University of Waterloo
CS240 Fall 2021
Programming Question 3

Due Date: Wednesday, Nov 24 at 5:00pm

The integrity of the grade you receive in this course is very important to you and the University of Waterloo. As part of every assessment in this course you must read and sign an Academic Integrity Declaration before you start working on the assessment and submit it **before the deadline of Nov 10th** along with your program; i.e. **read, sign and submit PQ03-AID.txt now or as soon as possible**. The agreement will indicate what you must do to ensure the integrity of your grade. If you are having difficulties with the assignment, course staff are there to help (provided it isn't last minute).

The Academic Integrity Declaration must be signed and submitted on time or the assessment will not be marked.

Please read `http://www.student.cs.uwaterloo.ca/~cs240/f21/guidelines.pdf` for guidelines on submission. Submit the file `kd.cpp` to MarkUs.

Problem 1 [10 marks]

In this programming question you are asked to implement an efficient algorithm to construct a *kd*-tree for 2-dimensions. Use the algorithm on Slide 17 of Module 8. You may assume points are in standard position; i.e. no two points lie on the same horizontal or vertical line. Also, make sure to follow the course convention where points on split lines belong to right/top side. For simplicity, all x and y coordinates will be integer values. For choosing the point on which to partition, you may use any of the *quick-select* implementations discussed in class.
You may not use any pre-existing code that would trivialize the implementation (e.g. built in data structures from STL, etc). You may, however, use smart pointers and the vector data structure.

Implement your program in C++ and provide a main function that accepts the following commands from stdin (you may assume that all inputs are valid):

- `r` - initializes an empty kd-tree. If a nonempty kd-tree already exists, it is destroyed and new empty data structures are created.
- `x` - properly terminates your program; i.e. no memory leaks, etc.

- `ikd n points` - creates a kd-tree for the given set of *points* where *points* is a list of $2n$ integers, separated by whitespace, representing x and y coordinates for n 2-dimensional points. The points list is ordered as follows: $x_0\ y_0\ x_1\ y_1 \ldots x_{n-1}\ y_{n-1}$.
- `pkd` - prints all nodes of the current kd-tree in the order they are visited during a pre-order traversal. Each node should be printed as a pair of integer values, representing the x and y coordinate of the point, followed by a newline; i.e. print a single point per line.
- `skd xmin xmax ymin ymax` - performs a range search query on the kd-tree for the given query rectangle defined by $[xmin, xmax)$ by $[ymin, ymax)$. Only the points found inside the query rectangle should be printed, in the order they are visited as leaf nodes during a pre-order traversal of the kd-tree. Again print each point as a pair of integer values, representing the x and y coordinate of the point, followed by a newline.

Notes:

- For simplicity, the query rectangle will be defined with non-negative integer values.
- The query rectangle will not be degenerate - the x and y ranges will not have 0 length.
- Warning: we do not have a strict limit on the number of points, so if you use `getline` to read the input, the standard buffer may not be large enough.
- Remember the course convention that a point on a line belongs to the region to the right/top.

Place your entire program in the file `kd.cpp`

Example: The following set of operations:

```
r
ikd 3 3 4 2 2 1 1
pkd
skd 1 2 1 2
x
```

would create a kd-tree on the 3 points p_1, p_2, p_3 = (3,4),(2,2),(1,1). These three points correspond to the following *kd*-tree:

```
                                       |
                                       |-----* p1------------
                                       |
                                       |
                                       |
         [x < p2.x?]                   |
        /    \                         |
     [p3]    [y < p1.y?]               |
            /    \                     * p2
          [p2]  [p1]                   |
                                       |
                                  *    |
                                  p3   |
```

and print out the following to stdout:

```
2 2
1 1
3 4
2 2
3 4
1 1
```