ASSIGNMENT 2

MELVIN RIJOHN T

10/12/2024

//1. Write a code to genrate random number between 135 and 257

class task1;

    rand bit[8:0] num;

    constraint con1 { num inside {[135 : 257]}; }

endclass

module task1\_tb;

        task1 t1;

    initial begin

        t1 = new();

        repeat(5) begin

            t1.randomize();

            $display("Number: %0d", t1.num);

        end

    end

endmodule

// OUTPUT

// # KERNEL: Number: 212

// # KERNEL: Number: 156

// # KERNEL: Number: 232

// # KERNEL: Number: 170

// # KERNEL: Number: 250

//2. write a constraint to generate a random even and od number between 20 and 100

class task2;

    rand bit[6:0] odd;

    rand bit[6:0] even;

    constraint con1 {

        odd >= 20 && odd <= 100 && odd % 2 == 1;

        even >= 20 && even <= 100 && even % 2 == 0;

    }

endclass

module task1\_tb;

        task2 t2;

    initial begin

        t2 = new();

        repeat(10) begin

            t2.randomize();

            $display("ODD: %0d", t2.odd);

            $display("EVEN: %0d", t2.even);

            $display("=======================");

        end

    end

endmodule

// OUTPUT

// # KERNEL: ODD: 35

// # KERNEL: EVEN: 46

// # KERNEL: =======================

// # KERNEL: ODD: 75

// # KERNEL: EVEN: 88

// # KERNEL: =======================

// # KERNEL: ODD: 93

// # KERNEL: EVEN: 36

// # KERNEL: =======================

// # KERNEL: ODD: 43

// # KERNEL: EVEN: 100

// # KERNEL: =======================

// # KERNEL: ODD: 89

// # KERNEL: EVEN: 82

// # KERNEL: =======================

//3. write a constraint such that even location contains odd number and odd location consits of even numbers

class task2;

    rand bit[6:0] odd;

    rand bit[6:0] even;

    constraint con1 {

        odd >= 20 && odd <= 100 && odd % 2 == 0;

        even >= 20 && even <= 100 && even % 2 == 1;

    }

endclass //0001

module task1\_tb;

        task2 t2;

    initial begin

        t2 = new();

        repeat(5) begin

            t2.randomize();

            $display("ODD: %0d", t2.odd);

            $display("EVEN: %0d", t2.even);

            $display("=======================");

        end

    end

endmodule

//OUTPUT

// # KERNEL: ODD: 40

// # KERNEL: EVEN: 51

// # KERNEL: =======================

// # KERNEL: ODD: 80

// # KERNEL: EVEN: 95

// # KERNEL: =======================

// # KERNEL: ODD: 78

// # KERNEL: EVEN: 79

// # KERNEL: =======================

// # KERNEL: ODD: 44

// # KERNEL: EVEN: 27

// # KERNEL: =======================

// # KERNEL: ODD: 26

// # KERNEL: EVEN: 31

// # KERNEL: =======================

//4. Write a sv program which contains a 32 bit rand variable which should have 16 bit postionsof 1 in non consecutive

class task4;

    rand bit [31:0] val;

    constraint c\_val {

        $countones(val) == 16;

        foreach (val[i]) {

            if (i < 31) val[i] + val[i+1] <= 1;

        }

    }

endclass

module task4\_tb;

    task4 r;

    initial begin

        r = new();

        repeat (5) begin

            r.randomize();

            $display("val = %b \t0x%0h", r.val,r.val);

        end

    end

endmodule

// OUTPUT

// # KERNEL: val = 10101010101010101010100101010101     0xaaaaa955

// # KERNEL: val = 10101010100101010101010101010101     0xaa955555

// # KERNEL: val = 10101010101010100101010101010101     0xaaaa5555

// # KERNEL: val = 10101010101010010101010101010101     0xaaa95555

// # KERNEL: val = 10101010101010100101010101010101     0xaaaa5555

//5. write a constrant to genrate factrial of first 10 numbers

class task5;

    randc int fact;

    constraint con1 {

       fact >= 0 && fact < 10;

    }

    function int factorial(int num);

        if (num == 0)

            return 1;

        else

            return num \* factorial(num - 1);

    endfunction

endclass

module task4\_tb;

    int result;

    task5 r;

    initial begin

        r = new();

        repeat(10) begin

            r.randomize();

            result = r.factorial(r.fact);

