Design

The design of the program can be split into three parts:

Stack.h file	(header file, initializing the stack functions)
Stack.c file	(c code for all the functions initialized in the header file)
Tower.c file	(the actual code for the recursion and stack implementations of the tower of Hanoi also containing the optarg code for command line)

The design of the stack.h file:

This was already given in the lab manual for assignment 3.

The design of the stack.c file:

The design for this file is from the lecture on stacks.

The pseudo-code of the stack.c file:

```
Creates the stack.
stack_create(capacity, name)
    S Stack initialize using malloc
    If stack is not true meaning that stack isn't initialized
        Returns 0
    If capacity is smaller than 1
        Round capacity up to 1
    S capacity = capacity setting the capacity given to the capacity of the stack
    S top = 0 setting pointer to 0
    S name = name setting the name given to the name of the stack
    S items are set to the length of the stack so the right amount of memory is
    allocated
    If s items are not true meaning their length of the stack isn't set
        Return 0
    Return the stack s
Erases the stack and prevent memory leaks and allows memory to be used more
efficiently.
Stack_delete(stack)
    Frees up the items
    Sets stack items to NULL
    Free stack


Takes the item on the top of the stack and sets the pointer back and returns the item taken
off the stack
Stack_pop(stack)
    If stack is not true meaning the stack is not allocated
        Return -1
    If top is bigger than 0 meaning that the stack is not empty
        The pointer for the stack is set back
        Return the item that was at the top
    If the stack is empty
        Return -1
Takes an item and adds it to the stack
stack_push(stack and item)
    If stack is not true meaning the stack is not allocated
        Return -1
    If stack pointer == the capacity meaning that the stack is full
        S capacity is multiplied by 2
        And new memory is allocated since the capacity increased
    If stack items is true it means that there are items in the stack
        In the stack set item at the current pointer
        Increase pointer by one
Checks if stack is empty
Stack_empty(stack)
    Return true if the stack pointer == 0 else return false
Returns the item that is on top of the stack
Stack_peek(stack)
    If stack is not true meaning the stack is not allocated
        Return -1
    If top is bigger than 0 meaning that the stack is not empty
        Return the item at the top of the stack
    Else if stack is empty
        Return -1
```

The design of the tower.c file:

Included stack.h file to have the stack functions in the tower.c file

Static int moves = 0 (this will count the moves this is static so it can be accessed outside of functions)

```
Recursion(number_disks, starting_peg, ending_peg, temp_peg)
    If number of disks == 1
        Print("moves disk" number_disks, "from peg" starting_peg "to peg" ending_peg)
        Increase moves + 1
    Else
        recursion(number_disks - 1, starting_peg, temp_peg, ending_peg)
        Print("moves disk" number_disks, "from peg" starting_peg "to peg" ending_peg)
        Increase moves + 1
        recursion(number_disks-1, temp_peg, ending_peg, starting_peg)
    Return
```

Summary of how the recursion function works

So it starts off with an if statement if it is disk number one move it forms the starting peg to the ending peg. But if this isn't the case it will call on the function again but decrease the number of disks and swap the from ending_peg with temp_peg. When it returns form that functions it move the disk from the starting peg to the ending peg. Then it calls the function again decreasing the number of disks but then swapping the order of peg, from starting_peg, temp_peg, ending_peg to temp_peg, ending_peg, staring_peg, this is to move the disks back to the right order.

