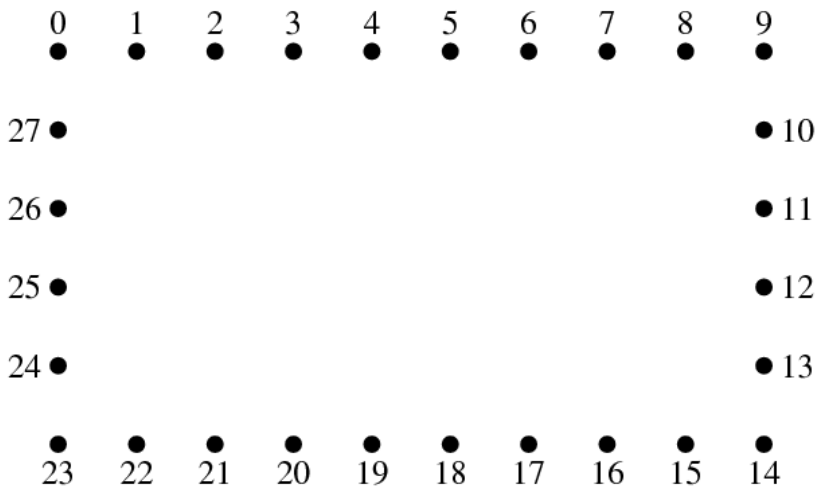
Assignment 3

There are TWO questions included in Assignment 3. Please find the zip file of the QT project for Assignment 3 in the BB system. **Please refer to the test program in "main.cpp" for each question to implement your programs and test them accordingly**.

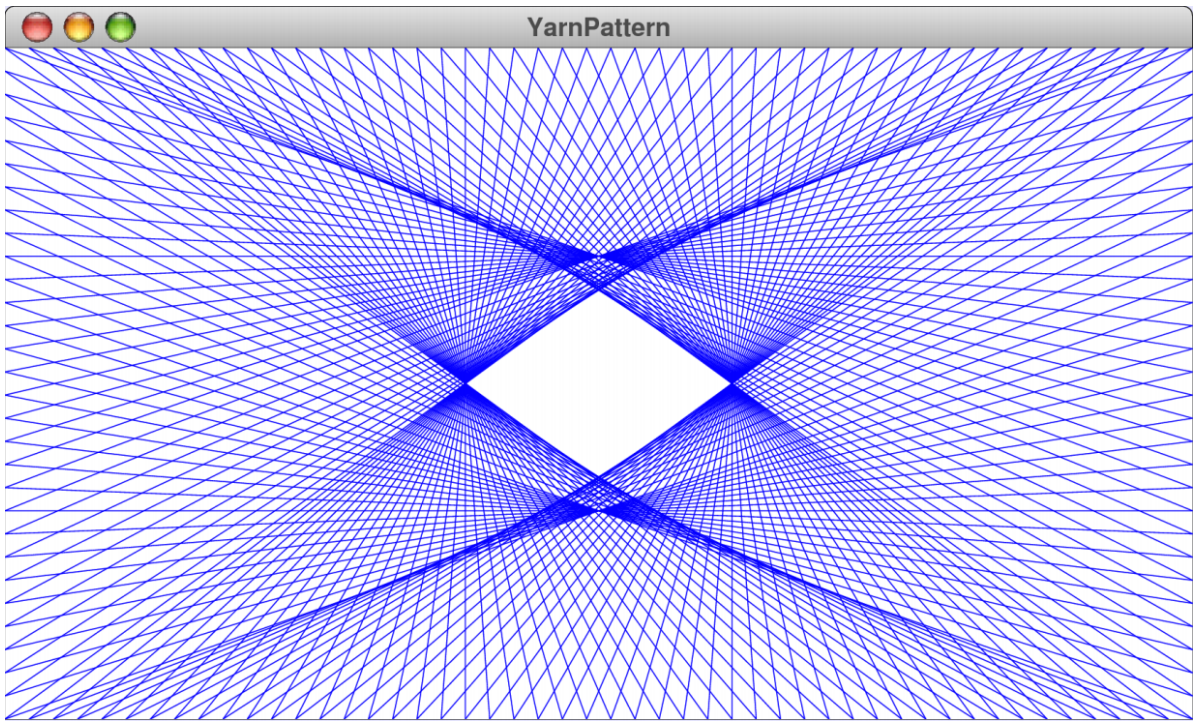
Question 1: Yarn Pattern

The classes exported by the **gtypes.h** interface described in the preceding exercise make it simpler to create intricate graphical patterns, in part because they make it easy to store coordinate information inside collection classes and other abstract data types. In this exercise, for example, you get to have some fun with a vector of **GPoint** objects. Imagine that you start with a rectangular board and then arrange pegs around the edges so that they are evenly spaced along all four edges, with **N_ACROSS** pegs along the top and bottom and **N_DOWN** pegs along the left and right edges. To model this process using the graphics window, what you want to do is create a **Vector<GPoint>** that holds the coordinates of each of these pegs, which are inserted into the vector starting at the upper left and then proceeding clockwise around the edges of the rectangle, as follows:



From here, you create a figure by drawing lines between the pegs, starting at peg 0 and then moving ahead a fixed number of spaces on each cycle, as specified by the constant **DELTA**. For example, if **DELTA** is 11, the first line goes from peg 0 to peg 11, the second goes from peg 11 to peg 22, and the third—which has to count 11 pegs clockwise past the beginning—goes from peg 22 to peg 5. The process continues in this way until the line returns to peg 0. As usual, implementing the wrap-around feature is much easier if you make use of the **%** operator.

