CS1100 - Lecture 22

Instructor : Jasine Babu

Teaching Assistants : Nikhila K N, Veena Prabhakaran

Towers of Hanoi

In this lecture, we will see an example in which the real power of recursive programs is evident. Towers of Hanoi is a puzzle in which there are three poles (P1, P2 and P3) and a set of n discs, each disc having a different radius. All these discs are initially placed in the first pole (P1) in such a way that no disc of larger radius is placed on a disc of lower radius.

Objective: The goal of the puzzle is to move all the discs from the first pole (P1) to the third pole (P3) such that finally all the discs are arranged in Pole 3 in such a way that no disc of larger radius is placed on a disc of lower radius. Pole 2 can be used for placing some discs at some intermediate points of the game. There are some rules that must be followed while moving the discs.

Rules:

- Move only one disc at a time.
- Never place a bigger disc on top of a smaller disc.
- In any move, moving only the top disc from a pole is allowed and the moved disc should be placed only as the top most disc in another pole.

Let the number of discs be `n`. Solutions to this puzzle for some values of `n` are shown below.

- For n=1,
  1. move disc from P1 to P3
- For n=2,
  1. move disc from P1 to P2
  2. move disc from P1 to P3
  3. move disc from P2 to P3
- For n=3,
  1. move disc from P1 to P3
  2. move disc from P1 to P2
  3. move disc from P3 to P2
  4. move disc from P1 to P3

5. move disc from P2 to P1
6. move disc from P2 to P3
7. move disc from P1 to P3

For `n=4`, the process of listing the required moves iteratively might be difficult and the difficulty increases as `n` value increases. To come up with a program which does the above task without using any recursion is very difficult. Technically it is possible to do so; but such solutions are very complicated and thus difficult to explain. However, there is an elegant and short recursive solution for this puzzle.

Looking at the solutions of the problem for `n` = 2 and 3, we can make some observations.

For `n=2`, after step 1, the smaller disc is on P2, bigger disc is on P1, and P3 is empty. In step 2, the bigger disc is moved from P1 to P3. After this, in step 3, the smaller disc in P2 is to moved to P3.

For `n=3`, after step 3, the smallest 2 discs are on P2, the biggest disc is on P1, and P3 is empty. In steps 1-3, the biggest disc is not moved at all. In step 4, the biggest disc is moved from P1 to P3. In steps 5-7, the smallest 2 discs get transferred from P2 to P3. During steps 5-7, the biggest disc is not moved.

Observe that if there are `2t+1` steps in total, in the first `t` steps, the smallest `n-1` discs get transferred from P1 to P2 without moving the largest disc and during this process P3 is used as an intermediate pole. In step `t+1`, the biggest disc is moved from P1 to P3. In the last `t` steps, the smallest n-1 discs get transferred from P2 to P3 without moving the largest disc and during this process P1 is used as an intermediate pole.

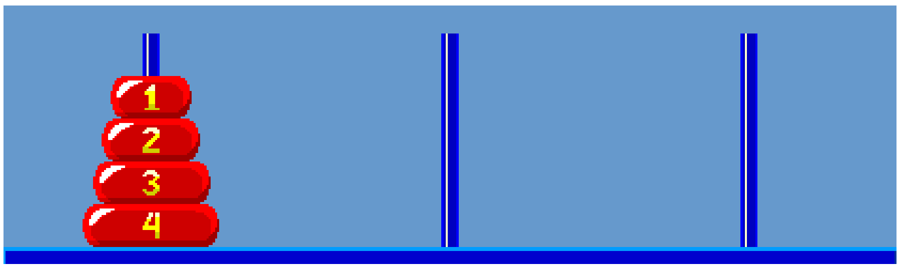
This leads to the following recursive algorithm for performing this task.

```
Tower(n,P1,P2,P3) //for moving n discs from P1 to P3
1. if(n==1)
    1.1 move disc from P1 to P3
2. else
    2.1 Recursively move top (n-1) discs from P1 to P2 using P3 as intermediate i.e.,
        Tower(n-1, P1, P3, P2)
    2.2 Move disc from P1 to P3
    2.3 Recursively move top (n-1) discs from P2 to P3 using P1 as intermediate i.e.,
        Tower(n-1, P2, P1, P3)
3. End
```

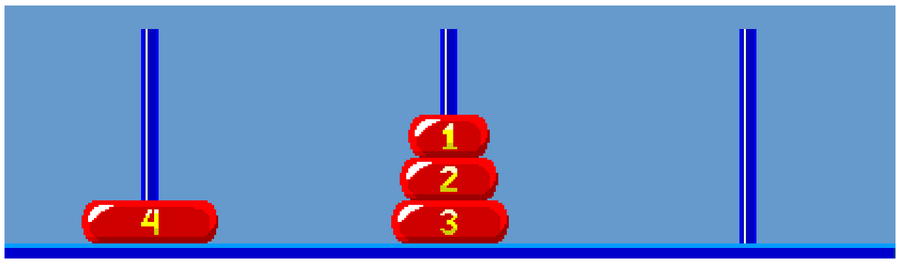
To give an understanding of this strategy, the configuration of discs at different stages of the game are given in the next page[1], for $n = 4$. Suppose there are $2t + 1$ disc movements in total.

[1]http://zylla.wipos.p.lodz.pl/games/hanoi4e.html

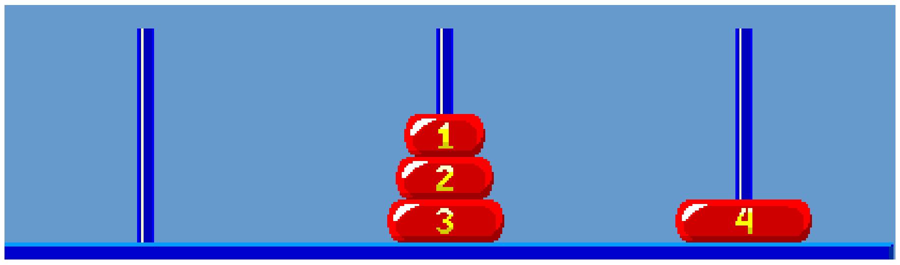
The initial configuration of discs is as follows.



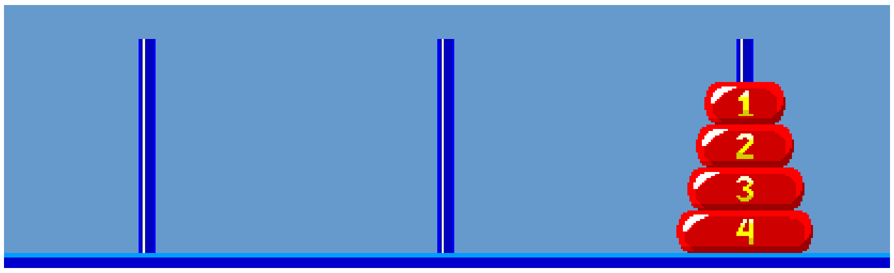
After t steps, the configuration is as follows.



After t+1 steps, the configuration becomes as shown in the figure below.



