Project 8 (C++ or Java): Solving the 8-puzzels problem using A* search, as taught in class. However, for easy programming, this project will not use CLOSE list.

*** Please make every effort to do this project on your own and not to consult anyone or get code from others!

The three A* functions uses in this program:

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 node.
f*(n) - g(n) + h*(n)

You are given two pairs of test data: first pair: Start1 and Goal1; 2nd pair: Start2 and Goal2.
First pair was illustrated in class.

Include in your hard copies:

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

****

Language: C++ or Java // of your choice. Please note: the project specs is written for C++. If you choose Java, you will have to modify the syntax from C++ to Java.
===================================

Points: 12 pts
Due Date: Soft copy (*.zip) and hard copies (*.pdf):
+1 (13/12 pts): early submission, 5/9/2023, Tuesday before midnight
-0 (12/12 pts): on time, 5/12/2023, Friday before midnight. NO LATE submission!

*** 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 3 lines of 3 numbers (0 to 8) represents the initial state of the 8-puzzel.
b) inFile2 (use argv [2]): A file contains 3 lines of 3 numbers (0 to 8) represents the goal states of the 8-puzzel.

****

II. Outputs:
a) outFile1: (use argv [3]) : For printing all intermediate of 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 display from each configuration to next configuration of 8-puzzels.
c) deBugFile: (use argv[5]): For all debugging prints.

****

III. Data structure:

****

- AstarNode class // To represent an 8-puzzel node
    - (int) config [9] // You have option of using 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 state.
    - (int) fStar // is gStar + hStar
    - (AstarNode*) next
    - (AstarNode*) parent //points to its parent node; initially point to null

methods:
- constructor (…)
- printNode (node, file) // node is a AstarNode*
  // print only node's fStar, config, and parent's fStar, configto file, in one text line.
  For example: if node's fStar is 8, its config is 6 3 4 8 7 0 5 2 1
  and its parent's fStar is 11 configis 6 3 4 8 7 1 5 2 0
  Then, print <11 [6 3 4 8 7 1 5 2 0] :: 8 [6 3 4 8 7 0 5 2 1] >

- AStar class
  - (AstarNode*) startNode
  - (AstarNode*) Open // A sorted linked list with a dummy AstarNode.
    // nodes in the list are ordered in ascending order w.r.t. fStar value of nodes.
  - (AstarNode*) childList // An un-sorted linked list with a dummy node; for storing the expended node's children.
  - (int) table[9][9] // A position table, as taught in class, for computing the h2* function and
    // for constructing the childList. You may hard code this table.
  - (int) initConfig [9] // to store the configuration of the initial state of 8-puzzle game.
  - (int) goalConfig [9] // to store the configuration of the goal state of 8-puzzle game.
  - (int) dummyConfig [9] // (-1, -1, -1, -1, -1, -1, -1, -1, -1)

  methods:
  - constructor (…)
  - (int) computeHstar (…) // See algorithm below.
  - (AstarNode*) expandChildList (…) // See algorithm below.
  - OpenInsert (node) // inserts node into Open w.r.t. to node's fStar. Reuse codes from your previous projects.
  - (AstarNode*) remove (list) // removes and returns the front node (after dummy) of a given list;
    // list can be OPEN or childList. You should know how to write this method.
  - (bool) match (config1, config2) // check to see if two configurations are identical; if they are identical, returns
    // true, otherwise returns false. You should know how to write this method
  - (bool) checkAncestors (child) // To avoid cycle. During the constructing the children of currentNode,
    // before inserting a child of currentNode into the childList, we need to check (upward to the
    // startNode) to see if the child has configuration as one of its ancestors, if yes, returns true,
    //otherwise, returns false. This method should call match(…) method to compare the two
    //configurations. You should know how to write this method.
  - printList (list, file) // call printNode () to print each node in the given list, to file, including dummy node.
  - printSolution (…) // Print the solution path to outFile2.
    // Print the node's configuration in 3 by 3 configuration instead of 1 by 9 configuration.

