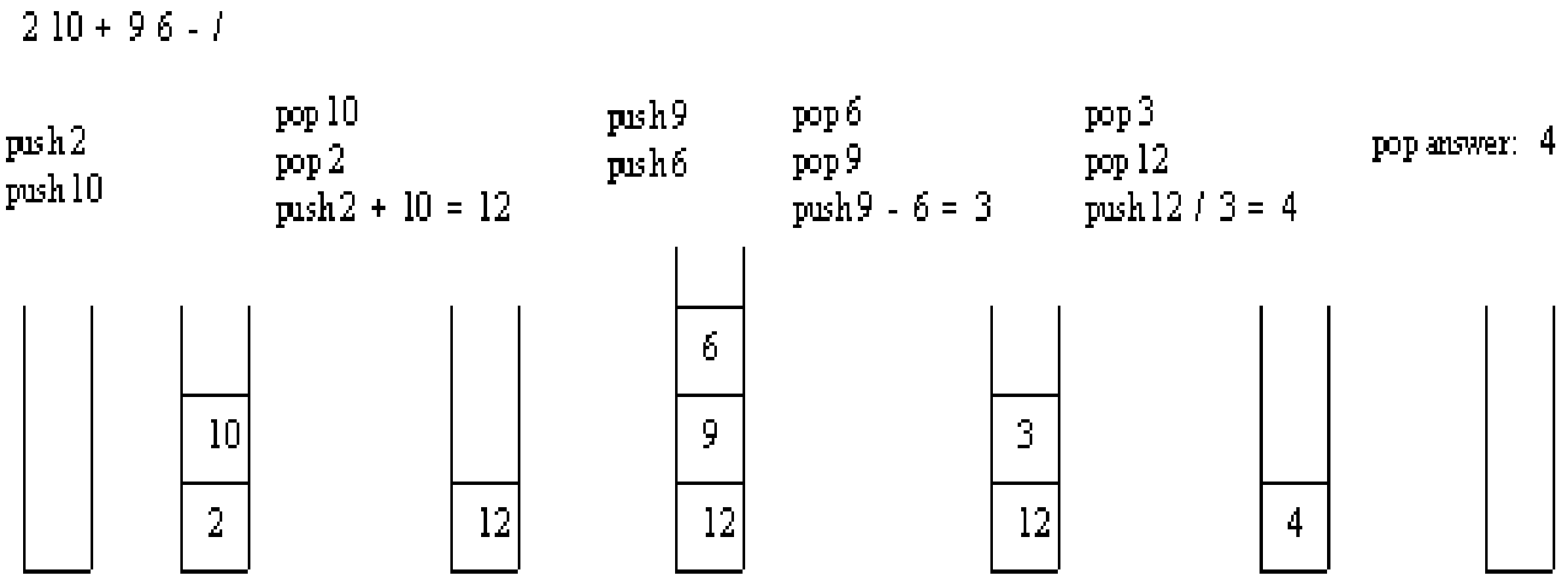
Postfix Evaluation using Stack

Postfix Evaluation Algorithm:

Step 1 − scan the expression from left to right
Step 2 − if it is an operand push it to stack
Step 3 − if it is an operator pull operand from stack and perform operation .
Step 4 − store the output of step 3, back to stack
Step 5 − scan the expression until all operands are consumed
Step 6 − pop the stack and perform operation

Postfix Evaluation using Stack..

Example:



Infix To Postfix Conversion Using Stack

One of the applications of Stack is in the conversion of arithmetic expressions in high-level programming languages into machine readable form. As our computer system can only understand and work on a binary language, it assumes that an arithmetic operation can take place in two operands only e.g., A+B, C*D,D/A etc. But in our usual form an arithmetic expression may consist of more than one operator and two operands e.g. (A+B)*C(D/(J+D)).

These complex arithmetic operations can be converted into reverse polish notation using stacks which then can be executed in two operands and an operator form.

Algorithm to convert Infix To Postfix:

Let, X is an arithmetic expression written in infix notation. This algorithm finds the equivalent postfix expression Y.

1. Push "(" onto Stack, and add ")" to the end of X.
2. Scan X from left to right and repeat Step 3 to 6 for each element of X until the Stack is empty.
3. If an operand is encountered, add it to Y.
4. If a left parenthesis is encountered, push it onto Stack.
5. If an operator is encountered ,then:

   (a) Repeatedly pop from Stack and add to Y each operator (on the top of Stack) which has the same precedence as or higher precedence than operator.

   (b) Add operator to Stack.

   [End of If]
6. If a right parenthesis is encountered ,then:

   (a) Repeatedly pop from Stack and add to Y each operator (on the top of Stack) until a left parenthesis is encountered.

   (b) Remove the left Parenthesis.

   [End of If]

   [End of If]
7. END.

