**EE6094 CAD for VLSI Design**

**Programming Assignment 3: Floorplanning**

Student ID: 108501023

Student Name: 李品賢

**Compile, Execute and Verification**

1. Pull the source code, i.e., *108501023\_PA3.cpp*, *Makefile*, *t10.txt*, *t20.txt*, *t500.txt* and *checker* into the workstation folder.

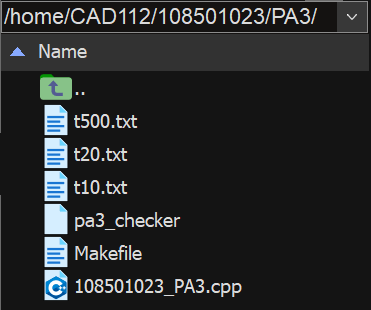


Fig. 1

1. Use *Makefile* as a trigger point to run the *108501023\_PA3.cpp* program, and then the output *t10\_out.txt* / *t20\_out.txt* / *t500\_out.txt* are generated.

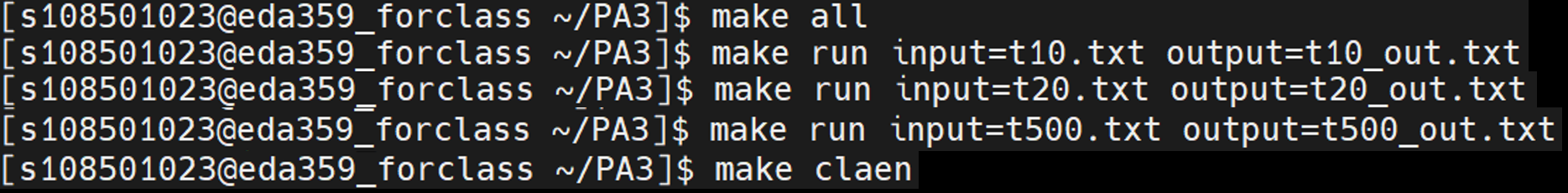


Fig. 2

* + make all
  + make run intput=t10.txt output=t10\_out.txt
  + make run intput=t20.txt output=t20\_out.txt
  + make run intput=t500.txt output=t500\_out.txt
  + make clean

1. Use checker to check whether output files fits the standard output format.
   * ./checker t10.txt t10\_out.txt
   * ./checker t20.txt t20\_out.txt
   * ./checker t500.txt t500\_out.txt

**Completion**

All three cases are successfully passed the checker, the screen shows three happy “Pepe the Frog”. The following three figure (Fig. 3, Fig. 4, Fig. 5) are the results.

|  |  |  |  |
| --- | --- | --- | --- |
|  | t10.txt | t20.txt | t500 |
| Completion | O | O | O |
| Area | 57323.05 | 89318.63 | 2636155.31 |

****

Fig. 3

****

Fig. 4

****

Fig. 5

**Data structure**

I use slicing tree and the corresponding polish expression to represent a single slicing floorplan. Take Fig. 6 as an example, the left-hand side is a slicing floorplan, and the right-hand side is its slicing tree. Additionally, we can do the postorder traversal on the tree, and then we get the sequence 21H67V45VH3HV which is the polish expression of this slicing tree.

