1. Constraint to generate the pattern 0102030405.

```
 1 module jk;
 2 class abc;
 3   rand int a[];
 4   constraint size {a.size==10;}
 5   constraint mirr { foreach(a[i])
 6                           if(i%2==0)
 7                               a[i]==0;
 8                           else
 9                               a[i]==(i+1)/2;
10                 }
11 endclass
12
13 abc m_h = new();
14
15 initial begin
16         m_h.randomize();
17 $display("\n\n");
18
19         $write("\t\t\t The required Pattern is: ");
20         foreach (m_h.a[i])
21           $write("%0d",m_h.a[i]);
22
23 $display("\n\n");
24 end
25
26 endmodule
```

SAMPLE OUTPUT:

```
                The required Pattern is: 0102030405

RNQUIE: Simulation is complete.
```

2. Constraint to generate unique numbers between 99 to 100.

```
 1 module JK();
 2   class eve;
 3     rand int val;
 4     real out_val;
 5
 6       constraint a1 {val inside {[990:1000]};}
 7
 8     function void post_randomize();
 9       out_val = val/10.0;
10       $display("\t\t\t Unique value is %2f", out_val);
11     endfunction
12
13    endclass
14
15    initial begin
16       eve p1=new();
17         repeat(20) begin
18                 p1.randomize();
19         end
20     end
21 endmodule
```

SAMPLE OUTPUT:

```
                Unique value is 99.600000
                Unique value is 99.500000
                Unique value is 99.000000
                Unique value is 99.400000
                Unique value is 99.900000
                Unique value is 99.300000
                Unique value is 99.300000
                Unique value is 99.300000
                Unique value is 99.700000
                Unique value is 99.100000
                Unique value is 99.100000
                Unique value is 99.800000
                Unique value is 99.300000
                Unique value is 99.200000
                Unique value is 99.700000
                Unique value is 99.800000
                Unique value is 99.100000
                Unique value is 99.400000
                Unique value is 99.200000
                Unique value is 99.000000
NQUIE: Simulation is complete.
```

3. Constraint - divisible by 5.

```
1  class abc;
2    rand bit[7:0] a;
3    constraint divby5 {a%5 == 0;}
4  endclass
5
6  module jk;
7  abc m_h=new();
8
9  initial begin
10         repeat(15) begin
11               m_h.randomize();
12               $display("A%%5 = %0d",m_h.a);
13         end
14 end
15 endmodule
```

SAMPLE OUTPUT:

```
A%5 = 145
A%5 = 170
A%5 = 15
A%5 = 10
A%5 = 35
A%5 = 115
A%5 = 75
A%5 = 100
A%5 = 50
A%5 = 240
A%5 = 175
A%5 = 85
A%5 = 225
A%5 = 210
A%5 = 135
RNQUIE: Simulation is complete.
```

4. Derive odd numbers within the range of 10 to 30 using SV constraint.

```
 1 class abc;
 2   rand bit[7:0] a;
 3   constraint odd {a inside {[10:30]};
 4                   a%2 == 1;
 5                  }
 6 endclass
 7
 8 module jk;
 9 abc m_h=new();
10
11 initial begin
12         repeat(15) begin
13   m_h.randomize;
14         if(m_h.a%2 == 1)
15           $display("\t\t\tODD number = %d",m_h.a);
16         else
17           $display("%0d is not an odd number",m_h.a);
18 end
19 end
20 endmodule
```

SAMPLE OUTPUT:

```
                ODD number =  19
                ODD number =  23
                ODD number =  29
                ODD number =  23
                ODD number =  13
                ODD number =  13
                ODD number =  13
                ODD number =  25
                ODD number =  23
                ODD number =  11
                ODD number =  17
                ODD number =  25
                ODD number =  21
                ODD number =  29
                ODD number =  29
RNQUIE: Simulation is complete.
```

5. Write a constraint for 4-bit dynamic array. The size of the array should be in between 15 to 20. There should be even number in odd locations and odd number in even locations.

```
class abc;
  rand bit[3:0] a[];
       int i;
  constraint arr_size {a.size() inside {[15:20]};}

  constraint odd_pos { foreach (a[i]) {
                         if(i%2 == 0)
                                 a[i]%2 == 1;
                         else
                                 a[i]%2 == 0;
                         }
                      }

function void display();
  $display("\t\t\t*------size = %0d------*\n",a.size());
        foreach(a[i]) begin
                if(i%2 == 0)
                         $write("\tindex = %0d(Even location)   ",i);
                else
                         $write("\tindex = %0d(Odd location)   ",i);

                 if(a[i]%2 == 0)
                    $display(" value = %0d (EVEN Number)",a[i]);
                 else
                    $display(" value = %0d (ODD Number)",a[i]);
        end
endfunction

endclass

module jk;
abc m_h=new();

initial begin
        repeat(25) begin
                 m_h.randomize();
                 m_h.display();
        end
end
endmodule
```

