1) **Pattern 1221221221221….**

```
class rakesh ;
  rand int data [9:0];
  constraint cons {foreach (data[i])
    if(i%3==1 || i%3==2)
      data[i] == 2;
                else
                  data[i] == 1;
                }
endclass

module ram();
  rakesh pkt;
  initial begin
    pkt=new();
    pkt.randomize();
    $display("data=%p",pkt.data);
  end
endmodule
//--------------output---------------
//  # KERNEL: data='{1, 2, 2, 1, 2, 2, 1, 2, 2, 1}
//------------------------------
```

Q2. Write a constraint for sorting elements in a dynamic array using constraints.

```
class rakesh ;
  rand int data [];
  rand int temp,i,j;
  constraint cons {data.size() == 10;
                   foreach (data[i])
                     data[i] inside {[1:100]};}
  function void post_randomize();
    $display(" before sorting is %p",data);
    foreach (data[i])
      begin
        for(j=i+1; j < $size(data) ; j++)
          begin
            if(data[i] > data[j])
              begin
                //data[i] = data[i] + data[j];
                //data[j] = data[i] - data[j];
                //data[i] = data[i] - data[j];
                temp = data[i];
```

```
            data[i]=data[j];
            data[j]=temp;
          end
        end
      end
  endfunction


endclass
module ram();
  rakesh pkt;
  initial begin
    pkt=new();
    pkt.randomize();
    $display("after sorting data=%p",pkt.data);
  end
endmodule

//--------------output---------------
# KERNEL:  before sorting is '{40, 75, 43, 37, 51, 95, 34, 42, 9, 6}
# KERNEL: after sorting data='{6, 9, 34, 37, 40, 42, 43, 51, 75, 95}
//------------------------------
```

Q3. Write a constraint to print unique elements in a 2D array without using the "unique" keyword

```
class rakesh ;
  rand bit[4:0] a [5][3];
  constraint cons {foreach (a[i])
    foreach (a[i][j])
      foreach (a[n])
        foreach (a[n][m])
          if (i != n && j != m)
            a[i][j] != a[n][m];
                }

endclass

module ram();
  rakesh pkt;
  initial begin
    pkt=new();
    pkt.randomize();
    $display("unique  data=%p",pkt.a);
  end
```

```
endmodule

//--------------output---------------
# KERNEL: unique  data='{'{18, 2, 31}, '{11, 17, 13}, '{7, 1, 0}, '{26, 6, 4}, '{11,
8, 0}}
//-----------------------------
```

Q4. Write a constraint for payload generation, where the size is between 11 and 22, and each
value is 2 greater than the previous.

```
class rakesh ;
  rand int a[];
  constraint  cons {a.size() inside {[11:22]};
                foreach (a[i])
                  if(i > 0)
                  a[i] == a[i-1]+2 ;
                  }
  constraint conss {foreach (a[i])
    a[i] inside {[1:100]};}
endclass
module ram();
  rakesh pkt;
  initial begin
    pkt=new();
    pkt.randomize();
    $display("payload  data=%p",pkt.a);
  end
endmodule

//--------------output---------------
# KERNEL: payload  data='{59, 61, 63, 65, 67, 69, 71, 73, 75, 77, 79, 81, 83, 85, 87,
89, 91, 93, 95, 97}
//-----------------------------
```

Q5. Write a code to simulate cyclic randomization behavior without using the "randc" keyword.

```
class rakesh;
  rand bit [2:0] a;
  int que[$];
  constraint cons {!(a inside {que});}
  function void post_randomize();
```

```
    que.push_back(a);
    if(que.size == 2**$size(a))
      //if(que.size==8)
        begin
          que={};
        end
  endfunction
endclass

module ram();
  rakesh pkt;
  initial begin
    pkt=new();
    repeat(8)begin
    pkt.randomize();
    $display("data=%d",pkt.a);
  end
end
endmodule

//--------------output---------------
# KERNEL: data=4
# KERNEL: data=2
# KERNEL: data=7
# KERNEL: data=0
# KERNEL: data=1
# KERNEL: data=5
# KERNEL: data=6
# KERNEL: data=3
//------------------------------
```

Q6. Write a constraint such that a 4-bit variable is not the same as the last five occurrences.

```
class rakesh;
  rand bit [3:0] a;
  int que[$];
  constraint cons {!(a inside {que});}
  function void post_randomize();
    que.push_front(a);
    if(que.size() ==6)
      que.pop_back();
  endfunction
endclass

module ram();
```

```
  rakesh pkt;
  initial begin
    pkt=new();
    repeat(8)begin
    pkt.randomize();
    $display("data=%d",pkt.a);
  end
end
endmodule

//--------------output---------------
# KERNEL: data=12
# KERNEL: data=10
# KERNEL: data=15
# KERNEL: data= 8
# KERNEL: data=14
# KERNEL: data= 7
# KERNEL: data= 5
# KERNEL: data=11
//-----------------------------
```