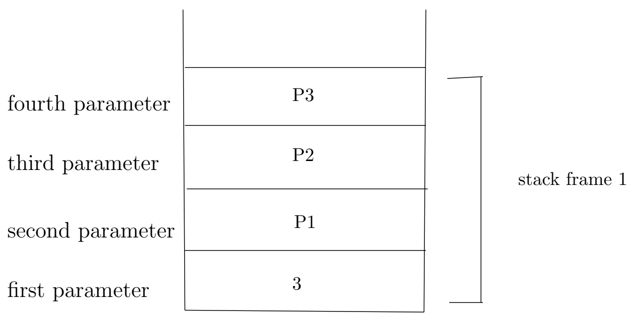
After 2t+1 steps, the final configuration is as follows.



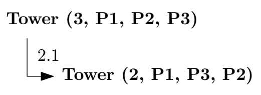
Unwinding the recursion for n=3

The working of the above recursive algorithm with input `n=3` is explained below.

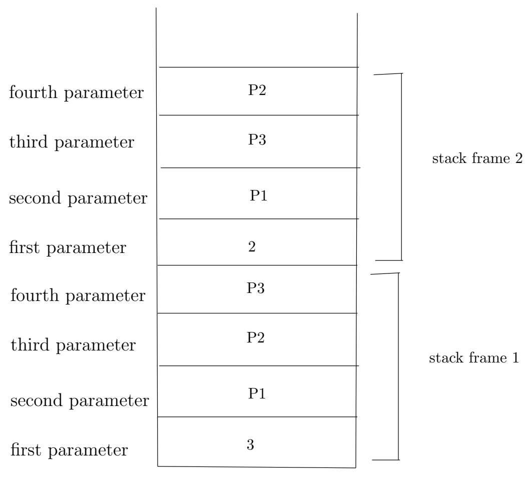
Step 1: The `main()` function calls the function `Tower()` with parameters `n=3`, `P1`, `P2` and `P3` in that order (i.e., `Tower (3, P1, P2, P3)` is invoked). Excluding the `main()` function, the stack frame in the memory is as shown below.



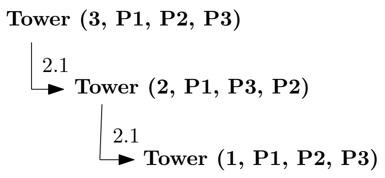
Step 2: The execution of `Tower (3, P1, P2, P3)` starts. Since `n` $\neq$ 1, the program control goes inside the `else` statement and in line 2.1 a call to the recursive function `Tower()` is made with its parameters `n=2`, `P1`, `P3` and `P2` in that order i.e., `Tower (2, P1, P3, P2)` gets invoked.



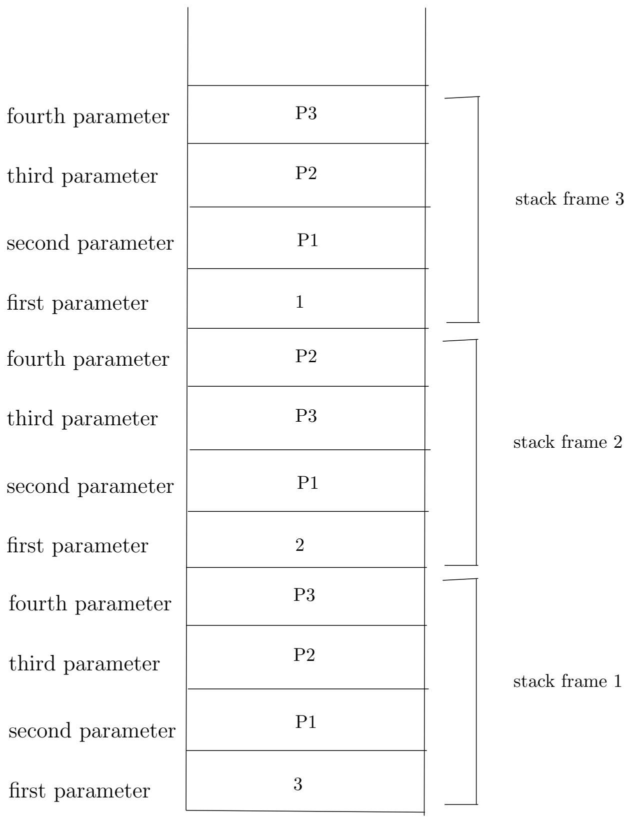
The stack frame in memory (except the stack frame of main) is as shown in the figure below.



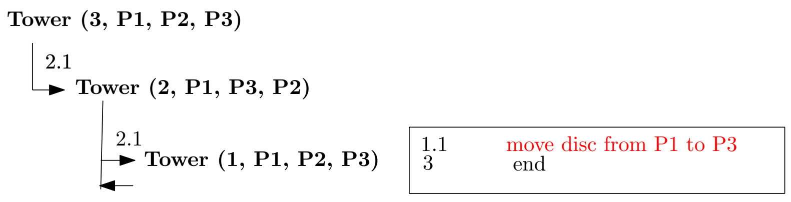
Step 3: The execution of `Tower (2, P1, P3, P2)` starts and again the `if` condition is *false* and in line 2.1, the function `Tower()` is called with parameters `n=1`, `P1`, `P2` and `P3` in that order i.e., `Tower (1, P1, P2, P3)` is invoked.



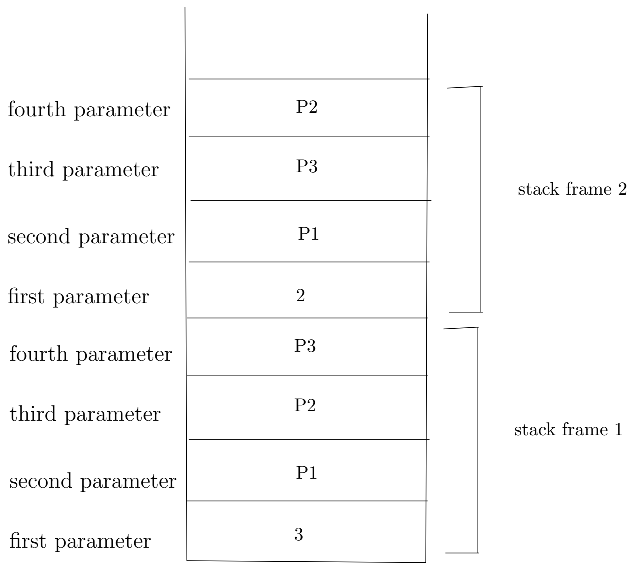
The memory stack frame diagram (except the stack frame of main) is as shown below.



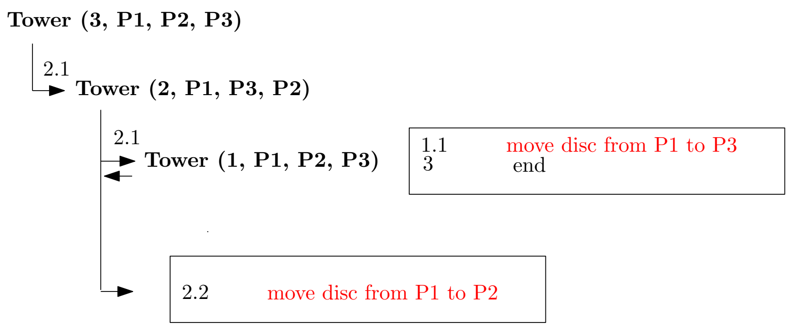
Step 4: Now, `Tower (1, P1, P2, P3)` starts executing and the condition `if(n==1)` becomes *true*. In line 1.1, the instruction is to move the disc from the second parameter to the fourth parameter i.e., from P1 to P3. After this, the program control moves to line 3 and this finishes the execution of `Tower (1, P1, P2, P3)` and the program control is returned to the calling function `Tower (2, P1, P3, P2)`.



