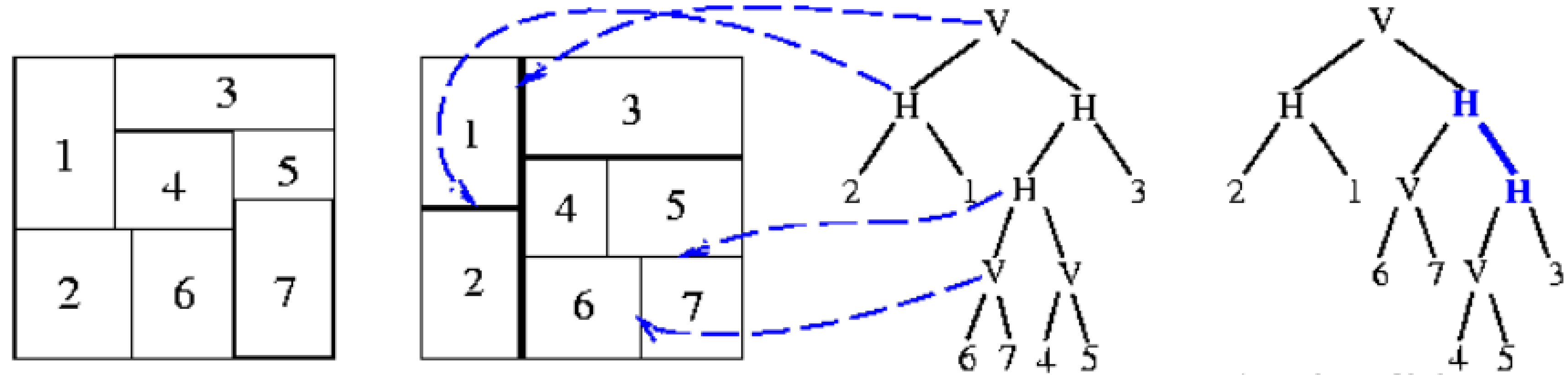
# Floorplanning Exercise

You are asked to implement the **slicing floorplan design algorithm** which is based on **SA and normalized polish expression** representation for solving rectangle packing problem. The rectangle packing problem is defined as follows: Given many rectangular modules of arbitrary sizes, place them without overlapping on a layer in the smallest bounding rectangle. It can be used to solve VLSI floorplan/placement problem.



