**Tasks 1, 2**

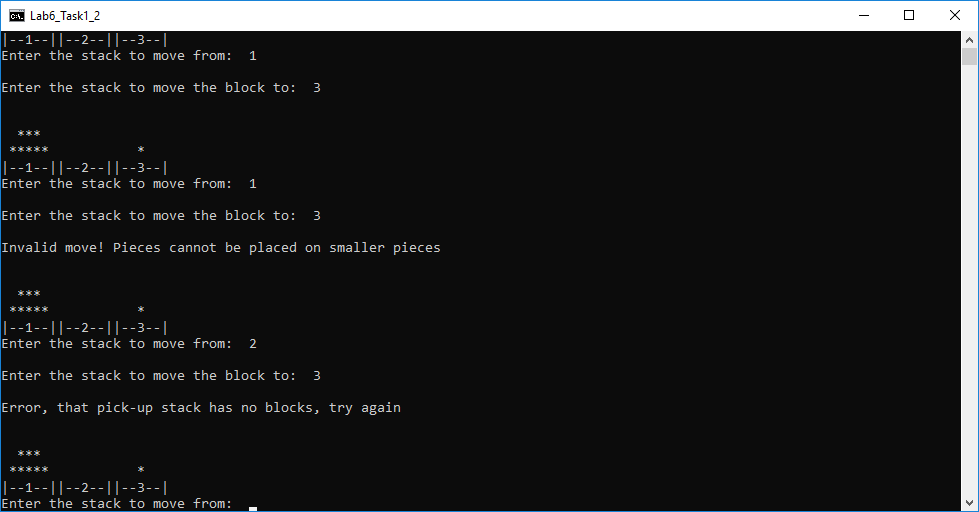
****

Figure 1 - Invalid move; no stacks on the given block

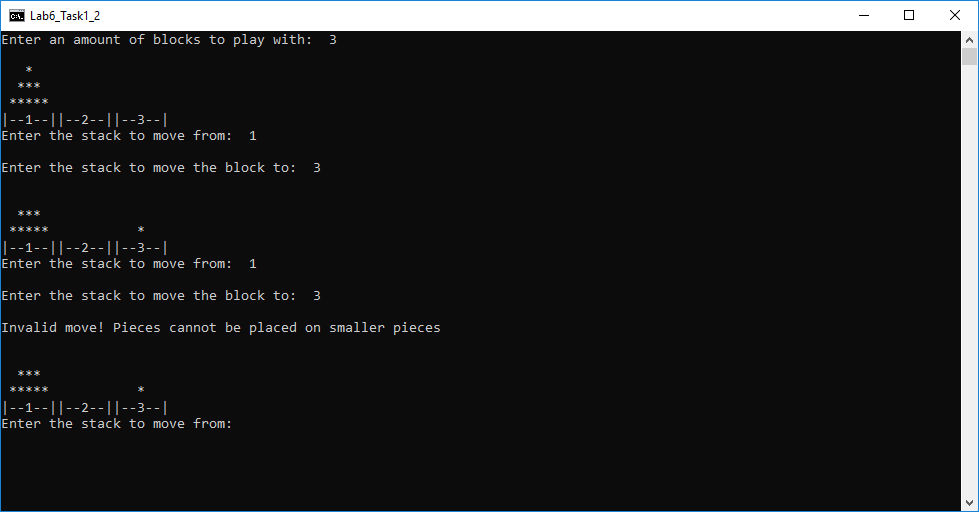


Figure 2 – Invalid move; stacking a bigger block on a smaller one error

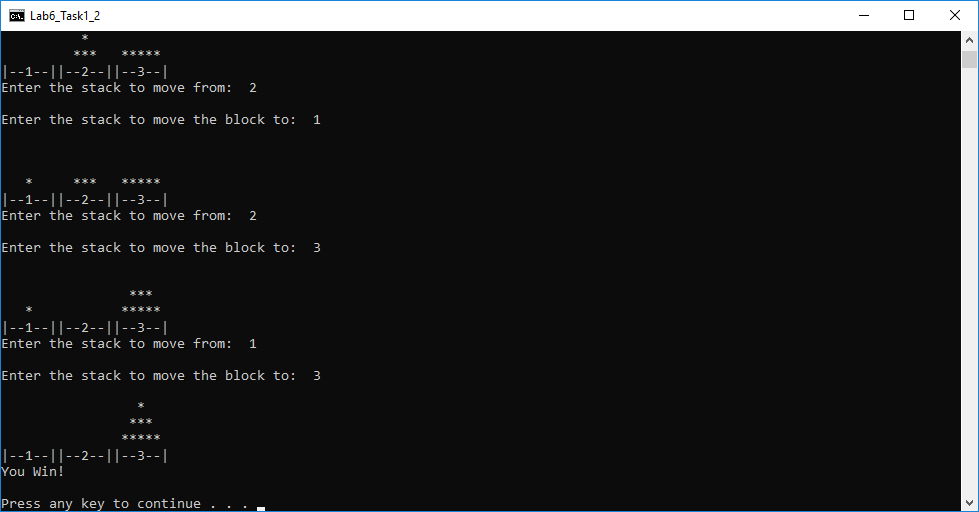


Figure 3 – Win condition is met, and program terminates

**Task 3**,

Figure 4 – Final 4 moves before player one wins

In this game, there are several strategies to win. From doing some testing, it doesn’t seem there is necessarily a best tactic to choose, but rather it is dependent on the opponent’s strategy. If both players play defensively, the game goes extremely slow and eventually comes down to whoever makes a minor mistake first. If one player plays very defensively while the other sets out to check their opponent whenever possible by placing their blocks on top of their opponents, the defensive player usually tends to win. This is likely because to be offensive in this game, you must also sacrifice your own objective to slow down your opponent’s. The best strategy in terms of win percentage seems to be ignoring the opponent. This game is rather simplistic and really comes down to making the least amount of moves possible. By actively trying to interfere with your opponent, either by protecting your own stack side or actively trying to slow down the opponent, you are still slowing yourself down and taking more moves. By just focusing only on the objective and adjusting as the opponent moves, you will tend to win more often.

