Mux9to1v

Create a 16-bit wide, 9-to-1 multiplexer. sel=0 chooses a, sel=1 chooses b, etc. For the unused cases (sel=9 to 15), set all output bits to '1'.

```
module top_module(
    input [15:0] a, b, c, d, e, f, g, h, i,
    input [3:0] sel,
    output [15:0] out );

    always@(*) begin
        case(sel)
            4'd0:out=a;
            4'd1:out=b;
            4'd2:out=c;
            4'd3:out=d;
            4'd4:out=e;
            4'd5:out=f;
            4'd6:out=g;
            4'd7:out=h;
            4'd8:out=i;
            default:out = 16'b1111_1111_1111_1111;
        endcase
    end
endmodule
```

Mux256to1

Create a 1-bit wide, 256-to-1 multiplexer. The 256 inputs are all packed into a single 256-bit input vector. sel=0 should select `in[0]`, sel=1 selects bits `in[1]`, sel=2 selects bits `in[2]`, etc.

- With this many options, a case statement isn't so useful.
- Vector indices can be variable, as long as the synthesizer can figure out that the width of the bits being selected is constant. In particular, selecting one bit out of a vector using a variable index will work.

```
module top_module(
    input [255:0] in,
    input [7:0] sel,
    output out );

    assign out = in[sel];

endmodule
```

Mux256to1v

Create a 4-bit wide, 256-to-1 multiplexer. The 256 4-bit inputs are all packed into a single 1024-bit input vector. sel=0 should select bits `in[3:0]`, sel=1 selects bits `in[7:4]`, sel=2 selects bits `in[11:8]`, etc.

Expected solution length: *Around 1–5 lines.*

- With this many options, a case statement isn't so useful.
- Vector indices can be variable, as long as the synthesizer can figure out that the width of the bits being selected is constant. It's not always good at this. An error saying "... is not a constant" means it couldn't prove that the select width is constant. In particular, `in[ sel*4+3 : sel*4 ]` does not work.
- Bit slicing ("Indexed vector part select", since Verilog-2001) has an even more compact syntax.

```
module top_module(
    input [1023:0] in,
    input [7:0] sel,
    output [3:0] out );

    assign out = {in[sel*4+3],in[sel*4+2],in[sel*4+1],in[sel*4]};

endmodule
```