SAMPLE OUTPUT:

```
                *------size = 20------*

 index = 0(Even location)    value = 15 (ODD Number)
 index = 1(Odd location)    value = 0 (EVEN Number)
 index = 2(Even location)    value = 7 (ODD Number)
 index = 3(Odd location)    value = 2 (EVEN Number)
 index = 4(Even location)    value = 7 (ODD Number)
 index = 5(Odd location)    value = 12 (EVEN Number)
 index = 6(Even location)    value = 11 (ODD Number)
 index = 7(Odd location)    value = 2 (EVEN Number)
 index = 8(Even location)    value = 9 (ODD Number)
 index = 9(Odd location)    value = 2 (EVEN Number)
 index = 10(Even location)    value = 5 (ODD Number)
 index = 11(Odd location)    value = 14 (EVEN Number)
 index = 12(Even location)    value = 13 (ODD Number)
 index = 13(Odd location)    value = 4 (EVEN Number)
 index = 14(Even location)    value = 13 (ODD Number)
 index = 15(Odd location)    value = 8 (EVEN Number)
 index = 16(Even location)    value = 13 (ODD Number)
 index = 17(Odd location)    value = 10 (EVEN Number)
 index = 18(Even location)    value = 11 (ODD Number)
 index = 19(Odd location)    value = 8 (EVEN Number)
*W,RNQUIE: Simulation is complete.
```

6. Write a constraint for two random variables such that one variable should not match with the other & the total number of bits toggled in one variable should be 5 w.r.t the other.

```
 1 class abc;
 2   rand bit[7:0] datain;
 3   rand bit[7:0] prev_data;
 4
 5   constraint d1 {datain != prev_data;}
 6   constraint ones {$countones(datain) == 5;}
 7
 8 endclass
 9
10 module jk;
11 abc m_h;
12
13 initial begin
14 m_h = new();
15         repeat(25) begin
16               m_h.randomize();
17             $display("\t\t\tPrevious_data = %0b", m_h.prev_data);
18             $display("\t\t\tdata          = %0b\n", m_h.datain);
19         end
20 end
21 endmodule
```

SAMPLE OUTPUT:

```
        Previous_data = 1011101
        data          = 1110011

        Previous_data = 11
        data          = 1101101

        Previous_data = 1000100
        data          = 10111010

        Previous_data = 11110
        data          = 111110

        Previous_data = 10001101
        data          = 10111001

        Previous_data = 1110110
        data          = 110111

        Previous_data = 1001110
        data          = 11010110

        Previous_data = 10011
        data          = 11010101
```

7. Write a constraint such that when rand bit[3:0] a is randomized, the value of “a” should not be same as 5 previous occurrences of the value of “a”.

```
 1 class abc;
 2   rand bit[3:0] a;
 3   int queue[$:7];
 4
 5   constraint c1 {!(a inside {queue});}
 6
 7 function void post_randomize();
 8         queue.push_front(a);
 9   $display("value of a = %0d \n",a);
10         if(queue.size == 6)
11                 queue.pop_back();
12   $display("PREV_OCCURENCES = %0p",queue);
13 endfunction
14 endclass
15
16 module jk;
17 abc m_h;
18 initial begin
19 m_h = new();
20         repeat(20) begin
21                 m_h.randomize();
22         end
23 end
24 endmodule
```

SAMPLE OUTPUT:

```
PREV_OCCURENCES = '{7, 11, 12, 4, 13}
value of a = 0

PREV_OCCURENCES = '{0, 7, 11, 12, 4}
value of a = 8

PREV_OCCURENCES = '{8, 0, 7, 11, 12}
value of a = 2

PREV_OCCURENCES = '{2, 8, 0, 7, 11}
value of a = 1

PREV_OCCURENCES = '{1, 2, 8, 0, 7}
value of a = 3

PREV_OCCURENCES = '{3, 1, 2, 8, 0}
value of a = 11

PREV_OCCURENCES = '{11, 3, 1, 2, 8}
value of a = 5

PREV_OCCURENCES = '{5, 11, 3, 1, 2}
value of a = 14
```

8. Constraint to generate 0, 1, x and z randomly.

```
 1 class rand_4_states;
 2   typedef enum {s0, s1, s2, s3} states;
 3
 4   rand states ss;
 5   logic a;
 6   rand bit ctrl;
 7 constraint ctrl_c {ctrl dist {0:=5, 1:=5};}
 8
 9 constraint c1 {
10               if(ctrl)
11                 ss inside {s0, s1};
12               else
13                 ss inside {s2, s3};
14               }
15
16 function void post_randomize();
17   case(ss)
18     s0: a =  0;
19     s1: a =  1;
20     s2: a = 'x;
21     s3: a = 'z;
22   endcase
23 endfunction
24
25 endclass
26

27 module tb;
28   rand_4_states c;
29   initial begin
30   c = new();
31     repeat(20) begin
32                  c.randomize();
33                  $display("\t\t\t State is %0d", c.a);
34         end
35
36   end
37 endmodule
```

