```c
/*
AUTHOR: Okoronkwo Alfred Chinedu
LANG: C
OBJECTIVE: To demonstrate the Towers of Hanoi puzzle by moving the disks
DESCRIPTION:
    In this solution we will illustrate the classic Towers of Hanoi puzzle
    by moving the disks around. We will define a data structure for the disk,
    the tower and another to hold the three towers together. Next, we
    initialize the towers given them names, making them ready to hold disks.
    After this, we load the source tower, which is the tower that will
    originally hold all the disks. To do this, we first create the disks, each
    with a unique diameter and then hook them on the tower.

    Each of these towers support last-in-first-out (LIFO), much like a stack.
    This helps to maintain the constraints of the problem. Remember, we do not
    want any disk to break in the process of transferring them from one tower
    to another. How can we break a disk? Well, the disks are customized such
    that a bigger disk cannot sit on a smaller one otherwise, the smaller one
    will break. If this happens, we have lost the game.

    Having our major constraint in mind, we need to ensure that while mounting
    a disk on a tower, that the tower is either empty or the disk below is
    larger than the one we are about to mount. With this in mind, we must
    choose to start with the biggest disk and move up in a decreasing order.
    But, we have the smaller disks on top. It's not too much of a problem. We
    will have to make use of the intermediary tower to ensure we don't break
    any disk.

    When loading the source tower, we will start with the biggest disk. Create
    it, hook it unto the source tower and so on. When we are done with this,
    the content of each tower is displayed. This assures us that going into the
    game, we have not been cheated. That all the disks are intact.

    It's time to move the disks to the destination tower. First of, we move the
    smaller N - 1 disks sitting atop to the middle tower and then, move the
    last disk to the destination. If you notice, we have all the other disks
    stacked on the middle tower. Again, we move N - 2 disks from the
    intermediary tower to the source tower and then move the last disk on the
    intermediary tower to the destination tower.

    This process continues this way until we have transferred all the disks
    to the destination tower and that none is broken in the process. How can we
    be sure, none is broken? With every move, we display the content of the
    towers.

    Did you know, it will take 2 ^ N to completely move a stack of N disks to
    the destination? We keep track of the total steps taken in our towers data
    structure and increment it before each move.  We also display it each time
    we show the content of the towers.

    Remember, do not break any disk else, it's game over. Enjoy the rest of the
    game.
```

```
REFERENCES:
    1. Data Structures and Algorithm in Java by Robert Lafore - Chapters 5 and
        6.
    2. Algorithmic Puzzles by Anany Levitin and Maria Levitin - Puzzle 83.
    3. Java How to Program (Ninth Edition) by Paul Deitel and Harvey Deitel -
        Pages 777 - 779.
*/
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>

struct disk{
    int diameter;
    struct disk *next;
};

typedef struct{
    char tower_name;
    struct disk *n_tower;
} tower;

typedef struct{
    long long n_steps;
    tower tower_a;
    tower tower_b;
    tower tower_c;
} towers;

towers tws;

void towers_initialize(char tower_a, char tower_b, char tower_c){
    tws.n_steps = 0;

    tws.tower_a.tower_name = tower_a;
    tws.tower_a.n_tower = NULL;

    tws.tower_b.tower_name = tower_b;
    tws.tower_b.n_tower = NULL;

    tws.tower_c.tower_name = tower_c;
    tws.tower_c.n_tower = NULL;
}

struct disk* create_disk(int diameter){
    struct disk *new_disk = malloc(sizeof(struct disk));
    new_disk->diameter = diameter;
    new_disk->next = NULL;
    return new_disk;
}

void load_source_tower(int n_disks){
    while (n_disks > 0){
        struct disk* new_disk = create_disk(n_disks);
```

```c
        --n_disks;
        new_disk->next = tws.tower_a.n_tower;
        tws.tower_a.n_tower = new_disk;
    }
}

void display_disks(tower tw){
    printf("Tower %c: ", tw.tower_name);
    struct disk *current = tw.n_tower;
    while (current != NULL){
        printf("%d ", current->diameter);
        current = current->next;
    }
    printf("\n");
}

void display_towers(){
    display_disks(tws.tower_a);
    display_disks(tws.tower_b);
    display_disks(tws.tower_c);

    printf("=====================================\n");
}

void move_disk(char from, char to){
    ++tws.n_steps;
    struct disk *new_disk;
    if (from == tws.tower_a.tower_name){
        new_disk = tws.tower_a.n_tower;
        tws.tower_a.n_tower = tws.tower_a.n_tower->next;
    } else if (from == tws.tower_b.tower_name){
        new_disk = tws.tower_b.n_tower;
        tws.tower_b.n_tower = tws.tower_b.n_tower->next;
    } else if (from == tws.tower_c.tower_name){
        new_disk = tws.tower_c.n_tower;
        tws.tower_c.n_tower = tws.tower_c.n_tower->next;
    }

    printf("Step %lld: Disk %d moved from %c to %c\n", tws.n_steps,
        new_disk->diameter, from, to);

    if (to == tws.tower_a.tower_name){
        new_disk->next = tws.tower_a.n_tower;
        tws.tower_a.n_tower = new_disk;
    } else if (to == tws.tower_b.tower_name){
        new_disk->next = tws.tower_b.n_tower;
        tws.tower_b.n_tower = new_disk;
    } else if (to == tws.tower_c.tower_name){
        new_disk->next = tws.tower_c.n_tower;
        tws.tower_c.n_tower = new_disk;
    }

    display_towers();
```

```c
}

void move_disks(int n_disks, char from, char inter, char to){
    if (n_disks == 1)
        move_disk(from, to);
    else {
        move_disks(n_disks - 1, from, to, inter);
        move_disk(from, to);
        move_disks(n_disks - 1, inter, from, to);
    }
}

void free_tower(tower tw){
    struct disk *current = NULL;
    while (tw.n_tower != NULL){
        current = tw.n_tower;
        tw.n_tower = tw.n_tower->next;
        free(current);
        current = NULL;
    }
}

int main(int argc, char *argv[]){
    printf("%d\n", argc);

    int n = atoi(argv[1]);
    char tw_a = 'A';
    char tw_b = 'B';
    char tw_c = 'C';

    towers_initialize(tw_a, tw_b, tw_c);
    load_source_tower(n);
    display_towers();
    move_disks(n, tw_a, tw_b, tw_c);
    free_tower(tws.tower_a);
    free_tower(tws.tower_b);
    free_tower(tws.tower_c);
    exit(0);
}
```