Q7. Write a constraint for two random variables such that one variable does not match the
other, and five bits are toggled.

```
class rakesh ;
  rand bit [9:0] var_1;
  rand bit [9:0] var_2;
  constraint cons {
                   var_1 != var_2;
                  $countones(var_1) == 5;
                  $countones(var_2) == 5;
                 }
 constraint toggle {foreach (var_1[i])
                    if(i>0 && var_1[i]==1)
                      var_1[i] != var_1[i-1];
                    foreach (var_2[j])
                    if(j>0 && var_2[j]==1)
                      var_2[j] != var_2[j-1];}



endclass

module ram();
```

```
  rakesh pkt;
  initial begin
    pkt=new;
    repeat(4)begin
      pkt.randomize();
      $display("(var1)%b != (var2)%b",pkt.var_1,pkt.var_2);
  end
end
Endmodule

//—————————————output———————
# KERNEL: (var1)1010101010 != (var2)0101010101
# KERNEL: (var1)0101010101 != (var2)1001010101
# KERNEL: (var1)1010100101 != (var2)0101010101
# KERNEL: (var1)0101010101 != (var2)1010101010
//————————————————————————————
```

Q8. Write a constraint such that the sum of any three consecutive elements in an array is even.

```
class rakesh;
  rand bit [7:0] even [10] ;
  constraint cons {foreach (even[i])
    if(i<=10)
      ((even[i]+even[i+1]+even[i+2]) % 2)== 0 ;

              }
endclass
module ram();
  rakesh pkt;
  initial begin
    pkt=new;
    repeat(4)begin
      pkt.randomize();
      $display("even = %p",pkt.even);
  end
end
endmodule


//————————————output————————————
# KERNEL: even = '{92, 128, 124, 76, 174, 126, 74, 204, 128, 90}
# KERNEL: even = '{22, 128, 200, 138, 254, 242, 178, 222, 196, 244}
# KERNEL: even = '{18, 178, 246, 248, 52, 98, 80, 160, 222, 248}
# KERNEL: even = '{96, 144, 144, 44, 102, 138, 10, 140, 132, 188}
//————————————————————————
```

Q9. Write a constraint for a variable such that the number of ones depends on the value of another variable.

```
class rakesh ;
  rand bit [7:0]a;
  rand int num_ones;
  constraint cons {num_ones inside {[0:7]};}
  constraint cons_1 {$countones(a) == num_ones;}
endclass

module ram();
  rakesh pkt;
  initial begin
    pkt=new();
    repeat(5)begin
    pkt.randomize();
      $display("a=%b,number_of ones=%0d",pkt.a,pkt.num_ones);
  end
end
endmodule
//—————————output—————————
# KERNEL: a=10000000,number_of ones=1
# KERNEL: a=01111100,number_of ones=5
# KERNEL: a=01011101,number_of ones=5
# KERNEL: a=01001010,number_of ones=3
# KERNEL: a=00000001,number_of ones=1
//------------------------------
```

Q10. Write a constraint on a 16-bit random vector to generate alternating pairs of 0's and 1's.

```
class rakesh ;
  rand bit [16:0] data;
  constraint ones {$countones(data) == 9;}
  constraint cons {foreach(data[i])
    if(data[i]>0 && data[i] == 1)
      data[i] != data[i-1];
              }
endclass

module ram();
  rakesh pkt;
  initial begin
    pkt=new();
    repeat(5)begin
    pkt.randomize();
      $display("data=%b",pkt.data);
```

```
  end
end
Endmodule
//—----------output—-------------
# KERNEL: data=10101010101010101
# KERNEL: data=10101010101010101
# KERNEL: data=10101010101010101
# KERNEL: data=10101010101010101
# KERNEL: data=10101010101010101
//—----------------------------------------
```

Q11. Write a constraint to randomly generate unique prime numbers in an array between 1 and 200, with 7 in the number.

```
class rakesh ;
  rand int a;
  constraint cons {a inside {[1:200]} && prime(a) == 1 && has_digit(a) == 1;}
  function int prime (int num);
    int i;
    if (num < 2) return 0;
    if(num == 2) return 1;
    for (i=3;i<=num/2;i++)begin
      if(num % i ==0)
        return 0;
      end
      return 1;
  endfunction
  function int has_digit (int num);
    int temp;
    temp =num;
    while (temp  != 0)begin
      if(temp % 10 == 7)
        return 1;
      temp = temp / 10;
    end
    return 0;
  endfunction
endclass

module ram();
  rakesh pkt;
  initial begin
    pkt=new();
    repeat(5)begin
      if(pkt.randomize())
        $display("prime_num with 7 =%d",pkt.a);
      else
        $display("randomization failed not a prime number = %d",pkt.a);
  end
```

```
end
endmodule
//-------------------------output-------------------------
 # KERNEL: prime_num with 7 =         47
# KERNEL: randomization failed not a prime number =         140
# KERNEL: randomization failed not a prime number =          98
# KERNEL: randomization failed not a prime number =         149
# KERNEL: randomization failed not a prime number =          82
//-------------------------------------------------------------
```

Q12. Write a constraint to generate a 32-bit number with exactly one bit high using $onehot().

```
 class rakesh ;
   rand bit [31:0] data;
   constraint cons {$onehot(data);}
 endclass
 module ram();
 rakesh pkt;
 initial begin
   pkt=new();
   repeat(5)begin
    pkt.randomize();
     $display("dat%b",pkt.data);
 end
end
endmodule
//—----------output—----------
# KERNEL: dat00000001000000000000000000000000
# KERNEL: dat00000000000010000000000000000000
# KERNEL: dat00000000000000000000000000000010
# KERNEL: dat00000000000000000001000000000000
# KERNEL: dat00000000000000010000000000000000
//—---------------------------------------
```

