Tower of Hanoi Assignment

**Class** : Advanced C Programming

**Submission Date** : 2024/04/05

**Student ID** : 2022742021

**Affiliation** : Kwangwoon University

**Full name** : Juho Kim

**Major** : Software Engineering

# Introduction

- Brief overview of the Tower of Hanoi assignment

For the sake of simplicity, contrary to the actual code, all functions are written without parameters in this report.

This is a text-based application of solving Tower of Hanoi puzzle in a console window by using C++. The objective of Hanoi Tower is to move all the disks to another rod stacked in descending order (largest to smallest from bottom to top). The game was developed applying the knowledge of vectors, error handling methods, Object Oriented Programming (OOP) and other concepts from advanced c programming class.

- Purpose of the report

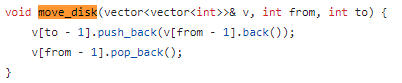
O

# Requirements Fulfillment

Following are the assignment requirements along with its corresponding code implementation

◼ Only one disk can be moved at a time

**move\_disk()** was created to pass v, rods with disks by reference and enable movement of one disk at a time.



◼ Each move consists of taking the upper disk from one of the stack and placing it on top of another stack or on an empty rod. No larger disk may be placed on top of a smaller disk.

The following logic was implemented in **IsMoveAllowed()** to meet the requirements:

1. Checking the highest disk on a rod



The highest disk placed on the rod moving from and to are checked for comparing the size of the disks later on.

1. Checking the existence and size of disks on each rods



To prevent user from moving disks from rod with no disks, the highest disks on the rod moving from is checked to see if it’s not zero (**fromPeakDisk != 0**).

In order to only allow user to move disks on rods with no disks or disks larger than the disk being moved, highest disks on the rod moving to is checked if there’s nothing and the size of the highest disks on rod moving from and to is compared (**toPeakDisk == 0 || fromPeakDisk < toPeakDisk**).

If the user tries to place the disks on the same rod or any other rod out of the range (that is a nonexistent rod), false is returned to indicate that the movement is not allowed.



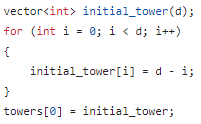
◼ All disks are stacked on the first rod in a decreasing order of sizes

The following was implemented in **initialize\_disks() :**

2D vector, towers is initialized with rows as the number of rods.



1D vector, initial\_tower is initialized with disks stacked in descending order of sizes using for loop and transferred to the first rod on towers vector. This ensures that all disks are stacked on the first rod in decreasing order of sizes.



◼ Indices of rods (starts from 1, not 0)

The following has been implemented in **print\_towers() :**



By adding 1 to indices of rods, the indices of the rods are displayed to start from 1, not 0.

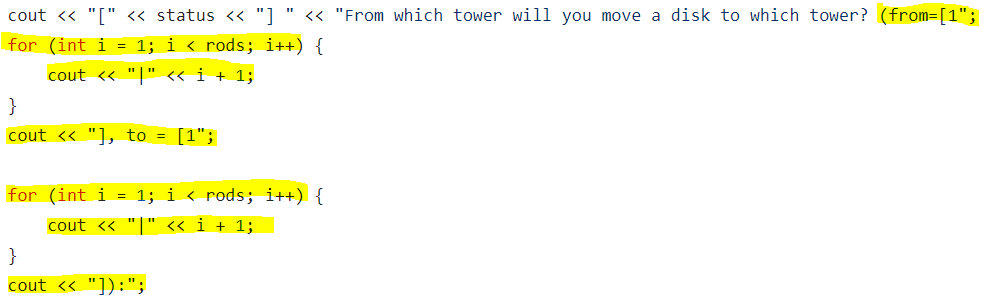
◼ Number of moves (starts from 1 again, not 0)

Initial status in play\_game() is set to 1 to ensure that the number of moves is always 1 at the beginning of the game.



◼ The allowed indices of the rod from which a disk will be moved

Snippet of logic from **show\_move\_prompt()**



Shows the allowed indices by adding indices until

# Flow chart of play\_game()

The general flow chart of hanoi tower has been specified based on the assignment code as follows :

|  |  |
| --- | --- |
| **General flowchart** | **Flowchart based on the assignment code** |
|  |  |

The game begins with **greet\_user()** displaying greeting message to the user and **setup\_game()** asking input for number of rods and disks. Once the number of rods and disks is set by the user via **initialize\_disks()**, its corresponding initial rods and disks is displayed by **print\_towers()**.

Then **ask\_input**() prompts the user to enter the rod they're moving the disk from and to. When movement is allowed, **IsMoveAllowed()** returns true enabling the user's movement to be entered as a parameter of **move\_disk()** for transferring the disk to a specific rod; otherwise, false is returned to display an error message. Once the disks have been moved, **checkGameSolved()** checks if game has been solved. If user hasn’t completed the game, the user enters a loop to repeat the game solving process starting from **ask\_input()** until game is solved.

Once all the disks has been transferred successfully to another rod, the game ends showing number of moves by user and **ask\_replay()** asking for replay. If user decides to replay by entering “y” **play\_game()** is called again to repeat all the steps mentioned above. Otherwise, when “n” is entered, the user breaks out of the loop causing the game to end.

- Explanation of how each requirement was fulfilled

- For each requirement:

- Description

- Implementation approach

- Code snippets (if applicable)

Additional Features

- Description of any additional features implemented

- Explanation of how each feature was implemented

- Code snippets (if applicable)

Code Correctness Demonstration

- Control flow correctness as specified in the flow chart

- Handling of valid and erroneous inputs

- Determination of puzzle solved status

- Screenshots for each example case

Conclusion

- Key challenges tackled

- Limitations addressed

- Future improvements