****

IV. main (…)

****

```
Step 0:  inFile1, inFile2, outFile1, outFile2, deBugFile ← open
         initConfig ← load from inFile1
         goalConfig ← load from inFile2
         startNode ← get an AstarNode with (initConfig, 0, 9999, 9999, null, null)
         Open ← get a dummy AstarNode (dummyConfig, 0, 0, 0, null, null) for Open to point to.
         Close ← get a dummy AstarNode (dummyConfig, 0, 0, 0, null, null) for Close to point to.
        childList ← get a dummy AstarNode (dummyConfig, 0, 0, 0, null, null) for childList to point to.

Step 1: startNode's gStar ← 0
        startNode's hStar  ← computeHstar (StartNode's configuration, goalConfig, deBugFile)
        startNode's fStar  ← startNode's gStar + startNode's hStar
        OpenInsert (startNode)
Step 2: currentNode ← remove (Open)
       deBugFile ← "this is currentNode"
        printNode (currentNode, deBugFile)
Step 3: if (currentNode != null) && match (currentNode's configuration, goalConfig) // found a solution.
                outFile2 ← "A solution is found!!
                printSolution (currentNode, outFile2)
                exit the program
Step 4:  childList ← expandChildList (currentNode, deBugFile)
Step 5: child ← remove (childList)
          deBugFile ← “In main(), remove node from childList, and printing”
          printNode (child, deBugFile)
Step 6: child's gStar ← currentNode’s gStar +1
          child's hStar ← computeHstar (child’s configuration, goalConfig, deBugFile)
          child's fStar ← child's gStar + child's hStar
          child's parent ← currentNode // back pointer
Step 7: OpenInsert (child)
Step 8: repeat Step 5 to Step 9 until childList is empty
Step 9: outFile1 ← "Below is Open list:"
           printList (Open, outFile1)
          Print up to 30 loops!
Step 10: repeat step 2 to step 9 until currentNode is a goal node or Open is empty.
Step 11: if Open is empty but currentNode is NOT a goal node,
                outFile1 ← “*** Message: no solution can be found in the search!"
Step 12: close all files
*******************************
V. (int) computeHstar (nodeConfig, goalConfig, deBugFile)
*******************************
Step 0: deBugFile ← “Entering computeHstar method”
          sum ← 0
          i ← 0
Step 1: p1 ← nodeConfig[i]
Step 2: j ← 0
Step 3: if goalConfig[j] == p1
              sum += table[i][j]
              break
Step 4: j++
Step 5: repeat Step 3 – Step 4 while j < 9
Step 6: i++
Step 7: repeat Step 1 to Step 6 while i < 9
Step 8: deBugFile ← “Leaving computeHstar method:  sum = ” // fill in value
Step 9: return sum

*******************************
VI. (AstarNode*) expandChildList (currentNode, deBugFile)
******************************
Step 0: deBugFile ← “Entering expandChildList method”
          deBugFile ← “Printing currentNode”
          printNode (currentNode, deBugFile)
          (AstarNode*) tmpList ← get a dummy AstarNode (dummyConfig, 0, 0, 0, null, null)
Step 1:  i ← 0
Step 2: if currentNode.config [i] != 0
              i++
Step 3: repeat Step 2 while currentNode.config [i] != 0 and i < 9
Step 4: if i >= 9
                 deBugFile ← “Something is wrong, currentNode does not have a zero in it”
                 return
          else
                zeroPosition ← i
                deBugFile ← “find the zero position in currentNode at position i =” // fill in value
```

```
Step 5: j ← 0  // looking for the neighbor of zeroPostion
Step 6: if table[zeroPosition][j] == 1  // found a position with 1 distance from the zeroPosition
                    (AstarNode*) newNode ← get a AstarNode with (currentNode.config, 999, 999, 999, null, currentNode)
                    newNode.config [j] ← 0
                    newNode.config [zeroPosition] ← currentNode.config[j]
                    if (checkAncestors (newNode) == false) // is not one of currentNode's ancestors.
                              newNode.next ← tmpList.next
                              tmpList.next ← newNode
Step 7: j++
Step 8: repeat Step 6 to Step 7 while j < 9
Step 9: deBugFile ← "Leaving expandChildList method and printing tmpList"
          printList (tmpList, deBugFile)
Step 10: return tmpList
```