SAMPLE OUTPUT:

```
                State is 1
                State is 0
                State is x
                State is z
                State is 1
                State is 1
                State is 0
                State is x
                State is 1
                State is x
                State is 1
                State is x
                State is z
                State is 0
                State is x
                State is z
                State is x
                State is 1
                State is x
                State is z
RNQUIE: Simulation is complete.
```

9. Write a program using dynamic array.
[i] array1: no. of elements should be between 30-40.
[ii] array2: sum of all elements should be < 100
[iii] array3: sum of all elements should be > 100

```
 1 class jk;
 2 rand int unsigned array1[];
 3 rand bit[3:0] array2[];
 4 rand int unsigned array3[];
 5
 6 constraint arr_size {array1.size() == 20;
 7                      array2.size() == 30;
 8                      array3.size() == 10;}
 9
10 constraint c1 {foreach(array1[i])
11                 array1[i] inside {[30:40]};}
12
13 function void display();
14         $display("SIZE Of: ARRAY1 = %0d, ARRAY2 = %0d, ARRAY3 = %0d\n",
15                             array1.size, array2.size, array3.size);
16         $display(" ELEMENTS OF THE ARRAYS ARE:");
17         $display(" ARRAY1: %0p \n", array1);
18         $display(" ARRAY2: %0p \n", array2);
19         $display(" ARRAY3: %0p \n", array3);
20         $display(" SUM of: ARRAY2 = %0d (<100), ARRAY3 = %d(>100)",
21                                         array2.sum, array3.sum);
22 endfunction
23 endclass
24
25 module dyn_arr_scenario;
26 jk c_h;
27
28 initial begin
29         c_h = new();
30
31 $display("\n\n");
32                 c_h.randomize() with{array2.sum < 100; array3.sum > 100;};
33                 c_h.display();
34 $display("\n\n");
35
36 end
37 endmodule
```

SAMPLE OUTPUT:

```
SIZE Of: ARRAY1 = 20, ARRAY2 = 30, ARRAY3 = 10

 ELEMENTS OF THE ARRAYS ARE:
 ARRAY1: '{38, 36, 39, 36, 36, 38, 35, 38, 33, 37, 34, 39, 33, 40, 40, 34, 38, 38, 31, 38}

 ARRAY2: '{'hf, 'hc, 'h1, 'h3, 'hf, 'hd, 'h2, 'h5, 'h7, 'h7, 'hb, 'hf, 'h0, 'h0, 'h2, 'hb, 'h6, 'hf,
, 'hd, 'h3, 'hb, 'ha, 'h3, 'h9, 'h5, 'h7, 'hb, 'h6, 'hc}

 ARRAY3: '{210399718, 253838307, 2484355680, 101771165, 2602896096, 1926113199, 1444724051, 647300115,
521252630, 324844528}

 SUM of: ARRAY2 = 15 (<100), ARRAY3 = 2927560897(>100)
```

10.There are two constraints applied to same variable. One will generate the value within the range of [25:50] and another expression should be greater than 40. What should be the value generated, and what is the reason?

```
1 class con;
2   rand bit [7:0] data;
3   constraint Const_1 {data inside {[25:50]};}
4   constraint Const_2 {data > 40;}
5 endclass
6
7 module jk;
8 con con_h;
9   initial begin
10    con_h = new();
11
12 $display("\n\n");
13     repeat(10) begin
14        con_h.randomize();
15        $display("\t\t data = %0d",con_h.data);
16
17      if(con_h.data < 40)
18           $display("\t\t data = %0d (under [25:50])",con_h.data);
19      end
20 $display("\n\n");
21
22  end
23 endmodule
```

SAMPLE OUTPUT:

```
data = 43
data = 46
data = 49
data = 43
data = 45
data = 50
data = 47
data = 47
data = 49
data = 42
```

REASON: 'Since the above two constraints are solved with the constraint solver, the solver will work on all the constraints in parallel way because constraints are **bidirectional** in nature. Thus, the constraint solver tried to satisfy both the above two constraints'.

11. Constraint with array size of 5 to 10 values & the array values should be in ascending order.

```
1 class jk;
2   rand bit[4:0] array[];
3
4   constraint c_asc { array.size inside {[5:10]};
5                      foreach (array[i])
6                         if(i>0)
7                           array[i] > array[i-1];
8                   }
9
10  function void display();
11     $display("\t\t\t size is '%0d'", array.size());
12             $display("\t\t Index  Value");
13         foreach(array[i]) begin
14             $display("\t\t [%0d]  |  %0d", i, array[i]);
15         end
16         $display("\t Ascending_order array = %p",array);
17 $display("\n\n");
18  endfunction
19
20 endclass
21
22 module ascending;
23 jk c_h;
24   initial begin
25     c_h = new();
26         repeat(5) begin
27           c_h.randomize();
28           c_h.display();
29         end
30   end
31 endmodule
```

