Nitro C++ Proficiency Test

The purpose of this test is to get some insights into your ability to write quality code to a specification. You are expected to complete this and return your submission within one week. If more time is required, please let your Nitro contact know as soon as possible.

Test

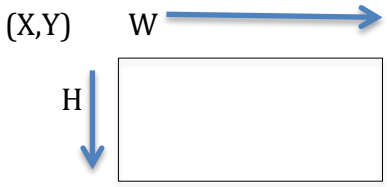
Create a console program in C++ that will take a command line argument identifying a JSON file. This file will contain rectangle definitions. Some of these rectangles may intersect with one or more others.

The objective is to report on 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 integers that are never negative. Rectangles have 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 input file; and the list of all intersections.

There are a few points to bear in mind:

- The JSON file may contain any number of input rectangles. Your program must be able to process JSON files containing at least 10 rectangles. You may discard the remaining rectangles after the first 10.
- 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.
- The result must include **all** intersections involving **two or more** rectangles.

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.
```

Judging:

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

- **Correctness** – does it do what the spec requires? Please understand that **two or more** does not stop at three. If the input includes six concentric rectangles, then the intersections reported should include entries with 2,3,4,5 and 6 contributors.
- **Verification** – does it include sufficient means to verify that it works as required? For example, how was the code tested? Providing unit tests here would be a good thing. The score will be high if you have good tests and low if the tests are poor.
- **Readability** – is it easily understandable? [Good] Is it using simple ideas? [Good]. Is it showing us how good you are at writing complicated code? [Bad].
- **Safety** – is it resource and input safe? If the JSON file is poorly formed will the program avoid a crash. If the JSON file has too much or too little content, will the result be clean. What about memory leaks? Did you protect against them? Are your abstractions easy to use correctly and difficult to use incorrectly?
- **Portability** – if it can only work on a Mac, on Windows or a specific flavor of Linux; if it depends on using a specific compiler, etc., it will score less than one that can be built for any platform that has a C++ compiler. Note: using the latest C++ standard is considered a good thing here even if only a few compilers currently support it. Requiring non-standard or special compiler extensions, for example, would be a bad thing.

The following constraints/recommendations apply:

- A single threaded solution is sufficient.
- The program must be able to deal with JSON files provided by the reviewer. Nitro will use a series of test files that exercise the submitted code to verify that it meets the specification. It will push the program into places where, if it is not written well, it is likely to fail.
- Maximizing performance is not a factor in this test.
- No compiler is specified nor is any particular build system specified.
- This is a C++ test, not a C test. Classes, objects and use of the STL are expected.
- Use of modern C++ (e.g. C++11, C++14 or C++17) is favoured over older C++ standards.
- Use of open source third party libraries is permitted and is recommended for such things as JSON parsing. Please don't write your own JSON parser. Please include sources for any such libraries in your repo or include steps to get them automatically as part of your build script, project, solution, etc. Don't overdo the use of third party libraries.

Submission instructions:
Please create a repository on github.org, bitbucket.org or any equivalent, to hold the files of your submission.

Create a root folder in the repo that contains your project files and subfolders (if used). This folder should be named as follows: NitroCppTest-xxxxxx where xxxxxx is your name without spaces. If your name contains characters that are illegal in a folder name, please substitute as needed. The reason for this is to ensure that when a reviewer pulls your code down to test it, that they will be easily able to associate your test code with you. Please help your reviewer.

Ensure the root folder has a README file detailing any steps needed to build the code. How easy it is to build the program is relevant. The root folder should also include a copy of this document. Finally, please include a link to your repository in your submission.

Please note: by submitting a ZIP file as an email attachment, or by sending an email with embedded code, or anything other than something that closely matches the instructions above, will only serve to demonstrate that you are not good at following specifications.

While you are expected to use online resources to research your submission, plagiarising is considered bad form for a test. To obviate this, you may be asked at interview to explain your code, to evaluate its efficiency and to describe the type of modifications that would be required were there a change made to the requirements. Examples of such a requirements change would include reporting the area of each intersection; increasing the number of rectangles possible in the input; excluding rectangles that are smaller than a certain size, etc., etc.