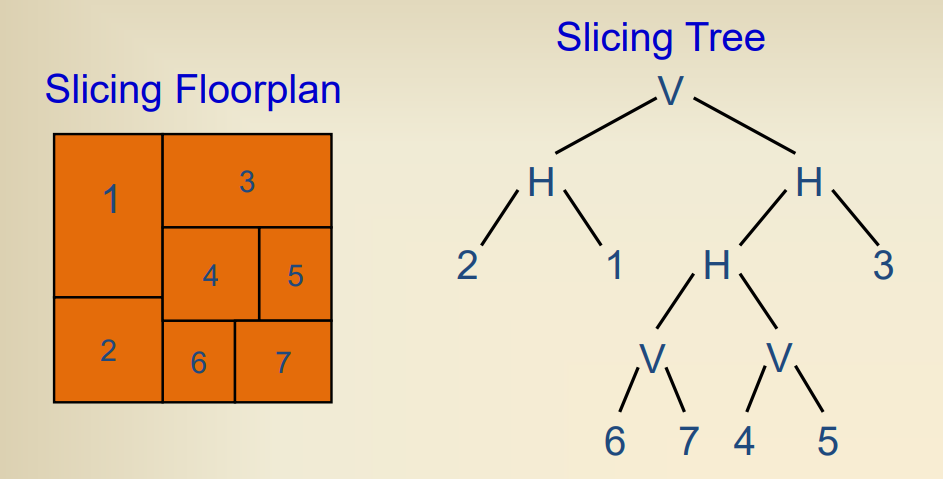
****

Fig. 6

**Algorithm**

The basic algorithm I used is Wong-Liu Algorithm (Fig. 7), and it’s also a simulated annealing based algorithm. The thesis indicates the how to choose the initial temperature and how to set the parameters. Also, the most important part is that it gives three types of move changing current state to its neighborhood state.

In my program, I take a pancake shape as an initial condition. Also, if the single SA have done before 2 hours, it would start another run of SA again and again until the time is up. The newly started SA take the best polish expression as an initial condition.

For the bottom-up procedure, I use Stockmeyer Algorithm (Fig. 8), the time complexity is O(m+n) which is extremely fast. It improves run time a lot.

