Data structures:
//Input name, logic value, time

- vector <tuple<string, bool, int>> cirInputs; (sorted)
- ordered_map<(int),bool,timestamp> //make this a namespace?

//Time, sched. events, act. List
//Enter time from .stim directly
Fill tuple with elements from cirInputs at the current time

- ordered_map<int, pair<vector<tuple<string, bool, int>>, queue<Logic_Gate>>> bigBoss; BB
- usedGates<Logic_Gate>
- Logic_Gate:
  - string name;
  - int num_Of_Inputs = 0;
  - int delay_ps = 0;
  - vector<pair<string,int>> cir_Input_Names;**
  - string cirCompName;
  - string cir_Output_Name;
  - string cirType; //Same value as string name

1. Initialize all inputs to 0
   a. Instead of cir_Input_Names initializing with -1 we will initialize with 0.
2. Initialize scheduled events with cirinputs

Algorithm:
Repeat until i == BB.size(){

1. Access the first element in bigBoss (access each element in the tuple at that time)

cout<< "Current element(s) from scheduled events at time: "<< (BB.first) <<'\t'<<get<0>((((BB.second).first).second))<<'\t'<<get<1>((((BB.second).first).second))<<'\t'<<get<2>((((BB.second).first).second))<<endl;

   a. Access each index in usedGates
      i. Access each pair in cir_Input_Names in gate (search with string //element name)
         1. Check if element is in them (error handling)
            a. If the value of element in the pair is different -> push into activity list
            b. If the value of element in the pair is the same -> don't push

cout<<"Current element(s) in activity list: "<<(((BB.second).second) <<//cout queue

Repeat until activity list is empty (cout a statement for each action)
{

2. Dequeue top element
3. Run operator
   a. Enqueue into scheduled events (i.e. cirInputs)

}

}