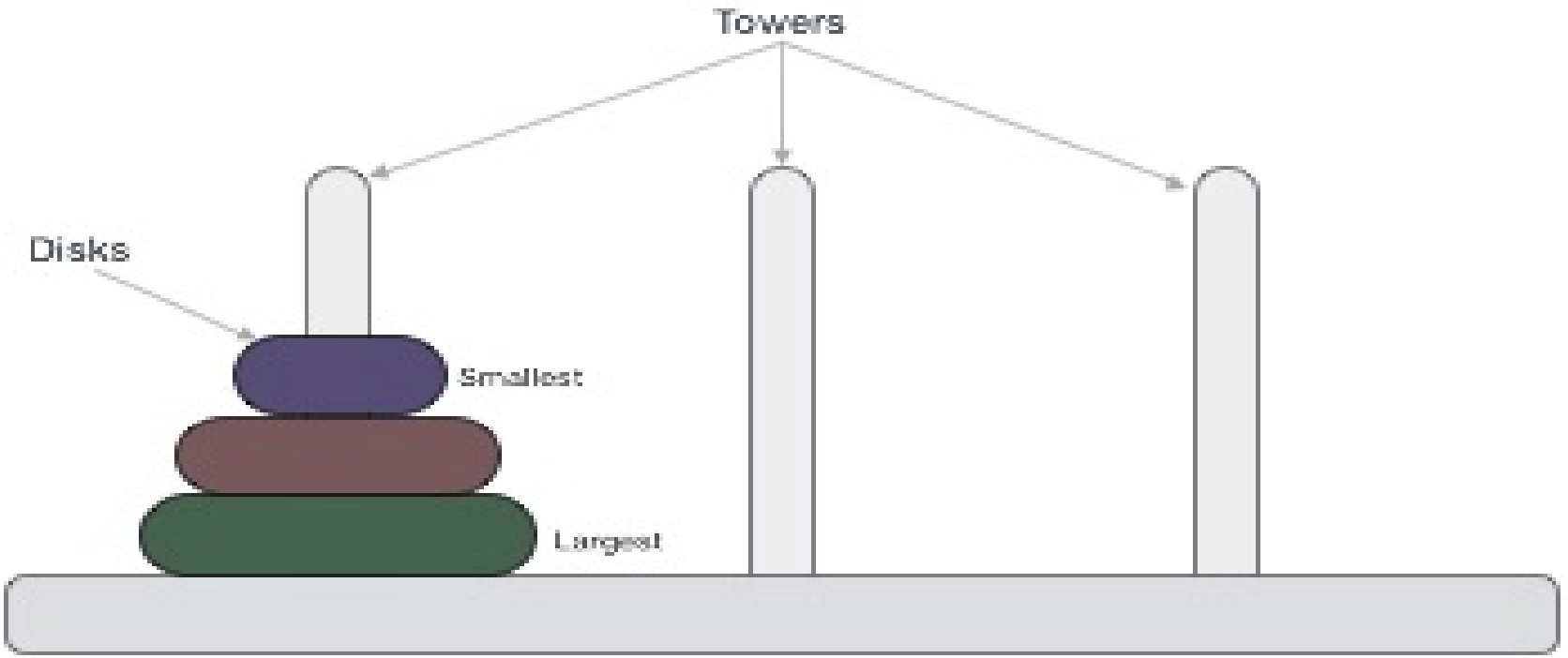
Example

Infix Expression: A+ (B*C-(D/E^F)*G)*H, where ^ is an exponential operator.

Symbol	Scanned	STACK	Postfix Expression	Description
1.		(		Start
2.	A	(	A	
3.	+	(+	A	
4.	(	(+(	A	
5.	B	(+(	AB	
6.	*	(+(*	AB	
7.	C	(+(*	ABC	
8.	-	(+(-	ABC*	'*' is at higher precedence than '-'
9.	(	(+(-(	ABC*	
10.	D	(+(-(	ABC*D	
11.	/	(+(-(/	ABC*D	
12.	E	(+(-(/	ABC*DE	
13.	^	(+(-(/^	ABC*DE	
14.	F	(+(-(/^	ABC*DEF	
15.	)	(+(-	ABC*DEF^/	Pop from top on Stack, that's why '^' Come first
16.	*	(+(-*	ABC*DEF^/	
17.	G	(+(-*	ABC*DEF^/G	
18.	)	(+	ABC*DEF^/G*-	Pop from top on Stack, that's why '^' Come first
19.	*	(+*	ABC*DEF^/G*-	
20.	H	(+*	ABC*DEF^/G*-H	
21.	)	Empty	ABC*DEF^/G*-H*+	END

Tower of Hanoi

Tower of Hanoi, is a mathematical puzzle which consists of three towers (pegs) and more than one rings is as depicted −



Tower of Hanoi..

These rings are of different sizes and stacked upon in an ascending order, i.e. the smaller one sits over the larger one. There are other variations of the puzzle where the number of disks increase, but the tower count remains the same.

Tower of Hanoi..

