RECURSION

An Application of Stack

Prof. Siddharth Shah

Concept

- A procedure that contains a procedure call to itself directly or indirectly is known as recursive procedure.

```
void funcA()
{
    .......
    prntf("In funcA\n");
    .......
    funcA();
}

int main()
{
    funcA();
}
```

Direct

```
void funcA()
{
    .......
    prntf("In funcA\n");
    .......
    funcB();
}

void funcB()
{
    .......
    prntf("In funcB\n");
    .......
    funcA();
}

int main()
{
    funcA();
}
```

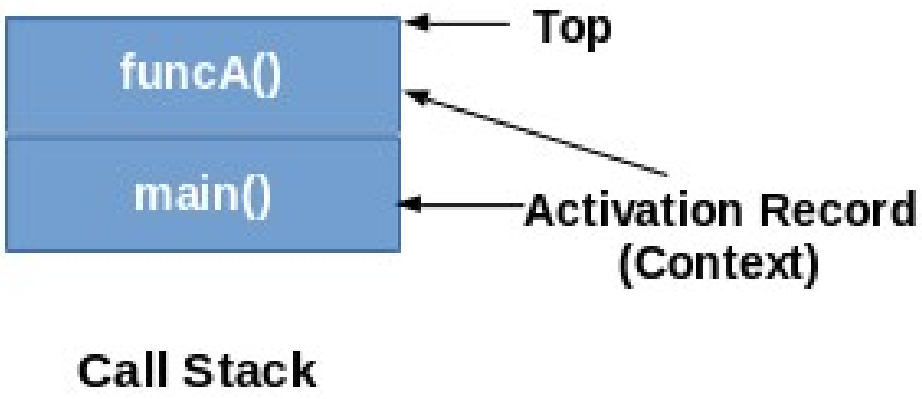
Indirect

Call Stack

- A call stack is a stack data structure that stores information about the active subroutines of a computer program.

```
void funcA()
{
    .......
    prntf("In funcA\n");
    .......
}

int main()
{
    funcA();
}
```



Activation Record

- The execution of a procedure is called its activation.
- An activation record contains all the necessary information required to call a procedure.
- Some common information are as below
  - Return address
  - Return Value
  - Local variables
  - Parameters
- Depending upon the source language used, there can be some variations.

Recursive Solution

- There are two important **conditions** that must be satisfied by any recursive procedure to solve the problem:

  1. Each time a procedure calls itself it must be nearer in some sense to a solution.

  2. There must be a decision criterion for stopping the process or computation (base condition).

Examples of Recursion

- Finding Factorial of n
- Tower of Hanoi

Factorial

Algorithm : Factorial

- Given integer N, this algorithm computes factorial of N.
- Stack A is used to store an activation record associated with each recursive call.
- Each activation record contains the current value of N and the current return address RET_ADDR.
- TEMP_REC is also a record which contains two variables PARAM & ADDRESS.
- TOP is a pointer to the top element of stack A.
- Initially return address is set to the main calling address.
- PARAM is set to initial value N.

Factorial

Algorithm : Factorial

1. [Save N and return Address]

CALL PUSH (A, TOP, TEMP_REC)

2. [Is the base criterion found?]

If N=0 Then

FACTORIAL← 1

GO TO Step 4

Else

PARAM← N-1

ADDRESS← Step 3

GO TO Step 1

3. [Calculate N!]

FACTORIAL← N * FACTORIAL

4. [Restore previous N and return address]

TEMP_REC← POP(A,TOP)

GO TO ADDRESS

Algorithm: Factorial

1. [Save N and return Address]

CALL PUSH (A, TOP, TEMP_REC)

2. [Is the base criterion found?]

If N=0 Then

FACTORIAL ← 1

GO TO Step 4

Else

PARAM ← N-1

ADDRESS ← Step 3

GO TO Step 1

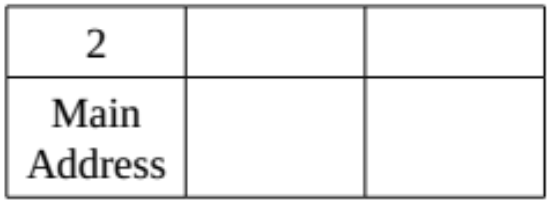
3. [Calculate N!]