SAMPLE OUTPUT:

```
                size is '5'
        Index  Value
        [0]  |  5
        [1]  |  8
        [2]  |  13
        [3]  |  28
        [4]  |  31
Ascending_order array = '{'h5, 'h8, 'hd, 'h1c, 'h1f}


                size is '10'
        Index  Value
        [0]  |  0
        [1]  |  2
        [2]  |  5
        [3]  |  7
        [4]  |  10
        [5]  |  21
        [6]  |  22
        [7]  |  25
        [8]  |  26
        [9]  |  27
Ascending_order array = '{'h0, 'h2, 'h5, 'h7, 'ha, 'h15, 'h16, 'h19, 'h1a, 'h1b}
```

12. Constraint with array size of 5 to 10 values & the array values should be in descending order.

```
1  class jk;
2    rand bit[5:0] array[];
3
4    constraint arr_size { array.size inside {[5:10]};}
5    constraint c_desc {foreach (array[i])
6                       if(i>0)
7                          array[i] < array[i-1];
8                    }
9
10  function void display();
11     $display("\t\t\t size is '%0d'", array.size());
12             $display("\t\t Index  Value");
13         foreach(array[i]) begin
14             $display("\t\t [%0d]  |  %0d", i, array[i]);
15         end
16         $display("\t Descending_order array = %p",array);
17 $display("\n");
18  endfunction
19
20 endclass
21
22 module ascending;
23 jk c_h;
24   initial begin
25     c_h = new();
26         repeat(5) begin
27           c_h.randomize();
28           c_h.display();
29         end
30   end
31 endmodule
```

SAMPLE OUTPUT:

```
                size is '5'
        Index  Value
        [0]  |  48
        [1]  |  33
        [2]  |  25
        [3]  |  5
        [4]  |  3
Descending_order array = '{'h30, 'h21, 'h19, 'h5, 'h3}


                size is '8'
        Index  Value
        [0]  |  60
        [1]  |  57
        [2]  |  49
        [3]  |  41
        [4]  |  28
        [5]  |  13
        [6]  |  8
        [7]  |  1
Descending_order array = '{'h3c, 'h39, 'h31, 'h29, 'h1c, 'hd, 'h8, 'h1}
```

13. Constraint - for 0-100 range 70% and for 101-255 range 30%

```
1  class abc;
2    rand bit[7:0] a;
3    constraint c_dist {a dist {[0:100] := 7, [101:255] := 3};}
4  endclass
5
6  module jk;
7  abc m_h=new();
8
9  initial begin
10         repeat(10) begin
11             m_h.randomize;
12         if(m_h.a < 100)
13             $display("\t\t\ta = %d (Under dist. [0:100])",m_h.a);
14         else
15             $display("\t\t\ta = %d (Under dist. [101:255])", m_h.a);
16         end
17 end
18 endmodule
```

SAMPLE OUTPUT:

```
a =   5 (Under dist. [0:100])
a = 153 (Under dist. [101:255])
a = 245 (Under dist. [101:255])
a =   6 (Under dist. [0:100])
a =  20 (Under dist. [0:100])
a = 158 (Under dist. [101:255])
a =  94 (Under dist. [0:100])
a =  34 (Under dist. [0:100])
a =  22 (Under dist. [0:100])
a =  43 (Under dist. [0:100])
```

14. Without inside operator generate random values for the range 34-43.

```
 1 class con;
 2
 3   rand logic [7:0] a;
 4
 5   constraint c_range {(a > 34) && (a < 43);}
 6 endclass
 7
 8
 9 module jk;
10 con con_h;
11   initial begin
12     con_h = new();
13
14   $display("\n Value of a (>34 and <43) is: ");
15     repeat(15) begin
16             con_h.randomize();
17             $display("\t\t a = %0d",con_h.a);
18      end
19   $display("\n\n");
20 end
21 endmodule
```

SAMPLE OUTPUT:

```
Value of a (>34 and <43) is:
                a = 39
                a = 42
                a = 35
                a = 41
                a = 37
                a = 36
                a = 39
                a = 41
                a = 41
                a = 40
                a = 37
                a = 37
                a = 40
                a = 40
                a = 39
```

15. Generate unique values without using rand or randc.

```
1  class con;
2
3    int data[15];
4
5    function new();
6          foreach (data[i]) begin
7                  data[i] = i;
8           end
9    endfunction
10
11   function void display();
12         $display("\n\n");
13         $display("\t\t Unique Values = %p",data);
14         $display("\n\n");
15   endfunction
16
17   function void data_shuffle();
18     data.shuffle();
19   endfunction
20 endclass
21
22 module jk;
23 con con_h;
24   initial begin
25     con_h = new();
26     con_h.data_shuffle();
27     con_h.display();
28   end
29 endmodule
```

SAMPLE OUTPUT:

```
Unique Values = '{4, 0, 12, 7, 2, 3, 11, 6, 10, 8, 14, 13, 5, 9, 1}
```

