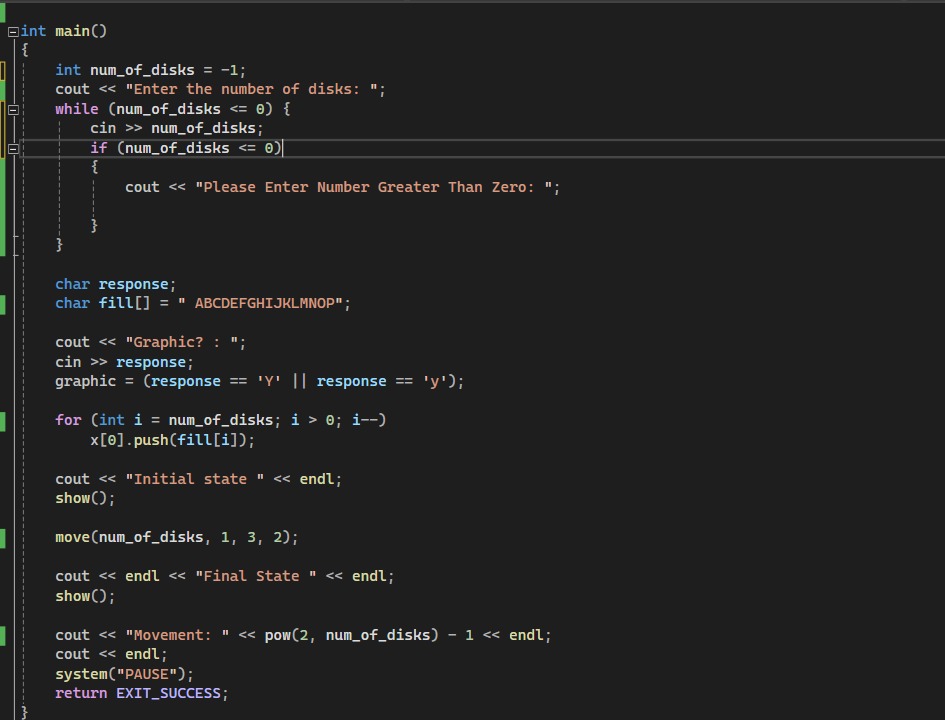
**Project Data Structures**

**Project Name : Tower of Hanoi**



1. User Input for Number of Disks

- int num\_of\_disks = -1;

- Declares an integer variable num\_of\_disks to store the number of disks.

- Initializes it to -1, indicating an invalid value yet.

- cout << "Enter the number of disks: ";

- Prompts the user to enter the number of disks.

- while (num\_of\_disks <= 0)

- Starts a loop that continues as long as num \_of\_ disks less than or equal to zero (non-positive).

- This ensures the user enters a valid number of disks (greater than zero).

- Inside the loop:

- cin >> num\_of\_disks;

- Reads the user's input and stores it in num\_of\_disks.

- if (num\_of\_disks <= 0)

- Checks if the input is invalid.

- If invalid, prompts the user to enter a number greater than zero.

2. Optional Graphic Display (commented out)

- These lines (commented out with //) seem to be for an optional graphic representation of the towers.

- char response;

- char fill [] = " ABCDEFGHIJKLMNOP"; (likely for labeling the disks)

- User is prompted whether they want a graphic display.

- The code snippet to set graphic based on the user's response is commented out.

3. Initializing Disks (commented out)

- These lines (commented out with //) seem to be for initializing the state of the towers.

- for (int i = num\_ of\_ disks; i > 0; i--)

- x [0]. push(fill[i]); (likely pushing disks onto a stack)

- The purpose of x and the details of this initialization are unclear without more context.

4. Solving the Puzzle (assuming a function move exists)

- cout << "Initial state " << endl; (likely for displaying the initial state)

- show (); (This function is likely for displaying the state of the towers, but it's not defined in the provided code)

- move (num\_of\_disks, 1, 3, 2);

- Calls a function move (not shown) that presumably implements the Tower of Hanoi algorithm to solve the puzzle.

- The arguments likely represent:

- num\_of\_disks: Number of disks to move.

- 1: The source rod (starting position).

- 3: The destination rod (final position).

- 2: The auxiliary rod (used for intermediate moves).

- cout << endl << "Final State " << endl; (likely for displaying the final state)

- show (); (Calling the display function again)

\*5. Calculating and Displaying Number of Moves

- cout << "Movement: " << pow (2, num\_of\_disks) - 1 << endl;

- Calculates the total number of moves required using the formula 2^n - 1, where n is the number of disks.

- Displays the number of moves.

