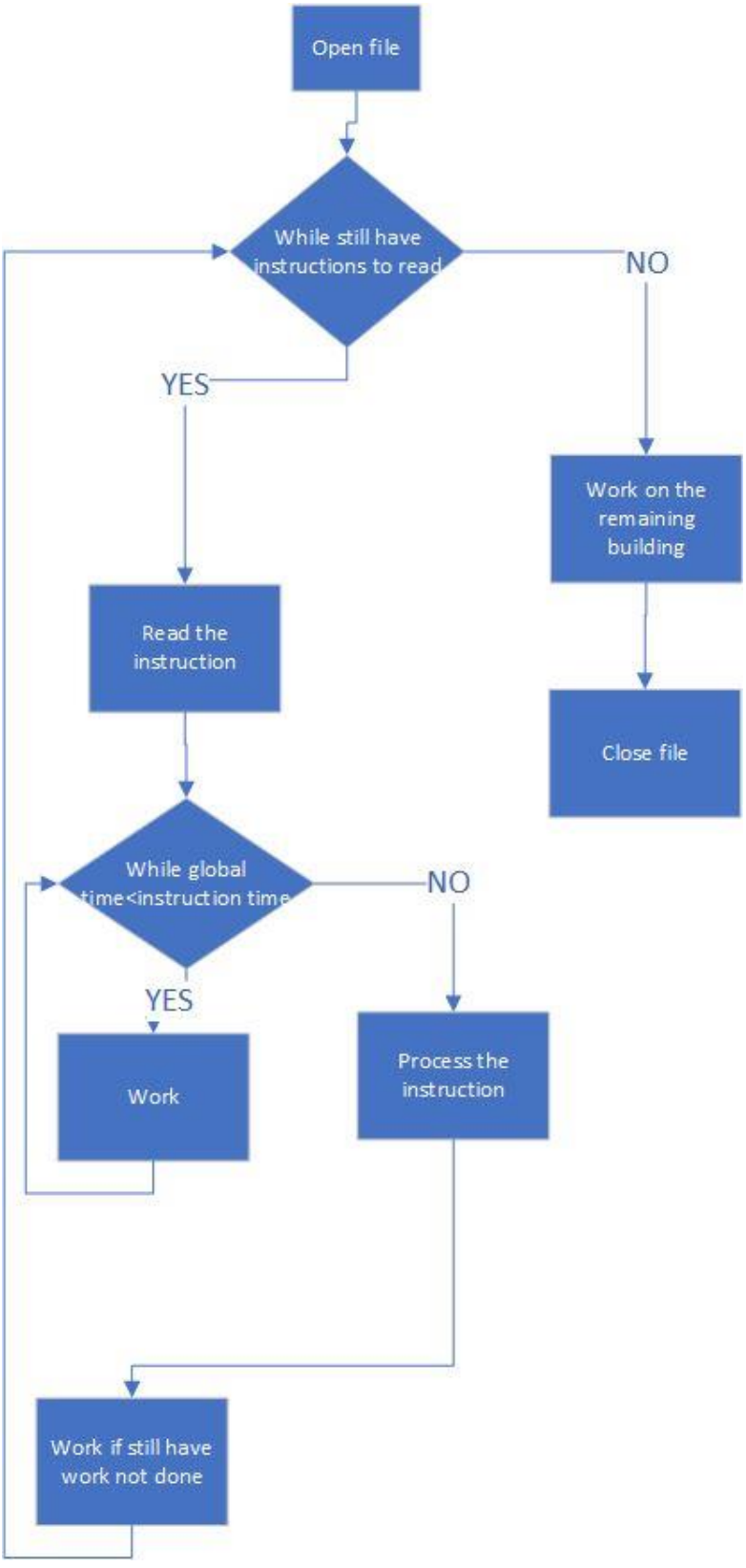
# ADS PROJECT REPORT

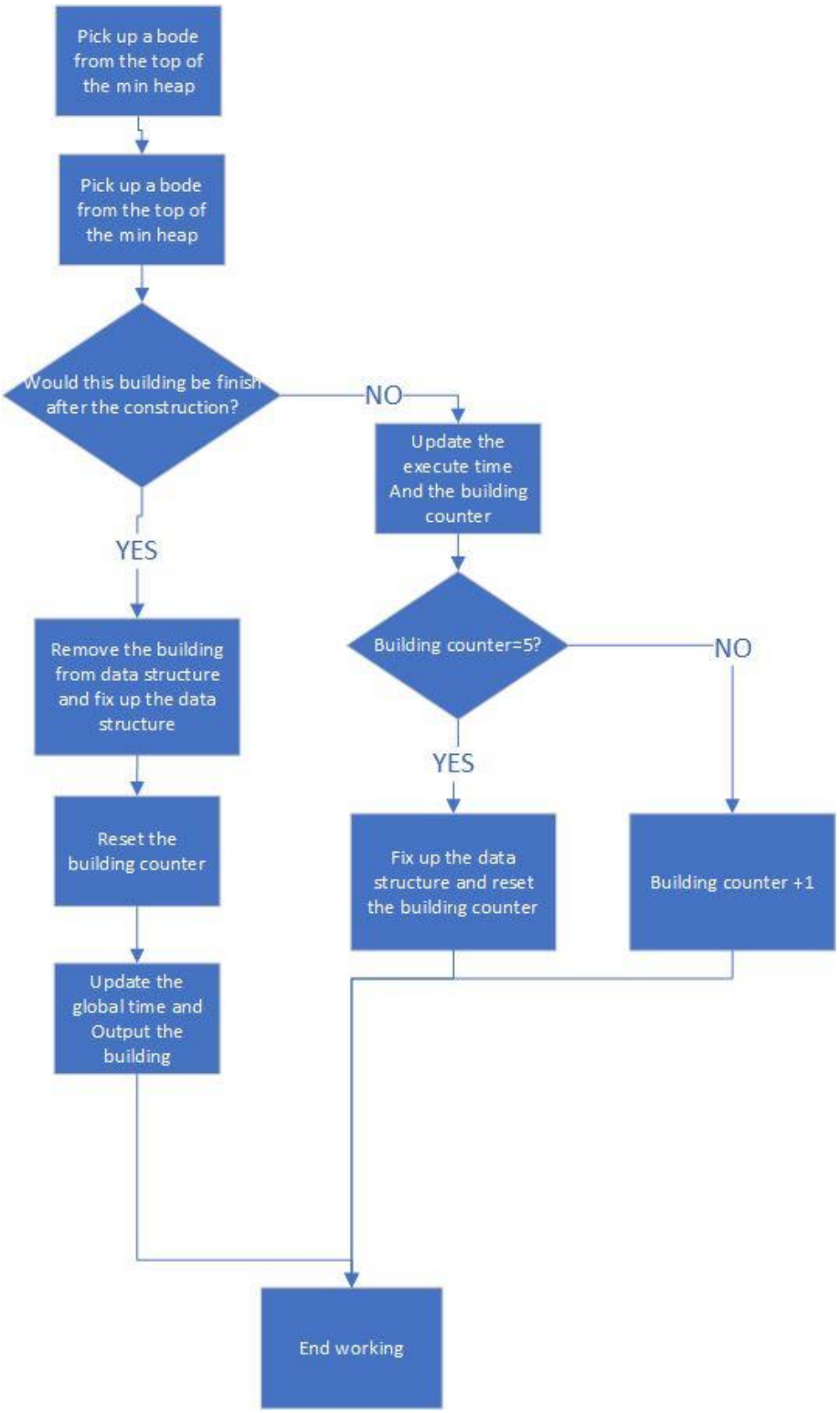
**Name: Yubo Wang**
**UFID: 5334-5862**
**UF Email: yubo.wang@ufl.edu**

## Code Structure:

### Structure of the main function:



## Structure of function ‘work’:



## Other tips:

Input format: java risingCity file_name (file_name should contain ".txt")

**From the structure of the code, we can get:**

1. If the print instruction happens at the same time with the output when building finished, remove the building first and then process the print instruction.