16. Randomize the below variables:

```
class randvar;
        rand bit[7:0] var1, var2, var3, var4;
endclass
```

i) Randomize all variables.
ii) Randomize only var2.
iii) Randomize var1 & var4.
iv) Randomize var1, var3 and var4.

```
1  class con;
2         rand bit [7:0] var1, var2, var3, var4;
3  endclass
4
5  module jk;
6  con con_h;
7    initial begin
8      con_h = new();
9
10 $display("\n\n");
11
12     $display("\t\t ----i) Randomizing all variable-----");
13     repeat(5) begin
14       con_h.randomize();
15        $display("\t\t var1 = %0d, var2 = %0d, var3 = %0d, var4 = %0d",
16                          con_h.var1,con_h.var2, con_h.var3, con_h.var4);
17     end
18     $display("\n");
19
20     $display("\t\t ----ii) Randomizing only var2 -----");
21     repeat(5) begin
22       con_h.randomize(var2);
23        $display("\t\t var1 = %0d, var2 = %0d, var3 = %0d, var4 = %0d",
24                          con_h.var1,con_h.var2, con_h.var3, con_h.var4);
25     end
26     $display("\n");
27
28     $display("\t\t ----iii) Randomizing var1, var4 -----");
29     repeat(5) begin
30        con_h.randomize(var1, var4);
31        $display("\t\t var1 = %0d, var2 = %0d, var3 = %0d, var4 = %0d",
32                         con_h.var1,con_h.var2, con_h.var3, con_h.var4);
33     end
34     $display("\n");
35
36     $display("\t\t ----iv) Randomizing var1,var3,var4 -----");
37     repeat(5) begin
38       con_h.randomize(var1, var3, var4);
39        $display("\t\t var1 = %0d, var2 = %0d, var3 = %0d, var4 = %0d",
40                         con_h.var1,con_h.var2, con_h.var3, con_h.var4);
41     end
42
43 $display("\n\n");
44
45   end
46 endmodule
```

SAMPLE OUTPUT:

```
----i) Randomizing all variable-----
var1 = 214, var2 = 104, var3 = 103, var4 = 172
var1 = 70, var2 = 116, var3 = 185, var4 = 10
var1 = 226, var2 = 10, var3 = 245, var4 = 230
var1 = 244, var2 = 12, var3 = 245, var4 = 197
var1 = 229, var2 = 29, var3 = 192, var4 = 195


----ii) Randomizing only var2 -----
var1 = 229, var2 = 126, var3 = 192, var4 = 195
var1 = 229, var2 = 46, var3 = 192, var4 = 195
var1 = 229, var2 = 37, var3 = 192, var4 = 195
var1 = 229, var2 = 211, var3 = 192, var4 = 195
var1 = 229, var2 = 220, var3 = 192, var4 = 195
```

```
----iii) Randomizing var1, var4 -----
var1 = 67, var2 = 220, var3 = 192, var4 = 98
var1 = 138, var2 = 220, var3 = 192, var4 = 194
var1 = 97, var2 = 220, var3 = 192, var4 = 102
var1 = 63, var2 = 220, var3 = 192, var4 = 30
var1 = 241, var2 = 220, var3 = 192, var4 = 208


----iv) Randomizing var1,var3,var4 -----
var1 = 99, var2 = 220, var3 = 16, var4 = 2
var1 = 180, var2 = 220, var3 = 80, var4 = 97
var1 = 26, var2 = 220, var3 = 253, var4 = 171
var1 = 108, var2 = 220, var3 = 173, var4 = 44
var1 = 101, var2 = 220, var3 = 115, var4 = 192
```

17. Write a single constraint to generate random values for bit[8:0] variable in the below range: 1-34, 127, 129-156, 192-202, 257-260.

```
class con;
 rand bit [8:0] data;
 constraint v {data inside {
                          [1:34],
                          127,
                          [129:156],
                          [192:202],
                          [257:260]
                          };}
function void display();
        if(data>=1 && data<=34)          $display("\t\t data = %0d (under [1:34])",data);
        else if(data>=129 && data<=156) $display("\t\t data = %0d (under [129:156])",data);
        else if(data>=192 && data<=202) $display("\t\t data = %0d (under [192:202])",data);
        else if(data>=257 && data<=260) $display("\t\t data = %0d (under [257:260])",data);
        else                             $display("\t\t data = %0d",data);
endfunction

endclass

module jk;
con con_h;
  initial begin
    con_h = new();

    repeat(20) begin
           con_h.randomize();
           con_h.display();
    end
 end
endmodule
```

SAMPLE OUTPUT:

```
        data = 132 (under [129:156])
        data = 131 (under [129:156])
        data = 146 (under [129:156])
        data = 144 (under [129:156])
        data = 33 (under [1:34])
        data = 260 (under [257:260])
        data = 136 (under [129:156])
        data = 257 (under [257:260])
        data = 17 (under [1:34])
        data = 28 (under [1:34])
        data = 130 (under [129:156])
        data = 15 (under [1:34])
        data = 32 (under [1:34])
        data = 133 (under [129:156])
        data = 21 (under [1:34])
        data = 195 (under [192:202])
        data = 34 (under [1:34])
        data = 259 (under [257:260])
        data = 192 (under [192:202])
        data = 149 (under [129:156])
RNQUIE: Simulation is complete.
```