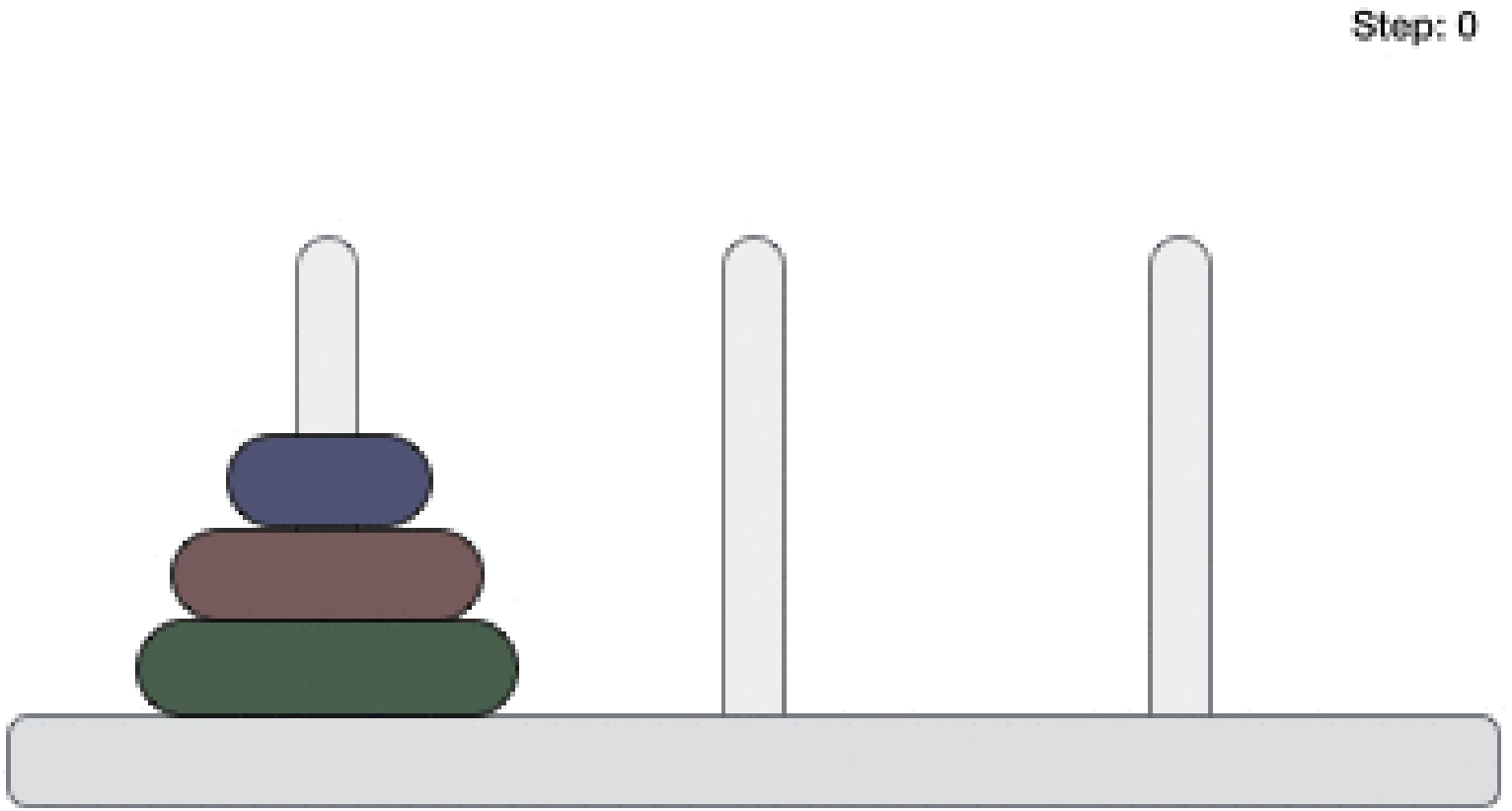
Rules:

The mission is to move all the disks to some another tower without violating the sequence of arrangement. A few rules to be followed for Tower of Hanoi are −

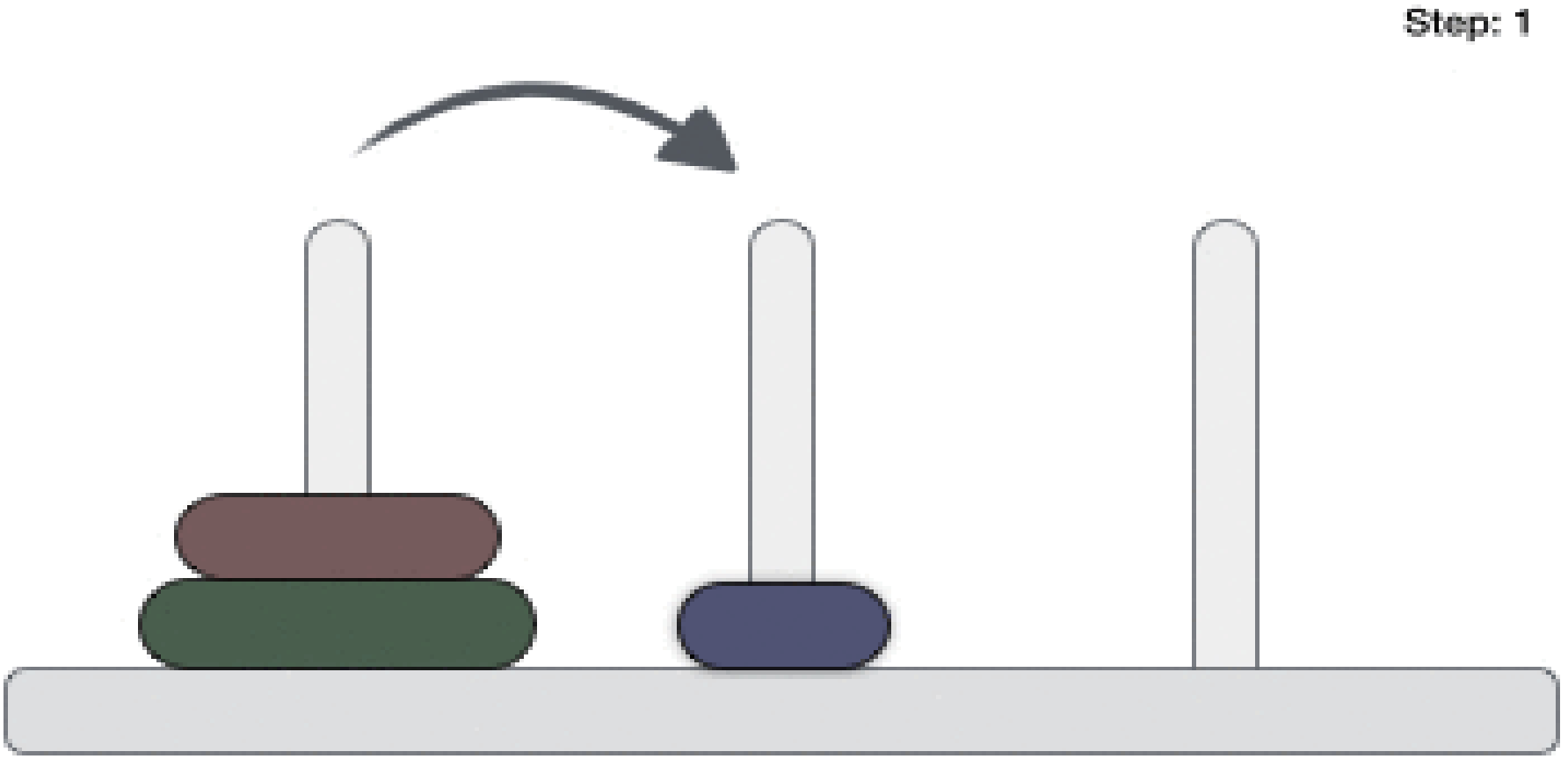
- Only one disk can be moved among the towers at any given time.
- Only the "top" disk can be removed.
- No large disk can sit over a small disk.