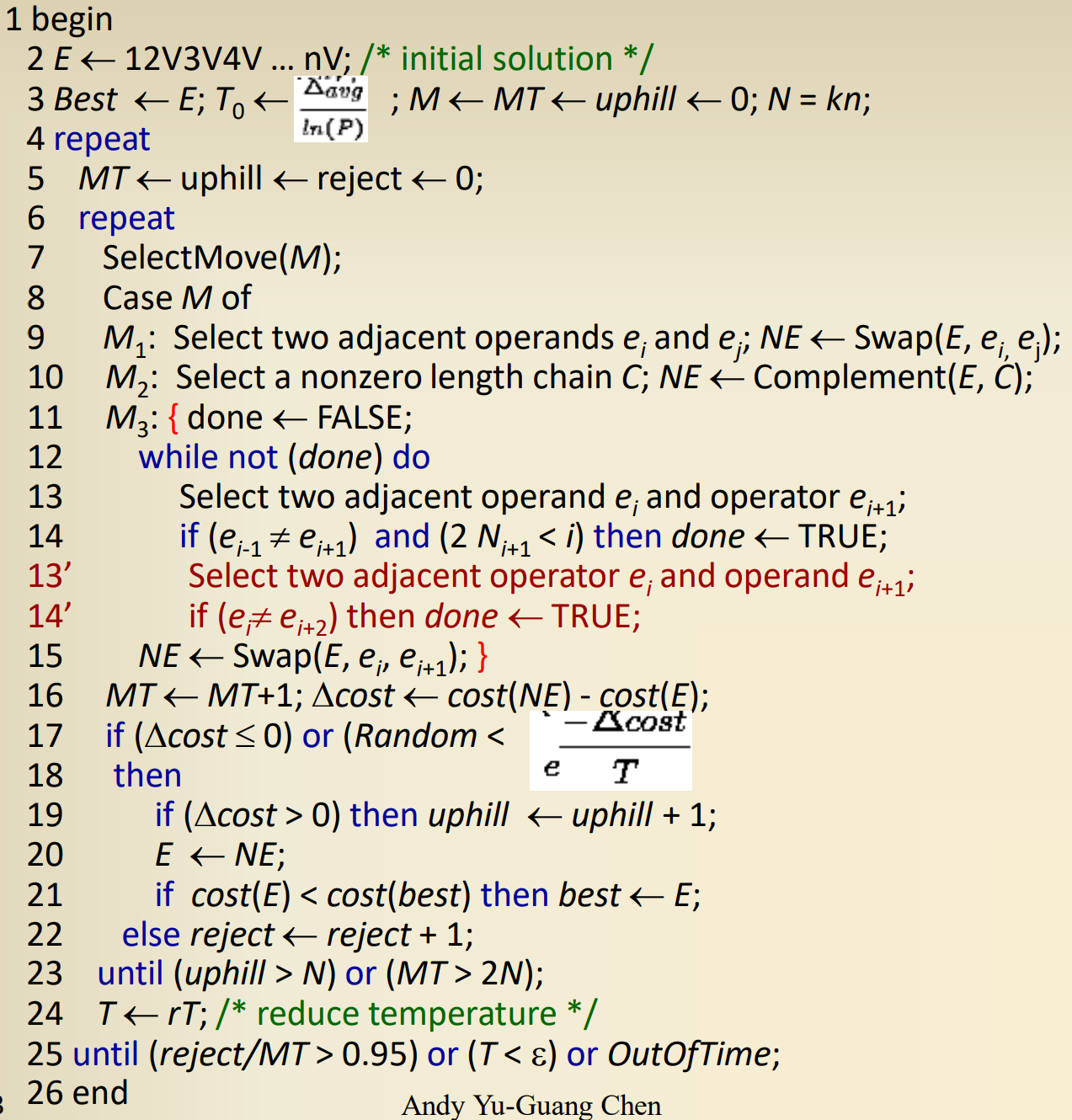


Fig. 7

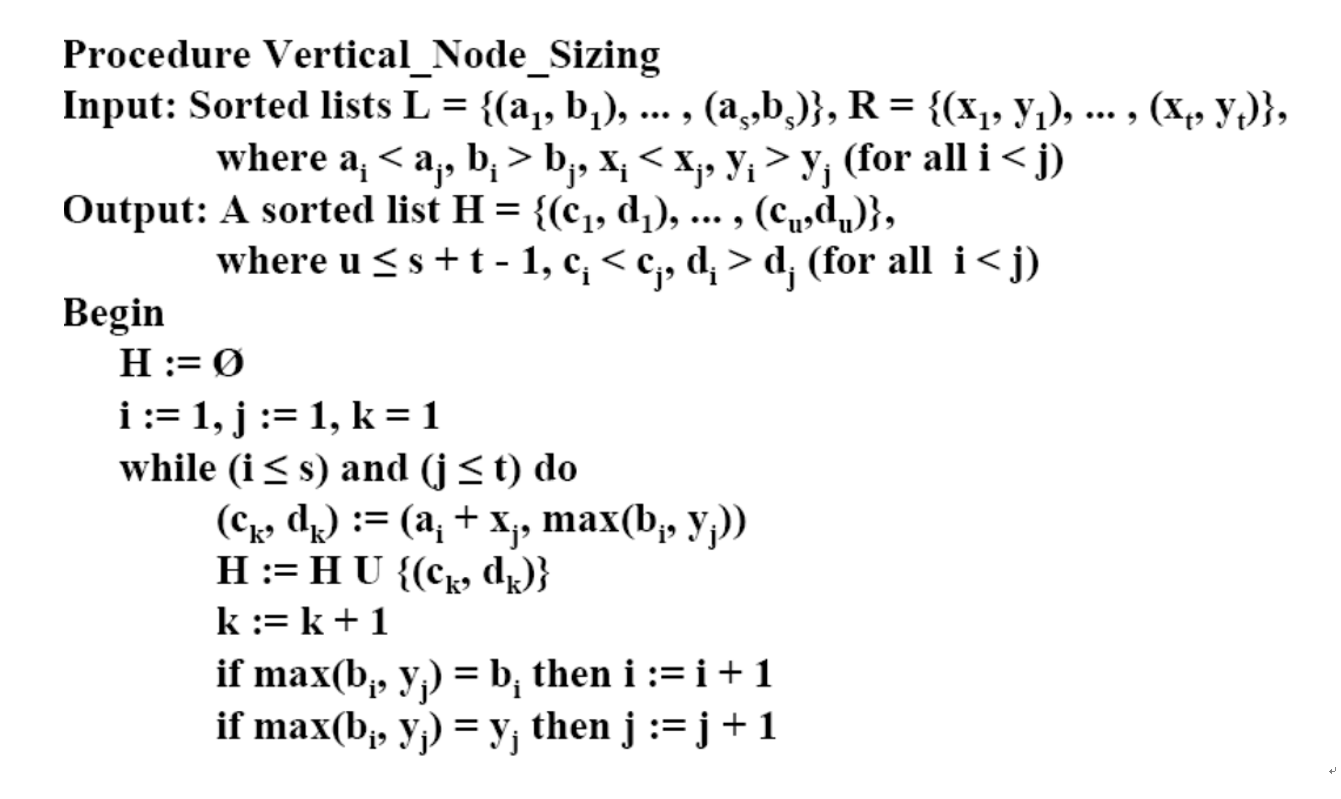


Fig.