\*6. Additional Code (commented out):

- system("PAUSE"); (This is Windows-specific and pauses the console, not generally recommended in C++).

- return EXIT\_SUCCESS; (Returns 0 to indicate successful program execution).

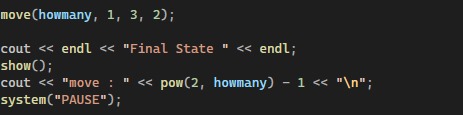
\*\*Overall, this code implements the core logic for solving the Tower of Hanoi puzzle. It takes user input for the number of disks, calls a function (not shown) to solve the puzzle using a recursive algorithm, and potentially displays the initial and final states (if the show function is implemented)

Key improvements:

- Uncomment and implement the missing parts (function definitions, show function) for complete functionality.

- Consider using more descriptive variable names (e.g., sourceRod instead of 1).

- Add comments to explain the code's logic, especially for the move function (not shown).



2. The provided code implements the Tower of Hanoi puzzle using recursion. Here's a breakdown of how it works:

Function Parameters:

count: Represents the number of disks to be moved.

from: The rod where the disks are initially located (numbered 1, 2, or 3).

to: The rod where the disks need to be moved to (numbered 1, 2, or 3).

temp: The temporary rod used as an intermediate storage (numbered 1, 2, or 3).

Base Case:

if (count == 0) return; This checks if there are no disks to move (i.e., count is 0). If so, the function returns without doing anything.

Recursive Steps:

Move n-1 disks from source to temporary rod using destination as intermediate:

Move (count - 1, from, temp, to): This recursive call moves count - 1 disks (smaller subproblem) from the from rod to the temp rod, using the to rod as the intermediate rod. This frees up the from rod to hold the largest disk.

Print the move instruction:

cout << "Move disk " << count << " from rod " << from << " to rod " << to << endl; This line prints a message indicating which disk is being moved from which rod to which rod.

Move the nth disk from source to destination:

x[to - 1].push(x[from - 1].pop()): This line simulates moving the largest disk (represented by count) from the from rod to the rod. It assumes there are stacks (x) representing the rods, where x[i] refers to the stack on rod i + 1. The pop () function removes the top element (largest disk) from the from rod stack, and push () adds it to the rod stack.

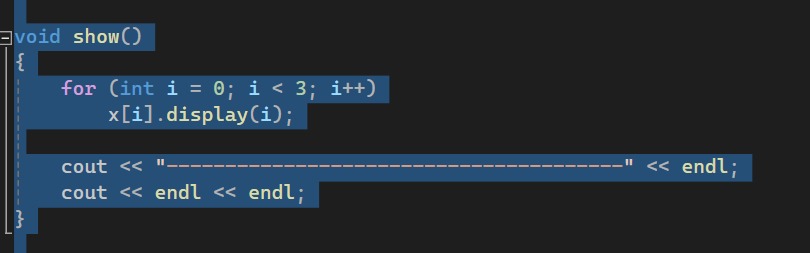
Optional visualization (if graphic is True):

if (graphic) show (); This line conditionally calls a function named show () if a boolean variable graphic is set to True. The show () function (not provided in the code snippet) presumably visualizes the state of the rods during the puzzle solving process.

Move n-1 disks from temporary rod to destination using source as intermediate:

Move (count - 1, temp, to, from): This final recursive call moves the n-1 disks that were placed on the temp rod in step 1, back to the rod (their final destination), using the from rod as the intermediate rod.

Overall Process:



The provided show () function visualizes the state of the rods in the Tower of Hanoi puzzle after each move. Here's how it works:

Iterating Through Rods:

for (int i = 0; i < 3; i++): This loop iterates three times, once for each rod (numbered 0, 1, and 2).

Calling display function for each rod:

x[i]. display(i); Inside the loop, it calls a function named display () on the current rod (x[i]). The display () function (presumably implemented elsewhere) takes the rod number (i) as an argument. This allows the display () function to know which rod it's visualizing and display its contents accordingly.

Separator:

cout << "---------------------------------------" << endl; After iterating through all rods, this line prints a separator line for better readability.

Newlines:

cout << endl << endl; This line prints two newlines, creating some space between the visualization of different moves.

Assumptions:

The code assumes a function called display () exists and is responsible for displaying the contents of each rod. This function likely iterates through the stack (x[i]) and prints the disks (or their sizes) in descending order, representing the order they are stacked on the rod

The code assumes x is an array of stacks (or data structures) that hold the disks for each rod.