Tower of Hanoi..

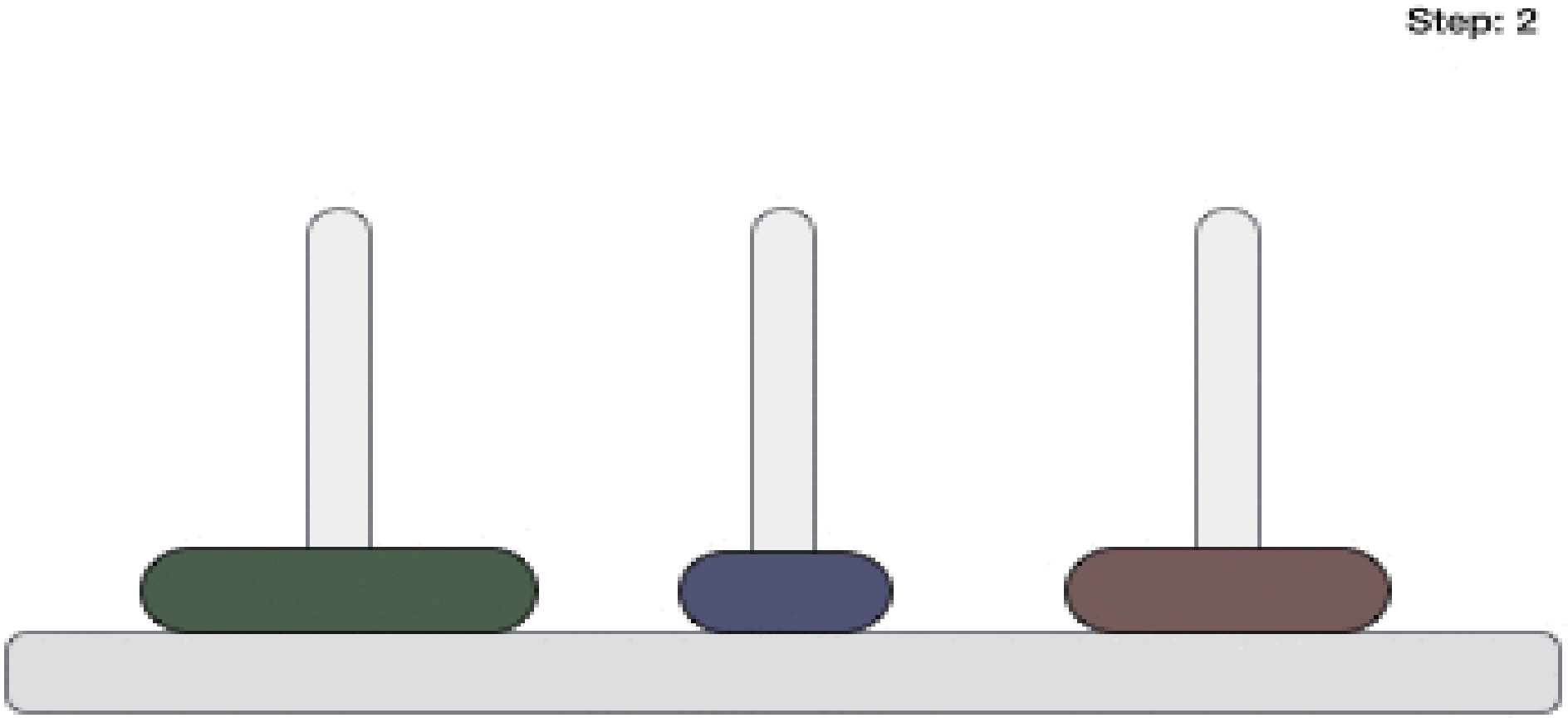
Following is an representation of solving a Tower of Hanoi puzzle with three disks.