14. Write a constraint for a variable where the range 0–100 is 70% and 101–255 is 30%.

```
class rakesh;
 rand bit [7:0] data;
 constraint cons {data dist {[0:100]:=70,[101:255]:=30};}
endclass
module ram();
 rakesh pkt;
 initial begin
   pkt=new();
   repeat(5)begin
     pkt.randomize();
     $display("data=%p",pkt.data);

 end
end
Endmodule
```

```
//—----------output—----------
# KERNEL: data=248
# KERNEL: data=249
# KERNEL: data=49
# KERNEL: data=44
# KERNEL: data=33
//—---------------------------
```

Q15. Write a constraint so that the elements in two queues are different.

```
class rakesh;
  rand int a [$];
  rand int b [$];
  constraint cons {a.size inside {[1:20]};
  b.size inside {[1:20]};
  }
  constraint cons_a {foreach(a[i]) a[i] inside {[1:100]};
                     foreach (b[i]) b [i] inside {[1:100]};
                     foreach(b[i]) !(b[i] inside {a});
                    }
endclass

module ram();
  rakesh pkt;
  initial begin
    pkt=new();
    repeat(5)begin
      pkt.randomize();
      $display("a=%p,b=%p",pkt.a,pkt.b);
  end
end
Endmodule
//—----------output—----------
# KERNEL: a='{98, 96, 19, 98, 85, 19, 75, 11},b='{65, 1, 2, 1, 64, 93, 100, 31, 58,
99, 89, 50, 100, 52, 99, 33, 45, 100, 84}
# KERNEL: a='{16, 30, 4, 100, 73, 98, 99, 31, 73, 99, 47, 70},b='{29, 32, 39, 77, 1}
# KERNEL: a='{100, 12, 43, 50, 77, 74, 85},b='{6, 68, 32, 44, 99, 98, 72, 75, 40, 54,
98, 97, 5, 36}
# KERNEL: a='{79, 16, 71, 46, 56, 40, 69, 84, 98, 4, 28, 92, 66, 47, 11, 48},b='{26,
29, 37}
# KERNEL: a='{39, 46, 12, 53, 22, 97, 50, 48, 17},b='{38, 87, 99, 78, 14}
//—-----------------------------------
```

Q16. Write a code to check whether the randomized number is an Armstrong number.

```
class rakesh;
  rand int arm;
  constraint cons {arm inside {370,345,456,407,455,544};}
  function void post_randomize();
    int sum,temp,r;
    temp = arm;
    for(int i=0;i<3;i++)begin
      r = arm % 10;
    sum = (r ** 3)+sum;
    arm = arm / 10;
```

```
    end
    if(temp == sum)
      $display("armstrong number = %d",temp);
    else
      $display("not a armstrong number = %d",temp);
    endfunction
endclass

module ram();
  rakesh pkt;
  initial begin
    pkt=new();
    repeat(5)begin
      pkt.randomize();
  end
end
Endmodule
//—-------------output—-------------
# KERNEL: not a armstrong number =          544
# KERNEL: not a armstrong number =          455
# KERNEL: not a armstrong number =          456
# KERNEL: armstrong number =          370
# KERNEL: armstrong number =          407
//—-----------------------------------------
```

Q17. Write a constraint to generate the Fibonacci sequence.

```
class rakesh;
  rand int fibo [10];
  constraint cons {foreach (fibo[i])
    if(i==0)
      fibo[i] == 0;
            else if (i == 1)
              fibo[i] == 1;
            else
              fibo[i] == fibo[i-1] + fibo[i-2];
            }
endclass

module ram();
  rakesh pkt;
  initial begin
    pkt=new();
    repeat(1)begin
      pkt.randomize();
      $display("fibonaccic sequence = %p",pkt.fibo);
  end
end
endmodule

//—-------------output—--------------
# KERNEL: fibonaccic sequence = '{0, 1, 1, 2, 3, 5, 8, 13, 21, 34}
—-----------------------------------
```

Q18. Write a constraint to check whether the randomized number is a palindrome.

```
class rakesh ;
  rand int poli;
  constraint cons {poli inside {454,343,678,756,777};}
  function void post_randomize();
    int r,sum,temp;
    temp = poli;
    for (int i=0; i<3; i++)begin
      r = poli % 10;
      sum = (sum *10) + r;
      poli = poli / 10;
    end
    if(temp == sum)
      $display("its a polidrome = %d",temp);
     else
    $display("its not a polidrome = %d",temp);
  endfunction
endclass

module ram();
  rakesh pkt;
  initial begin
    pkt=new();
    repeat(5)begin
      pkt.randomize();
  end
end
endmodule

//—---------------------------------
# KERNEL: its a polidrome =         454
# KERNEL: its a polidrome =         343
# KERNEL: its not a polidrome =         756
# KERNEL: its not a polidrome =         678
# KERNEL: its not a polidrome =         756
//—-------------------------------
```

