```
function error_locs = chien_search(lambda, GF, prnt_flag)
%{
CHIEN_SEARCH Completes the chien search algorithm on the error locator
 polynomial to find the roots of the lambda function in GF(2^m) (the
 values
 of x where lambda = 0 (inf in GF(2^m)) Finds the error locations with
 the
 inverse of these roots
INPUTS:
    lambda - the power form error locator polynomial array whose
 elements
    represent the powers of the alpha coefficients of the polynomial.
    highest degree coeff in lowest index

    GF - Array of cells mapping the binary groups to GF(2^m) alpha
 coeff
OUTPUTS:
    error_locs - a list representing the inverse of roots of the
 lambda polynomial.
    Numbers are powers of alpha. These are the error locations in the
    recieved codeword
%}

%see if need to set default value of prnt_flag
if ~exist('prnt_flag','var')
    prnt_flag = false;
end

deg_lambda = size(lambda, 2) - 1;

%create a roots list, number of roots corresponds to the degree of
 lambda
%polynomial
roots = zeros(1,deg_lambda);
roots(1:end) = -1;
error_locs = roots;

field_len = size(GF,1); %number of elements in the GF(2^m)
root_cnt = 0; %number of non-zero roots found

if(prnt_flag)
    fprintf(" - This part attempts to find the roots of lambda(x)\n");
    fprintf(" - The inverse of the roots of lambda(x) correspond to
\n");
    fprintf("   the power of the x-term that denotes the location of
\n");
    fprintf("   of the symbol error in the received codeword.\n");
    fprintf("  1.) Loop through non-zero powers of the primitive root
 alpha\n");
    fprintf("    and use them to evaluate the error locator
 polynomial.\n");
```

```
    fprintf("    If lambda(a^?) = 0, that power of alpha is a root.
\n");
end

%the indices of GF correlate to the powers of alpha that map to the
 binary
%number stored at that index location. The mapping is a^i-2 = GF{i}
%start at 2 because were looking for non 0 roots
for gf_idx = 2:field_len

    %power of alpha is gf_idx-2 which represents the x_input to the
 lambda
    %function to evaluate at each possible root
    root = gf_idx - 2;
    eval = EvalPolyGF2(lambda, root, GF);
    if(eval == -1)
        root_cnt = root_cnt + 1;
        if(root_cnt > deg_lambda)
            error("Too many roots found for the lambda polynomial of
 degree %d!\n", deg_lambda);
            return;
        end
        roots(root_cnt) = root;
        if(prnt_flag)
            fprintf("    %d Root(s) found! lambda(a^%d) = 0\n",
 root_cnt, root);
        end
    end
end

if(root_cnt ~= deg_lambda)
    error("The number of roots found (%d) is less the degree of
 lambda(x) (%d)!\n", root_cnt, deg_lambda);
end

%get the inverse of the roots, these are the locations of the errors
 in the
%received codeword
if(prnt_flag)
    fprintf("  2.) There were %d roots of lambda(x) found without any
 errors.\n", root_cnt);
    fprintf("    Therefore, taking the inverse of the roots gives us
\n");
    fprintf("    the error locations in R(x) shown below:\n");
end

for i = 1:root_cnt
    error_locs(i) = DivGF2(0,roots(i), GF);
    if(prnt_flag)
        fprintf("    - error %d: 1/a^%d = a^%d. The symbol coefficient
\n", i, roots(i), error_locs(i));
        fprintf("      at x^%d in R(x) has an error\n",
 error_locs(i));
    end
```

```
end
```

```
end
```

*Published with MATLAB® R2018b*