**Flow Chart**

1. Overall program

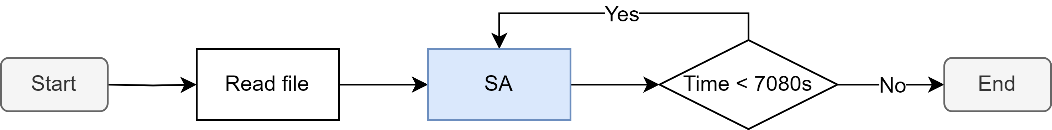


Fig. 9

1. Simulated annealing procedure

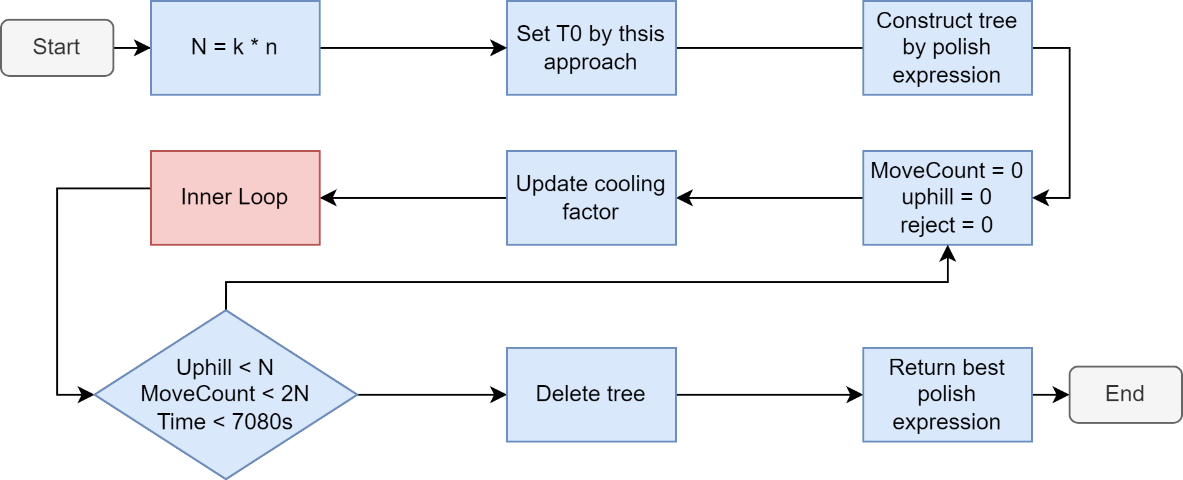
****

Fig. 10

1. Inner loop of simulated annealing

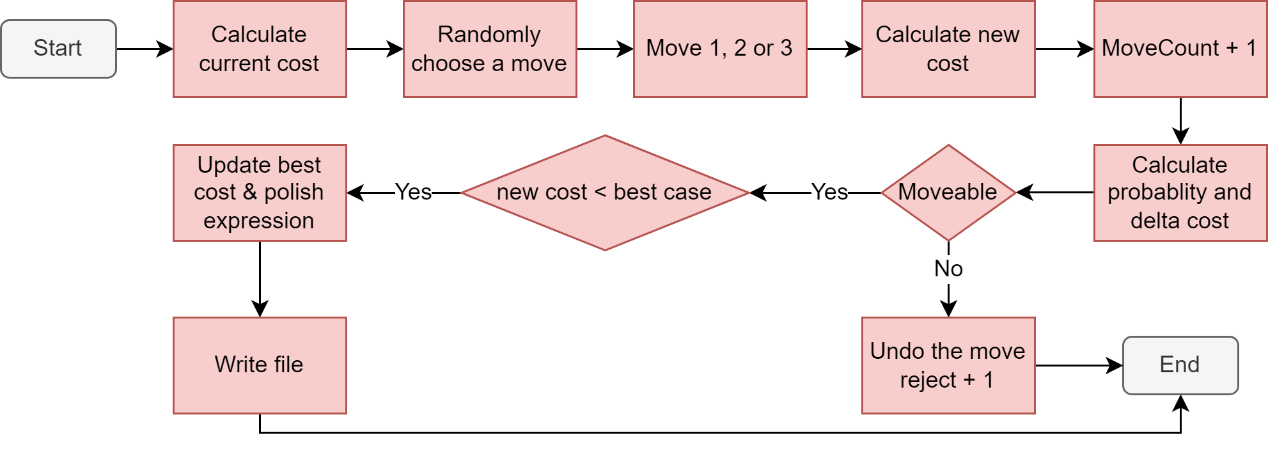
****

Fig. 11

**Data structure in program**

1. **class Size** – indicate the aspect of an area

|  |  |  |
| --- | --- | --- |
| **Data type** | **Name** | **Purpose** |
| double | width | record the width of an area |
| double | height | record the height of an area |

1. **class Module** – record the area and final selected aspect of the input module

|  |  |  |
| --- | --- | --- |
| **Data type** | **Name** | **Purpose** |
| double | area | the area of input module |
| Size | aspect | the final selected aspect of an input module |

1. **class Block** – record one possibility of the aspect combining left-child aspect and right-child aspect, and also their index

|  |  |  |
| --- | --- | --- |
| **Data type** | **Name** | **Purpose** |
| int | Lindex | the index of selected aspect of left-child in this block |
| int | Rindex | the index of selected aspect of right-child in this block |
| Size | aspect | the aspect combining left-child aspect and right-child aspect |

1. **class Node** – a tree node including pointers, slicing information and all possibilities of combined aspect

|  |  |  |
| --- | --- | --- |
| **Data type** | **Name** | **Purpose** |
| class Node \* | left | point to its left child |
| class Node \* | right | point to its right child |
| class Node \* | parent | point to its parent |
| string | data | the information of slicing tree, either is a character “V” and “H” or a module number |
| bool | update | false indicate need to update; true indicate no need to update |
| vector<Block> | blks | all possibilities of combined aspect |

**Important variables**

1. In class Floorplanning

|  |  |  |
| --- | --- | --- |
| **Data type** | **Name** | **Purpose** |
| int | moduleCount | number of modules |
| double | total\_area | sum of area of input modules |
| double | global\_cost | best cost between all SA result |
| int | pts | number of points picked up in shape curve |
| double | iter\_T | the parameter k of equation N = kn in the thesis |
| Node \* | root | point to the root of a slicing tree |
| Node \* | avail | point to the available node that can be reused |
| vector<Module> | module | the information of input module |
| vector<string> | SPE | the polish expression of a slicing floorplan |