Q19. Write a constraint for a 2D dynamic array to print consecutive elements.

```
class rakesh;
  rand int  data[2:0] [6];
  constraint cons {foreach(data[i])
    foreach (data[i][j])
      data[i][j] == i+j;
              }
endclass

module ram();
  rakesh pkt;
  initial begin
    pkt=new();
    repeat(1)begin
      pkt.randomize();
```

```
    $display("data=%p",pkt.data);
  end
end
endmodule

//—-------------------------------
# KERNEL: data='{'{2, 3, 4, 5, 6, 7}, '{1, 2, 3, 4, 5, 6}, '{0, 1, 2, 3, 4, 5}}
//—--------------------
```

Q20. Write a code to generate unique elements in an array without using the "unique" keyword or constraints.

```
class rakesh;
  rand int data [10];
  constraint cons {foreach (data[i]) data[i] inside {[1:100]};
    foreach (data[i])
    foreach (data[j])
      if(i!=j)
        data[i] != data[j];
              }

endclass

module ram();
  rakesh pkt;
  initial begin
    pkt=new();
    repeat(3)begin
      pkt.randomize();
      $display("data=%p",pkt.data);
  end
end
endmodule

  //—-------------------------------------
# KERNEL: data='{96, 44, 100, 2, 45, 98, 99, 87, 21, 36}
# KERNEL: data='{41, 98, 16, 72, 19, 64, 3, 61, 15, 87}
# KERNEL: data='{83, 3, 48, 43, 67, 8, 61, 75, 72, 71}
//—-------------------------------------------------
```

Q21. Write a constraint demonstrating the use of the "solve before" constraint

```
class rakesh;
  rand bit [7:0] data;
  rand bit en;
  constraint cons {solve en before data;
              if(en==1)  data inside {[1:100]};
              else if (en==0)  data inside {[101:255]};
               }
endclass

module ram();
  rakesh pkt;
```

```
 initial begin
   pkt=new();
   repeat(3)begin
     pkt.randomize();
     $display("en=%d,data=%d",pkt.en,pkt.data);
  end
end
endmodule

//—---------------------------------------
# KERNEL: en=0,data=254
# KERNEL: en=1,data= 50
# KERNEL: en=1,data= 21
# KERNEL: en=1,data= 71
# KERNEL: en=1,data= 36
//—-------------------------
```

Q22. Write a constraint to generate a 10-bit variable with alternating values (e.g., 1010101010).

```
class rakesh;
  rand bit [9:0] data;
  constraint cons {$countones(data)==5;
    foreach(data[i])
    if(i>0 && data[i]==1)
      data[i] != data[i-1];
              }
endclass

module ram();
  rakesh pkt;
  initial begin
    pkt=new();
    repeat(5)begin
      pkt.randomize();
      $display("data=%b",pkt.data);
  end
end
endmodule
//—-----------------output—---------------------
# KERNEL: data=1010101010
# KERNEL: data=1001010101
# KERNEL: data=1001010101
# KERNEL: data=1010101001
# KERNEL: data=1010101010
//—--------------------------
```

Q23. Write a constraint to generate even numbers between 10 to 30 using a fixed-size array,
dynamic array, and queue.

```
class rakesh;
  rand int dynamic [];
  rand int fixed [10];
```

```
 rand int que [$];
 constraint cons_dyan {dynamic.size() == 10;
                       que.size() == 10;
                       foreach(dynamic[i]) dynamic[i] inside {[10:30]}  && dynamic[i] %2==0;
                       foreach(fixed[i]) fixed[i] inside {[10:30]}  && fixed[i] %2 ==0;
                       foreach(que[i]) que[i] inside {[10:30]} && que[i] %2 ==0;
                      }
endclass

module ram();
 rakesh pkt;
 initial begin
   pkt=new();
   repeat(1)begin
     pkt.randomize();
     $display("dynamic=%p,fixed=%p,que=%p",pkt.dynamic,pkt.fixed,pkt.que);
 end
end
endmodule

//—----------------------------output—----------
# KERNEL: dynamic='{16, 26, 20, 10, 22, 20, 28, 30, 16, 22},fixed='{20, 14, 10, 12,
12, 24, 22, 26, 10, 12},que='{16, 24, 30, 20, 24, 10, 22, 28, 10, 30}
//—---------------------------
```

Q24. Write a constraint such that the array size is between 5 to 10, and the values are in ascending order.

```
class rakesh;
 rand int ascending_order [];
 constraint cons {ascending_order.size() inside {[5:10]};
                  foreach(ascending_order[i]) ascending_order[i] inside {[1:100]};}
 constraint cons_1 {foreach(ascending_order[i])
   if(i>0)
     ascending_order[i] > ascending_order[i-1];
                  }
endclass

module ram();
 rakesh pkt;
 initial begin
   pkt=new();
   repeat(3)begin
     pkt.randomize();
     $display("ascending_order=%p",pkt.ascending_order);
 end
end
endmodule

//—----------output—-------------
# KERNEL: ascending_order='{1, 13, 15, 17, 19, 75, 77, 87, 90, 92}
# KERNEL: ascending_order='{62, 69, 78, 82, 95}
# KERNEL: ascending_order='{2, 3, 28, 60, 75}
//—-------------------------
```