            $display("factorial of %0d is %0d", r.fact, result);

        end

    end

endmodule

// OUTPUT

// # KERNEL: factorial of 7 is 5040

// # KERNEL: factorial of 0 is 1

// # KERNEL: factorial of 5 is 120

// # KERNEL: factorial of 6 is 720

// # KERNEL: factorial of 3 is 6

// # KERNEL: factorial of 2 is 2

// # KERNEL: factorial of 9 is 362880

// # KERNEL: factorial of 1 is 1

// # KERNEL: factorial of 4 is 24

// # KERNEL: factorial of 8 is 40320

//6. write a constrant to genrate factrial of first 5 even numbers

class task6;

    randc int fact;

    constraint con1 {

       fact >= 0 && fact < 5 && fact % 2 == 0;

    }

    function int factorial(int num);

        if (num == 0)

            return 1;

        else

            return num \* factorial(num - 1);

    endfunction

endclass

module task6\_tb;

    int result;

    task6 r;

    initial begin

        r = new();

        repeat(5) begin

            r.randomize();

            result = r.factorial(r.fact);

            $display("factorial of %0d is %0d", r.fact, result);

        end

    end

endmodule

// OUTPUT

// # KERNEL: factorial of 2 is 2

// # KERNEL: factorial of 0 is 1

// # KERNEL: factorial of 4 is 24

// # KERNEL: factorial of 0 is 1

// # KERNEL: factorial of 2 is 2

//7. write a constrant to genrate factrial of first 5 odd numbers

class task7;

    randc int fact;

    constraint con1 {

       fact >= 0 && fact < 5 && fact % 2 == 1;

    }

    function int factorial(int num);

        if (num == 0)

            return 1;

        else

            return num \* factorial(num - 1);

    endfunction

endclass

module task7\_tb;

    int result;

    task7 r;

    initial begin

        r = new();

        repeat(5) begin

            r.randomize();

            result = r.factorial(r.fact);

            $display("factorial of %0d is %0d", r.fact, result);

        end

    end

endmodule

// OUTPUT

// # KERNEL: factorial of 3 is 6

// # KERNEL: factorial of 1 is 1

// # KERNEL: factorial of 1 is 1

// # KERNEL: factorial of 3 is 6

// # KERNEL: factorial of 1 is 1

//8. write a sv program to randomize 32bit variable but only randomize the 20th bit

class task8;

    randc bit pos;

    int num = 32'hffffffff;

    function void post\_randomize;

      num[20] = pos;

    endfunction

endclass

module task8\_tb;

    int result;

    task8 r;

    initial begin

        r = new();

        repeat(3) begin

            r.randomize();

            $display("number: %0b \t 0x%0h", r.num, r.num);

        end

    end

endmodule

// OUTPUT

// # KERNEL: number: 11111111111011111111111111111111    0xffefffff

// # KERNEL: number: 11111111111111111111111111111111    0xffffffff

// # KERNEL: number: 11111111111011111111111111111111    0xffefffff

//9. write a constraint such that sum of any 3 conceutive elements should be an even number

class task9;

  randc bit[4:0] num[5];

    constraint con1 {

       foreach (num[i]) {

        (num[i] + num[i + 1] + num[i + 2]) % 2 == 0;

       }

    }

endclass

module task9\_tb;

    int result;

    task9 r;

    initial begin

        r = new();

        repeat(5) begin

            r.randomize();

            $display("%p", r.num);

        end

    end

endmodule

// OUTPUT

// # KERNEL: '{18, 18, 14, 12, 26}

// # KERNEL: '{2, 26, 30, 16, 14}

// # KERNEL: '{28, 10, 6, 8, 30}

// # KERNEL: '{30, 4, 18, 26, 6}

// # KERNEL: '{10, 2, 22, 28, 28}

**//10. write a constraint ona 16bit number to generate alternate pair of zeros and ones**

class task10;

    rand bit [15:0] val;

    constraint c\_val {

        foreach (val[i]) {

          if (i % 2 == 0) {

                (val[i +: 2] == 2'b11 && val[i+2 +: 2] == 2'b00) || (val[i +: 2] == 2'b00 && val[i+2 +: 2] == 2'b11);