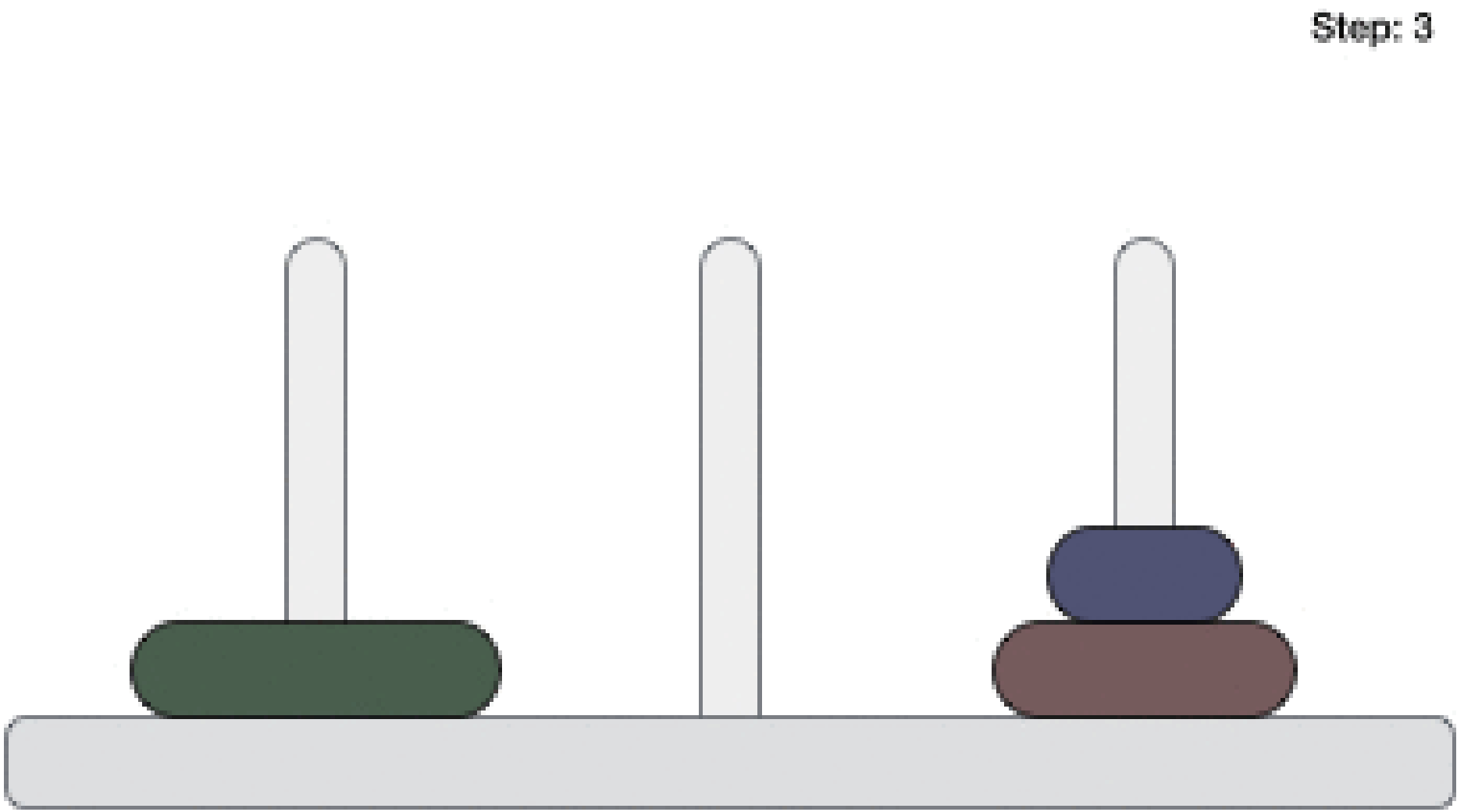
Tower of Hanoi..



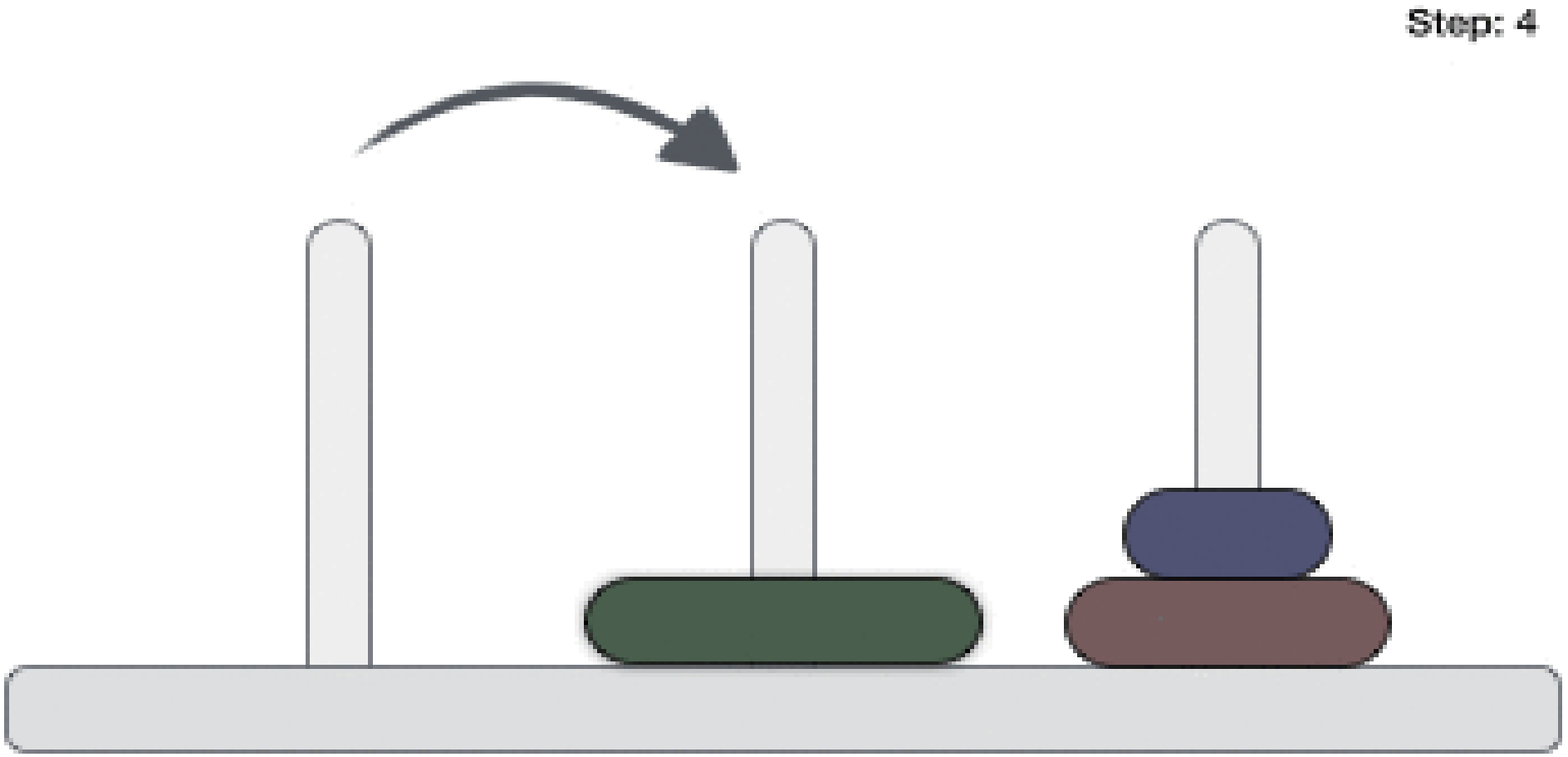
Tower of Hanoi..