FACTORIAL ← N * FACTORIAL

4. [Restore previous N and return address]

TEMP_REC ← POP(A,TOP)

GO TO ADDRESS

Level Number	Description	Stack Content
Enter Level 1 (main call)	Step 1: PUSH(A,0,(2,main address)) Step 2: N ≠ 0 PARAM ← 1, ADDR ← Step 3	

Algorithm: Factorial

1. [Save N and return Address]

CALL PUSH (A, TOP, TEMP_REC)

2. [Is the base criterion found?]

If N=0 Then

FACTORIAL ← 1

GO TO Step 4

Else

PARAM ← N-1

ADDRESS ← Step 3

GO TO Step 1

3. [Calculate N!]

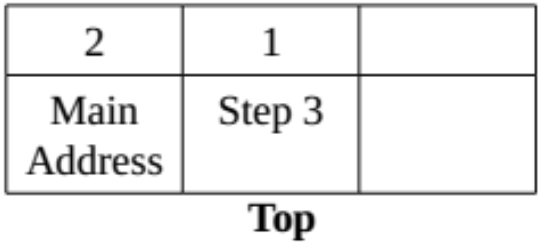
FACTORIAL ← N * FACTORIAL

4. [Restore previous N and return address]

TEMP_REC ← POP(A,TOP)

GO TO ADDRESS

Level Number	Description	Stack Content
Enter Level 2 (first recursive call)	Step 1: PUSH(A,1,(1,Step 3)) Step 2: N ≠ 0 PARAM ← 0, ADDR ← Step 3	



Algorithm: Factorial

1. [Save N and return Address]

CALL PUSH (A, TOP, TEMP_REC)

2. [Is the base criterion found?]

If N=0 Then

FACTORIAL ← 1

GO TO Step 4

Else

PARAM ← N-1

ADDRESS ← Step 3

GO TO Step 1

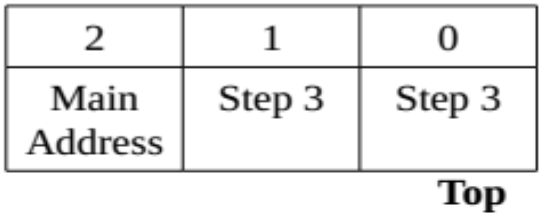
3. [Calculate N!]

FACTORIAL ← N * FACTORIAL

4. [Restore previous N and return address]

TEMP_REC ← POP(A,TOP)

GO TO ADDRESS

Level Number	Description	Stack Content
Enter Level 3 (second recursive call)	Step 1: PUSH(A,2,(0,Step 3)) Step 2: N = 0 FACTORIAL ← 1	
	Step 4: POP(A,3) go to Step 3	

Algorithm: Factorial

1. [Save N and return Address]

CALL PUSH (A, TOP, TEMP_REC)

2. [Is the base criterion found?]

If N=0 Then

FACTORIAL ← 1

GO TO Step 4

Else

PARAM ← N-1

ADDRESS ← Step 3

GO TO Step 1

3. [Calculate N!]

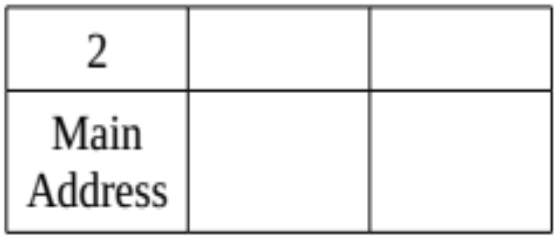
FACTORIAL ← N * FACTORIAL

4. [Restore previous N and return address]

TEMP_REC ← POP(A,TOP)

GO TO ADDRESS

Level Number	Description	Stack Content
Return to Level 2	Step 3: FACTORIAL ← 1*1 Step 4: POP(A,2) go to Step 3	



Algorithm: Factorial

1. [Save N and return Address]

CALL PUSH (A, TOP, TEMP_REC)

2. [Is the base criterion found?]

If N=0 Then

FACTORIAL ← 1

GO TO Step 4

Else

PARAM ← N-1

ADDRESS ← Step 3

GO TO Step 1

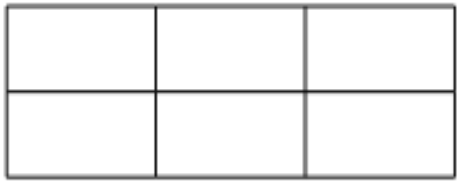
3. [Calculate N!]

