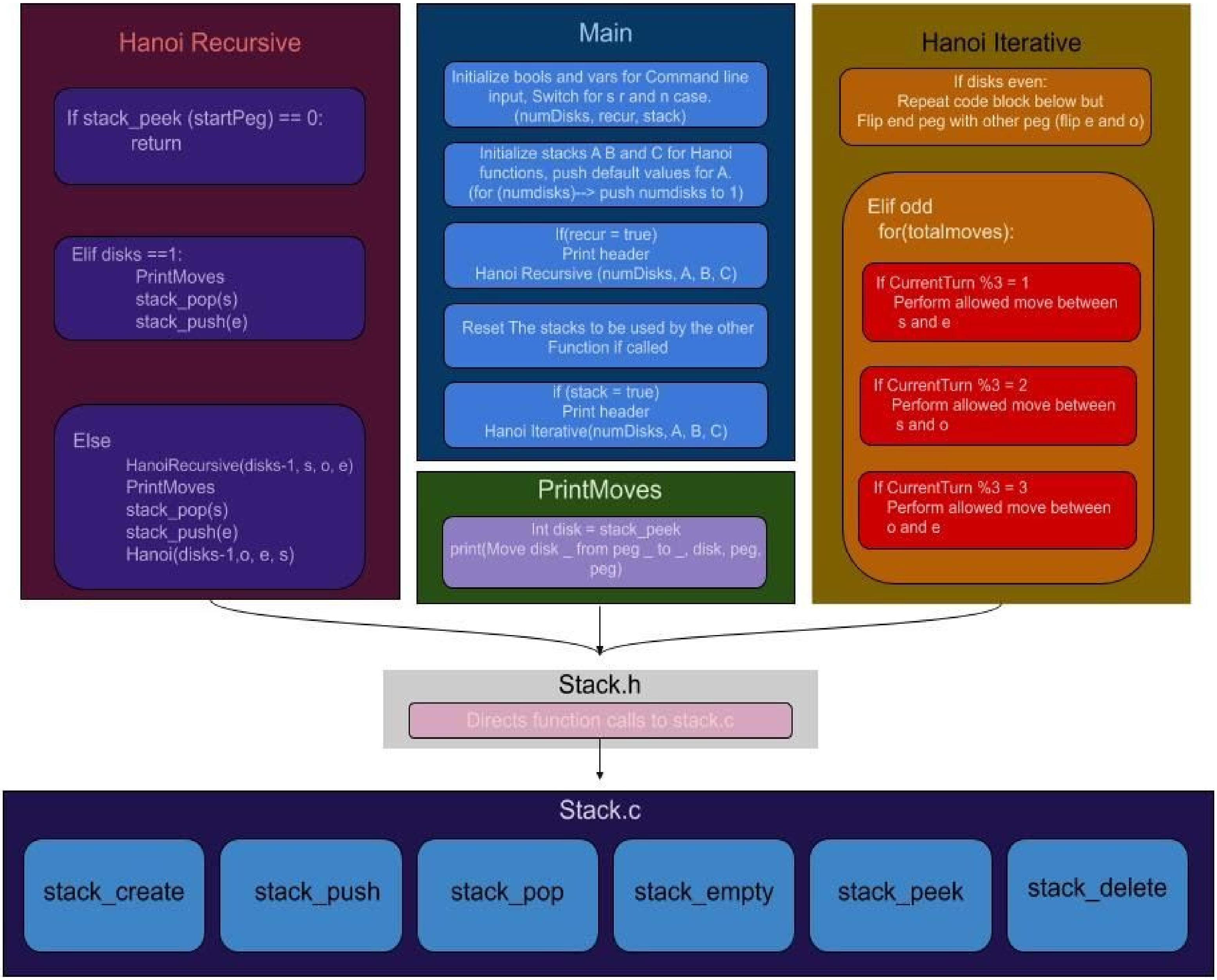
# DESIGN



## Hanoi Recursive

```
If stack_peek (startPeg) == 0:
        return

Elif disks ==1:
        PrintMoves
        stack_pop(s)
        stack_push(e)

Else
        HanoiRecursive(disks-1, s, o, e)
        PrintMoves
        stack_pop(s)
        stack_push(e)
        Hanoi(disks-1,o, e, s)
```

## Main

Initialize bools and vars for Command line input, Switch for s r and n case. (numDisks, recur, stack)

Initialize stacks A B and C for Hanoi functions, push default values for A. (for (numdisks)--> push numdisks to 1)

```
If(recur = true)
Print header
Hanoi Recursive (numDisks, A, B, C)
```

Reset The stacks to be used by the other Function if called

```
if (stack = true)
Print header
Hanoi Iterative(numDisks, A, B, C)
```

## PrintMoves

```
Int disk = stack_peek
print(Move disk _ from peg _ to _, disk, peg, peg)
```

## Hanoi Iterative

If disks even:
Repeat code block below but
Flip end peg with other peg (flip e and o)

```
Elif odd
  for(totalmoves):
    If CurrentTurn %3 = 1
      Perform allowed move between
      s and e
    If CurrentTurn %3 = 2
      Perform allowed move between
      s and o
    If CurrentTurn %3 = 3
      Perform allowed move between
      o and e
```

## Stack.h

Directs function calls to stack.c

## Stack.c

stack_create

stack_push

stack_pop

stack_empty

stack_peek

stack_delete

# Main

Simply parses command line inputs, setting bools to true if the user called for it so that the desired function will run. Also assigns Numdisk with the user imputed value, and initializes the Stacks for the Hanoi Functions.

# Hanoi Recursive

Takes inputs of Number of disks, and three stacks in order of Start (s) End (e) and Other (o). The order is important as it is dependent on knowing which peg is the destination and which is the start. Continuously calls itself, effectively breaking the problem into smaller pieces, until the puzzle is solved.

# Hanoi Iterative

Takes inputs of Number of disks, and three stacks in order of Start (s) End (e) and Other (o). The order is important as it is dependent on knowing which peg is the destination and which is the start.

First decision made is whether the number of total disks is even. If so, a version of the algorithm where the End and Other stacks have been swapped is used. If not it continues as shown in the chart above, it takes the current move mod three to find and decide which moves it should perform, and then performs any allowed move within those bounds. Continues this until the required amount of moves required is depleted, at which point the puzzle should be solved.

# Stack.c

Separate page with basic functions for stack management.

# Stack.h

Redirects stack calls from main to Stack.c