Tower of Hanoi
Design

Pseudocode

```
define options for 3 arguments

int main(get command argument) {

  while arguments supplied {
    switch {
        case n:
            set n_discs to user input,
            otherwise default 5 disks
    case stack:
      set stack to true
    case recursion
      set recursion to true
    }
  }

  if stack {
    create stack of each rod with n_discs as capacity
    print number of moves stack function takes
    using stack_iterator

    delete each stack created before to free memory
  }
 if recursion {
    print number of moves recursion function takes
    using recursion
    }
 }
```

Helper Functions

Recursive function that solves ToH by making smaller ToH games
Ends when reaching smallest ToH game with 1 disc

```
int hanoi(n_disc, start rod, end rod) {
     n_moves = 0
    increment n_moves each recur
     if n_discs is 1 {
        print disc # and movement
    } else {
     find aux rod
     recur hanoi with 1 less disc from start rod to aux rod
     print disc # and movement
     recur hanoi with 2 less discs from aux rod to end rod
    }
    return n_moves hanoi function took
    }
```

Checks to see what legal move should be made between two input rods by looking at top disc of each one

```
void stack_move_disc(start rod, end rod, rod 1 name, rod 2 name) {

  if start rod is empty {
     pop disc from end rod
     push disc from end rod to start
     print movement of disc
}  else if end rod is empty {
     pop disc from start rod
     push disc from start rod to end
     print movement of disc
}    else if start rod top disc > end rod top disc {
       pop disc from end rod
       push disc from end rod to start
       print movement of disc
 }    else start rod top disc < end rod top disc {
         pop disc from start rod
         push disc from start rod to end
         print movement of disc
  }
 }
```

Helper Functions

Finds minimum moves needed to solve game with n_discs and iterates through each turn. Depending on move #, will move the a disc from two certain rods. If n_discs is even, switches aux and end rod as even n_discs requires different moving pattern

```
int stack_iterator(n_disc, Start stack, Aux stack, End stack) {
   declare n_moves and moves_taken, set moves_taken to 0
   set rod/stack names to A, C, B, respectively according to inputs
   if even n_discs {
      set C (aux) rod to B (end) rod and vice versa.
 }
calculate minimum n_moves using 2^(n_discs) - 1
and set = to n_moves

for i = n_discs to 1, decrementing i {
  push i into Start stack, setting up game
}
for i (move) = 1 to n_moves, incrementing i {
   if move % 3 =1 {
      stack_move_disc(start rod, end rod, A, B)
   }   else if move % 3 =2 {
          stack_move_disc(start rod, aux rod, A, C)
   }   else if move % 3 =0 {
          stack_move_disc(aux rod, end rod, C, B)
   }

   increment moves_taken
}
  return moves_taken
}
```

Functions from stack.c that are used in tower.c to create stacks, push and pop items, delete, etc.

```
Stack *stack_create( capacity, name) {
allocate memory for stack
if no name, return 0
if capacity < 1, set capacity to 1
 set top of stack to be leading 0
 allocate memory in stack = to capacity
return stack
 }

stack_delete(stack name) {
     free stack items from memory
     free stack from memory
}

stack_pop(stack name) {
     if no name, return -1
     if top of stack has item,
       decrement top of stack pointer
       and return first item
     if stack is empty,
       return -1
}

stack_push(stack name, item) {
    if no name, return
    if top of stack = capacity,
      double capacity and allocate memory for it
      in stack
    if space in stack,
      add item to top of stack items array
      and increment stack top pointer
 }

stack_empty(stack name) {
  check if stack has any item and if not,
  return True
}

stack_peek(stack name) {
  if stack top not 0,
    return top item in stack using pointer = [top - 1]
  else, return 0;
}
```

Notes

Recursive function derived from video: https://www.youtube.com/watch?v=rf6uf3jNjbo&ab_channel=Reducible

Video discussed how ToH could be solved by treating each movement of a disc as a smaller ToH
Video also created and discussed recursive code for ToH game in Python

Stack Iterator function derived from: https://www.youtube.com/watch?v=ZWNK34T0YKM&ab_channel=PooyaTaheri

Video discussed that depending on the move, certain rods will have a movement of disc
Video also mentioned a recursion method

I later realized that although I know which rods to move a disc from if I know which move it is,
I still need to check what move I should make. stack_move_disc checks the top discs of each of its
2 input rods and see which move is legal

stack.h header was given to us by lab, but actual stack functions were derived and edited from
Lecture 14 discussing Stacks and Recursion