Q25. Write a constraint to randomly generate 10 unique numbers between 99 and 100.

```
class rakesh ;
  rand int a;
  real b;
  constraint cons {a inside {[990:1000]};
                   unique {a};
                  }
  function void post_randomize();
   b=a/10.0;
    $display("real = %f",b);
  endfunction
endclass

module ram();
  rakesh pkt;
  initial begin
    pkt=new();
    repeat(5)begin
      pkt.randomize();
  end
end
endmodule

//—----------------output—----------------
# KERNEL: real = 99.200000
# KERNEL: real = 99.500000
# KERNEL: real = 99.800000
# KERNEL: real = 99.500000
# KERNEL: real = 99.400000
//—----------------------------------------
```

Q26. Write a constraint to generate consecutive and non-consecutive elements in a fixed-size

```
class rakesh;
  rand int con[10];
  rand int non_con[10];
  constraint cons{foreach (con[i])
    con[i] == i+1;
               }
  constraint non_cons {foreach(non_con[i])
    non_con[i] == i+i;}
endclass

module ram();
  rakesh pkt;
  initial begin
    pkt=new();
    repeat(1)begin
      pkt.randomize();
      $display("con = %p",pkt.con);
      $display("non_con = %p",pkt.non_con);
```

```
  end
end
endmodule

//—---------------------------------------output—----------
# KERNEL: con = '{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
# KERNEL: non_con = '{0, 2, 4, 6, 8, 10, 12, 14, 16, 18}
//—---------------------------------------
```

Q27. Write a constraint to generate a variable with 0–31 bits as 1 and 32–61 bits as 0.

```
class rakesh ;
  rand bit [61:0] data;
  constraint cons {foreach (data[i])
    if (i<=31)
      data[i]==1;
      else
        data[i] == 0;
                }
 endclass

module ram();
  rakesh pkt;
  initial begin
    pkt=new();
    repeat(1)begin
      pkt.randomize();
      $display("data=%b",pkt.data);
  end
end
endmodule
//—---------------------------------------output—----------
# KERNEL: data=00000000000000000000000000000011111111111111111111111111111111
//—---------------------------------------------------------
```

Q28. Write a constraint to generate prime numbers between the range of 1 to 100.

```
class rakesh;
  rand int a ;
  constraint cons {a inside {[1:100]};
                   prime(a) == 1;
                  }
  function int prime (int pr);
    int i;
    if(pr<2) return 0;
    if(pr==2) return 1;
    for(int i=3;i<= pr/2;i++)begin
      if(pr%i == 0)
        return 0;
    end
        return 1;
  endfunction
endclass

module ram();
```

```
 rakesh pkt;
 initial begin
   pkt=new();
   repeat(10)begin
    if(pkt.randomize())
    $display("prime num=%d",pkt.a);
    else
     $display("randomization failure %d is not a prime number",pkt.a);
 end
end
endmodule
//—----------------output—---------------
# KERNEL: prime num=          43
# KERNEL: prime num=          37
# RCKERNEL: Warning: RC_0024 testbench.sv(389): Randomization failed. The condition of
randomize call cannot be satisfied.
# RCKERNEL: Info: RC_0103 testbench.sv(389): ... the following condition cannot be
met: (1==(pkt.prime(pkt.a)=0))
# RCKERNEL: Info: RC_1007 testbench.sv(367): ... see class 'rakesh' declaration.
# KERNEL: randomization failure          42 is not a prime number
# RCKERNEL: Warning: RC_0024 testbench.sv(389): Randomization failed. The condition of
randomize call cannot be satisfied.
# RCKERNEL: Info: RC_0103 testbench.sv(389): ... the following condition cannot be
met: (1==(pkt.prime(pkt.a)=0))
# RCKERNEL: Info: RC_1007 testbench.sv(367): ... see class 'rakesh' declaration.
# KERNEL: randomization failure          50 is not a prime number
# KERNEL: prime num=          37
# KERNEL: prime num=          41
# KERNEL: prime num=          41
# KERNEL: prime num=          73
# KERNEL: prime num=          31
//—----------------------------------------
```

Q30. Write a constraint to generate an array with unique values and multiples of 3

```
class rakesh ;
 rand bit [7:0] data;
 constraint cons {unique{data};
              data %3 == 0;
            }
 endclass

module ram();
 rakesh pkt;
 initial begin
   pkt=new();
   repeat(5)begin
     pkt.randomize();
     $display("data=%d",pkt.data);
 end
end
endmodule
```

```
//—---------------------------------output—----------------
# KERNEL: data=120
# KERNEL: data= 90
# KERNEL: data=243
# KERNEL: data=156
# KERNEL: data= 48
//—---------------------------------------------------
```

Q31. Write a constraint on a two-dimensional array to generate even numbers in the first 4 locations and odd numbers in the next 4 locations.

```
class rakesh;
  rand int a [2][4];
  constraint cons_1 {foreach (a[i])
    foreach (a[i][j])
      a[i][j] inside {[1:100]};
    }
  constraint cons {foreach (a[i])
    foreach (a[i][j])
    if(i==0)
      a[i][j] %2==0;
                else
                  a[i][j] %2==1;
                }
endclass

module ram();
  rakesh pkt;
  initial begin
    pkt=new();
    repeat(2)begin
      pkt.randomize();
      $display("data=%p",pkt.a);
  end
end
endmodule

//—---------------------------------output—----------
# KERNEL: data='{'{48, 26, 58, 38}, '{23, 39, 95, 11}}
# KERNEL: data='{'{40, 44, 4, 10}, '{15, 75, 13, 37}}
//—---------------------------------------------
```