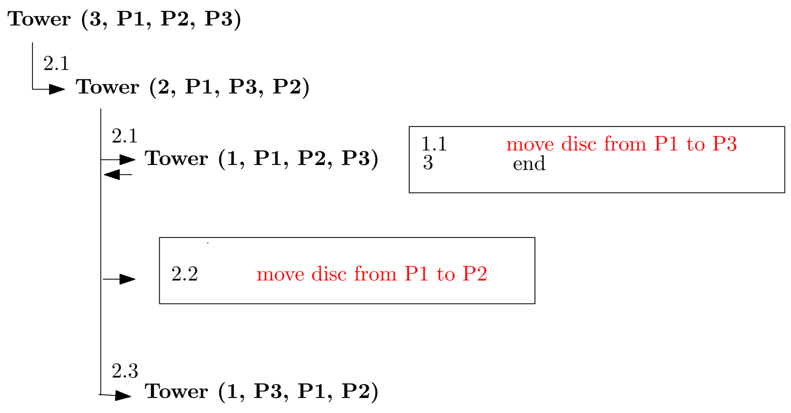
The memory stack frame diagram (except the stack frame of main) is as shown below.



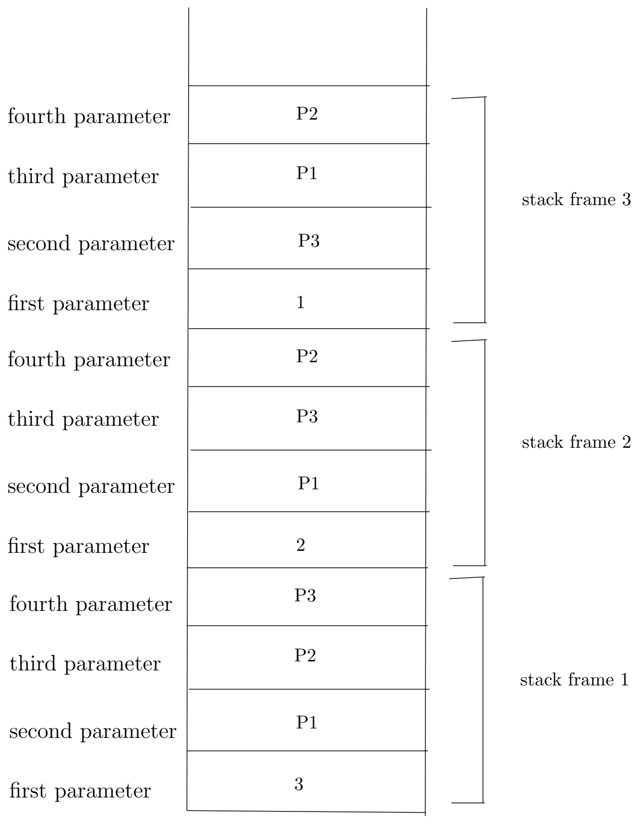
Step 5: Now, the program control is in the function `Tower (2, P1, P3, P2)` and the program execution continues from line number 2.2 and in this step, a disc is moved from the second parameter to the fourth parameter of the function call `Tower (2, P1, P3, P2)` i.e., from P1 to P2.



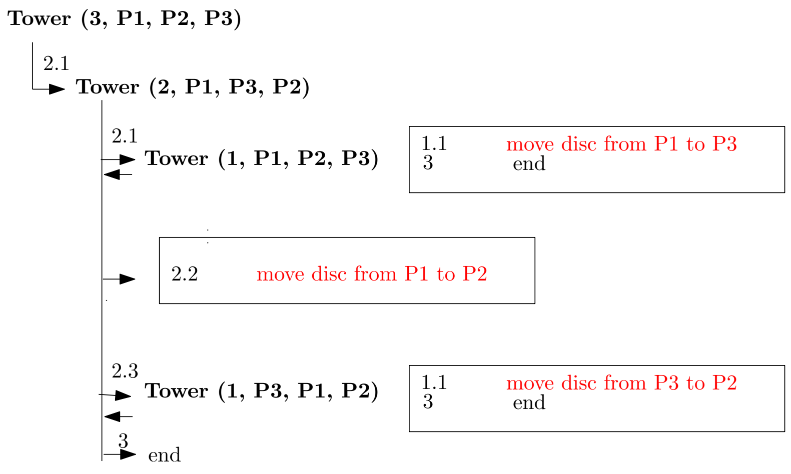
Step 6: After this step, in line number 2.3 of `Tower (2, P1, P3, P2)`, the recursive call to `Tower()` is made with parameters `n=1`, `P3`, `P1` and `P2` in that order i.e., `Tower (1, P3, P1, P2)` is invoked.



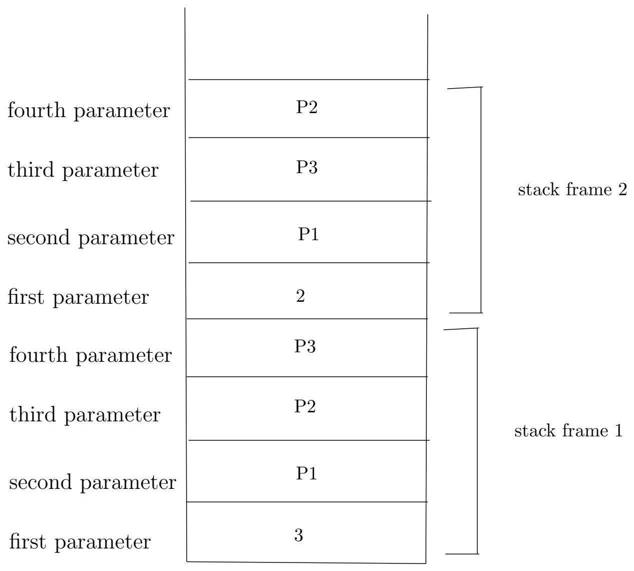
The memory stack frame diagram (except the stack frame of main) is as shown below.



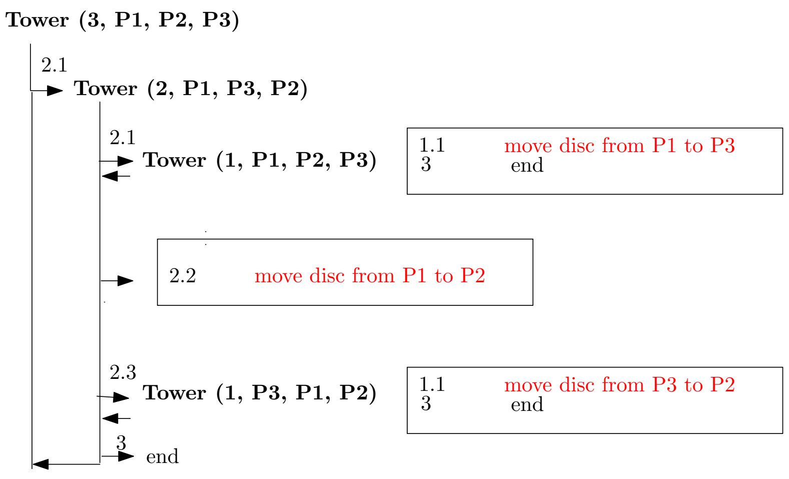
Step 7: Now, `Tower (1, P3, P1, P2)` starts executing and the condition `if(n==1)` becomes *true*. In line 1.1, the instruction is to move the disc from the second parameter to the fourth parameter i.e., from P3 to P2. After this, the program control moves to line 3 and this finishes the execution of `Tower (1, P3, P1, P2)` and the program control is returned to the calling function `Tower (2, P1, P3, P2)` to line 3.