FACTORIAL ← N * FACTORIAL

4. [Restore previous N and return address]

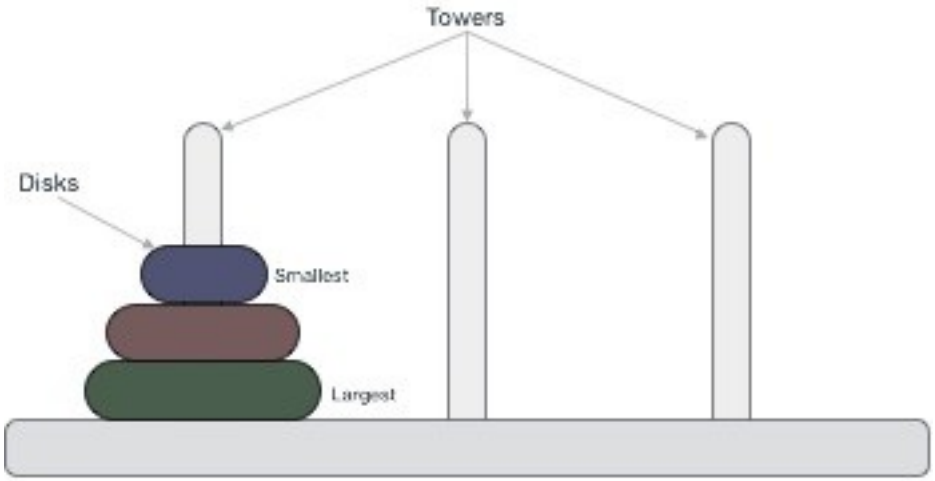
TEMP_REC ← POP(A,TOP)

GO TO ADDRESS

Level Number	**Description**	**Stack Content**
Return to Level 1	Step 3: FACTORIAL ← 2*1	
	Step 4: POP(A,1) go to main address	**Top**

Tower of Hanoi

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



- 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

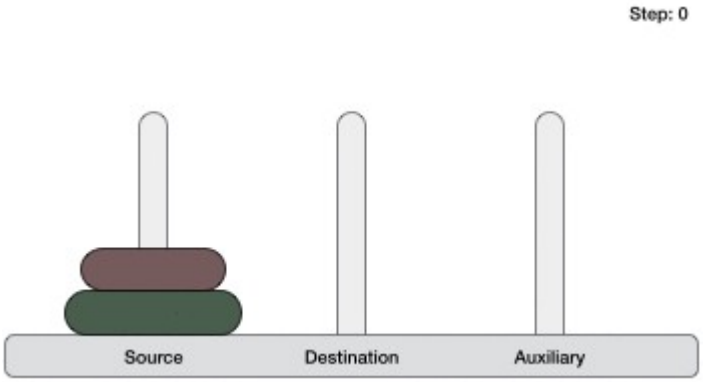
RULES:

- 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 puzzle with ***n*** disks can be solved in minimum 2^n-1 steps.

Tower of Hanoi

SOLUTION:

- 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, move the smaller (top) disk to aux peg.
  - Then, move the larger (bottom) disk to destination peg.
  - And finally, move the smaller disk from aux to destination peg.



Tower of Hanoi

SOLUTION:

- To design an algorithm for Tower of Hanoi with more than two disks, divide the stack of disks in two parts.
- The largest disk (**n^{th}** disk) is in one part and all other (**n-1**) disks are in the second part.
- The ultimate aim is to move **n^{th}** disk from source to destination and then put all other (**n-1**) disks onto it.
- Same logic can be applied in a recursive way for all given set of disks.
- The steps to follow are −

  Step 1− Move **n-1** disks from source to auxiliary

  Step 2− Move **n^{th}** disks from source to destination

  Step 3− Move **n-1** disks from auxiliary to destination

Tower of Hanoi

Algorithm: Hanoi (disk, source, auxiliary, destination)

1. [Check the base criterion]

```
If disk = 1 Then
        move disk from source to destination
Else
    Hanoi(disk - 1, source, destination, auxiliary)    // Step 1
    move disk from source to destination               // Step 2
    Hanoi(disk - 1, auxiliary, source ,destination)    // Step 3
```

1. [Finished]

```
Exit
```

Tower of Hanoi

Example: Trace algorithm Hanoi for 3 disks