Problem 1

Rewrite the following bluespec function using a loop.

```bluespec
function Bit#(16) reverse16(Bit#(16) in);
    Bit#(16) rev;
    rev[0] = in[15];
    rev[1] = in[14];
    rev[2] = in[13];
    rev[3] = in[12];
    rev[5] = in[10];
    rev[6] = in[9];
    rev[7] = in[8];
    rev[8] = in[7];
    rev[9] = in[6];
    rev[10] = in[5];
    rev[12] = in[3];
    rev[14] = in[1];
    rev[15] = in[0];
    return rev;
endfunction
```

The new function should be able to reverse any sized input. Its function specification should look like:

```bluespec
function Bit#(w) reverseN(Bit#(w) in);
```

```bluespec
function Bit#(w) reverseN(Bit#(w) in);
    Bit#(w) ret;
    for (Integer i=0; i< valueOf(w); i=i+1) begin
        ret[i] = in[valueOf(w)-1-i];
    end
    return ret;
endfunction
```
**Problem 2:**

Identify whether each of the follow code snippets contain valid BSV syntax. If not recommend how they can be fixed.

Assume the following `addN` function exists.

```bsv
defunction Bit#(w) addN(Bit#(w) a, Bit#(w) b, Bit#(1) c_in);
```

a) ```bsv
function Bit#(64) Add64 (Bit#(64) a, Bit#(64) b);
    Bit#(64) Ret = addN(a, b, 0);
    return Ret;
endfunction
```

**Answer:** Not valid. Function/variable names cannot begin with a capital letter.

b) ```bsv
function Bit#(64) add64 (Bit#(64) a, Bit#(32) b);
    return addN(a, b, 0);
endfunction
```

**Answer:** Not valid. No variation of addN can take a 64 bit ‘a’ and a 32-bit ‘b’.

**Solution:**
```bsv
return addN(a, signExtend(b), 0)
```

c) ```bsv
function UInt#(64) add64 (UInt#(64) a, UInt#(64) b);
    return addN(a, b, 0);
endfunction
```

**Answer:** Not valid, UInt64 has to be converted into Bits before being fed into add64, and the output bits has to be unpacked into a UInt.

```bsv
return unpack(addN(pack(a), pack(b)));
```

d) ```bsv
function Bool isZero (Bit#(8) a);
    return (a==0) ? 1: 0;
endfunction
```

**Answer:** Not valid. Literals ‘1’ and ‘0’ are of type Bit#(1). Several fixes are possible

```bsv
return (a==0);
return (a==0? True: False);
return unpack(a==0? 1: 0);
```
// Performs function (in >> shiftBy)
function Bit#(32) shift ( Bit#(32) in, Bit#(5) shiftBy);
    Bit#(32) ret = in;
    for (Integer i=0; i<shiftBy; i=i+1) begin
        ret = {1'b0, ret[31:1]};
    end
    return ret;
endfunction

Answer: Not valid. For loop bounds must be compile time constants.

f) typedef enum {Red, Green, Blue} Color deriving {Bits, Eq};
function Color getColorFromBits (Bit#(2) in);
    return unpack(in);
endfunction

Answer: This is valid. Although for (a == 2’b11), the output is undefined. This will not be valid if ‘a’ is anything but 2 bits wide.

Problem 3:

Parameterize the bit-scan-reverse function from Lab1 to take as input a w-bit vector and output the index of the first non-zero bit scanned from the largest index.

function Bit#(TLog#(w)) bitScanReverse (Bit#(w) in);  
    Bit#(TLog#(w)) ret = 0;
    for (Integer i=0; i<valueOf(w); i=i+1) begin
        ret = (in[i] == 1) ? fromInteger(i) : ret;
    end
    return ret;
endfunction

Note: TLog#(w) returns a type of ceiling(log2(w))

Problem 4:

Write a BSV function to compute the parity of an n-bit vector. The propagation delay of your circuit should be O(log n). You can assume that n is a power of 2. The parity of a vector is 1 if the vector has an odd number of ‘1’s and 0 otherwise.
function Bit#(1) parity ( Bit#(n) in);
  Bit#(n) out = in;
  // Creates a reduction tree.
  // Each iteration of the outer loop represents a single
  // tree level, while each iteration of the inner loop
  // represents a single xor gate within a level
  for (Integer s=valueOf(n)/2; s>0; s = s / 2) begin
    for (Integer i =0 ; i< s; i = i+1) begin
      out[i] = out[2*i] ^ out[2*i + 1];
    end
  end
  return out[0];
endfunction