SYSTEM VERILOG

CONSTRAINTS – Part 3

Write a constraint to generate prime numbers between 1 to 100.

```
class constraint_21;
	int prime_num[$];
	rand int da[];
	constraint c1{da.size == 100;}
	constraint c2{foreach(da[i])
				da[i] == prime_fun(i); }

	function int prime_fun(input int i);
		if(i>1 && (! ((i%2 == 0 && i!=2) || (i%3==0 && 
i!=3) || (i%5==0 && i!=5) || (i%7==0 && i!=7))))
			return i;
		else
			return 0;
	endfunction

	function void post_randomize();
		foreach(da[i])
			begin
				if(da[i] != 0)
					begin
						prime_num.push_back(da[i]);
					end
			end
	endfunction
endclass
```

```
constraint_21 c1;

module test();
        initial
                begin
                        c1=new;
                        assert(c1.randomize());
                        $display("da: %p",c1.da);
                end
endmodule
```

Generate ascending and descending order using constraints.

```
class constraint_22;
        rand int a[];
        rand int b[];
        constraint c1{a.size == 10; b.size == 10;}
        constraint c2{foreach(a[i])
                                a[i] inside {[1:50]};
                        foreach(b[i])
                                b[i] inside {[1:50]};}
        constraint ascending_order{foreach(a[i])
                                        if(i>0)
                                                a[i] > a[i-1]; }
        constraint descending_oredr{foreach(b[i])
                                        if(i>0)
                                                b[i] < b[i-1]; }
endclass
```

```
constraint_22 c1;

module test();
	initial
		begin
			c1=new;
			assert(c1.randomize());
			$display("ASCENDING ORDER: %p",c1.a);
			$display("DESCENDING ORDER: %p",c1.b);
		end
endmodule
```

Write a constraint to declare an MXN array where the sum of all row elements equal to the last element.

```
class constraint_23;
	rand int arr[5][6];
	int i;
	constraint c1{foreach(arr[i])
			foreach(arr[i][j])
				arr[i][j] inside {[1:100]};}
	constraint c2{foreach(arr[i])
			foreach(arr[i][j])
				arr[i][5] == arr[i][0] + arr[i][1] +
arr[i][2] + arr[i][3] + arr[i][4]; }

endclass
```

```
constraint_23 c1;

module test();
	initial
		begin
			c1=new;
			assert(c1.randomize());
			$display("arr: %p",c1.arr);
		end
endmodule
```

Write a constraint that randomize an array of 10 numbers such that the sum is always divisible by 5.

```
class constraint_24;
	rand int da[];
	constraint c1{da.size == 10;}
	constraint c2{foreach(da[i])
				da[i] inside {[0:30]};}
	constraint c3{(da.sum() % 5) == 0;}
endclass

constraint_24 c1;

module test();
	initial
		begin
			repeat(3)
				begin
					c1=new;
					assert(c1.randomize());
					$display("da: %p",c1.da);
				end
```

```
        end
endmodule
```

Write a class to randomize an array of size 6 such that the sum of the array is always 30.

```
class constraint_25;
    rand int da[];
    constraint c1{da.size == 6;}
    constraint c2{foreach(da[i])
                    da[i] inside {[0:15]};}
    constraint c3{da.sum() == 30;}
endclass

constraint_25 c1;

module test();
    initial
        begin
            repeat(3)
                begin
                    c1=new;
                    assert(c1.randomize());
                    $display("da: %p",c1.da);
                end
        end
endmodule
```

Randomize a 6-element array such that the difference between each adjacent element is exactly 3.

```
class constraint_26;
	rand int da[];
	rand int a;
	constraint c1{da.size == 6;}
	constraint c2{a inside {[20:30]};}
	constraint c3{foreach(da[i])
				if(i==0)
					da[i] == a;
				else
					da[i] == da[i-1] - 3; }
endclass

constraint_26 c1;

module test();
	initial
		begin
			repeat(3)
				begin
					c1=new;
					assert(c1.randomize());
					$display("da: %p",c1.da);
				end
		end
endmodule
```

Randomize an array of size 6 with values from 1 to 9 such that no value repeats and the sum is divisible by 3.

```
class constraint_27;
	rand int da[];
	constraint c1{da.size == 6;}
	constraint c2{foreach(da[i])
				da[i] inside {[1:9]};}
	constraint c3{unique {da};}
	constraint c4{(da.sum() % 3) == 0;}
endclass

constraint_27 c1;

module test();
	initial
		begin
			repeat(3)
				begin
					c1=new;
					assert(c1.randomize());
					$display("da: %p",c1.da);
				end
		end
endmodule
```

Randomize a variable ‘count’ such that it is divisible by both 3 and 5, but not 2 and lies between 30 to 150.

```
class constraint_28;
	rand int count;
	constraint c1{(count%3 == 0) && (count%5 == 0) &&
(count%2 != 0);}
```

```
    constraint c2{count inside {[30:150]};}
endclass

constraint_28 c1;

module test();
    initial
        begin
            repeat(5)
                begin
                    c1=new;
                    assert(c1.randomize());
                    $display("count=%d",c1.count);
                end
        end
endmodule
```

Generate a constraint to randomize a binary string of length 10 such that it starts and ends with 1.

```
class constraint_29;
     rand int da[];
    constraint c1{da.size == 10;}

    //method 1
    constraint c2{foreach(da[i])
                        if(i==0 || i==9)
                            da[i] == 1;
                        else
                            da[i] == 0; }
```

```
    //method 2
    constraint c2{foreach(da[i])
                        if(i%9 == 0)
                            da[i] == 1;
                        else
                            da[i] == 0; }
endclass

constraint_29 c1;

module test();
    initial
        begin
            c1=new;
            assert(c1.randomize());
            $display("da: %p",c1.da);
        end
endmodule
```

Write a class with 'rand' members and write inline constraint to limit values between 100 to 200 that are divisible by 4 but not by 8.

```
class constraint_30;
    rand bit [7:0] a;
    constraint c2{(a%4==0) && (a%8!=0);}
endclass

constraint_30 c1;
```

```
module test();
	initial
		begin
			repeat(5)
				begin
					c1=new;
					assert(c1.randomize() with {a>100
&& a<200;});
					$display("a: %d",c1.a);
				end
		end
endmodule
```