Calibre D2S R&D C++ Project

1. Problem Formulation:

You will have number of rectangles. They may overlap or not. You need to group the rectangles into minimum number of groups. Each group should has non overlapped squares.

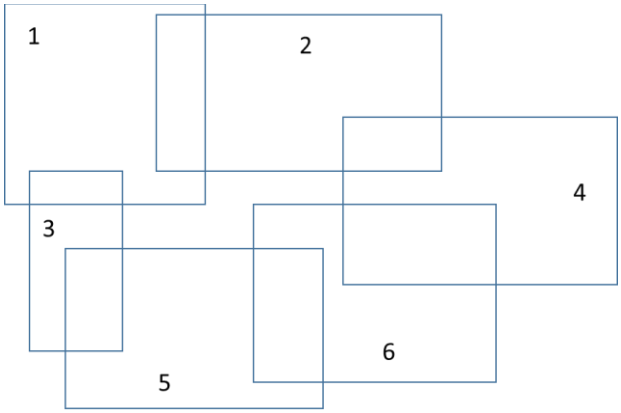


Figure 1: Original rectangles set

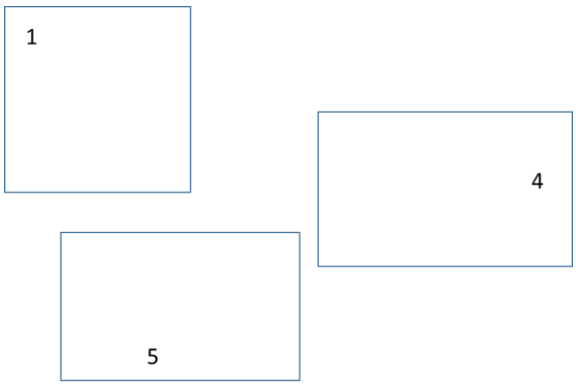


Figure 2: Group 1

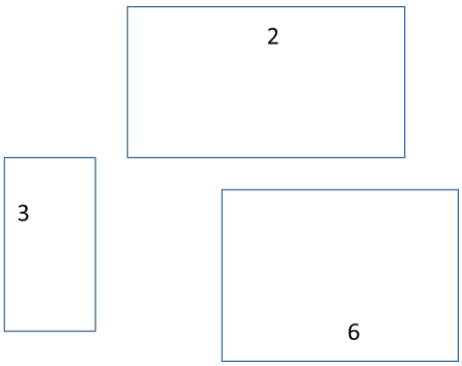
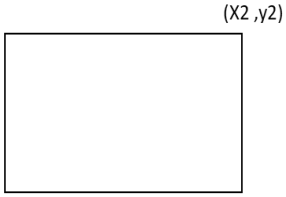


Figure 3: Group 2

2. Input:

- Text file contains the coordinates of rectangles.
- Each line contain coordinates of one rectangle in format (x1 y1 x2 y2).

```
39076 -618 42180 46302
14244 -618 20452 46302
20452 -618 26660 46302
```



- LONG_MIN <= x1, y1, x2, y2 <= LONG_MAX
- You will have 16 data set for testing. 5 small, 5 medium, 5 large and 1 huge.

3. Output:

- Multiple text files with the same input format, each file contain set of non-overlapped rectangles.
- Output files should be named as: group_1, group_2, ... etc.

4. Deliverables:

- Source code files.
- Output files for each data set in a separate directory.
- A detailed report for the project will be explained in next section.
- All deliverables should be compressed in one file ".zip format" and send via mail.

5. Report:

The report should contain the following:

- Mention the platform used: (windows, Linux).
- Flow chart explains the algorithm used.
- Pseudocode for the algorithm used.
- For each input data set: The number of output groups, run time elapsed to finish and the memory usage.
  All organized in the following table:

Data set	Number of inputs	Number of output groups	Runtime (sec)	Memory usage (MB)
data_set_1.txt				
data_set_2.txt				
data_set_3.txt				
data_set_4.txt				
data_set_5.txt				
data_set_6.txt				
data_set_7.txt				
data_set_8.txt				
data_set_9.txt				
data_set_10.txt				
data_set_11.txt				
data_set_12.txt				
data_set_13.txt				
data_set_14.txt				
data_set_15.txt				
data_set_16.txt				

- Testing methodology:
  Explain how had you tested your algorithm?

- Some Screen shots for the output of your program.

6. General notes:

- The delivered code must compile. Codes with compilation issues will be rejected.
- The delivered code must be organized, clean and well documented.
- The code should handle corner cases (for example: no inputs, inputs with wrong formats, …etc).
- Try to provide the most efficient solution (runtime & memory usage).
- It is okay if your algorithm could not handle some input data sets (explain why).