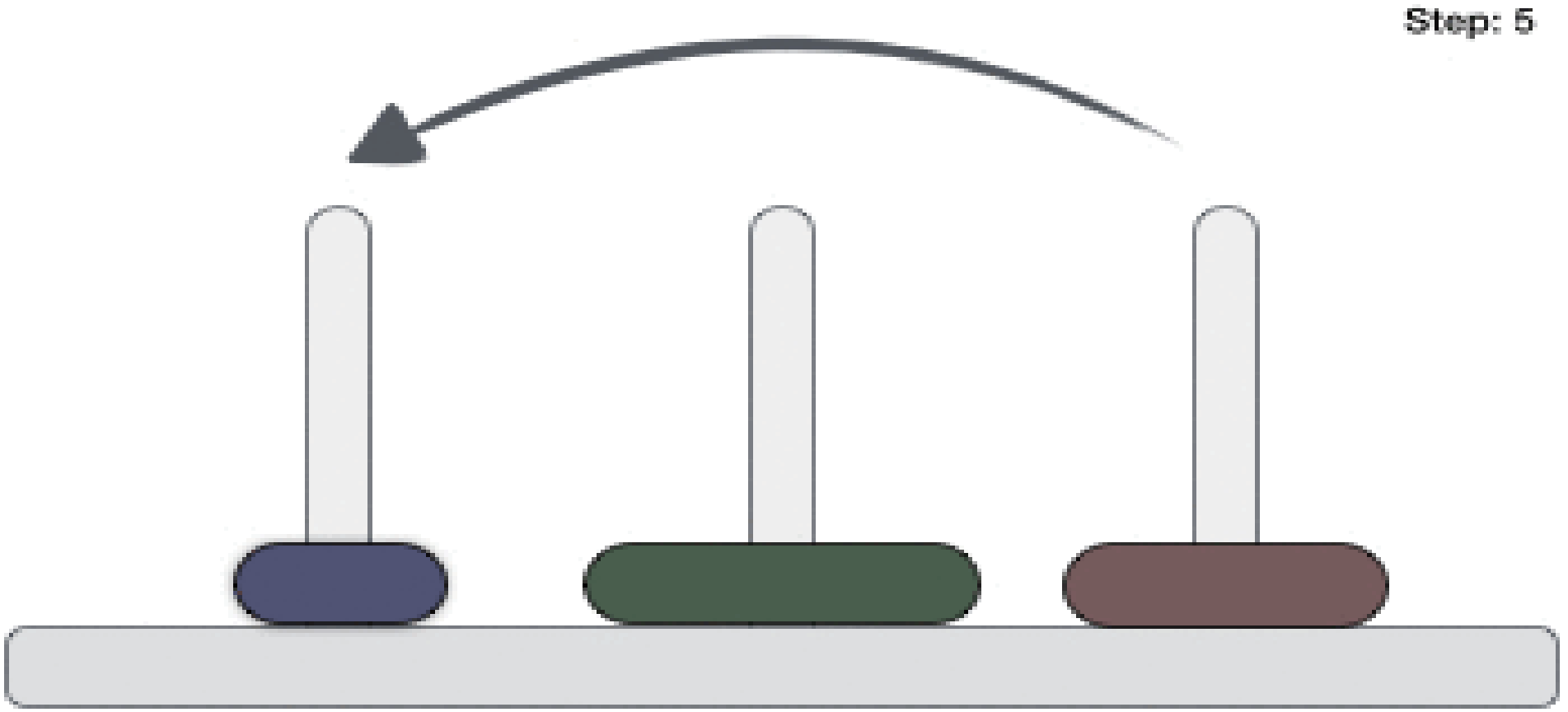
Tower of Hanoi..