1. In SA function

|  |  |  |
| --- | --- | --- |
| **Data type** | **Name** | **Purpose** |
| vector<string> | best\_SPE | the polish expression leading to the best area in a single SA |
| Node \* | temp\_root | point to a root of temporary tree especially for move 3 |
| double | T0 | the initial temperature |
| double | T | the current temperature |
| double | w | the cooling factor |
| double | curr\_cost | the cost of current state of a floorplan |
| double | new\_cost | the cost of neighborhood state of a floorplan |
| double | delta\_cost | the difference between current cost and new cost |
| double | best\_cost | the smallest cost in a single SA |
| int | Move\_type | the integer type ranging from 1 to 3 |
| int | MoveCount | the move count in a specific temperature |
| int | uphill | the number of times accepting worse solution |
| int | reject | the number of times being rejected worse solution |

**Important functions**

(the list ignores the Floorplanning:: mark and the arguments of function)

1. the functions used in SA

|  |  |
| --- | --- |
| 1. | double init\_temp() |
|  | move 50 times to get the average cost difference of uphill, and divide this value by ln(P), where P is near 1.  T0 = ave/ln(P), ave = (uphill cost-current cost)/uphill times |
| 2. | Node \* SPE\_to\_tree() |
|  | turn a polish expression into slicing tree |
| 3. | void bottom\_up() |
|  | use postorder traversal to combine the left-child aspect and right-child aspect, record indexes of them, eliminate unnecessary combinations, and also set the limit of combination number by picking up the smaller area. |
| 4. | double update\_cooling\_factor() |
|  | w is 0.85 for T above 20, while w is 0.9 for T under 20 |
| 5. | double calculate\_cost() |
|  | cost is normalized by the total input area and multiply 100 |
| 6. | int select\_move() |
|  | randomly pick up a move ranging from 1 to 3 |
| 7. | bool is\_movable() |
|  | if the cost difference is negative or the cost difference is positive but its random number is lower than the possibility, it returns true, otherwise, returns false. |
| 8. | void write\_file() |
|  | write the output result when the best cost in a single SA is smaller than the global cost between SAs |
| 9. | void M1() |
|  | randomly pick up two operand and swap them, either in the slicing tree or in the polish expression |
| 10. | void M2() |
|  | randomly pick up a chain of operators and invert them, either in the slicing tree or in the polish expression |

