TOWER OF BRAHAMA

THE PROBLEM

- MOVE ALL DISKS FROM COLUMN A TO B
- ONE DISK AT A TIME
- NO LARGER DISKS ON SMALLER DISKS

IMPLEMENTATION 1 = RECURSION (STANDARD)

IMPLEMENTATION 2 = STACK:
- EACH PEG IS ITS OWN STACK
- TAKES $X_n = 2X_{n-1} + 1$ → $X_n = 2^{n-1}$ MOVES

- stack.h HEADER FILE GIVEN - IMPLEMENT FUNCTION

TASKS

- PARSE COMMAND LINE OPERATIONS
  - "n x" : SETS # OF DISKS TO X DEFAULT 5
  - "s" : PRINT OUT MOVES PERFORMED W/ STACK
  - "r" : PRINT OUT MOVES PERFORMED W/ RECURSION
- PRINT # OF MOVES USED
- MULTIPLE COMMAND LINE ARGUMENTS CAN BE CALLED

- tower.c CONTAINS MAIN CODE

- stack.c CONTAINS STACK FUNCTIONS USED IN tower.c

STACK / ITERATIVE NOTES

- DISK ONE MOVES EVERY OTHER MOVE

① - AFTER IT MAKE LEGAL MOVE W/ SMALLEST DISK POSSIBLE

2a - NEXT 1 MOVE TO SMALLEST DISK (IF YOU JUST MOVED AN EVEN DISK)

- IF CHOOSING BETWEEN DISK + EMPTY CHOOSE DISK! - OR DISK JUST MOVED

PEEK RETURNS 0 FOR EMPTY STACKS/PEGS

2b - NEXT 1 MOVE TO LARGEST DISK (IF YOU JUST MOVED AN ODD DISK)

- IF CHOOSING BETWEEN DISK + EMPTY CHOOSE EMPTY - OR PEG THAT WASN'T JUST MOVED

FIRST MOVE:

0-5

- ODD # DISKS FIRST MOVE = DESTINATION
- EVEN # DISKS FIRST MOVE = OTHER / AUX DISK

```
    DO FIRST MOVE (EVEN OR ODD DISKS);

  WHILE ( PEG A != EMPTY && PEG C != EMPTY ) {
①      MOVE SMALLEST DISK POSSIBLE
           PRINT MOVE

2a/2b  MOVE DISK 1 ( KEEP TRACK IF LAST DISK WAS EVEN OR ODD)
           PRINT MOVE

③      CHECK IF GAME OVER — A/C EMPTY
```

```
Stack* move_smallest_disk ( MAYBE PASS PEG LAST MOVED TO) {
    - LOOK ON TWO PEGS THAT WERE NOT JUST MOVED
      TO
          - USE STACK_PEEK + COMPARE VALUES
    - LOCATE SMALLER TOP DISK
          - IF ONE PEG IS 0 USE OTHER ONE

    - MOVE SMALLER TOP DISK TO OTHER ELIGIBLE
      PEG
          - POP FROM ORIGINAL PEG -> STORE IN VARIABLE
          - PUSH TO NEW PEG

    - RETURN REFERENCE/ COPY OF LAST PEG MOVED TO
}

     -> RETURN                                        -> PEG LAST
                                                         MOVED TO
Stack* move_disk1 ( int last_disk#, stack *s ) {

    LOCATE DISK 1 - STACK PEER + CHECK + STORE IN VAR
    SWITCH ( last_disk# % 2 ) :

    case 0:
        - MOVE DISK 1 TO PEG JUST ADDED TO
        - POP FROM PEG DISK 1 IS @
        - PUSH DISK 1 TO PEG LAST MOVED TO

    case 1:
        - MOVE DISK 1 TO PEG NOT JUST ADDED TO
              - AND OBV. NOT YOUR OWN PEG
        - POP FROM PEG DISK 1 IS @
        - PUSH DISK 1 TO PEG LAST MOVED TO

        - RETURN REFERENCE TO/COPY OF LAST PEG
          MOVED TO
}
```

RECURSIVE IMPLEMENTATION

BROAD RECURSIVE STRATEGY: CREATING SUBTASKS

1) BASE CASE (ASSUME FUNCTION NAME IS h)
   - h(1) = 1 DISK = MOVE DISK FROM STARTING POINT TO DESTINATION ✓

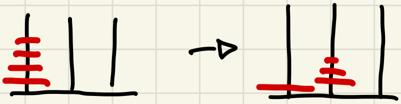
2) ASSUME h(n-1) WORKS
3) PROVE h(n) WORKS USING h(n-1)

PSUEDOCODE:

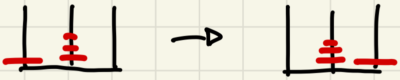
```
h( disks, start, end, other) {

    base case : if ( disks == 1) {
        - move disk from start to end
    }

                                          → NOT USED
      h( disks-1, start, other, end)
          - moves disks-1 disks to the intermediate
            peg : resulting in
```



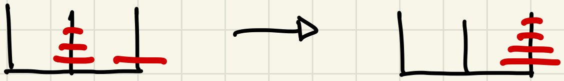
```
                                                   → NOT USED
      move_disk( disk, start, end, other)
          - moves largest disk from start peg to
           destination peg
```



NOT USED

h(disks-1, other, end, start)

- finally moves disks-1 disks from intermediate peg to the destination peg on top of the largest disk.



- THESE THREE STEPS IN CONJUNCTION WITH THE BASE CASE BREAK UP THE PROBLEM INTO SUB-PROBLEMS WITH n-1 DISKS, THEN N-2, AND SO ON UNTIL THERE IS ONLY ONE DISK LEFT TO MOVE.