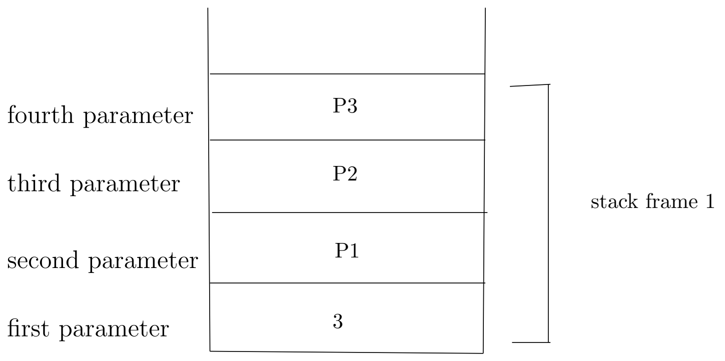
The memory stack frame diagram (except the stack frame of main) is as shown below.



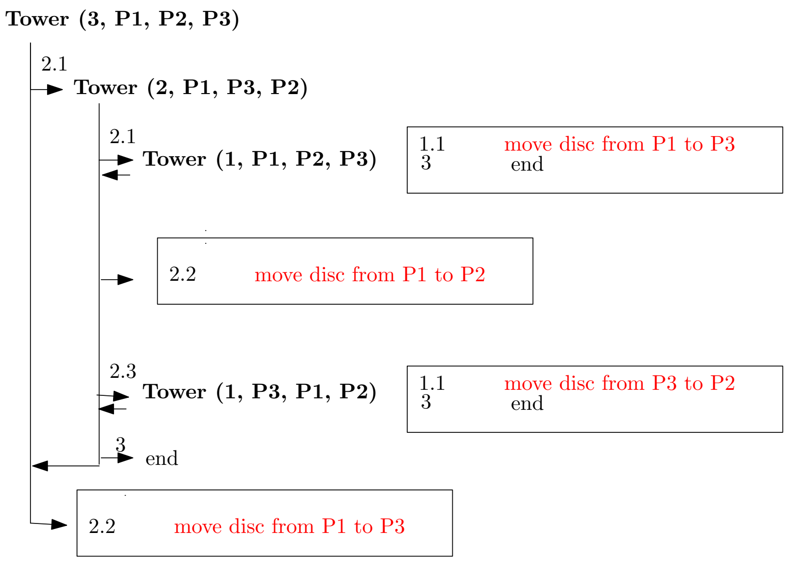
[Step 8:] After this, line 3 in `Tower (2, P1, P3, P2)` is executed and program control returns to the calling function `Tower (3, P1, P2, P3)`.


Tower (3, P1, P2, P3)
2.1
Tower (2, P1, P3, P2)
2.1
Tower (1, P1, P2, P3)
1.1 move disc from P1 to P3
3 end
2.2 move disc from P1 to P2
2.3
Tower (1, P3, P1, P2)
1.1 move disc from P3 to P2
3 end
3
end

The memory stack frame diagram (except the stack frame of main) is as shown below.

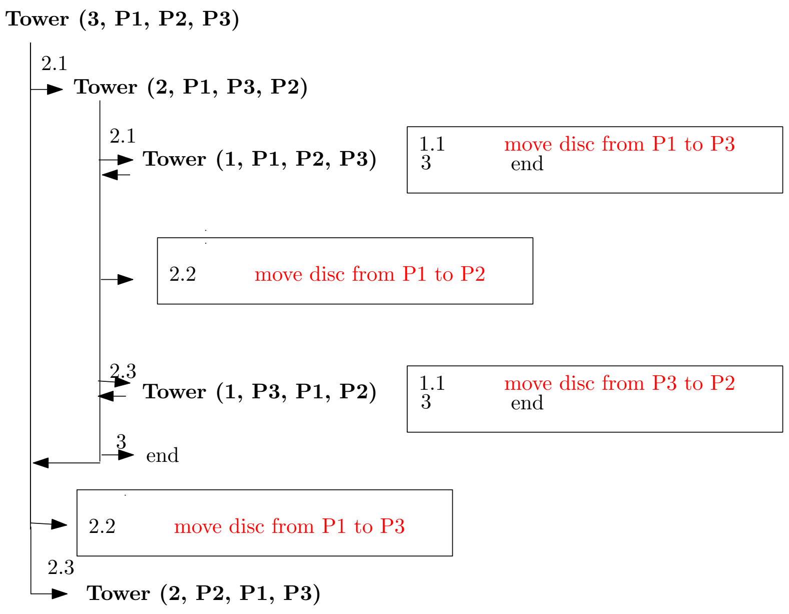


Step 9: Now, the program control is in the function `Tower (3, P1, P2, P3)` and the program execution continues from line number 2.2 and in this step, a disc is moved from the second parameter to the fourth parameter of the function call `Tower (3, P1, P2, P3)` i.e., from P1 to P3.

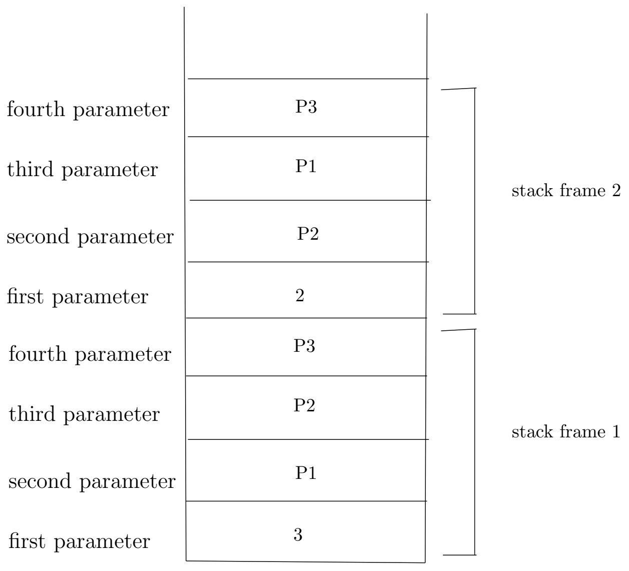


Tower (3, P1, P2, P3)
2.1
Tower (2, P1, P3, P2)
2.1
Tower (1, P1, P2, P3)
1.1 move disc from P1 to P3
3 end
2.2 move disc from P1 to P2
2.3
Tower (1, P3, P1, P2)
1.1 move disc from P3 to P2
3 end
3
end
2.2 move disc from P1 to P3

