General Information:

- Rectangles are defined with a name, centroid coordinates, width [distance from leftmost and rightmost points], and height [distance from bottommost and upmost points]
- When a rectangle is created, it will be placed in the binary tree dictionary of the project [from the data structure code we provided after part 1, this is in the Rectangle struct itself)
    - The dictionary itself is a binary tree that stores the rectangles based on name.
- When adding rectangles to the x or y axis tries:
    - If the rectangle is touching/intersecting one of the axes add them into the respective axis trie
    - If the rectangle is touching/intersecting both, add it to the Y axis trie (wouldn't want to store the same data twice).
- Smallest size a rectangle could have would be a 2x2 (width = height = 2).
- We are only using int values for coordinates, width, and height (implementing with floats is not fun).
- A Rectangle is closed on the left and bottom sides [INCLUSIVE] and closed on the right and top sides [EXCLUSIVE].

Part 3 Operations

1) INIT_QUADTREE(WIDTH):

Initialize a quadtree with the dimensions of 2^{width} x 2^{width}. If there is already an existing quadtree, then delete it and initialize the new quadtree.

- Output: "initialized a quadtree of width 2^{width}" (where 2^{width} would be the actual numeric value)
- Failure Conditions: NONE

2) DIPLAY():

Generate a display of a 2^WIDTH ×2^WIDTH square from the MX-CIF quadtree. To draw the MX-CIF quadtree, you are to use the drawing routines provide. Use the utilities SHOWQUAD and PRINTQUAD. A dashed (broken) line should be used to draw quadrant lines, but the rectangles should be solid. Rectangle names should be output somewhere near the rectangle or within the rectangle. Along with the name of a rectangle R.

- Output: PNG File (or window display) of the drawing for the MXCIF quadtree and print statements from printquad.
- Failure Conditions: NONE

3) TRACE(ON/OFF):

When initiated, output the nodes of the quadtree (both leaf and non-leaf) visited by functions. The root of the quadtree is assigned the number 0. Given a node with number N, its NW, NE, SW ,and SE children are labeled 4·N +1, 4·N +2, 4·N +3, and 4·N +4, respectively. For example, starting at the root, the NE child is numbered 2, while the SE child of the NW child of the root is numbered 4*(4*0+1)+4=8.

- Output: There is no direct output for the call of Trace, so TRACE ON returns nothing initially. Then for each operation that involves traversal through the MX-CIF Quadtree will output an additional list of nodes that were traversed during the Search_Point and Rectangle_Search functions. (More functions from Part 4 will have lists outputted as well)
- Failure Conditions: NONE

NOTE: This returns the nodes that were traversed, so if the child of a node is empty (null) we do not count that as traversing it (unless the MXCIF quadtree is empty, which we count the root as being traversed).

NOTE: I will not be asking for tracing the traversal of the x or y axis tries for part 3. I will ask for this in part 4.

4) LIST_RECTANGLES():

List the names of all the rectangles in the dictionary in lexicographical order, with their coordinate values, width, and height alongside them. For the lexicographical order, this means that letters come before digits in the collating sequence. Similarly, shorter identifiers precede longer ones. For example, a sorted list is A, AB, A3D, 3DA, 5.

- Output: "listing k rectangles:" (where k is the total number of rectangles in the dictionary)

  $Name_1$ CX_1 CY_1 W_1 L_1

  ….

  $Name_K$ CX_K CY_K W_k L_k

- Failure Conditions: NONE

5) CREATE_RECTANGLE(name, XC, YC, W, H):

Create a rectangle with the following qualities:

- string Name
- int Centroid Coordinates (XC, YC)
- int Width (W)
- int Height (H)

The width and the height refer to the entire value for the shape (border to border). Place the created rectangle into the rectangle dictionary. Rectangles will be closed (inclusive) on its left and bottom sides, and open (exclusive) on its right and top sides.

- Output: "created rectangle 'name'"
- Failure Conditions: NONE
  NOTE: We will not be testing Create_Rectangle calling the same name twice.
  NOTE: This places the rectangle into the dictionary, not the MXCIF Quadtree.

6) INSERT_RECTANGLE(name):

Try to insert the rectangle with the name “name” into the MX-CIF Quadtree. Prior to inserting the rectangle, call the RECTANGLE_SEARCH function to see if the rectangle would intersect any other rectangle already in the MXCIF quadtree and make sure that the rectangle does not go out of bounds.

- Output: “inserted rectangle ‘name’”
- Failure Conditions:
    - If the rectangle intersects another rectangle: “failed: intersects with …”
    - If the rectangle is out of bounds: “failed: out of bounds”

NOTE: We will only call insert on rectangles that exist in the dictionary

7) SEARCH_POINT(PX, PY):

Return the rectangle that contains the point (so inside it, on its bottom and/or left side).

- Output: “found rectangle name” (where name is the name of the rectangle)
- Failure Conditions: In the instance that no rectangles store the point return:
  “no rectangle found”

NOTE: This function will return the nodes traversed if TRACE is set to ON.

8) RECTANGLE_SEARCH(name):

Return the names of the rectangles in the MXCIF Quadtree that would intersect with the Rectangle “name”.

- Output:
    - If the rectangle intersects multiple - “’name’ intersects rectangles: ‘name1’ …. ‘nameN’”
    - If the rectangle only intersects one - “’name’ intersects rectangle: ‘name1’”
- Failure Conditions:
    - If the rectangle does not intersect any rectangles – “’name’ does not intersect an existing rectangle”

NOTE: This function will return the nodes traversed if TRACE is set to ON.

NOTE: This function should return a Boolean that’s based on the rectangle intersecting others in the MXCIF. This is because you should call this method prior to inserting a rectangle into the MXCIF.

NOTE: If the edges and/or end points of rectangles touch, it does not count as intersecting (must be cutting through the rectangles).

NOTE: This only compares the rectangles that currently exist in the MXCIF quadtree (ones that are just in the dictionary do not count and ones that have been deleted do not count either [unless reinserted after deletion]).

9) DELETE_RECTANGLE(name):

Delete from the MXCIF Quadtree, the rectangle with the name value “name”.

- Output: “deleted rectangle ‘name’”
- Failure Conditions:
    - If the rectangle is not in the MXCIF Quadtree: “rectangle ‘name’ does not exist”

NOTE: Only deletes from MXCIF Quadtree, not from the dictionary

10) DELETE_POINT(PX, PY):

Delete from the MXCIF Quadtree, the rectangle that contains the point defined by the coordinates PX,PY.

- Output: “deleted rectangle ‘name’”
- Failure Conditions:
    - If there are no rectangles that contain the point: “no rectangle deleted”

NOTE: Won’t ask for coordinates outside the bounds of the MXCIF Quadtree

11) MOVE(name, CX, CY):

Move the center of the rectangle with the name “name” by CX units horizontally and CY units vertically.

- Output: “rectangle name moved successfully”
- Failure Conditions:
    - If you attempt to move the rectangle outside of bounds
        - “Failure: Rectangle ‘name’ moved out of bounds”
    - If you attempt to move the rectangle so that it intersects another rectangle
        - “Failure: Rectangle ‘name’ moved into another rectangle”
    - if the rectangle does not exist in the MXCIF Quadtree.
        - “Failure: Rectangle ‘name’ not found”

NOTE: If the move would result in a failure, keep the rectangle in its original place.