Overall Functionality:

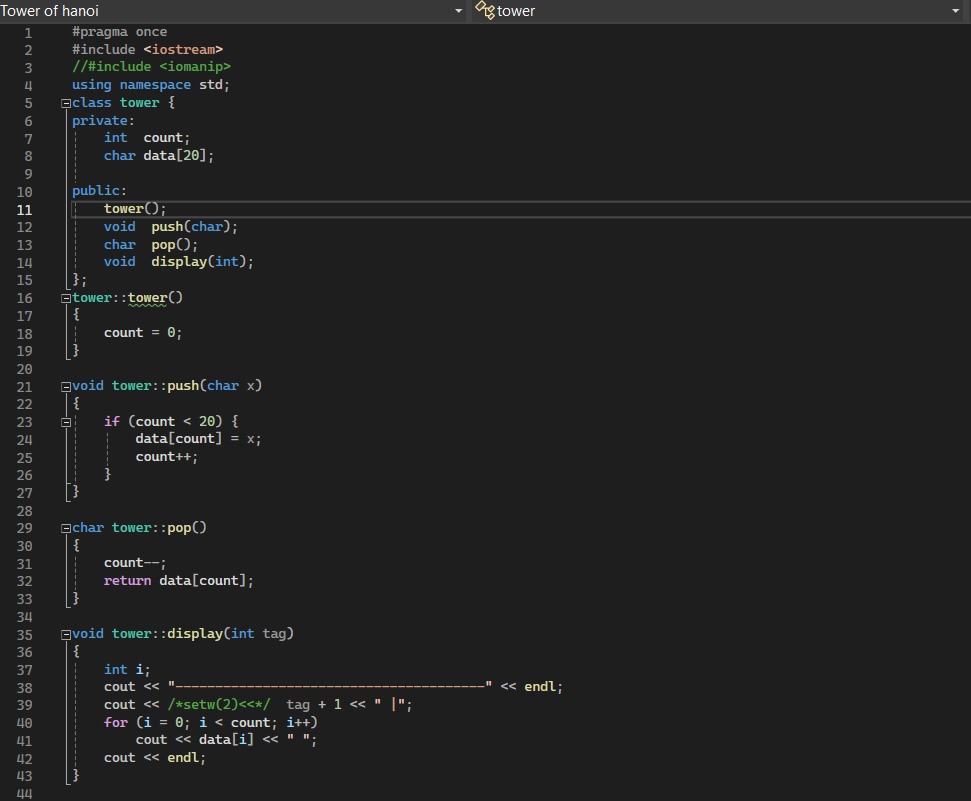
By calling show (), the user can see a textual representation of the rods after each move in the Tower of Hanoi solving process. This helps visualize the movement of disks and verify the solution.

4. `tower:tower()`:

- This is the constructor of the `tower` class.

- It initializes the `count` variable to 0, indicating an empty tower.

**Header file**

****

**- private Members declared here are accessible only within the tower class.**

**- int count: An integer variable to keep track of the number of disks currently in the tower (maximum 20).**

**- char data [20]: An array of characters to store the disks (represented by characters).**

**- public Members declared here are accessible outside the tower class.**

**- tower (): The constructor for the tower class, initializes count to 0.**

**- void push(char): A method to push a disk (represented by a character) onto the top of the tower.**

**- char pop (): A method to remove and return the disk from the top of the tower.**

**- void display(int): A method to display the contents of the tower, along with a tag number.**

**tower Class Member Functions:**

**- tower () Constructor:**

**- Initializes count to 0, indicating an empty tower initially.**

**- \*\*void push (char x):**

**- Checks if the tower is not full (count < 20).**

**- If there's space:**

**- Adds the character x (representing the disk) to the end of the data array (data[count]).**

**- Increments count to reflect the new disk.**

**- \*\*char pop ():**

**- Decrements count to remove the top element from consideration.**

**- Returns the character at the (now decreased) count index, which was the top disk.**

**- \*\*void display (int tag):**

**- Prints a separator line.**

**- Prints the tag number (tag + 1) followed by a pipe (|).**

**- Loops through the data array from 0 to count-1 (up to the last element):**

**- Prints the character representing the disk (data[i]).**

**- Prints a space (` `) for separation.**

**- Prints a newline (endl) after the disks.**

**\*\*Overall, this code defines a reusable tower class for the Tower of Hanoi puzzle. It provides methods for pushing, popping, and displaying the disks on a tower.**