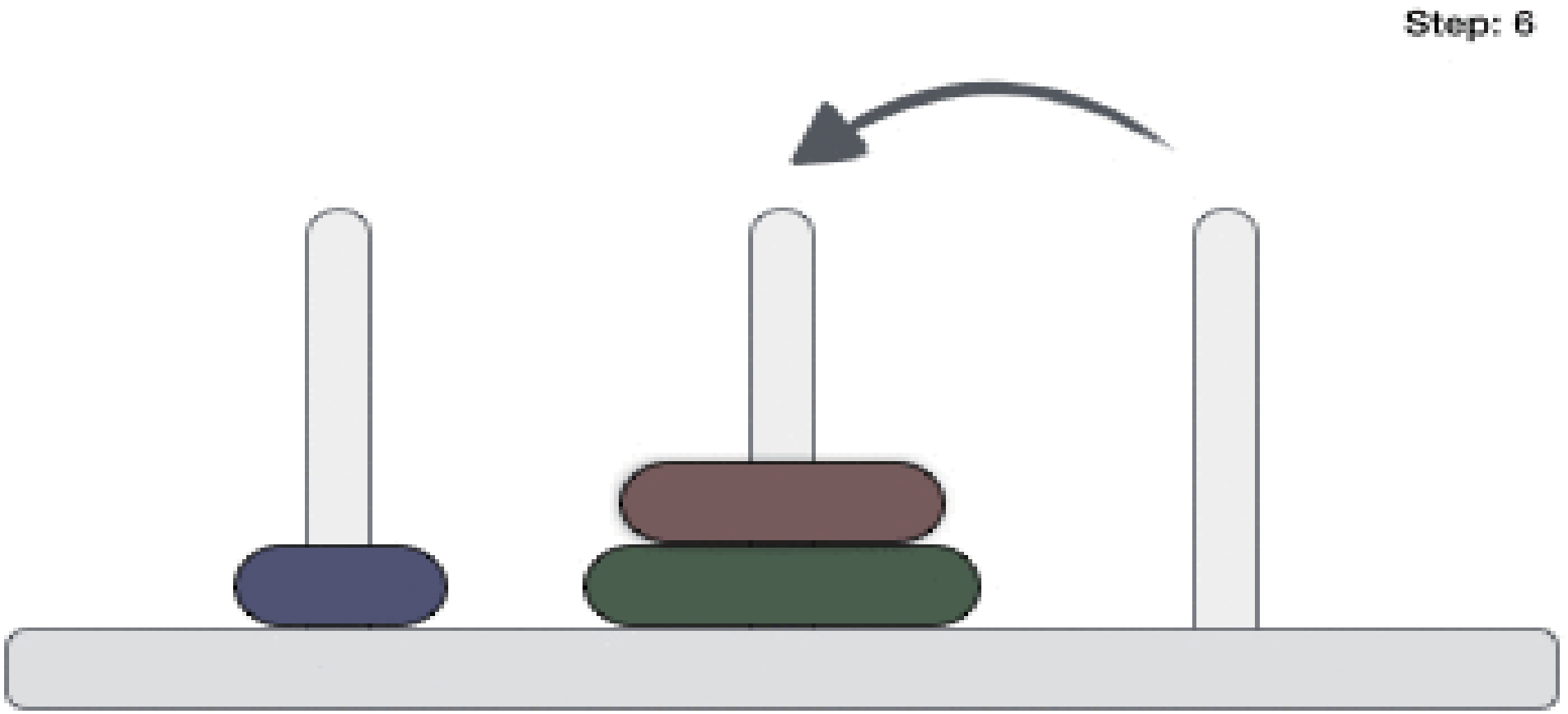
Tower of Hanoi..



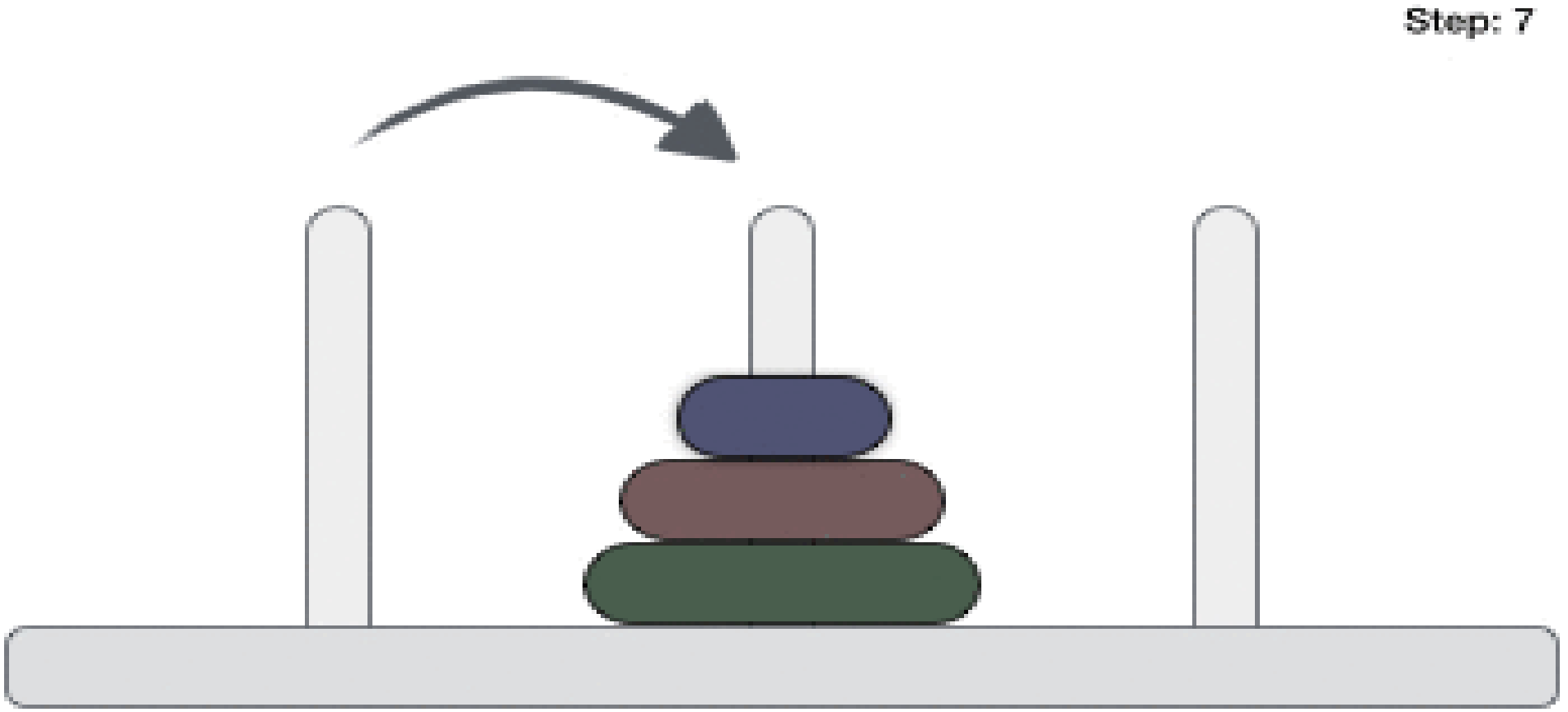
Tower of Hanoi..