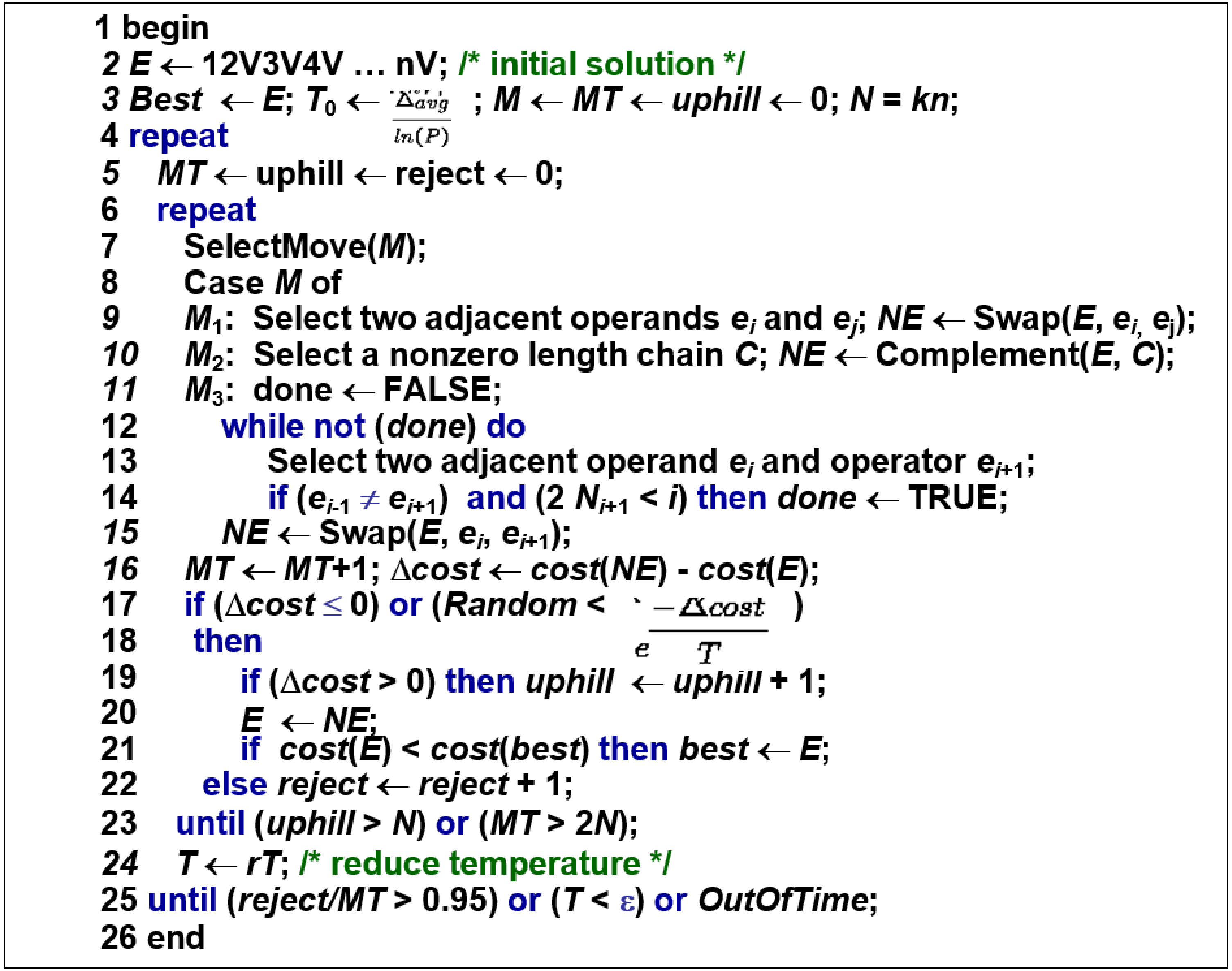
Non–slicing floorplan

Slicing floorplan

A slicing tree (skewed)

Another slicing tree (non–skewed)

# Wong-Liu Algorithm



```
begin
 E ← 12V3V4V … nV; /* initial solution */
 Best ← E; $T_0 \leftarrow \frac{\Delta_{avg}}{ln(P)}$ ; M ← MT ← uphill ← 0; N = kn;
 repeat
   MT ← uphill ← reject ← 0;
   repeat
     SelectMove(M);
     Case M of
     $M_1$: Select two adjacent operands $e_i$ and $e_j$; NE ← Swap(E, $e_i$, $e_j$);
     $M_2$: Select a nonzero length chain C; NE ← Complement(E, C);
     $M_3$: done ← FALSE;
        while not (done) do
            Select two adjacent operand $e_i$ and operator $e_{i+1}$;
            if ($e_{i-1} \neq e_{i+1}$) and ($2 N_{i+1} < i$) then done ← TRUE;
        NE ← Swap(E, $e_i$, $e_{i+1}$);
     MT ← MT+1; Δcost ← cost(NE) - cost(E);
     if (Δcost ≤ 0) or (Random < $e^{\frac{-\Delta cost}{T}}$ )
      then
        if (Δcost > 0) then uphill ← uphill + 1;
        E ← NE;
        if cost(E) < cost(best) then best ← E;
      else reject ← reject + 1;
   until (uphill > N) or (MT > 2N);
   T ← rT; /* reduce temperature */
 until (reject/MT > 0.95) or (T < ε) or OutOfTime;
 end
```

# Definition of Rectangle Packing Problem

- Rectangle Packing Problem in this assignment is defined as follows:

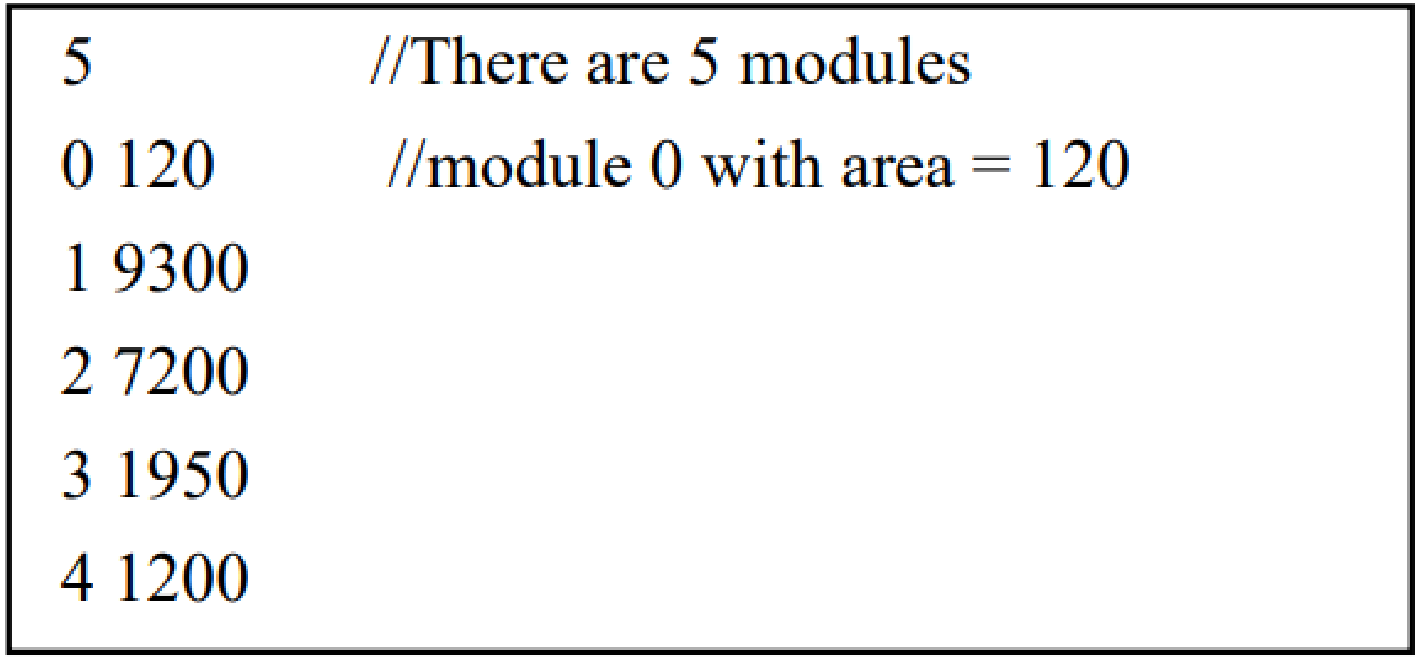
**Input:** Given a set of rectangular modules each of which is a soft module with aspect ratio is ranging from 0.5 to 2