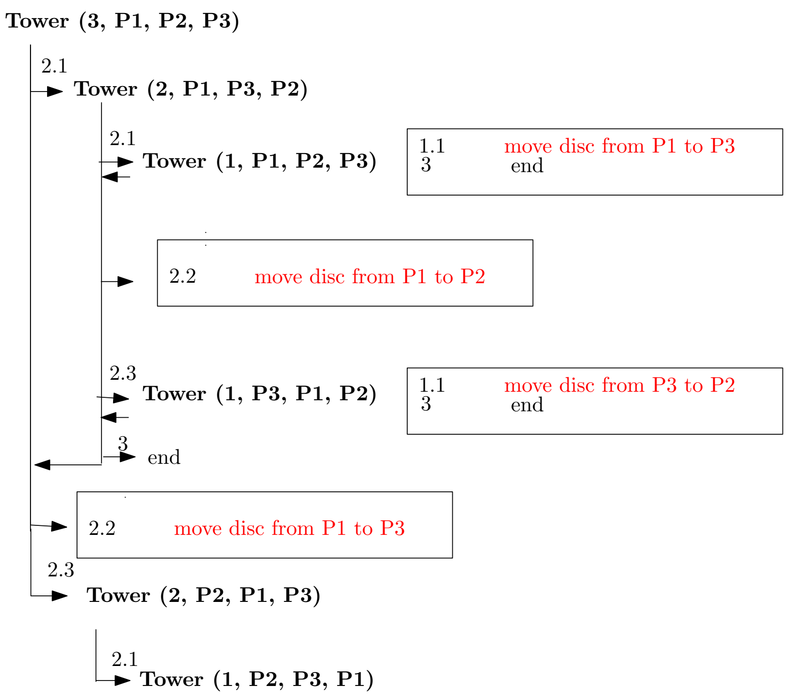
Step 10: After this step, in line number 2.3 of `Tower (3, P1, P2, P3)`, the recursive call to `Tower()` is made with parameters `n=2`, `P2`, `P1` and `P3` in that order i.e., `Tower (2, P2, P1, P3)` is invoked.



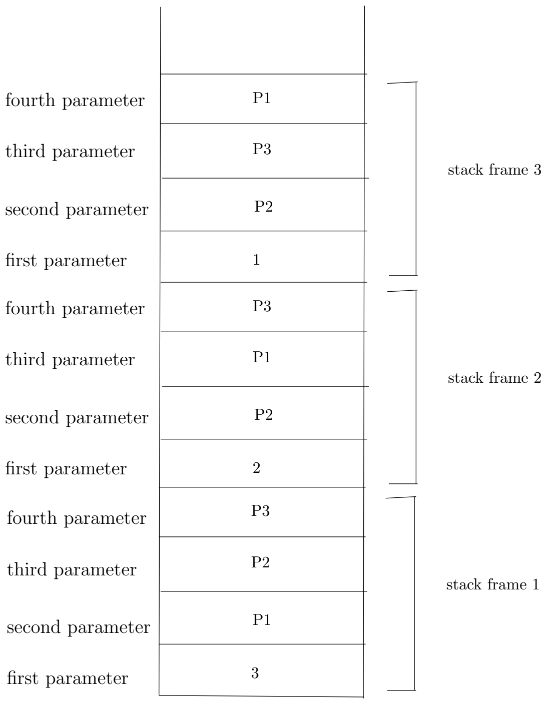
The memory stack frame diagram (except the stack frame of main) is as shown below.



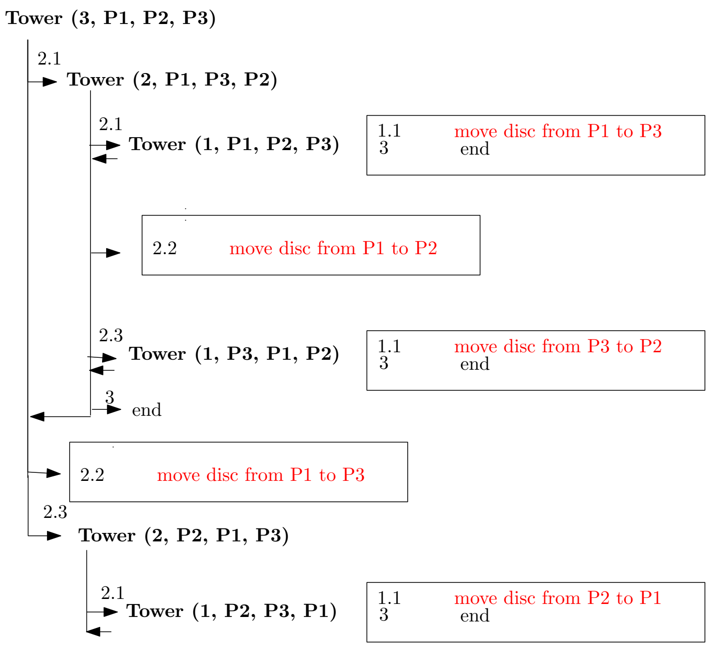
Step 11: The execution of `Tower (2, P2, P1, P3)` starts and the `if` condition is *false* and in line 2.1, the function `Tower()` is called with parameters `n=1`, `P2`, `P3` and `P1` in that order i.e., `Tower (1, P2, P3, P1)` is invoked.



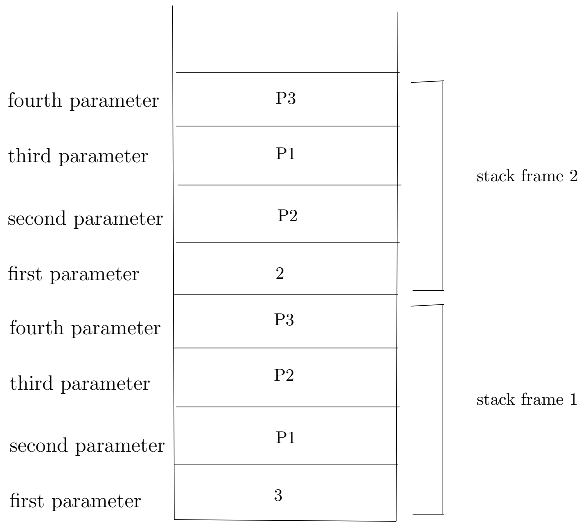
The memory stack frame diagram (except the stack frame of main) is as shown below.



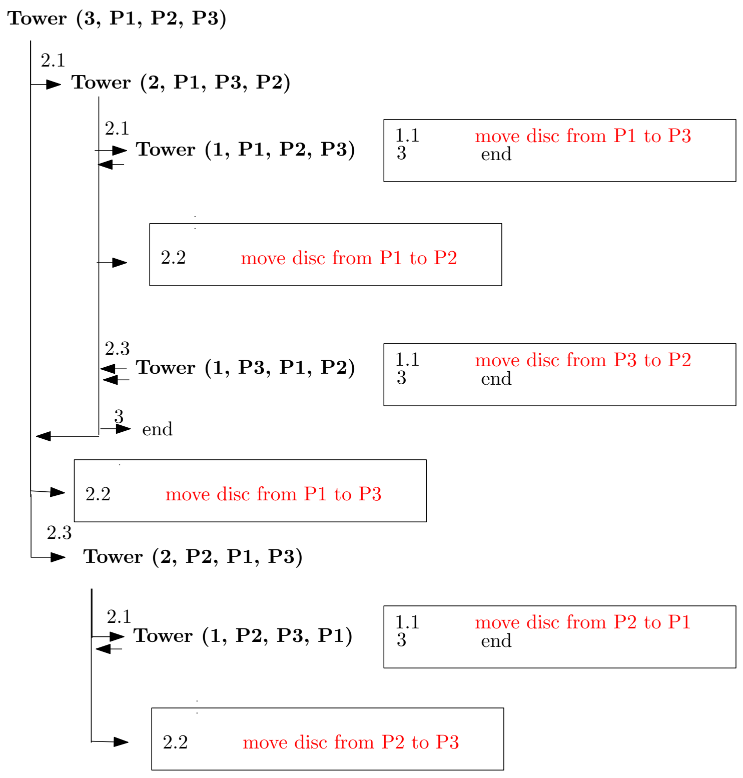
Step 12: Now, `Tower (1, P2, P3, P1)` starts executing and the condition `if(n==1)` becomes *true*. In line 1.1, the instruction is to move the disc from the second parameter to the fourth parameter i.e., from P2 to P1. After this, the program control moves to line 3 and this finishes the execution of `Tower (1, P2, P3, P1)` and the program control is returned to the calling function `Tower (2, P2, P1, P3)`.