Q32. Write a program to randomize a 32-bit variable but only randomize the 12th bit.

```
class rakesh;
  rand bit [31:0] data;
  constraint cons {foreach(data[i])
    if(i==12)
      data[i] inside {[1:0]};
else
  data[i] == 0;
                }
```

```
endclass
module ram();
  rakesh pkt;
  initial begin
    pkt=new();
    repeat(10)begin
      pkt.randomize();
      $display("data=%b",pkt.data);
  end
end
endmodule

//—--output———————————
# KERNEL: data=00000000000000000000000000000000
# KERNEL: data=00000000000000000000000000000000
# KERNEL: data=00000000000000000000000000000000
# KERNEL: data=00000000000000000000000000000000
# KERNEL: data=00000000000000000001000000000000
# KERNEL: data=00000000000000000001000000000000
# KERNEL: data=00000000000000000001000000000000
# KERNEL: data=00000000000000000000000000000000
# KERNEL: data=00000000000000000000000000000000
# KERNEL: data=00000000000000000001000000000000
//—————————————————
```

Q33. Write a constraint such that even locations contain odd numbers and odd locations contain even numbers.

```
class rakesh;
  rand int a [15];
  constraint cons{foreach (a[i])
    a[i] inside {[1:100]};
                foreach(a[i])
                if(i%2==0)
                  a[i] %2==1;
                else
                 a[i] % 2==0;}
 endclass

module ram();
  rakesh pkt;
  initial begin
    pkt=new();
    repeat(1)begin
      pkt.randomize();
      $display("data=%p",pkt.a);
  end
end
endmodule

//—————-output———————————
# KERNEL: data='{97, 94, 81, 30, 73, 60, 95, 36, 41, 44, 91, 18, 15, 68, 61}
//—————————————————
```

Q34. Write a constraint to generate the factorial of the first 5 even and odd numbers.

```
 class rakesh;
  rand int a[5]; // First 5 odd numbers
  rand int b[5]; // First 5 even numbers
  int fact_a[5]; // Factorials of a[]
  int fact_b[5]; // Factorials of b[]

  // Constraint for generating first 5 odd and even numbers
  constraint cons {
    foreach (a[i]) a[i] == (i * 2) + 1; // 1, 3, 5, 7, 9
    foreach (b[i]) b[i] == (i + 1) * 2; // 2, 4, 6, 8, 10
  }

  // Post-randomize: Compute factorials
  function void post_randomize();
    foreach (fact_a[i]) fact_a[i] = compute_factorial(a[i]);
    foreach (fact_b[i]) fact_b[i] = compute_factorial(b[i]);
  endfunction

  // Factorial function
  function int compute_factorial(int n);
    int result = 1;
    for (int i = 2; i <= n; i++)
      result = result* i;
    return result;
  endfunction
endclass

module ram();
  rakesh pkt;
  initial begin
    pkt = new();
    if (pkt.randomize()) begin
      $display("Odd numbers (a[])     = %p", pkt.a);
      $display("Factorials of a[]     = %p", pkt.fact_a);
      $display("Even numbers (b[])    = %p", pkt.b);
      $display("Factorials of b[]     = %p", pkt.fact_b);
    end else begin
      $display("Randomization failed.");
    end
  end
endmodule
//----------------------output----------------------------//
# KERNEL: Odd numbers (a[])     = '{1, 3, 5, 7, 9}
# KERNEL: Factorials of a[]     = '{1, 6, 120, 5040, 362880}
# KERNEL: Even numbers (b[])    = '{2, 4, 6, 8, 10}
# KERNEL: Factorials of b[]     = '{2, 24, 720, 40320, 3628800}
//--------------------------------------------------------------//
```

Q35. Write a constraint for a 32-bit random variable to have 12 number of 1's non-consecutively.

```
class rakesh;
  rand bit [31:0] a;
  constraint cons {$countones(a)==12;}
  constraint cons_1 {foreach (a[i])
    if(i>0 && a[i] == 1)
      a[i] != a[i-1];
                }
endclass

module ram();
  rakesh pkt;
  initial begin
    pkt=new();
    repeat(5)begin
      pkt.randomize();
      $display("data=%b",pkt.a);
      $display("count ones = %p",$countones(pkt.a));
  end
end
endmodule
//—----------------------output—----------------
# KERNEL: data=10001010100101010010001010101000
# KERNEL: count ones = 12
# KERNEL: data=10100010101000010100101010001001
# KERNEL: count ones = 12
# KERNEL: data=10010001010001001001010101010010
# KERNEL: count ones = 12
# KERNEL: data=00100100101010101010100010100001
# KERNEL: count ones = 12
# KERNEL: data=10100101001010100100101001000010
# KERNEL: count ones = 12
//—----------------------------------------------
```

Q37. Write a constraint to generate random values 25, 27, 30, 36, 40, 45 without using "set membership

```
class rakesh;
  rand int a;
  constraint cons { a>24 ; a<46;
                (a%5==0) || (a%9==0); a!=35 ;
               }
endclass

module ram();
  rakesh pkt;
  initial begin
    pkt=new();
    repeat(6)begin
      pkt.randomize();
      $display("data=%d",pkt.a);
```

```
  end
end
endmodule
//—-------------output—--------------------
# KERNEL: data=          40
# KERNEL: data=          27
# KERNEL: data=          36
# KERNEL: data=          30
# KERNEL: data=          36
# KERNEL: data=          27
//—-----------------------------------
```