2. The print and the insert instruction themselves do not spend times, they could be processed at the same time with the construction work.

3. When doing insert, the code first insert the new node as a leaf node of the min heap, then still work on the root node if the root node was not work on for 5 consecutive days, after it done the 5 consecutive working or it finished, then do the heapify or red black tree fixup.

4. When updating the data structure, the code first delete the updated root node, then do the heapify or red black tree fixup, and then reinsert the previous node and then do the heapify or red black tree fixup again.

5. The finish time of the city is just the finish time of the last building of the city, it would be printed as the last output.

## Function Prototypes:

**public class risingCity:**

**static function1 f1 = new function1():** function related to the min heap

**static function_rbt frbt = new function_rbt():** function related to the red black tree

**public static int counter:** global time counter

**public static int count:** building time counter, make sure building is worked on until complete or for 5 days

**static building_record_rbt Root = null:** root of the red black tree

**public static int[] work(buliding_record[] br, int[] t):** constructioning

**static void addet(buliding_record[] br):** used for updating the executed_time and the datastructure after printing.

**static void output(int[] ans, BufferedWriter out) throws IOException:** write output buffer

**static void process(String line, buliding_record[] br, BufferedWriter out) throws IOException:** procress one instruction readed from file

**public static void main(String args[]):** main function

**public class function1:**

**int returnparent(int l):** return the parent of a node

**int rlength(buliding_record[] b):** return the length of the min heap

**buliding_record[] move(buliding_record[] a, int l):** check and move the last leaf node bottom-up until it's executed_time> it's parent node's executed_time

**int check(buliding_record[] a, buliding_record b):** to check whether the building number already exist

**public buliding_record[] insertarray(buliding_record[] a, buliding_record b):** insert a new node into the min heap

**public void movetd(buliding_record[] br, int s):** check and move the last leaf node bottom-up until it's executed_time> it's parent node's executed_time

**public buliding_record[] del_top(buliding_record[] br):** delete the root node of the min heap, and then do the heapify if the min heap contains more than 1 node.

**public class function_rbt:**

**public building_record_rbt leftRotate(building_record_rbt x, building_record_rbt Root):** do the left rotate on a particular node

**private building_record_rbt rightRotate(building_record_rbt y, building_record_rbt Root) :** do the right rotate on a particular node

**public building_record_rbt insert(building_record_rbt node, building_record_rbt Root):** insert a node and use **insertFixUp** to make it a red black tree

**public int[] lookup(int bn, building_record_rbt Root):** look up a particular node in the red black tree

**public void find_section(int bn_start, int bn_end, building_record_rbt br, List<building_record_rbt> result):** find the section of print recursively

**public building_record_rbt search(int bn, building_record_rbt Root):** search a particular node in the red black tree

**private building_record_rbt insertFixUp(building_record_rbt node, building_record_rbt Root):** fix the node after updating, make the tree a red black tree

**public building_record_rbt remove(building_record_rbt node, building_record_rbt Root):** remove a node from a red black tree and return the removed node, use **removeFixUp** to make it a red black tree

**private building_record_rbt removeFixUp(building_record_rbt node, building_record_rbt parent, building_record_rbt Root):** fix the node after removing, make the tree a red black tree

**class buliding_record:**

**int buildingNum:** stores the building number

**int executed_time:** stores the executed time

**int total_time:** stores the total_time

**building_record_rbt rbt:** point to the same node in the red black tree

**class building_record_rbt:**

**int buildingNum:** stores the building number

**int executed_time:** stores the executed time

**int total_time:** stores the total_time

**boolean colorisred:** store the color of the node, value is true if node is red

**building_record_rbt left:** point to the left child node

**building_record_rbt right:** point to the right child node

**building_record_rbt parent:** point to the parent node

**buliding_record array:** point to the same node in the min heap tree