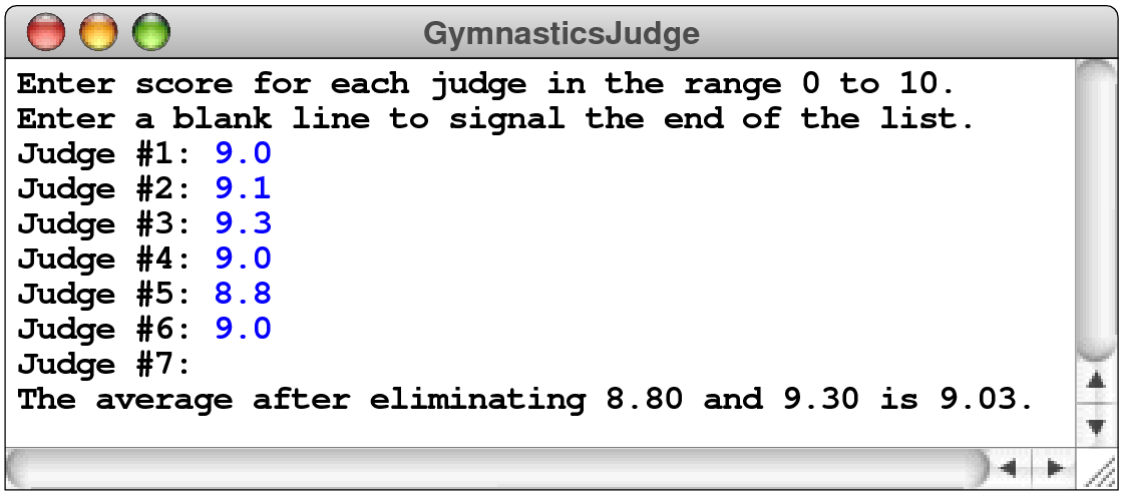
Write a program that simulates this process on the graphics window using larger values for **N_ACROSS** and **N_DOWN**. The output of the program with **N_ACROSS** equal to 50, **N_DOWN** equal to 30, and **DELTA** equal to 67 appears in Figure. By changing those constants, you can create other wonderful patterns composed entirely of straight lines.



Please implement *initPegVector* and *showYarnPattern*function in *YarnPattern.cpp*. **The hyperparameters in *YarnPattern.h* do not need to be modified.**

Question 2: Gymnastics Judge

Using the definitions of **MAX_JUDGES** and **scores** on page 498 as a starting point, write a program that reads in gymnastics scores between 0 and 10 from a set of judges and then computes the average of the scores after eliminating both the highest and lowest scores from consideration. Your program should accept input values until the maximum number of judges is reached or the user enters a blank line. A sample run of this program might look like this:



Please implement *sumArray*, *findLargest*, *findSmallest* and *readScores* function in *GymnasticsJudge.cpp*. **Illegal inputs should not be considered. The hyperparameters in *GymnasticsJudge.h* do not need to be modified.**

File Tree

We will deliver an archive with the following structure, when submitting, please also keep the same structure.

```
.
├── lib
|   └── <some files>       # do not modify
├── res
|   └── <some files>         # do not modify
├── assignment3.pro      # do not modify
└── src
    ├── YarnPattern.cpp        # change it
    ├── YarnPattern.h          # do not modify
    ├── GymnasticsJudge.cpp    # change it
    ├── GymnasticsJudge.h      # do not modify
    ├── main.cpp             # do not modify
```

Special Notes

1. Please refer to the grading criteria documents for detailed descriptions
2. Do not rename functions or change its type signature.
3. We are expecting **beautiful code layout** and **consistent code style**.
4. One should refer to the provided files to check the recommended comment style:

   If you want to add some additional functions, we are expecting to see something like

   ```
   /*
    * Function: permutations(int n, int k)
    * Usage: int n = permutations(n, k);
    * ----------------------------------
    * Returns the number of permutations of n objects taken k at a time.
    * In a permutation, the order of elements is important, so that the
    * combination AB represents a different permutation than BA.
    */
   ```

   before your function prototype and

   ```
   /*
    * Implementation notes: monthToString
    * -----------------------------------
    * The monthToString function must return some value if the month does not
    * match any of the enumeration constants.  Here, as in the Direction
    * type, the function returns ???.
    */
   ```

   before you implementation.
5. We recommend you to use Qt creator to write the code. Though **not required**, we hope you could eliminate " "warnings" as much as you can. They are definitely useful for you to get rid of some apparent mistakes.