Design of the Recursion

The idea with recursion is to divide up the problem into groups. To move the bottom disk you got to move the top disks, so when making the program I knew I had developed a function that it would do that. I know it had to start with an if statement that stops the function of calling itself over and over again. This why I put the if statement there that stops the recursion if it is the last peg to the ending peg. This is when I got lost because I didn't really know how to go further so I went to Eugene's lab section on Wednesday and he went over the problem and explained the steps. This where I found out that I needed to swap the pegs when calling the function and that I had to call it twice this is because you need to move the disks back and forth.

```
stack_func(number_disks, starting_peg, ending_peg, temp_peg)
    Moves  = 0
    Stack *A = stack_create(number_disks, starting_peg)
    Stack *B = stack_create(number_disks, ending_peg)
    Stack *A = stack_create(number_disks, temp_peg)
    For (i = number_disks; i >0; i -= 1)(puts all the disks on the A peg)
        Stack_push(A,i)

    if  even number of disks
        While true
            If b pointer == number of disks (meaning all the disks are on B peg)
                Break the loop
            If top of A <  top C or stack C is empty
                print ("Move disk" stack_peek(A)"form peg" A "to peg" C)
                Add top of A to stack C
                Pop top off stack A
                Increase moves by 1
            Else if top of C <  top A or stack A is empty
                print ("Move disk" stack_peek(C)"form peg" C "to peg" A)
                Add top of C to stack A
                Pop top off stack C
                Increase moves by 1
            If top of A <  top B or stack B is empty
                print ("Move disk" stack_peek(A)"form peg" A "to peg" B)
                Add top of A to stack B
                Pop top off stack A
                Increase moves by 1
                If b pointer == num_disks (if all disks are on B stop the loop)
                    break
            Else if top of B <  top A or stack C is empty
                print ("Move disk" stack_peek(B)"form peg" B "to peg" A)
                Add top of B to stack A
                Pop top off stack B
                Increase moves by 1
            If top of C <  top B or stack B is empty
                print ("Move disk" stack_peek(C)"form peg" C "to peg" B)
                Add top of C to stack B
                Pop top off stack C
                Increase moves by 1
                If b pointer == num_disks (if all disks are on B stop the loop)
                    break
            Else if top of B <  top C or stack C is empty
                print ("Move disk" stack_peek(B"form peg" B "to peg" C)
                Add top of B to stack C
                Pop top off stack B
                Increase moves by 1
```

```
if an odd number of disks
    While true
        If top of A <  top B or stack B is empty
            print (“Move disk” stack_peek(A)”form peg” A “to peg” B)
            Add top of A to stack B
            Pop top off stack A
            Increase moves by 1
            If b pointer == num_disks (if all disks are on B stop the loop)
                break
        Else if top of B <  top A or stack C is empty
            print (“Move disk” stack_peek(B)”form peg” B “to peg” A)
            Add top of B to stack A
            Pop top off stack B
            Increase moves by 1
        If top of A <  top C or stack C is empty
            print (“Move disk” stack_peek(A)”form peg” A “to peg” C)
            Add top of A to stack C
            Pop top off stack A
            Increase moves by 1
        Else if top of C <  top A or stack A is empty
            print (“Move disk” stack_peek(C)”form peg” C “to peg” A)
            Add top of C to stack A
            Pop top off stack C
            Increase moves by 1

        If top of C <  top B or stack B is empty
            print (“Move disk” stack_peek(C)”form peg” C “to peg” B)
            Add top of C to stack B
            Pop top off stack C
            Increase moves by 1
            If b pointer == num_disks (if all disks are on B stop the loop)
                break
        Else if top of B <  top C or stack C is empty
            print (“Move disk” stack_peek(B”form peg” B “to peg” C)
            Add top of B to stack C
            Pop top off stack B
            Increase moves by 1


stack_delete(A)
stack_delete(B)
stack_delete(C)
(This is done to prevent memory leaks and free up memory)

Return
```

Summary of how the stack function works

If it is an even number it loops through the same three moves while still abiding with the rules of the tower of Hanoi.

1. Between a and c
2. Between a and b
3. Between b and c

If it is an odd number it loops through the same three moves while still abiding with the rules of the tower of Hanoi.

4. Between a and b
5. Between a and c
6. Between b and c

The design process of the Stack function

I really didn't know how to implement this but during Eugene's Lab section on Wednesday, he had explained that one way to do it, is to look at the differences between odd and even number of disks. So I have two different loops one if it is an even number of disks and one for if it is an odd number of disks. Then when just looking at the even number of disks solution I noticed there was a continues pattern till it solves, it would always loop through three same steps

1. Between a and c
2. Between a and b
3. Between b and c

I also noticed a pattern in the odd number of disks with the three same steps

1. Between a and b
2. Between a and c
3. Between b and c

Therefore I implemented the if statement to determine which way was the legal move. And added some checks so it won't take from an empty stack or that it still could add to an empty stack.

```
main(argc, **argv)
    C = 0
    Starting_peg = "A"
    Ending_peg = "B"
    Temp_peg = "C"
    Num_disks = 5(default number of disks if not give)
    Recur = false
    Stack = false

    The getopt while loop taken from assignment 2
        And edited so when -n x(any number) is in imputed it will take the argument x
        and sets it to num_disks

    If no arg is provided ;
        It will print an error
```

```
If num_disks is less than 1
    It will print an error
If stack and recur are false
    It will print an error
If stack is true
    It will print the stack header
    And run the stack_func function
    Print number of moves
If recur is true
    It will print the stack header
    And run the recursion function
    Print number of moves
```

Design for main function

The design for this function actually comes form assignment 2 since we learned how to use getopt during that assignment

Citation

Eugene’s Lab section on Wednesday at 10 am(both recursion and stack design used pseudo-code and design ideas)