Project 8 (C++): Solving the 8-puzzels problem using A* search, as taught in class and in lecture notes. The three A* functions you used in this program are:

g(n) - # of moves from initial state to node n.
h*(n) - the total distance for all tiles to move from node n to the goal state.
f*(n) - g(n) + h*(n)

You are given two pairs of test data: first pair: Start_1 and Goal_1; 2nd pair: Start_2 and Goal_2.

Include in your hard copies:

- cover sheet
- source code
- print outFile1 for the first pair
- print outFile2 for the first pair
- print outFile1 for the second pair
- print outFile2 for the second pair

****

Language: C++
Points: 12 pts
Due Date: Soft copy (*.zip) and hard copies (*.pdf):

-0 (12/12 pts): on time, 5/12/2022 Thursday before midnight
+1 (13/12 pts): early submission, 5/9/2022, Monday before midnight
-1 (11/12 pts): 1 day late, 5/13/2022 Friday before midnight
-2 (10/12 pts): 2 days late, 5/14/2022 Saturday before midnight
(-12/12 pts): non submission, 5/14/2022 Saturday after midnight

*** Name your soft copy and hard copy files using the naming convention as given in the project submission requirement.
*** All on-line submission MUST include Soft copy (*.zip) and hard copy (*.pdf) in **the same email attachments** with correct email subject as stated in the email requirement; otherwise, your submission will be rejected.
****

I. Inputs:
a) inFile1 (use argv [1]) : A file contains 9 numbers, 0 to 8, represents the initial configuration of the 8-puzzel.
b) inFile2 (use argv [2]) : A file contains 9 numbers, 0 to 8, represents the goal configuration of the 8-puzzel.

****

II. Outputs:
a) outFile1: (use argv [3]) : For all intermediate Open list and Close list and expanded child list.
b) outFile2: (use argv [4]): For the display of the sequence of moves from initial state to the goal state.
Make a very nice display from each configuration to next configuration of 8-puzzels.

****

III. Data structure:

****

- AstarNode class // To represent an 8-puzzel node
    - configuration [9] - you can use an integer array of size 9 or a string length of 9.
    - (int) gStar // # moves so far from initial state to current state
    - (int) hStar // the estimated distance from the currentNode to the goal stateNode
    - (int) fStar // is gStar + hStar
    - (AstarNode*) next
    - (AstarNode*) parent //points to its parent node; initially point to null

    methods:
    - constructor (…)
    - printNode (node)
        // print only node's fStar, configuration, and parent's configuration, in one text line.
        For example: if node's fStar is 9, its configuration is 6 3 4 8 7 0 5 2 1
            and its parent's fStar is 11 configuration is 6 3 4 8 7 1 5 2 0
        Then, print <9:: 6 3 4 8 7 0 5 2 1 :: 10:: 6 3 4 8 7 1 5 2 0>

- AStar class
    - (AstarNode) startNode
    - (AstarNode) goalNode
    - (AstarNode*) Open // A sorted linked list with a dummy node.
      // It maintains an ordered list of nodes, w.r.t. the fStar value of nodes.
      // Reuse code from your previous project.
    - (AstarNode*) Close // a linked list stack (unsorted) with a dummy node.
      // It maintains a list of nodes that already been processed
    - (AstarNode*) childList // a linked list Stack for the expend node's children.
    - (int) table[9][9] // The distances for tiles in a position (a total of 9 positions) move
      // from current configuration to goal configuration. You may hard code this table.

    methods: // all methods are on your own.
    - (int) computeGstar (node) // equal to node's parent's gStar + 1 // one move
    - (int) computeHstar (node) // count the total distance/moves of tiles from node to goalNode.
    - (bool) isGoalNode (node) // returns true if node's configuration is identical to goalNode's configuration,
      // return false otherwise.
    - OpenInsert (node) // inserts node into Open w.r.t. node's fStar value. <u>Reuse codes from your previous projects</u>.
    - CloseInsert (node) // insert node into Close w.r.t. node's fStar value. <u>Reuse codes from your previous projects</u>.
    - (AstarNode) remove (OpenList) // removes and returns the front node of Open after dummy.
    - (AstarNode*) findSpot (Close, child) // Searching thru Close list to find a node's configuration is
      // the same as child node's configuration. It returns null if child is not in Close,
      // else returns Spot where the node after Spot has the same configuration as Child node's.
    - CloseDelete (Spot) // delete the node after Spot in Close list.
    - (bool) match (configuration1, configuration2) // check to see if two configurations are identical.
      // if they are identical, returns true, otherwise returns false.
    - (bool) checkAncestors (currentNode) // To avoid cycle; it starts from currentNode, call match () method
      //to see if currentNode's configuration is identical to its parent's, and recursively call
      // upward until reaches the startNode. If it matches with one of currentNode's ancestor, //returns
      true, otherwise return false.
    - (AstarNode*) constructChildList (currentNode) // Constructs a linked list Stack to store the children of
      //currentNode (i.e., all possible moves from currentNode, but NOT one of the currentNode's ancestors.
      // returns the linked list head. <u>**This method must call checkAncestors method!**</u>
    - printList (listHead, outFile1) // call printNode () to print each node in OpenList, including dummy node;
    - printSolution (currentNode, outFile2) // Print the solution to outFile2, make it pretty to look at.

****

IV. main () // The algorithm may contain bugs, debugging is yours

****

```
Step 0:  inFile1, inFile2, outFile1, outFile2 ← open
         initialConfiguration ← get from inFile1
         goalConfiguration ← get from inFile2
         startNode ← Use the constructor to create a AstarNode with initialConfiguration
         goalNode ← Use the constructor to create a AstarNode with goalConfiguration
         Open ← Use the constructor to create a dummy AstarNode
         Close ← Use the constructor to create a dummy AstarNode

Step 1: startNode's gStar ← 0
        startNode's hStar ← computeHstar (StartNode)
        startNode's fStar ← startNode's gStar + startNode's hStar
        OpenInsert (startNode)
Step 2: currentNode ← remove (Open)
Step 3: if (currentNode != null) && isGoalNode (currentNode) // found a solution.
             outFile2 ← "A solution is found!!
             printSolution (currentNode, outFile2)
             exit the program
Step 4: childList ← constructChildList (currentNode)
Step 5: child ← pop (childList)
Step 6: child's gStar ← computeGstar (child)
        child's hStar ← computeHstar (child)
        child's fStar ← child's gStar + child's hStar
        child's parent ← currentNode // back pointer
Step 7:  Spot ← findSpot (Close, child)
Step 8: if Spot's next != null &&  Spot->next->fStar > child's fStar
                CloseDelete (Spot)
Step 9: OpenInsert (child)
Step 10: repeat Step 5 to Step 9 until childList is empty
Step 11: CloseInsert (currentNode)
Step 12: Print "This is Open list:" to outFile1
         printList (Open, outFile1)
         Print "This is CLOSE list:" to outFile1
        printList (Close, outFile1)
        Print up to 20 loops!
Step 13: repeat step 2 to step 12 until currentNode is a goal node or Open is empty.
Step 14: if Open is empty but currentNode is NOT a goal node,
                print error message: "no solution can be found in the search!" to outFile1
Step 15: close all files
```