Tower of Hanoi..



Tower of Hanoi..



Tower of Hanoi..

Tower of Hanoi puzzle with n disks can be solved in minimum 2^n−1 steps. This presentation shows that a puzzle with 3 disks has taken 2^3 - 1 = 7 steps.

Tower of Hanoi..

Algorithm:

To write an algorithm for Tower of Hanoi, first we need to learn how to solve this problem with lesser amount of disks, say → 1 or 2. We mark three towers with name, source, destination and aux (only to help moving the disks). If we have only one disk, then it can easily be moved from source to destination peg.

If we have 2 disks −

First, we move the smaller (top) disk to aux peg.

Then, we move the larger (bottom) disk to destination peg.

And finally, we move the smaller disk from aux to destination peg.

Tower of Hanoi..

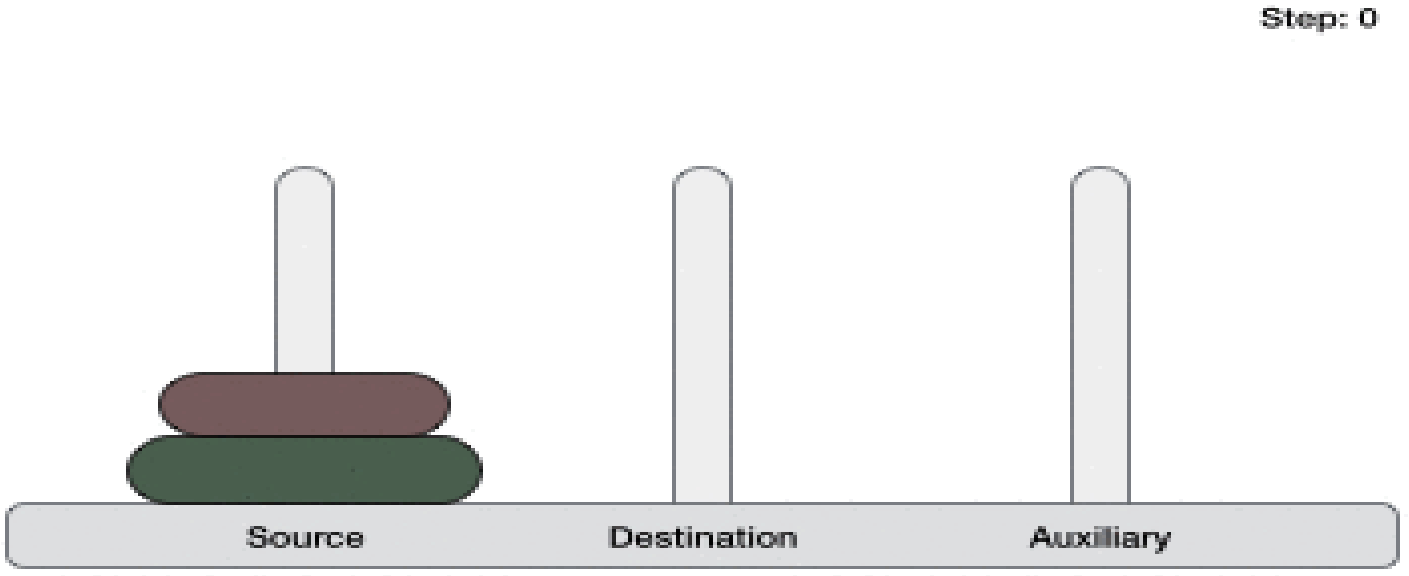
So now, we are in a position to design an algorithm for Tower of Hanoi with more than two disks. We divide the stack of disks in two parts. The largest disk (nth disk) is in one part and all other (n-1) disks are in the second part.

Our ultimate aim is to move disk n from source to destination and then put all other (n1) disks onto it. We can imagine to apply the same in a recursive way for all given set of disks.

Tower of Hanoi..

The steps to follow are −

Step 1 − Move n-1 disks from source to aux
Step 2 − Move nth disk from source to dest
Step 3 − Move n-1 disks from aux to dest



Tower of Hanoi..

A recursive algorithm for Tower of Hanoi can be driven as follows −

```
START
Procedure Hanoi(disk, source, dest, aux)

   IF disk == 1, THEN
      move disk from source to dest
   ELSE
      Hanoi(disk - 1, source, aux, dest)     // Step 1
      move disk from source to dest          // Step 2
      Hanoi(disk - 1, aux, dest, source)     // Step 3
   END IF

END Procedure
STOP
```

Reverse a String using STACK

This algorithm will read a string and reverse the string using Stack push and pop operations.

Reversing string is an operation of Stack by using Stack we can reverse any string.

Algorithm:

1. Read a string.
2. Push all characters until NULL is not found - Characters will be stored in stack variable.
3. Pop all characters until NULL is not found - As we know stack is a LIFO technique, so last character will be pushed first and finally we will get reversed string in a variable in which we store inputted string.

Factorial using stack

Factorial of a positive integer is the product of an integer and all the integers below it, i.e., the factorial of number n (represented by n!) would be given by

$$n! = 1*2*3*4* \ldots . *n$$

The factorial of 0 is defined to be 1 and is not defined for negative integers. There are multiple ways to find it which are listed below-

- Factorial Program in C using For loop
- Factorial Program using Functions
- Factorial Program using Recursion
- Factorial Program using Stack

Thank You