1. Introduction.

Physical layouts are composed of many layers of geometric data. Each layer, by nature, includes a large amount of polygons. They are usually represented as hierarchical structures of cells to reduce complexities. The polygons in different cells might be overlapped hierarchically. This is not good for some operations applied on the layout data. Consider the flattened area computation problem in 2003 IC/CAD contest. If there is no any hierarchical overlapping, then we can compute the flattened area by 1) computing the area of the polygons cell by cell and 2) summing up the areas bottom-up on the hierarchy. For example, consider Fig. 1.

![Fig. 1](image1.png)

The areas of polygons in cells B and A are 400 and 300. Since A contains two B and there is no any overlapping, the flattened area of cell A is 300+2*400=1100. Consider the second example in Fig. 2.

![Fig. 2](image2.png)
The answer would be 1300. But how could we compute the answer without flattening all polygons in B? If we explode only the right polygon of B into A, we will get the result in Fig. 3.

![Fig. 3](image)

Now, the areas of polygons for B and A in Fig. 3 are 200 and 300+200+400=900 respectively. Since A contains two B and there is no any overlapping in Fig. 3, the flattened area of cell A is 900+2*200=1300.

Hierarchical overlapping removing is an important operation on hierarchical layout data. Given a hierarchy of a layer data, the objective of the problem is to produce another hierarchy of data such that there is no hierarchical overlapping of polygons between different cells. The polygons in one cell could be overlapped. There is no need to merge overlapped polygons into a larger one. Notice that edge touching is considered as one kind of overlapping in this problem but point touching is not. Notice also that the result is not unique. There are many feasible results. The result that explodes fewer polygons is considered as a better result.

The instance placement of referenced cells in their parent cells could be reflected about the X-axis and rotated 0, 90, 180 or 270 degrees. In order to simplify the complexity of this problem, only rectangles will be considered as input polygons.

Fig. 4 gives a more complicated input case without coordinates. Fig. 5 is one of its possible results.
2. Input/Output Format.

The following is the Bachus Naur Form (BNF) of our input format:

\[
\text{<Input>} ::= \text{‘BGNLIB’} \text{<Cell>* ‘ENDLIB’}
\]

\[
\text{<Cell>} ::= \text{‘BGNCEL’} \text{<Name> <Element>* ‘ENDCEL’}
\]
Each token is separated by white space, tab or new line characters. `<String>` could be composed by any printable ASCII characters including digits and non-alphabet characters. `<String>` should not conflict with any keywords used in this format. `<Number>` could be any 32-bit signed integer. A cell reference could be rotated about its original point (0,0) or reflected about the X-axis. The values 0, 1, 2 and 3 of `<Rotation>` mean to rotate the corresponding cell to the original point (0,0) counterclockwise by 0, 90, 180 and 270 degrees respectively. If `<Reflection>` is 1, the cell is reflected about the X-axis BEFORE rotation. Otherwise, no reflection is necessary. Please notice that, for a cell reference B in the definition of cell A, cell B will be put into a containing cell A by the following steps: First, do the necessary reflection and rotation for cell B according to the values of `<Reflection>` and `<Rotation>`. Then, put the result into A by matching the original point (0,0) of the result of B with the point `<XY>` of A. For a `<Rectangle>`, the first `<XY>` is the coordinate for the bottom-left corner and the second `<XY>` is for the top-right corner. Notice that it is ok if there is no obvious (unique) top cell in an input file.

Consider the example in Fig. 2. The input file could be as follows:

```
BGNLIB
BGNCEL A
BGNREF B 20 10 0 0 ENDREF
BGNRECT 10 40 40 50 ENDRECT
BGNREF B 60 10 0 0 ENDREF
BGNRECT 80 20 90 50 ENDRECT
ENDCEL
BGNCEL B
BGNRECT 0 0 10 20 ENDRECT
```
III. Language and Platform

1. Language: C or C++
2. Compiler: GNU gcc or g++ with optimization flag -O
3. Platform: Sun Solaris and Linux

IV. Performance Evaluation.

The performance evaluation will base upon correctness, sum of cell areas, speed and memory consumption. The output hierarchy will be checked for no hierarchical overlapping for correctness first. Then the area for all rectangles in every cell will be computed and summed up. The area of overlapped rectangles in the same cell will not be computed twice.

Please make sure that your source code could be compiled successfully with optimization flag –O by gcc or g++ under Solaris AND Linux platforms. Your program should accept two arguments in the command line. The first one is the
input file name and the second one is the name of the cell that you want to compute the area. Suppose that the file name of the compiled executable is “a.out” and the input file name is “case1.dat” and the name of the top cell is ‘TOP’. Please make sure your program could be launched by the following command in UNIX:

% a.out case1.dat TOP

Please always use “result.dat” as the output file name.