18. Generate unique random values without using unique constraint.

```
 1 class con;
 2   bit[4:0] data[10];
 3
 4   function void pre_randomize();
 5     data.shuffle();
 6   endfunction
 7
 8   function void post_randomize();
 9         $display("\n\n");
10         $display("\t\t Unique values = %p",data);
11         $display("\n\n");
12   endfunction
13
14   function new();
15     for(int i = 0; i< 20; i++)
16       data[i] = i;
17   endfunction
18 endclass
19
20
21 module jk;
22 con con_h;
23   initial begin
24     con_h = new();
25     con_h.randomize();
26   end
27 endmodule
```

SAMPLE OUTPUT:

```
Unique values = '{'h4, 'h2, 'h7, 'h0, 'h5, 'h6, 'h9, 'h8, 'h3, 'h1}
```

19. What is wrong with the below code? What's the correct process to write the constraint?

```
class const;
        rand bit[7:0] low, mid, high;
        constraint const_1 {low<mid<high;}
endclass
```

Without Correction:

```
1 class con;
2  rand bit [7:0] low, mid, high;
3  constraint c1 {low < mid < high;}
4 endclass
5
6 module jk;
7 con con_h;
8   initial begin
9     con_h = new();
10
11 $display("\n\t For the given constraint without correction: ");
12     repeat(10) begin
13       con_h.randomize();
14       $display("\t\t low = %0d, mid = %0d, high = %0d",con_h.low, con_h.mid, con_h.high, );
15       if(con_h.high < con_h.mid)
16         $display("\t\t low = %0d, mid = %0d, high = %0d [HIGH < MID]",con_h.low, con_h.mid, con_h.high, );
17     end
18 $display("\n\n");
19
20   end
21 endmodule
```

SAMPLE OUTPUT:

```
For the given constraint without correction:
        low = 251, mid = 177, high = 200
        low = 209, mid = 28, high = 174
        low = 134, mid = 207, high = 56
        low = 134, mid = 207, high = 56 [HIGH < MID]
        low = 41, mid = 133, high = 103
        low = 41, mid = 133, high = 103 [HIGH < MID]
        low = 41, mid = 156, high = 35
        low = 41, mid = 156, high = 35 [HIGH < MID]
        low = 33, mid = 235, high = 9
        low = 33, mid = 235, high = 9 [HIGH < MID]
        low = 62, mid = 35, high = 117
        low = 86, mid = 237, high = 139
        low = 86, mid = 237, high = 139 [HIGH < MID]
        low = 229, mid = 119, high = 224
        low = 226, mid = 246, high = 187
        low = 226, mid = 246, high = 187 [HIGH < MID]
```

With Correction:

```
1 class con;
2  rand bit [7:0] low, mid, high;
3
4  constraint c1 {low < mid;
5                 mid < high;}
6 endclass
7
8 module jk;
9 con con_h;
10   initial begin
11     con_h = new();
12
13 $display("\n\t\t With Correction:");
14     repeat(10) begin
15       con_h.randomize();
16       $display("\t\t low = %0d, mid = %0d, high = %0d",con_h.low, con_h.mid, con_h.high, );
17       if(con_h.high < con_h.mid)
18         $display("\t\t low = %0d, mid = %0d, high = %0d [HIGH < MID]",con_h.low, con_h.mid, con_h.high, );
19     end
20 $display("\n\n");
21
22   end
23 endmodule
```

SAMPLE OUTPUT:

```
With Correction:
low = 169, mid = 182, high = 220
low = 54, mid = 74, high = 154
low = 65, mid = 185, high = 239
low = 182, mid = 223, high = 245
low = 17, mid = 19, high = 154
low = 172, mid = 190, high = 202
low = 206, mid = 223, high = 231
low = 23, mid = 100, high = 203
low = 153, mid = 186, high = 221
low = 117, mid = 175, high = 224
```

20. Write a constraint for 16-bit variable such that no two consecutive (continuous) ones should be generated.

```
 1 class jk;
 2 rand bit[15:0] value;
 3
 4 constraint c1 {foreach(value[i]) {
 5                     if(value[i] == 1 && i<15)
 6                          value[i+1] == 0;}}
 7 endclass
 8
 9 module no_consecutive_ones;
10 jk c_h;
11
12 initial begin
13         c_h = new();
14
15 $display("\n\n");
16         $display("\t\t    binary-format \t  corresponding decimal");
17         repeat(10) begin
18                 c_h.randomize();
19                 $display("\t\ta = %b \t->\t %d",c_h.value, c_h.value);
20         end
21 $display("\n\n");
22
23 end
24 endmodule
```

SAMPLE OUTPUT:

```
        binary-format          corresponding decimal
    a = 0010101001001010    ->        10826
    a = 1000000101000001    ->        33089
    a = 0010010001010010    ->         9298
    a = 1010001010010010    ->        41618
    a = 1010101000000101    ->        43525
    a = 0010000100000101    ->         8453
    a = 1010100010101000    ->        43176
    a = 0000000100010000    ->          272
    a = 1000100001000001    ->        34881
    a = 1001000100000001    ->        37121
```