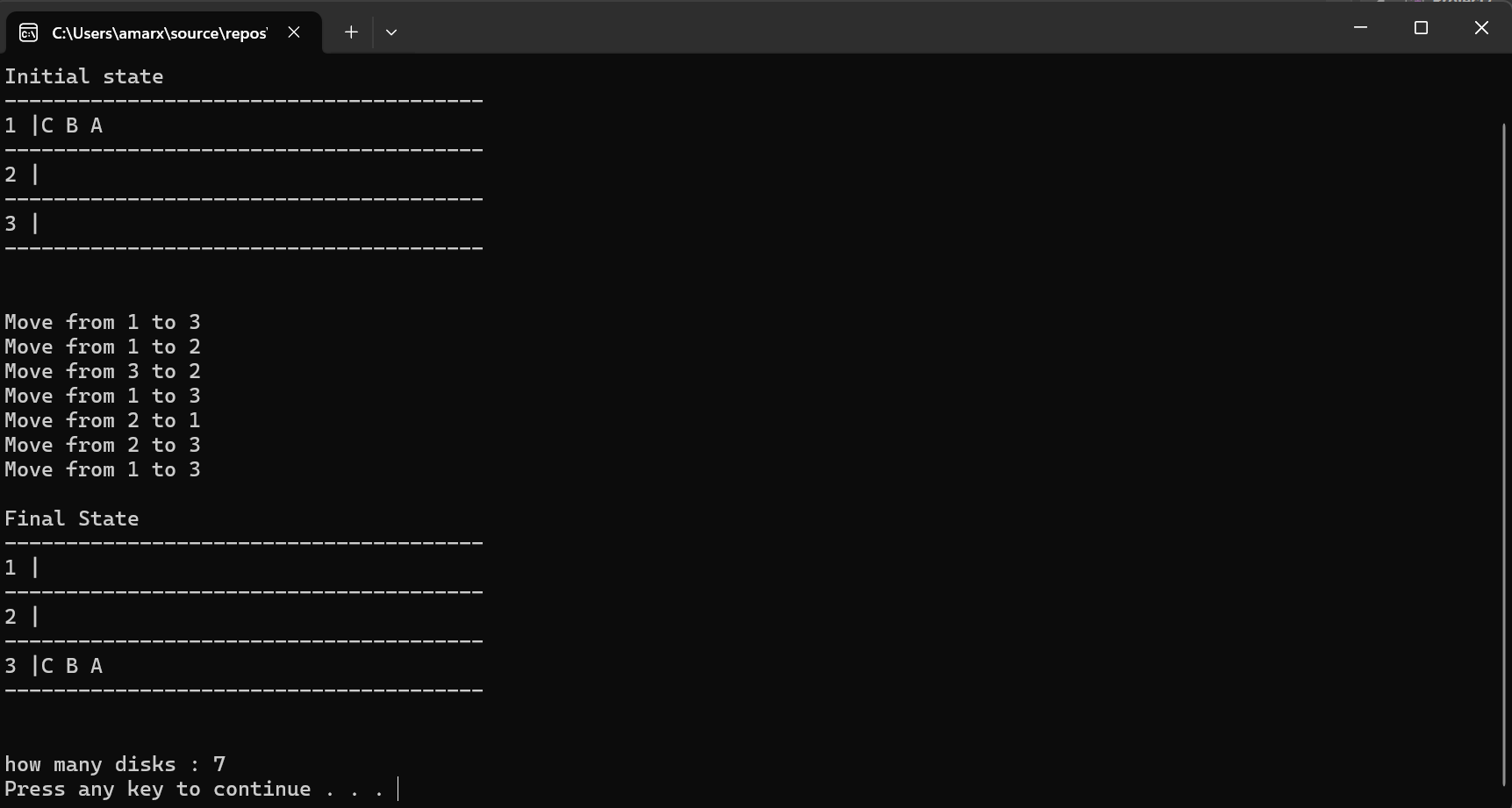
**\*Note: To use this tower class in a complete Tower of Hanoi solution, you'll need additional code that:**