|  |  |
| --- | --- |
| 11. | void M3() |
|  | randomly pick up an operator and an operand and swap them, either in the slicing tree or in the polish expression |
| 12. | void delete\_tree() |
|  | free nodes in the slicing tree to *Node \*avail* pointer when reach the end of SA |

1. the function used in write\_file()

|  |  |
| --- | --- |
| 1. | Node \* top\_down () |
|  | use preorder traversal to give the index of selected aspect of each child, and assign aspects into variable *vector<Module> module*. Then, we can write file by those information. |

**Parameter adjustment and observation**

1. **Observation 1:**

The module counts and # of sampling point of shape curve strongly impact the run time. No need to say, the more sampling point an area has, the smaller overall area we get.

* I find that I can run multiple SA on both t10.txt and t20.txt even if they have over 10000 sampling points, so I try to increase # of points to find optimal solution.
* As the module count increase, I decrease # of sampling point so that the program can run SA at least one time. (My program only can handle module count under 4000, or the # of sampling is lower than 2 which is meaningless)

1. **Observation 2:**

In case t500.txt, no matter how I change the initial condition, the pancake shape is still the best solution after finishing SA.

* I turn to increase # of sampling point of the case t500.txt at first time, then output the pancake solution. Then I decrease # of sampling point, then start to run SA.

1. **Observation 3:**

If I use the non-normalized area as the cost function, the initial temperature of bigger case will be very high. The cost function using normalized area has a problem that opposite to the one I just mentioned.

* Therefore, I use normalized area multiply 100 as cost function.

1. **Observation 4:**

The better solution tends to emerge at lower temperature.

* I use the factor 0.85 recommended in thesis for T > 20, while it’s 0.9 for T <= 20.

**Makefile**

Because I use single .cpp file in this project, there is only one executable file created, i.e., *108501023\_PA3.o*. The following is source code of Makefile.

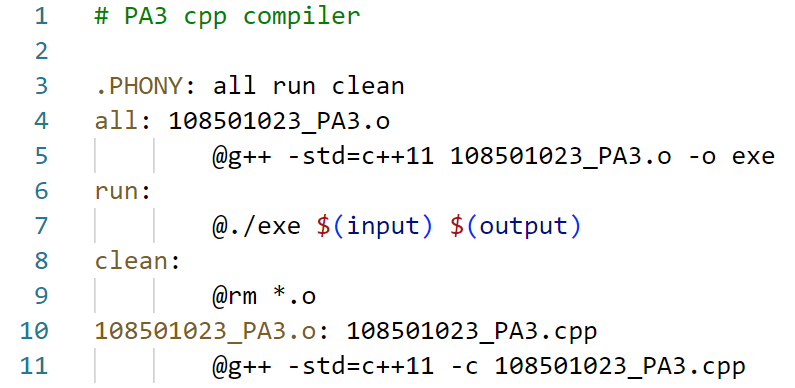


Fig. 12

**Hardness**

The hardest part is the debugging procedure after the overall program architecture is nearly finished. The SA engine starts smoothly but the output file is not correct, so I have to trace code to find where the bug is. Independent Move1, Move2 and Move are executed correctly. When I mix them into SA, it turns out to be wrong. That’s quite annoying.

Also, the adjustment of parameters is also a great challenge. During these time, I still can’t figure out a systematic approach to find the best parameter. The only thing I do is trial-and-error through little observation. Therefore, I learn that I should study more thesis so that I can get more knowledge and come up with more ingenious ideas.

**Suggestion**

I am grateful for having this project, it helps me integrating data structure background into this project.

**Reference**

1. The class slide of chapter 9. (2023Spring\_EE6094\_CAD\_Chapter9\_FloorPlanning)
2. Wong & Liu, “A new algorithm for floorplan design,” DAC-86.