**Task 4**

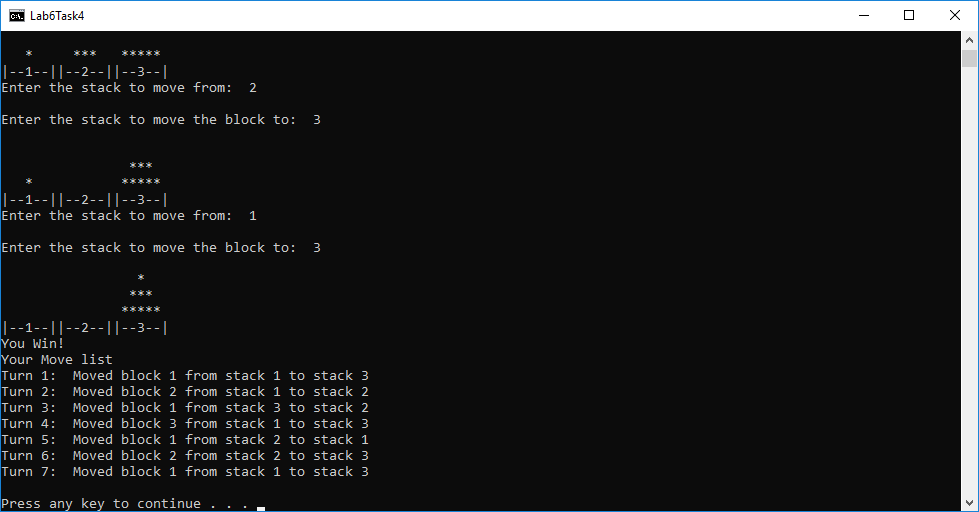


Figure 5 – Move list of a successful win

In task 4 a move data structure was created to store each move made by the player and store it in a queue. When you look at the Tower of Hanoi game, there are three fundamental parts to a move: the stack a block is being moved from, the stack a block is being moved to, and the size of the block being moved. The simplest way to represent this was to store each of the three pieces of data as ints in our move struct. The stack numbers are numbers 1 to 3 and the block number increases with the size of the blocks. This setup works seamlessly with our already existing switch statements to label the stacks from 1 to 3.

**Submission**

Portions of the stack class pertaining to the data structure alone were coded by Zach while the functions needed for the game itself were created by Dustin. Task 3 was implemented by Dustin. Task 4 was implemented by Zach. The work is equitable.

Lab6\_Task1\_2 contains the main for tasks 1 and 2

Lab6Task3 contains the main for task 3

Lab6Task4 contains the main for task 4

All programs should be compiled using the g++ compiler with default Windows settings