21. Write a constraint using $countones.

```
1  class jk;
2    rand bit[15:0] data;
3    constraint ones {$countones(data);}
4  endclass
5
6  module jk;
7  jk c_h;
8      initial begin
9          c_h = new();
10
11       $display("\t\t\t\t binary format \t\tCorresponding decimal format");
12          repeat(20) begin
13            c_h.randomize();
14            $display("\t\t\t data = %b ->\t\t%0d", c_h.data, c_h.data);
15          end
16       $display("\n\n");
17      end
18 endmodule
```

SAMPLE OUTPUT:

```
                binary format            Corresponding decimal format
      data = 0110110000111011 ->                27707
      data = 1011000111111011 ->                45563
      data = 0111001011100110 ->                29414
      data = 0110010011011011 ->                25819
      data = 1011011100010111 ->                46871
      data = 0111101010010011 ->                31379
      data = 0101000000110010 ->                20530
      data = 1100111101101001 ->                53097
      data = 0011101110010110 ->                15254
      data = 1000011001010100 ->                34388
      data = 0011111000100101 ->                15909
      data = 0110110110010011 ->                28051
      data = 1001110001010100 ->                40020
      data = 0001110001010001 ->                7249
      data = 0001110011011100 ->                7388
      data = 1110101001010000 ->                59984
      data = 1111111100111111 ->                65343
      data = 0110110001101010 ->                27754
      data = 0111010010010101 ->                29845
      data = 0100010101101011 ->                17771
```

22. Generate 32-bit random number with only one bit set (should not use $countones).

```
1 class jk;
2 rand bit[31:0] number;
3 rand bit[4:0] shift;
4 constraint c1 {number == 1 << shift;}
5 endclass
6
7 module no_countones;
8 jk c_h;
9
10 initial begin
11         c_h = new();
12
13 $display("\n\n");
14         $display("\t\t\tONE BIT HIGH SET \t   CORRESPONDING DECIMAL FORMAT");
15         repeat(20) begin
16                 c_h.randomize();
17                 $display("\t number = %b ->\t %0d",c_h.number, c_h.number);
18         end
19 $display("\n\n");
20
21 end
22 endmodule
```

SAMPLE OUTPUT:

```
                ONE BIT HIGH SET           CORRESPONDING DECIMAL FORMAT
    number = 00000100000000000000000000000000 ->    67108864
    number = 00010000000000000000000000000000 ->    268435456
    number = 00000000000000000000000010000000 ->    128
    number = 00001000000000000000000000000000 ->    134217728
    number = 00000000000000000000000000001000 ->    8
    number = 01000000000000000000000000000000 ->    1073741824
    number = 00000000000000000000000000100000 ->    32
    number = 00000100000000000000000000000000 ->    67108864
    number = 00100000000000000000000000000000 ->    536870912
    number = 00000000000000100000000000000000 ->    131072
    number = 00000000000010000000000000000000 ->    524288
    number = 00000000100000000000000000000000 ->    4194304
    number = 00100000000000000000000000000000 ->    536870912
    number = 00001000000000000000000000000000 ->    134217728
    number = 00000000000000100000000000000000 ->    262144
    number = 00000000000000000000100000000000 ->    2048
    number = 00000000000010000000000000000000 ->    524288
    number = 00000000001000000000000000000000 ->    1048576
    number = 00000000100000000000000000000000 ->    8388608
    number = 00000000000000000000000000001000 ->    8
```

23. Having 16-bit of variable, only single bit high values need to be accessed. Write a constraint for that.

```
1  class jk;
2    rand bit[15:0] data;
3    constraint ones {$onehot(data);}
4  endclass
5
6  module jk;
7  jk c_h;
8      initial begin
9          c_h = new();
10
11       $display("\t\t\t\t binary format \t\tCorresponding decimal format");
12          repeat(20) begin
13            c_h.randomize();
14            $display("\t\t\t data = %b ->\t\t%0d", c_h.data, c_h.data);
15          end
16       $display("\n\n");
17      end
18 endmodule
```

SAMPLE OUTPUT:

```
                binary format            Corresponding decimal format
          data = 0000000001000000 ->              64
          data = 0000001000000000 ->              512
          data = 0000010000000000 ->              1024
          data = 0000000100000000 ->              256
          data = 0000000000001000 ->              8
          data = 0000100000000000 ->              2048
          data = 0000100000000000 ->              2048
          data = 0000000000000010 ->              2
          data = 0000000000000100 ->              4
          data = 0000000000010000 ->              16
          data = 0000100000000000 ->              2048
          data = 0000001000000000 ->              512
          data = 0000000010000000 ->              128
          data = 0000000100000000 ->              256
          data = 0000100000000000 ->              2048
          data = 0000000000001000 ->              8
          data = 0000000000010000 ->              16
          data = 0100000000000000 ->              16384
          data = 0000000000000010 ->              2
          data = 0000000100000000 ->              256
```