Q38. Write a constraint to generate the pattern 0102030405.

```
class rakesh;
  rand int a [10];
  constraint cons {foreach(a[i])
    if(i%2==0)
      a[i]==0;
               else
                 a[i] == (i+2)/2;
               }
endclass

module ram();
  rakesh pkt;
  initial begin
    pkt=new();
    repeat(1)begin
      pkt.randomize();
      $display("data=%p",pkt.a);
  end
end
endmodule

//—-------------------output—--------------------
# KERNEL: data='{0, 1, 0, 2, 0, 3, 0, 4, 0, 5}
//—-----------------------------------------
```

Q39. Write a code to generate a random number between 1.35 and 2.57

```
class rakesh;
  rand int a;
  real b;
  constraint cons {a inside {[1350:2570]};}
  function void post_randomize();
    b=a/1000.0;
    $display("b=%0f",b);
  endfunction
endclass

module ram();
  rakesh pkt;
  initial begin
```

```
    pkt=new();
    repeat(10)begin
      pkt.randomize();
  end
end
endmodule

//—---------------output—--------------------
# KERNEL: b=2.552000
# KERNEL: b=2.547000
# KERNEL: b=2.533000
# KERNEL: b=2.104000
# KERNEL: b=1.869000
# KERNEL: b=2.130000
# KERNEL: b=2.181000
# KERNEL: b=1.759000
# KERNEL: b=1.67300
//—-----------------------------------------
```

Q40. Write a constraint to generate the pattern 1122334455.

```
class rakesh;
  rand int  a [10];
  constraint cons {foreach(a[i])
    a[i] == (i+2)/2;
               }
endclass

module ram();
  rakesh pkt;
  initial begin
    pkt=new();
    repeat(1)begin
      pkt.randomize();
      $display("a=P",pkt.a);
  end
end
endmodule
//—----------------------output—-------------
# KERNEL: a=P'{1, 1, 2, 2, 3, 3, 4, 4, 5, 5}
//—-------------------------------------------------
```

Q41. Write a constraint to generate the pattern 5 -10 15 -20 25 -30.

```
class rakesh;
  rand int a [];
  constraint connn {a.size==8;}
  constraint cons {foreach(a[i])
    if(i%2==0)
      a[i] == 5+(i*5) ;
                else
                 a[i] == -5*(i+1);
               }
endclass
```

```
module ram();
  rakesh pkt;
  initial begin
    pkt=new();
    repeat(1)begin
      pkt.randomize();
      $display("a=%P",pkt.a);
  end
end
endmodule
```

Q42. Write a constraint to generate the pattern 9 19 29 39 49 59 69 79.

```
class rakesh ;
  rand int a[10];
  constraint cons {foreach(a[i])
    a[i] == (i*10) + 9;
              }
endclass

module ram();
  rakesh pkt;
  initial begin
    pkt=new();
    repeat(1)begin
      pkt.randomize();
      $display("a=%P",pkt.a);
  end
end
endmodule
//—-------------------output—-----------
# KERNEL: a='{9, 19, 29, 39, 49, 59, 69, 79, 89, 99}
//—-------------------------------
```

Q43. Write a constraint to generate the pattern 1234554321.

```
class rakesh;
  rand int a[10];
  constraint cons {foreach (a[i])
    if(i<5)
      a[i] == i+1;
              else
                a[i] == a[9-i];
              }
  function void post_randomize();
    $display("a=%P",a);
  endfunction
endclass

  module ram();
  rakesh pkt;
  initial begin
```

```
    pkt=new();
    repeat(1)begin
      pkt.randomize();

  end
end
endmodule
//—---------------output—-------------------
# KERNEL: a='{1, 2, 3, 4, 5, 5, 4, 3, 2, 1}
//—---------------------------------------
```

Q44. Write a constraint to generate the pattern 0101010101.

```
  class rakesh;
  rand int a[10];
  constraint cons {foreach (a[i])
    if(i%2==0)
      a[i] == 0;
                else
                  a[i] == 1;
                }
  function void post_randomize();
    $display("a=%P",a);
  endfunction
endclass

  module ram();
  rakesh pkt;
  initial begin
    pkt=new();
    repeat(1)begin
      pkt.randomize();

  end
end
endmodule
//—---------------------output—---------------
# KERNEL: a='{0, 1, 0, 1, 0, 1, 0, 1, 0, 1}
//—---------------------------------------
```

Q45. Write a constraint to generate a 64-bit number where only the lower and upper 8 bits are /random and the rest are zeros.

```
class rakesh ;
  rand bit [63:0] data;
  constraint cons {foreach (data[i])
    if (i <=7)
      data[i] inside {[1:0]};
                else if (i >= 55)
                  data[i] inside {[1:0]};
                else
                  data[i] == 0;
                  }
```

```
endclass

module ram();
  rakesh pkt;
  initial begin
    pkt=new();
    repeat(5)begin
      pkt.randomize();
      $display("data=%b",pkt.data);
  end
end
endmodule

  //—----------output—----------
# KERNEL: data=11000000100000000000000000000000000000000000000000000000000101110
# KERNEL: data=10010001000000000000000000000000000000000000000000000000011000000
# KERNEL: data=11101110000000000000000000000000000000000000000000000000000111110
# KERNEL: data=11101111000000000000000000000000000000000000000000000000000100110
# KERNEL: data=01000111000000000000000000000000000000000000000000000000011010111
//—-----------------------------------
```