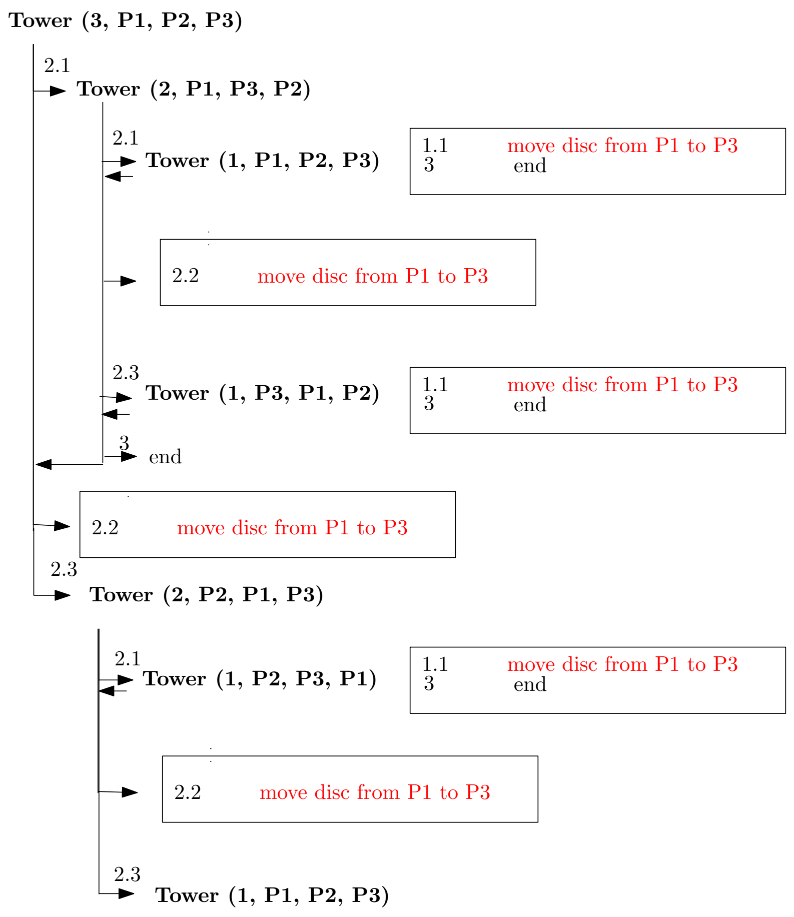
The memory stack frame diagram (except the stack frame of main) is as shown below.



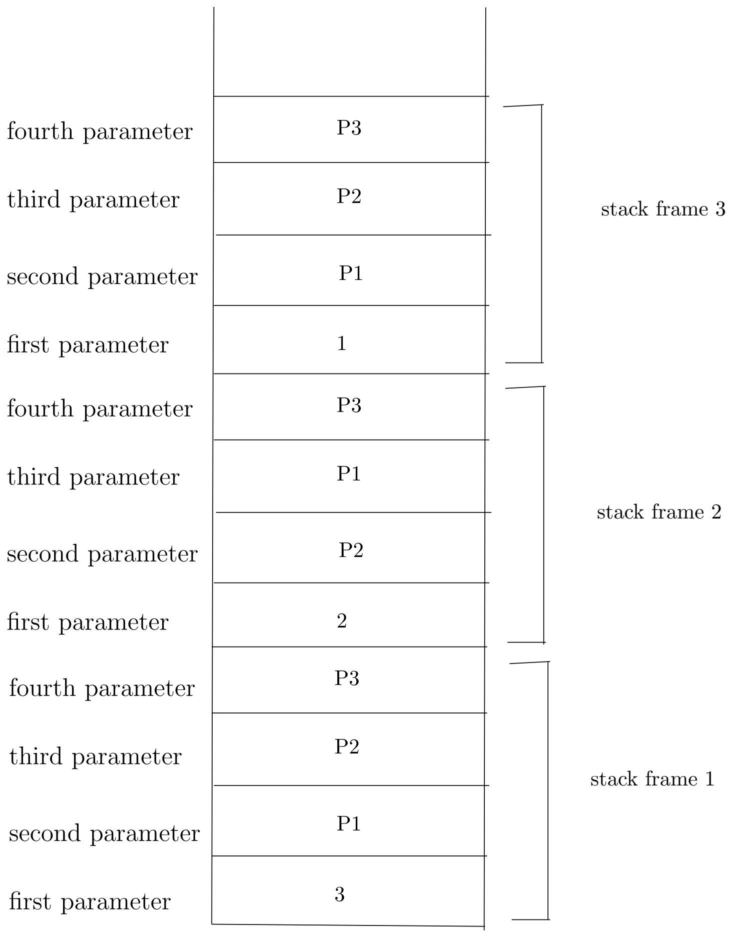
Step 13: Now, the program control is in the function `Tower (2, P2, P1, P3)` and the program execution continues from line number 2.2 and in this step, a disc is moved from the second parameter to the fourth parameter of the function call `Tower (2, P2, P1, P3)` i.e., from P2 to P3.


Tower (3, P1, P2, P3)
2.1
Tower (2, P1, P3, P2)
2.1
Tower (1, P1, P2, P3)
1.1 move disc from P1 to P3
3 end
2.2 move disc from P1 to P2
2.3
Tower (1, P3, P1, P2)
1.1 move disc from P3 to P2
3 end
3 end
2.2 move disc from P1 to P3
2.3
Tower (2, P2, P1, P3)
2.1
Tower (1, P2, P3, P1)
1.1 move disc from P2 to P1
3 end
2.2 move disc from P2 to P3

