Lab 04 - Queues & Breadth First Search

Instructions:

- **The lab requires writing a complete cpp file and cpp file within an hour. It requires completing 2 tasks.**
- **Accompanying this file is a template cpp file that you must modify. You cannot include additional libraries to or remove any libraries from the template file. 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 Lab04 assessment. The file must remain a cpp file.**
- **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

Along with the **Queue.h** file, you are provided the **Path.h** file that contains the classes *Position* and *Path*.
The class *Position* represents a valid (x, y) coordinate of an 8×8 grid where $0 \leq x \leq 7$ and $0 \leq y \leq 7$. Its methods are

- `GetX()` - it returns the x value.
- `GetY()` - it returns the y value.
- `SetX(`*v*`)` - if *v* is valid, it makes the x value equal to *v*; otherwise, it does nothing.
- `SetY(`*v*`)` - if *v* is valid, it makes the y value equal to *v*; otherwise, it does nothing.
- `MoveN()` - if the x value is greater than 0, it decrements the x value by 1 and returns true; otherwise, it returns false.
- `MoveS()` - if the x value is less than 7, it increments the x value by 1 and returns true; otherwise, it returns false.
- `MoveW()` - if the y value is greater than 0, it decrements the y value by 1 and returns true; otherwise, it returns false.
- `MoveE()` - if the y value is less than 7, it increments the y value by 1 and returns true; otherwise, it returns false.
- `MoveNW()` - if the x value and the y value are both greater than 0, it decrements the x value and the y value by 1 and returns true; otherwise, it returns false.
- `MoveNE()` - if the x value is greater than 0 and the y value is less than 7, it decrements the x value by 1, increments the y value by 1 and returns true; otherwise, it returns false.
- `MoveSW()` - if the x value is less than 7 and the y value is greater than 0, it increments the x value by 1, decrements the y value by 1 and returns true; otherwise, it returns false.
- `MoveSE()` - if the x value and the y value are both less than 7, it increments the x value and the y value by 1 and returns true; otherwise, it returns false.
- `operator==()` - it returns true if the x and y values of the parameters are equal.
- `ToString()` - it returns a string of either the x and y values enclosed in parentheses separated by a comma.
- `operator<<()` - it displays is in the same format as the return of `ToString()`.

And the class *Path* represents an 8×8 char grid with as corresponding 8×8 bool grid. Its methods are

- `GetValue(`*p*`)` - it returns the element of the char grid at position *p*.
- `GetState(`*p*`)` - it returns the element of the bool grid at position *p*.
- `Toggle()` - it switches between returning a string of the char grid or the bool grid for the `ToString()` method. If it returns true, a string of the char grid is returned; otherwise, a string of the bool grid is returned.
- `GeneratePath()` - it populates a random char grid of pluses (`+`) and spaces ( ), and it makes the bool grid all false.
- `ToString()` - it returns a string of either the char grid or the bool grid.
- `operator<<()` - it displays is in the same format as the return of `ToString()`.

Your objective is to write a complete program that uses a breadth first search to determine if a path exists from a given start point to a given end point. A breadth first search allows you search through a graph or tree by using a queue. Its algorithm is as follows:

A1. add start position to the queue.
A2. if the queue is empty, then
    B1. terminate algorithm as a failure.
A3. else,
    B1. remove the front from the queue and make it the current node.
    B2. if the current node is the target, then
        a. terminate algorithm as a success.
    B3. else if the current node has not been visited, then
        a. add the adjacent nodes of the current node to the queue.
    B4. mark current node as visited.
    B5. goto to step A2.

To accomplish your objective, complete the following tasks

☐ define a void function named `GetAdjacencies()` that takes a *Position Queue* reference parameter and a constant *Position* reference parameter respectively. It will inserts all positions that are one diagonal space away from the *position* object.

☐ define a bool function named `HasPath()` that takes a constant *Path* reference parameter and two constant *Position* reference parameters respectively. It determines if a path exists in the *Path* object starting from the first *Position* parameter and ending at the second *Position* parameter. A path exists if there exists an ordered collection of elements from the *Path* object such that the char values of all the elements in the collection are the space character with the possible exceptions of the first and last elements and each sequential element in the collection is one diagonal element away from each other.