**3. Tower of Hanoi wihout recursion**

The iterative solution can be figured out analyzing the recursive solution. Two things worth notice are that:

Total no. of moves required are 2n-1 where n is the number of disks.

If the bars are arranged in space then for the even number of disks the movement of disks will start in clockwise direction and if the number of disks is odd then the movement will start in anticlockwise direction.

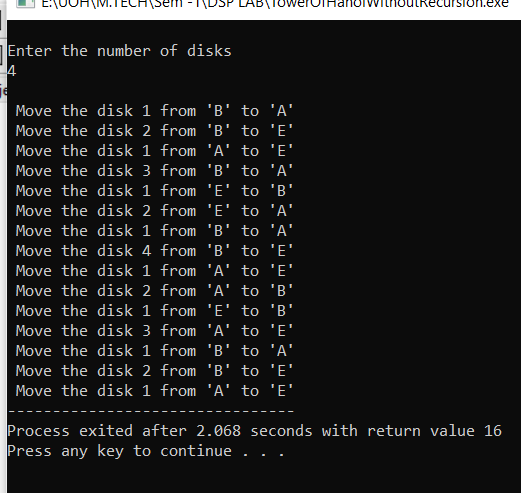
With the help of above two observations we can devise the algorithm as:

TowerOfHanoi(source, destination, auxiliary, numDisks)  
1. Calculate total no. of moves as pow(2, numDisks) - 1. numDisks is no. of disks.   
2. If numDisks is even then interchange the destination pole with the auxiliary pole. (This is to ensure that moves are in clockwise for even disks and anticlockwise for odd disks)  
3. for i = 1 to number of moves calculate in step 1:  
a. if i%3 == 1: legal movement of topmost disk between source and destination.  
b. if i%3 == 2: legal movement of topmost disk between source and auxiliary.  
c. if i%3 == 0:legal movement of topmost disk between auxiliary and destination.

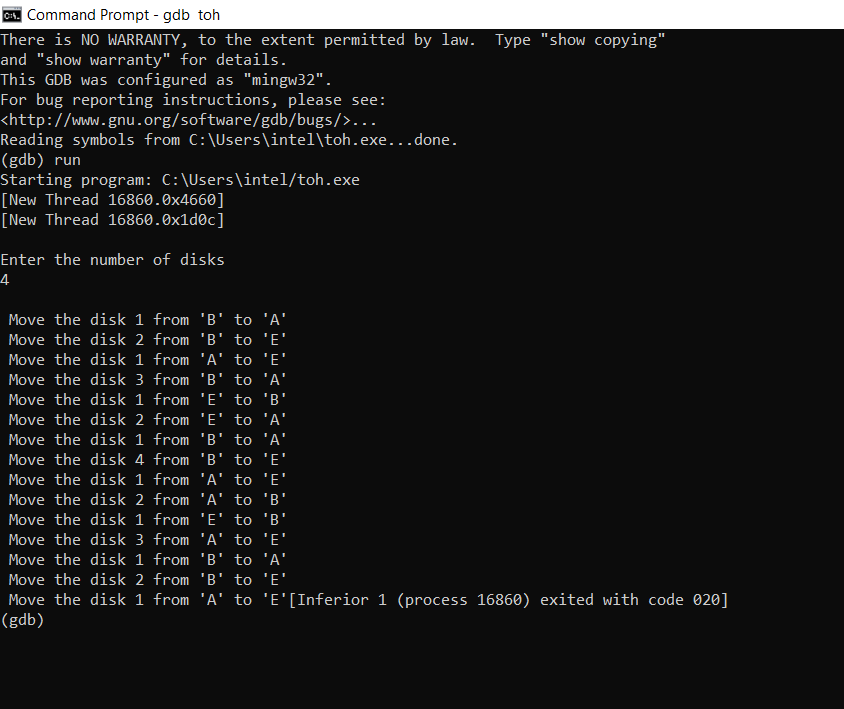
**“Cases** for legal movement:

1. **When one of the two poles is empty** we must move the disk from non empty pole to the empty pole.
2. **When the top disk of one pole is smaller than the other** we move the smaller of two disks to the pole with larger disk.”

* We have implemented a structure for the stack
* **createStack()**: is used to create a stack of a given capacity.
* **Push()** and **Pop()**: are used to add or delete elements from our stack.
* **Movedisk():** is the function that actually moves the disks from one pole to another.
* **moveDisksBetweenTwoPoles():** decides the movement of disks from one pole to another , satisfying the above mentioned rules.
* **TOH():** The actual function that implements the problem. If total number of disks is even, we swap auxiliary tower with the destination tower.
* **Main():**The main driving function. Three different stacks are created for each of the three towers using createstack() and then, toh() is called which solves our TOH problem.

**Execution Screenshots:  
**

**Debug Screenshots:**

****