Lab 05 - Stack Game

Instructions:

- **The lab requires writing a complete cpp file and cpp file within an hour. It requires completing a few tasks.**
- **Accompanying this file is a template header files that you must modify. You cannot include additional libraries to or remove any libraries from the template files. All other modifications are allowed.**
- **Your submission must be submitted to the Labs directory of your github repository and/or as an attachment on Google classroom under the Lab05 assessment. The files must remain header files.**
- **Cheating of any kind is prohibited and will not be tolerated.**
- **Violating and/or failing to follow any of the rules will result in an automatic zero (0) for the lab.**

TO ACKNOWLEDGE THAT YOU HAVE READ AND UNDERSTOOD THE INSTRUCTIONS ABOVE, AT THE BEGINNING OF YOUR SUBMISSION(S), ADD A COMMENT THAT CONSISTS OF YOUR NAME AND THE DATE

The game Tower of Hanoi is implemented by using a stack. That is, the towers of the game only allow the movement of the top disk, which is the last disk added to the tower. However, you cannot move a disk to a tower unless the tower is empty or the new disk has a smaller circumference than the current top disk of the tower its being moved to. Your objective is to define two classes, one for the tower and the other for the game. To accomplish your objective, complete the following:

☐ in the header file **Tower.h**, define the class *Tower* that contains

- a private int array field named *tower* that has a size of 4.
- a private int field named *size*.
- a public default constructor. It assigns 0 to *size*.
- a public copy constructor.
- a public assignment operator.
- a public empty destructor.
- a public bool method named `Push()` that takes an int parameter. If *tower* is not full, and *tower* is either empty or the parameter is less than the top element of *tower*, the function inserts the parameter to the top of *tower*, increments *size* by 1 and returns true; otherwise, it returns false.
- a public void method named `Pop()` that takes no parameters. If *tower* is not empty, the function decrements *size* by 1; otherwise, it does nothing.
- a public constant int method named `Top` that takes no parameters. If *tower* is not empty, the function returns the top element of *tower*; otherwise, it throws the error message `"empty tower"`.
- a public constant bool method named `IsEmpty()` that takes no parameters. It return true if *tower* is empty [*size* equals 0].
- a public constant bool method named `IsFull()` that takes no parameters. It return true if *tower* is full [*size* equals 4].
- a public void method named `MakeEmpty()` that takes no paramaters. It assigns 0 to *size*.
- a public string constant method named `ToString()` that takes no parameters. It returns a string in the format

  `[`*a*`|`*b*`|`*c*`|`*d*`]`

  where *a*, *b*, *c* and *d* are the values of the elements of *tower* whose indices are 3 through 0 respectively. For the elements whose indices are greater than or equal to *size*, write a space instead of its value.
- a friend ostream operator. Its display is in same format as the return of `ToString()`.

☐ in the header file **Game.h**, define the class *Game* that contains

- a private *Tower* array field named *towers* that has a size of 3.
- a public default constructor. It inserts the numbers 4 through 1 into the first element of *towers*.
- a public copy constructor.
- a public assignment operator.
- a public empty destructor.
- a public bool method named `Move()` that takes two int parameters. If the parameters are valid distinct indices of *towers* and the top of the element of *towers* whose index is equal to the first parameter is less than the top of the element of *towers* whose index is equal to the second parameter, the function performs the move and returns true; otherwise, it returns false.
- a public void method named `Reset()` that takes no parameters. It emptys all the elements of *towers*, and then, inserts the numbers 4 through 1 into the first element of *towers*.
- a public constant bool method named `Success()` that takes no parameters. It return true if the last element of *towers* is full and the other two elements are empty.
- a public string constant method named `ToString()` that takes no parameters. It returns a string in the format

  `1:`*a*
  `2:`*b*
  `3:`*c*

  where *a*, *b*, and *c* are the values of the elements of *towers* whose indices are 0 through 2 respectively.
- a friend ostream operator. Its display is in same format as the return of `ToString()`.