24. Write a constraint to generate random values for var1[7:0] within 50 and var2 [7:0] with the non-repeated value in every randomization.

```
1  class con;
2    rand logic [7:0] var1;
3    rand logic [7:0] var2;
4
5    constraint c1 {var1 inside {[0:50]};
6                   unique{var2};}
7  endclass
8
9  module jk;
10 con con_h;
11   initial begin
12     con_h = new();
13
14   $display("\n\n");
15     repeat(5) begin
16             con_h.randomize();
17             $display("\t\t var1 = %d (UNDER [0:50]) | var2 = %d",
18                                             con_h.var1, con_h.var2);
19
20      end
21   $display("\n\n");
22 end
23 endmodule
```

SAMPLE OUTPUT:

```
var1 =  35 (UNDER [0:50]) | var2 = 103
var1 =  10 (UNDER [0:50]) | var2 = 214
var1 =  38 (UNDER [0:50]) | var2 = 185
var1 =   6 (UNDER [0:50]) | var2 =  70
var1 =  31 (UNDER [0:50]) | var2 = 245
```

25. Write a constraint to randomly generate unique prime numbers in an array between 1 and 200. The generated prime numbers should have 7 in it (Eg.: 7, 17, 37..)

```
 1 class prime;
 2   rand int a[];
 3   rand int b[$];
 4   constraint c1 {a.size() == 200;}
 5   constraint c2 {foreach(a[i])
 6                    a[i] == prime(i);}
 7
 8   function integer prime(int val);
 9     if(val <= 1)
10         return 2;
11
12     for(int i=2; i<val; i++) begin
13       if(val%i == 0)
14           return 2;
15      end
16
17     prime = val;
18    endfunction
19
20   function void post_randomize();
21    for(int i=0; i<a.size(); i++) begin
22      if(a[i]%10 == 7)
23        b.push_back(a[i]);
24    end
25   endfunction
26
27 endclass
28
29 module jk;
30 prime c_h;
31   initial begin
32     c_h = new();
33         $display("\n\n");
34           c_h.randomize();
35           $display("\t Required Prime Nos. = %0p",c_h.b);
36         $display("\n\n");
37   end
38 endmodule
```

SAMPLE OUTPUT:

```
  Required Prime Nos. = '{7, 17, 37, 47, 67, 97, 107, 127, 137, 157, 167, 197}


*W,RNQUIE: Simulation is complete.
```

26. Write a constraint to generate multiples of power 2.

```
1  class power_of_2;
2    rand logic[7:0] a;
3    rand logic[4:0] power;
4    constraint d1 {a == 2**power;}
5  endclass
6
7  module jk;
8  power_of_2 c_h;
9
10 initial begin
11 c_h = new();
12
13     $display("\n\n");
14         repeat(20) begin
15             c_h.randomize();
16             $display("\t\t Power = %d -> a = %d (2^%0d)",
17                                 c_h.power, c_h.a, c_h.power);
18         end
19     $display("\n\n");
20 end
21 endmodule
```

SAMPLE OUTPUT:

```
Power =  4 -> a =  16 (2^4)
Power =  3 -> a =   8 (2^3)
Power =  7 -> a = 128 (2^7)
Power =  7 -> a = 128 (2^7)
Power =  2 -> a =   4 (2^2)
Power =  3 -> a =   8 (2^3)
Power =  6 -> a =  64 (2^6)
Power =  5 -> a =  32 (2^5)
Power =  5 -> a =  32 (2^5)
Power =  3 -> a =   8 (2^3)
Power =  6 -> a =  64 (2^6)
Power =  2 -> a =   4 (2^2)
Power =  6 -> a =  64 (2^6)
Power =  0 -> a =   1 (2^0)
Power =  3 -> a =   8 (2^3)
Power =  2 -> a =   4 (2^2)
Power =  0 -> a =   1 (2^0)
Power =  0 -> a =   1 (2^0)
Power =  7 -> a = 128 (2^7)
Power =  3 -> a =   8 (2^3)
```

Alternate Method:

```
1 class power_of_2;
2   rand logic[5:0] a;
3   constraint d1 {a != 0;
4                  (a & (a-1)) == 0;}
5 endclass
6
7 module jk;
8 power_of_2 c_h;
9
10 initial begin
11 c_h = new();
12
13     $display("\n\n");
14         $display("\t\t  value\t\t Corresponding binary format");
15         repeat(15) begin
16             c_h.randomize();
17             $display("\t\t a = %d  \t->\t %b",c_h.a, c_h.a);
18         end
19     $display("\n\n");
20 end
21 endmodule
```

SAMPLE OUTPUT:

```
     value          Corresponding binary format
    a = 16          ->       010000
    a = 32          ->       100000
    a =  8          ->       001000
    a =  8          ->       001000
    a = 16          ->       010000
    a =  4          ->       000100
    a =  4          ->       000100
    a =  8          ->       001000
    a =  1          ->       000001
    a = 16          ->       010000
    a =  8          ->       001000
    a = 16          ->       010000
    a =  2          ->       000010
    a =  4          ->       000100
    a =  4          ->       000100
```

THANK YOU 

- Augustin JK