Nitro C++ Proficiency Test for Senior Candidates

The purpose of this test is to get some insights into your ability to write quality code to a specification. It is a simple test that candidates are requested to complete over a few days but generally no longer than one week. Because of that, the expectations are high. It is important that the result be demonstrably correct and robust. A single threaded solution is sufficient. Maximizing performance is not a factor in this test although highly wasteful code will be marked down. No compiler is specified nor is any particular build system specified. The code should ideally be portable. Also, this is a C++ test. Classes, objects and use of the STL are expected.

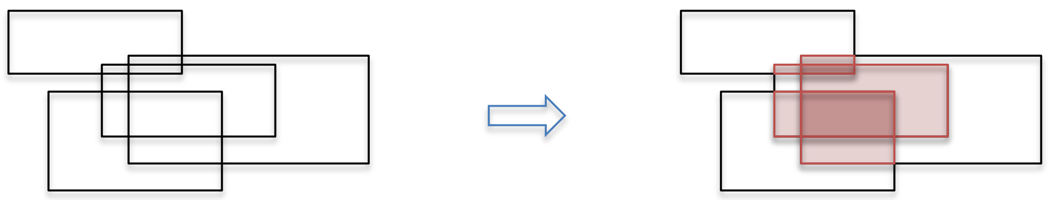
Please create a repository on bitbucket.org or equivalent, with your submission. Ensure the repository has a README file in the root detailing any steps needed to build the code. How easy it is to build the program is relevant. Please include a copy of this document in your root folder.

Test

Create a console program in C++ that will take a command line argument identifying a JSON file containing rectangle definitions. Some of these rectangles may intersect with one or more others.

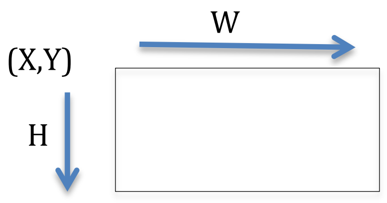
The objective is to report the set of those intersections.

For example, the rectangles on the left result in the intersections on the right.



Each of the lightly shaded areas is the intersection between two rectangles. Each of the darkly shaded areas is an intersection between three rectangles.

The JSON file defines a collection of rectangles in a 2D space using X, Y integer coordinates for one corner and a width and height value. Width and height are always non-negative integers. Rectangles are oriented with X,Y at the top left corner.



On start-up, the JSON file will be loaded and validated. Then two lists will be outputted: the list of all rectangles in the file; and the list of all intersections.

There are a few points to bear in mind:

- Multiple input rectangles with the same coordinates and sizes are nevertheless distinct. All should be included when determining intersections.
- Rectangle A∩B is the same as B∩A and should result in only one rectangle in the result. Take care not to accidentally include both.
- With the exception of A∩B and B∩A above, duplicate matching intersections should be included in the result. Despite being the same, they are distinct intersections.
- Intersections with zero widths or heights should be discarded
- The result must include **all** intersections involving **two or more** rectangles.
- The JSON file may contain any number of input rectangles.
- Your program must be able to process at least **1000** rectangles. You may discard any after the first 1000.

Thus, the following is an example JSON file that could be used as an input file

```
{
        "rects": [
                {"x": 100, "y": 100, "w": 250, "h": 80 },
                {"x": 120, "y": 200, "w": 250, "h": 150 },
                {"x": 140, "y": 160, "w": 250, "h": 100 },
                {"x": 160, "y": 140, "w": 350, "h": 190 }
        ]
}
```

The program will take a single command line argument, the name of the JSON file. The output**,** for the JSON above, will look something like:

```
Input:
        1: Rectangle at (100,100), w=250, h=80.
        2: Rectangle at (120,200), w=250, h=150.
        3: Rectangle at (140,160), w=250, h=100.
        4: Rectangle at (160,140), w=350, h=190.

Intersections
        Between rectangle 1 and 3 at (140,160), w=210, h=20.
        Between rectangle 1 and 4 at (160,140), w=190, h=40.
        Between rectangle 2 and 3 at (140,200), w=230, h=60.
        Between rectangle 2 and 4 at (160,200), w=210, h=130.
        Between rectangle 3 and 4 at (160,160), w=230, h=100.
        Between rectangle 1, 3 and 4 at (160,160), w=190, h=20.
        Between rectangle 2, 3 and 4 at (160,200), w=210, h=60.
```

Your submission should gracefully deal with outputs that take an excessive amount of time or produce excessively long output listings. For example, processing 1000 *concentric* rectangles will encounter heat-death of the universe before its output completes.

Judging:
The submission will be judged by assessing, among other things, the following factors:

- **Correctness** – does it do what the spec requires? **Critical**
- **Verification** – does the submission verify thoroughly that it behaves as required? **Critical**
- **Readability** – is it easily read without excessive scrolling, are entities well named, etc.?
- **Clarity** – does it make clear what it is trying to do?
- **Simplicity** – is the code relatively uncomplicated?
- **Safety** – is it resource, exception and input safe? Are the abstractions easy to use correctly and hard to use incorrectly
- **Expertise** – does the code feel like it was written by an expert?
- **Craftsmanship** – how maintainable is the code by others?

Critical: Correctness and Verification must be adequately addressed, or the submission will be summarily rejected. Assuming correctness and verification are adequately dealt with, the other judging criteria are then considered. These are the main differentiating criteria between candidates' submission and, hence, you should be aware that these are examined more thoroughly.

Feel free to use any open source third party libraries you like, either include them in your repo or include steps to get them automatically as part of you build script/project/solution/etc.

While you are expected to use online resources to research your submission, plagiarising is considered bad form for a test. To obviate this, you will be asked at interview to explain your code, to evaluate its efficiency and to describe any modifications necessary when a substantive change is made to the requirements.