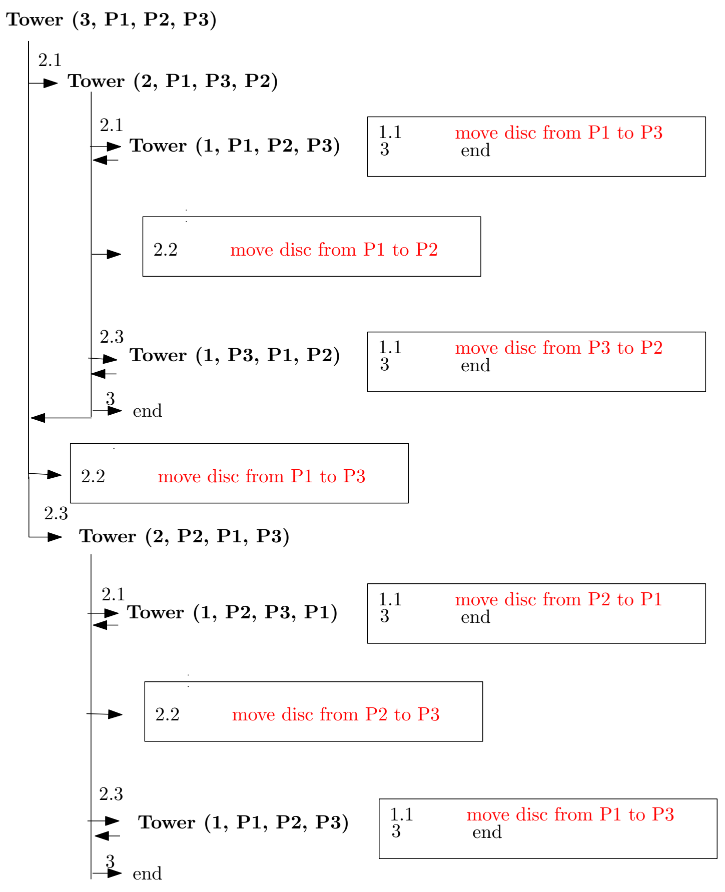
Step 14: After this step, in line number 2.3 of `Tower (2, P2, P1, P3)`, the recursive call to `Tower()` is made with parameters `n=1`, `P1`, `P2` and `P3` in that order i.e., `Tower (1, P1, P2, P3)` is invoked.



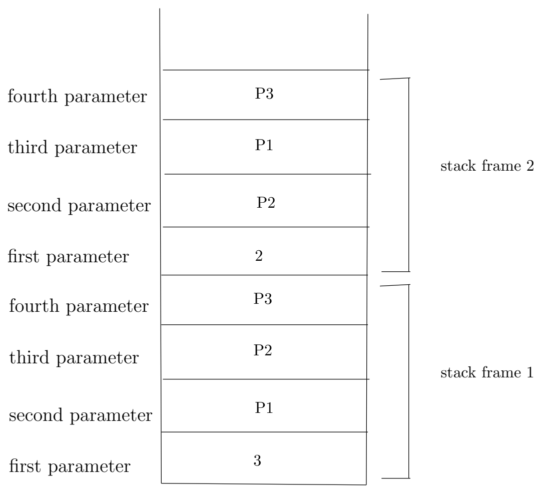
The memory stack frame diagram (except the stack frame of main) is as shown below.



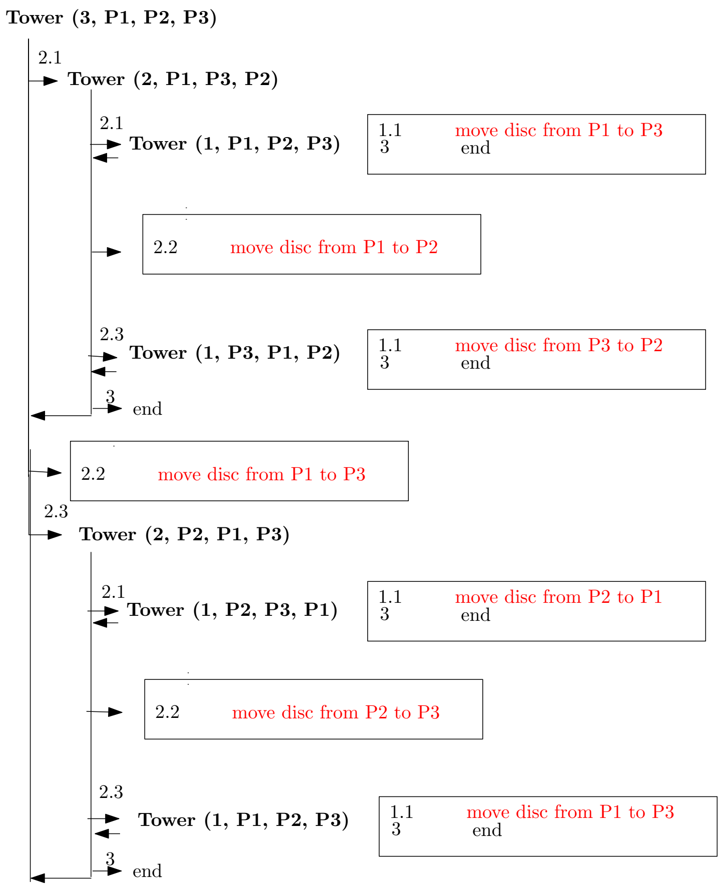
Step 15: Now, `Tower (1, P1, P2, P3)` starts executing and the condition `if(n==1)` becomes *true*. In line 1.1, the instruction is to move the disc from the second parameter to the fourth parameter i.e., from P1 to P3. After this, the program control moves to line 3 and this finishes the execution of `Tower (1, P1, P2, P3)` and the program control is returned to the calling function `Tower (2, P2, P1, P3)` to line 3.



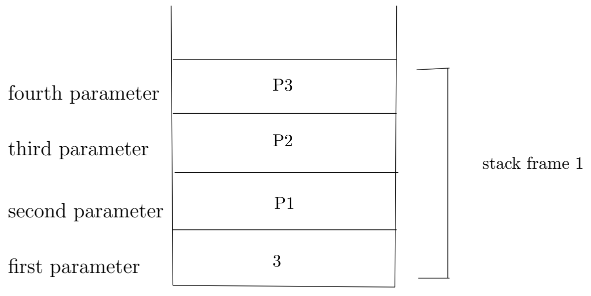
The memory stack frame diagram (except the stack frame of main) at this point is as shown below.



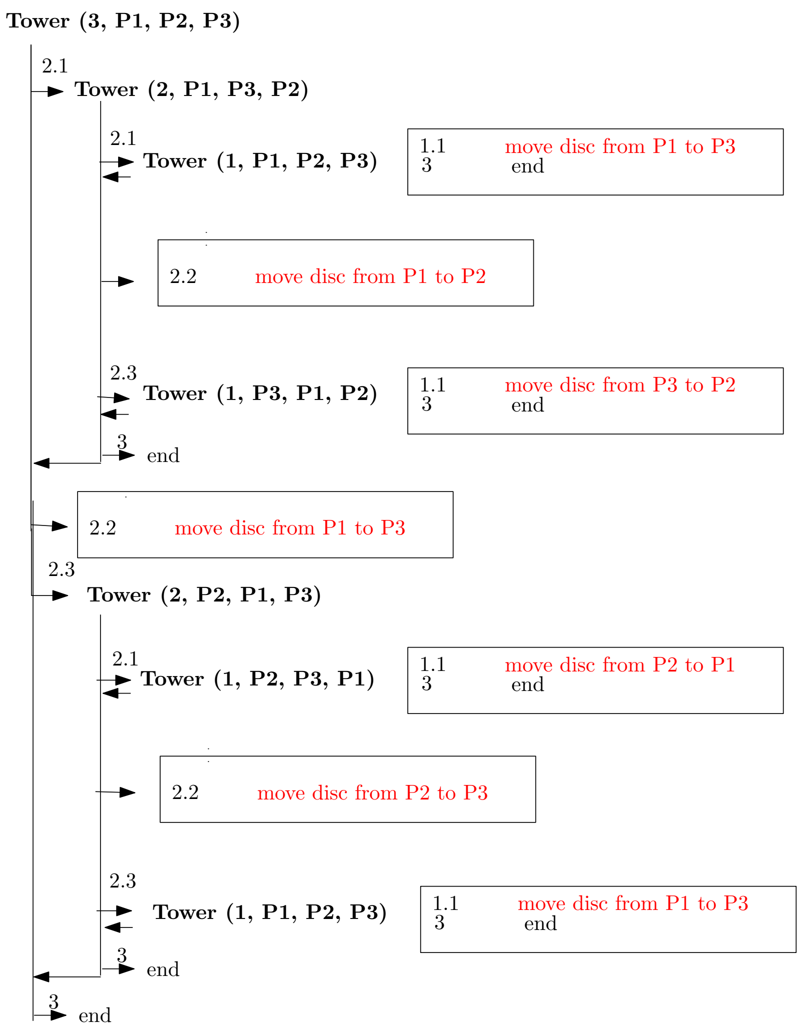
Step 16: Now, `Tower (2, P2, P1, P3)` finishes its execution by executing line number 3 and the program control is in turn transfered to line number 3 of `Tower (3, P1, P2, P3)`.