Q47. Write a constraint for a dynamic array where every element is a multiple of its index (starting from index 1).

```
class rakesh ;
  rand int a [];
  constraint cons {a.size() == 10;
                   foreach (a[i])
                     a[i] inside {[1:100]};}
  constraint conss {foreach (a[i])
    if(i>=1)
        a[i] % i ==0;
                   }
endclass

module ram();
  rakesh pkt;
  initial begin
    pkt=new();
    repeat(1)begin
      pkt.randomize();
      $display("a=%P",pkt.a);
  end
end
endmodule

//—---------output—-------------
# KERNEL: a='{40, 75, 58, 15, 56, 35, 36, 42, 40, 18}
//—----------------------------
```

Q48. Write a constraint for generating a Gray code sequence of 5 bits.

```
class rakesh ;
```

```
  rand bit [4:0] gray [];
  constraint cons {gray.size == 32;}
  constraint conss {foreach (gray[i])
    if (i> 0)
      $countones(gray[i] ^ gray[i-1]) ==1;
                }
endclass

module ram();
  rakesh pkt;
  initial begin
    pkt = new();
    if (pkt.randomize()) begin
      $display("Gray Code Sequence:");
      foreach(pkt.gray[i])
        $display("gray[%0d] = %05b", i, pkt.gray[i]);
    end else
      $display("Randomization failed!");
  end
endmodule
//—----------------output—-----------------\
   KERNEL: gray[0] = 11110
# KERNEL: gray[1] = 10110
# KERNEL: gray[2] = 11110
# KERNEL: gray[3] = 11111
# KERNEL: gray[4] = 11101
# KERNEL: gray[5] = 11111
# KERNEL: gray[6] = 11101
# KERNEL: gray[7] = 10101
# KERNEL: gray[8] = 00101
# KERNEL: gray[9] = 00001
# KERNEL: gray[10] = 00011
# KERNEL: gray[11] = 00111
# KERNEL: gray[12] = 00101
# KERNEL: gray[13] = 00100
# KERNEL: gray[14] = 00101
# KERNEL: gray[15] = 10101
# KERNEL: gray[16] = 10111
# KERNEL: gray[17] = 10011
# KERNEL: gray[18] = 10010
# KERNEL: gray[19] = 11010
# KERNEL: gray[20] = 10010
# KERNEL: gray[21] = 10000
# KERNEL: gray[22] = 10010
# KERNEL: gray[23] = 11010
# KERNEL: gray[24] = 11110
# KERNEL: gray[25] = 11111
# KERNEL: gray[26] = 10111
# KERNEL: gray[27] = 00111
# KERNEL: gray[28] = 00011
# KERNEL: gray[29] = 10011
# KERNEL: gray[30] = 10001
# KERNEL: gray[31] = 10000
//—---------------------------------------
```

Q49. Write a constraint such that all elements in an array are powers of 2 and sorted in descending order.

```
class rakesh;
  rand int a[10];

  constraint cons {foreach (a[i])
    $onehot(a[i]);
                 foreach (a[i])
                   a[i] inside {[1:1000]};
                 unique{a};
                }
  constraint conss {foreach (a[i])
    if (i>0)
      a[i] <=  a[i-1];}
endclass

module ram();
  rakesh pkt;
  initial begin
    pkt=new();
    repeat(2)begin
      pkt.randomize();
      $display("a=%P",pkt.a);
  end
end
endmodule
//—------------------------output—------------------
# KERNEL: a='{512, 256, 128, 64, 32, 16, 8, 4, 2, 1}
# KERNEL: a='{512, 256, 128, 64, 32, 16, 8, 4, 2, 1}
//—-----------------------------------------------------
```

Q50. Write a constraint for a 32-bit variable such that the number of trailing zeros is between
5 and 10.

```
class rakesh ;
  rand bit [31:0] data;
  constraint cons {foreach (data[i])
    if(i>5 && i<10)
      data[i] inside {[1:0]};
                 else
                   data[i] == 0;
                }
endclass


module ram();
  rakesh pkt;
  initial begin
    pkt=new();
    repeat(10)begin
      pkt.randomize();
```

```
    $display("a=%b",pkt.data);
  end
end
endmodule

//—----------------------output—-------------------------
# KERNEL: a=00000000000000000000001110000000
# KERNEL: a=00000000000000000000000000000000
# KERNEL: a=00000000000000000000001110000000
# KERNEL: a=00000000000000000000000110000000
# KERNEL: a=00000000000000000000000111000000
# KERNEL: a=00000000000000000000001111000000
# KERNEL: a=00000000000000000000000101000000
# KERNEL: a=00000000000000000000000110000000
# KERNEL: a=00000000000000000000000000000000
# KERNEL: a=00000000000000000000001101000000
//—-------------------------------------------------------------
```