            }

        }

    }

endclass

module task10\_tb;

    task10 r;

    initial begin

        r = new();

      repeat (3) begin

            r.randomize();

            $display("val = %b \t0x%0h", r.val,r.val);

        end

    end

endmodule

// OUTPUT

// # KERNEL: val = 1100110011001100     0xcccc

// # KERNEL: val = 1100110011001100     0xcccc

// # KERNEL: val = 1100110011001100     0xcccc

//11. generate the below pattern

// 1

// 11

// 111

// 1111

// 11111

// 111111

// 1111111

// 11111111

// 111111111

class PatternGen;

    rand int num\_rows;

    function new();

        num\_rows = 9;

    endfunction

    constraint con1t {

        num\_rows inside {1, 2, 3, 4};

    }

    function void generate\_pyramid();

        int i;

        string row\_pattern;

        for (i = 1; i <= num\_rows; i++) begin

            row\_pattern = {i{"1"}};

            $display("%s", row\_pattern);

        end

    endfunction

  endclass

module pattern\_generator;

    initial begin

        PatternGen gen;

        gen = new();

        gen.generate\_pyramid();

    end

endmodule

// # KERNEL: 1

// # KERNEL: 11

// # KERNEL: 111

// # KERNEL: 1111

// # KERNEL: 11111

// # KERNEL: 111111

// # KERNEL: 1111111

// # KERNEL: 11111111

// # KERNEL: 111111111

//12. generate the below pattern

//123456789

//12345678

//1234567

//123456

//12345

//1234

//123

//12

//1

class PatternGen;

    rand int num\_rows;

    function new();

        num\_rows = 9;

    endfunction

    constraint con1t {

        num\_rows inside {[1:9]};

    }

    function void generate\_pattern();

        int i, j;

        string row\_pattern;

        for (i = num\_rows; i > 0; i--) begin

            row\_pattern = "";

            for (j = 1; j <= i; j++) begin

              row\_pattern = {row\_pattern, $sformatf("%0d", j)};

            end

            $display("%s", row\_pattern);

        end

    endfunction

  endclass

module pattern\_generator;

    initial begin

        PatternGen gen;

        gen = new();

        gen.generate\_pattern();

    end

endmodule

//OUTPUT

// # KERNEL: 123456789

// # KERNEL: 12345678

// # KERNEL: 1234567

// # KERNEL: 123456

// # KERNEL: 12345

// # KERNEL: 1234

// # KERNEL: 123

// # KERNEL: 12

// # KERNEL: 1

//13. generate the below pattern

//1

//12

//123

//1234

//12345

//123456

//1234567

//12345678

//123456789

class PatternGen;

    rand int num\_rows;

    function new();

        num\_rows = 9;

    endfunction

    constraint con1t {

        num\_rows inside {[1:9]};

    }

    function void generate\_pattern();

        int i, j;

        string row\_pattern;

        for (i = 1; i <= num\_rows; i++) begin

            row\_pattern = "";

            for (j = 1; j <= i; j++) begin

              row\_pattern = {row\_pattern, $sformatf("%0d", j)};

            end

            $display("%s", row\_pattern);

        end

    endfunction

  endclass

module pattern\_generator;

    initial begin

        PatternGen gen;

        gen = new();

        gen.generate\_pattern();

    end

endmodule

//OUTPUT

// # KERNEL: 1

// # KERNEL: 12

// # KERNEL: 123

// # KERNEL: 1234

// # KERNEL: 12345

// # KERNEL: 123456

// # KERNEL: 1234567

// # KERNEL: 12345678

// # KERNEL: 123456789

//15. Write a constraints for genrating numbers in ascending order

class task15;

    rand int num;

    int prev\_num = 0;

    constraint ascending\_c {

      num >= prev\_num + 1;

      num <= 100;

    }

    task generate\_num();

        if (this.randomize()) begin

            $display("%0d", num);

            prev\_num = num;

        end else begin

            $fatal("Randomization failed!");

        end

    endtask

endclass

module task15\_tb;

    task15 asc\_gen = new();

    initial begin

        for (int i = 0; i < 10; i++) begin

            asc\_gen.generate\_num();

        end

        $finish;

    end

endmodule

// OUTPUT

// # KERNEL: 77

// # KERNEL: 80

// # KERNEL: 83

// # KERNEL: 91

// # KERNEL: 97

// # KERNEL: 100