The memory stack frame diagram at this point (except the stack frame of main) is as shown below.



Step 17: Now, `Tower (3, P1, P2, P3)` finishes its execution by executing line number 3.



After this, the program control is transfered to the function `main()`. The sequence of disc movements are indicated in the figure in Red color, the order of movements can be obtained by reading the Red lines in the figure from top to the bottom order.

Proof of Correctness

The proof of correctness of the above recursive algorithm for the Tower of Hanoi problem can be done using the method of mathematical induction.

We will prove a more generalized statement about the algorithm, where there could be more than n discs on the poles. We will prove the following statement:

S(n): Tower(n, P1, P2, P3) moves the top n discs from $P1$ to $P3$ keeping all rules of the Towers of Hanoi game without moving any other discs other than these n discs at any of the intermediate stages.

Base Case (n=1): From the algorithm, it is clear that for $n = 1$, the solution keeps all rules of the game, it terminates and the objective is achieved. It can be easily verified that **S(n)** is true for n=1.

Induction hypothesis: Let $k > 1$ be any fixed integer and assume that **S(n)** is true for all $n < k$.

Inductive step: Consider the statement **S(n)** for $n = k$. Suppose a call to the function Tower(n, P1, P2, P3) is made with $n = k$.

In step 2.1, the algorithm calls Tower(n-1, P1, P3, P2). By induction hypothesis $S(k-1)$, this step moves the top $k-1$ discs (i.e., the smaller $k-1$ discs) from $P1$ to $P2$ keeping all rules of the Towers of Hanoi game without moving the largest disc at all in any of the intermediate stages. By induction hypothesis, among the $k-1$ discs getting moved, a larger disc is never placed on top of a smaller disc. Hypothesis also guarantees that the largest disc is never moved. So, the largest disc remains at the bottom of $P1$ and it is never placed on top of any other disc. At some intermediate stages, some discs may get placed over the largest disc on $P1$. But, this does not violate any rules of the game. After completing Step 2.1, only the largest disc is there in $P1$. All other $k-1$ discs are in $P2$, in the increasing order of their radius, from top to bottom. Pole $P3$ is empty.

In step 2.2, the top disc from pole $P1$ (which is the largest disc) is moved to pole $P3$. This step is a valid move and it does not violate any rules of the game. After this step, pole $P1$ is empty, the largest disc is on $P3$, and all other $k-1$ discs are in $P2$, in the increasing order of their radius, from top to bottom.

In step 2.3, the algorithm calls Tower(n-1, P2, P1, P3). By induction hypothesis $S(k-1)$, this step moves the top $k-1$ discs (i.e., the smaller $k-1$ discs) from $P2$ to $P3$ keeping all rules of the Towers of Hanoi game without moving the largest disc at all in any of the intermediate stages. By induction hypothesis, among the $k-1$ smaller discs getting moved, a larger disc is never placed on top of a smaller disc. Hypothesis also guarantees that the largest disc is never moved. So, the largest disc remains at the bottom of $P3$ and it is never placed on top of any other disc. Some discs get placed over the largest disc on $P3$. But, this is allowed, as per the rules of the game. After completing Step 2.3, the largest disc remains at the bottom of $P3$. and all the other $k-1$ smaller discs also get transferred from $P2$ to $P3$, and they will be getting arranged in $P3$ in the increasing order of radius, from top to bottom. Thus, all k discs get arranged in the required way in $P3$. Poles $P1$ and P_2 are empty.

This proves that $S(n)$ is true for $n = k$ as well. Hence by induction, $S(n)$ is true for all natural numbers n.

Implementation in C

A C program to solve Towers of Hanoi for an input `n` is given below.

```
#include<stdio.h>
void Tower(int, int, int, int);
void main()
{  int n, p1=1, p2=2, p3=3;
   printf("Enter the number of discs\n");
   scanf("%d", &n);
   if (n <=0)
   {   printf("no discs to move!\n");
       return;
   }
   printf("---------------Sequence of disc movements------------------\n\n");
   Tower(n, p1, p2, p3);
}
```

```
void Tower(int n, int p1, int p2, int p3)
{
  // To move n discs from pole p1 to pole p3, using intermediate pole p2.
   if (n==1)
   {   printf("move disc from pole %d to pole %d \n",p1, p3);
      return;
   }

   Tower(n-1,p1, p3, p2);
   printf("move disc from pole %d to pole %d \n",p1, p3);
   Tower(n-1,p2, p1, p3);
}
```

The output of the program for n=2 is given below.

```
Enter the number of discs
2
----------------Sequence of disc movements-------------------

move disc from pole 1 to pole 2
move disc from pole 1 to pole 3
move disc from pole 2 to pole 3
```

The output of the program for n=3 is given below.

```
Enter the number of discs
3
----------------Sequence of disc movements-------------------

move disc from pole 1 to pole 3
move disc from pole 1 to pole 2
move disc from pole 3 to pole 2
move disc from pole 1 to pole 3
move disc from pole 2 to pole 1
move disc from pole 2 to pole 3
move disc from pole 1 to pole 3
```

Some observations:
From the above two outputs, it can be easily noticed that there is some similarity in the output of `n=2` and the first three steps of the output for `n=3`. The only difference is in the pole numbers. In the output for `n=2`, two discs are getting moved from Pole 1 to Pole 3, using intermediate pole Pole 2. In the first three steps of the output for `n=3`, two discs are getting moved from Pole 1 to Pole 2, using intermediate pole Pole 2. The first three steps of `n=3` can be obtained by swapping Pole 2 and Pole 3 in the output of $n = 2$.

In the same way, there is also some similarity in the last three steps of the output for `n=3` and the output of `n=2`. Just as the above case, the difference is only in the pole numbers. The first three steps of `n=3` can be obtained by swapping Pole 1 and Pole 2 in the output of $n = 2$,.

Number of moves

Since in the solution for n discs, there are two recursive calls to the algorithm with $n-1$ discs (steps 2.1 and 2.3), and one move in step 2.2, the number of moves, M(n) used by our algorithm to move `n` discs from Pole 1 to Pole 3 can be calculated by the following recursive function.

$$M(n) = \begin{cases} 1 & : n = 1 \\ 2M(n-1) + 1 & : n > 1 \end{cases}$$

This gives the closed form expression $M(n) = 2^n - 1$, $n \geq 1$.