**Output:** A legal floorplan/placement result (no overlapping)

**Objective:** The area of the packing area is as small as possible

# Input File Format

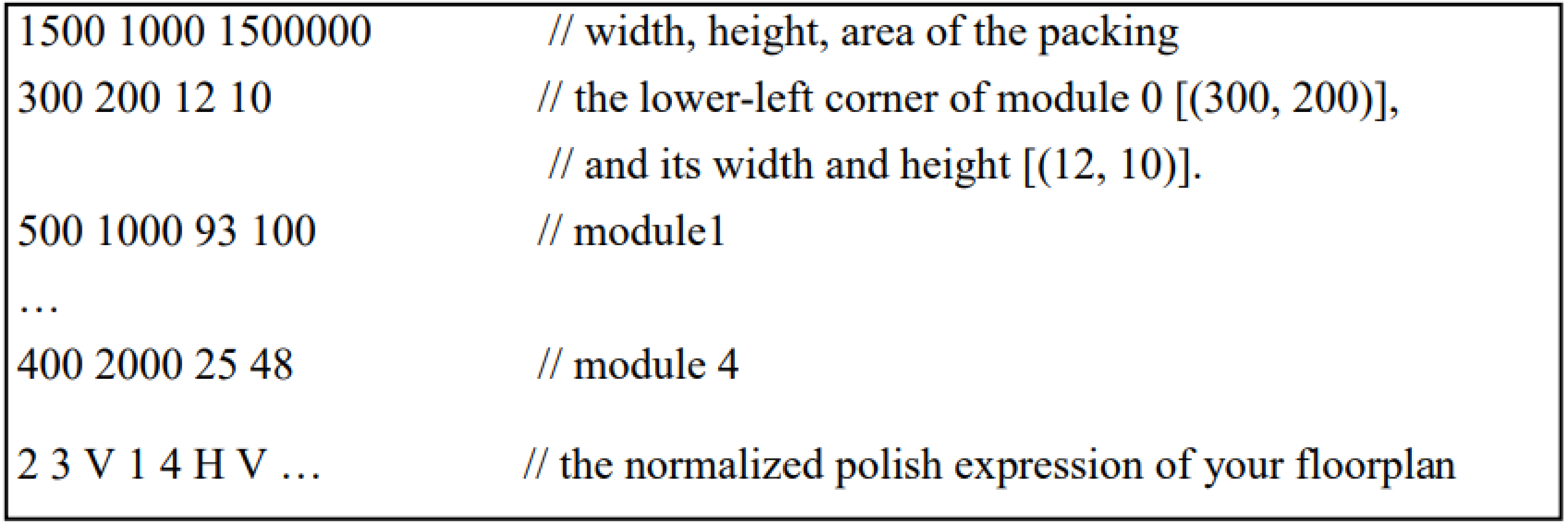
- The first line gives the number of modules, denoted by n. From line 2 through line n+1, each line specifies the index and area of a module.



```
5              //There are 5 modules
0 120           //module 0 with area = 120
1 9300
2 7200
3 1950
4 1200
```

# Output File Format

- For each test case, output the width, height, and area of the best packing found by your program in line 1 with “blank” characters separating them. For each of the next n lines (with the increasing order of module indices), output the coordinates of the lower-left corner (x, y), width, and height of a module. All of the data are separated by a “blank” character. The final line is the normalized polish expression of your floorplan. Every two consecutive elements in the expression are separated by a “blank” character

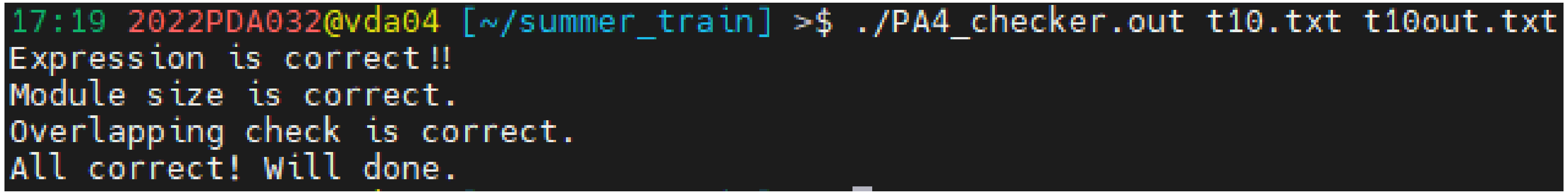


```
1500 1000 1500000         // width, height, area of the packing
300 200 12 10             // the lower-left corner of module 0 [(300, 200)],
                          // and its width and height [(12, 10)].
500 1000 93 100           // module1
…
400 2000 25 48            // module 4

2 3 V 1 4 H V …          // the normalized polish expression of your floorplan
```

# Checker

- How to use:
- **./Checker.out  your_input_file  your_output_file**
- Before you use this checker, you have to type **chmod 755 Checker.out** in your command line. After that. you can use this checker.
- If your program is correct, you will get the following prompt message:



```
17:19 2022PDA032@vda04 [~/summer_train] >$ ./PA4_checker.out t10.txt t10out.txt
Expression is correct!!
Module size is correct.
Overlapping check is correct.
All correct! Will done.
```

- On the contract, if your program has some mistake, you will get which state is incorrect. for example, if your normalized polish expression is incorrect, you will get " Expression has some mistake." and so on.