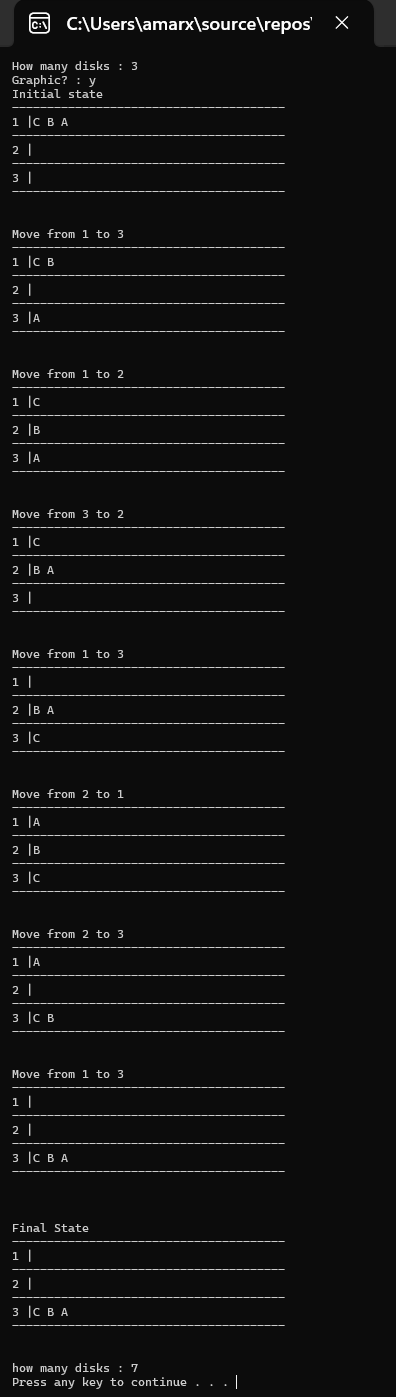
**- Creates three tower objects to represent the source, destination, and auxiliary rods.**

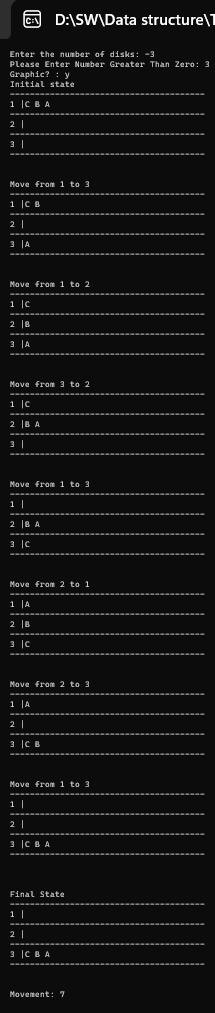
**- Implements the Tower of Hanoi algorithm to move disks between towers recursively.**

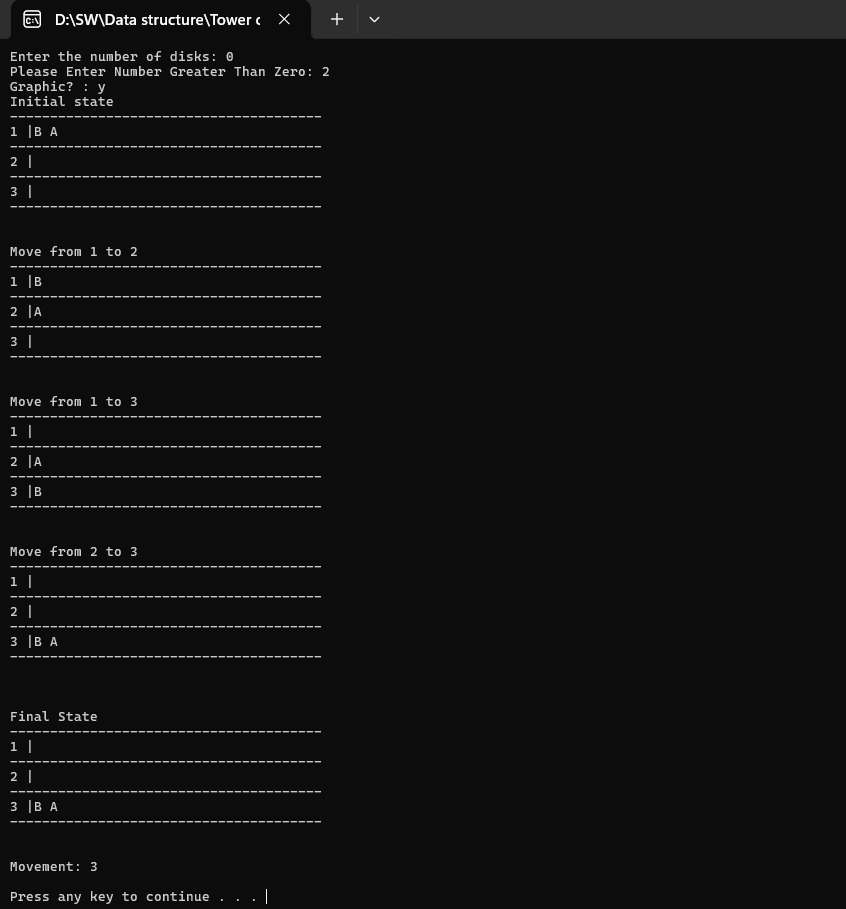
**- Manages user interaction and displays the overall game state.**

**\*Outputs\***

****

****

****

****