```
MODULE MLS;


(*
   Title: Using maximum length sequences (MLS) for impulse response measurements
   LastEdit: 16th November 2006
   Author: Jens Hee, Denmark
   Programming Language: Originally C - translated to Component Pascal
   References: http://jenshee.dk/signalprocessing/mls.pdf:
      Impulse response measurements using MLS by Jens Hee
*)


IMPORT Out;  (* Out is only imported to output the calculated data to the standard output *)


PROCEDURE GenerateSignal

   (mls: POINTER TO ARRAY OF BYTE; signal: POINTER TO ARRAY OF REAL; p: INTEGER);

VAR

   i: INTEGER;
   input: POINTER TO ARRAY OF REAL;

BEGIN

   NEW (input, p);

   FOR i := 0 TO p - 1 DO input [i] := -2 * mls [i] + 1 END;   (* Change 0 to 1 and 1 to -1 *)

   FOR i := 0 TO p - 1 DO  (* Simulate a system with h = {2, 0.4, 0.2, -0.1, -0.8}, just an example *)

      signal [i] :=

         2.0 * input [(p + i) MOD p] +
         0.4 * input [(p + i - 1) MOD p] +
         0.2 * input [(p + i - 2) MOD p] -
         0.1 * input [(p + i - 3) MOD p] -
         0.8 * input [(p + i - 4) MOD p];

   END;

END GenerateSignal;
```

```
PROCEDURE GenerateMls (VAR mls: POINTER TO ARRAY OF BYTE; p, n: INTEGER);

(* Generate the Maximum length sequence *)

CONST

   MaxNoTaps = 18;

VAR

   i, j, sum: INTEGER;
   tapsTab: POINTER TO ARRAY OF ARRAY OF BYTE;
   taps, delayLine: POINTER TO ARRAY OF BYTE;

BEGIN

   NEW (tapsTab, 16, MaxNoTaps);
   FOR i := 0 TO LEN (tapsTab) - 1 DO FOR j := 0 TO LEN (tapsTab [0]) - 1 DO tapsTab [i, j] := 0 END END;

   tapsTab [0, 10] := 1; tapsTab [0, 17] := 1;
   tapsTab [1, 13] := 1; tapsTab [1, 16] := 1;
   tapsTab [2, 3] := 1; tapsTab [2, 12] := 1; tapsTab [2, 14] := 1; tapsTab [2, 15] := 1;
   tapsTab [3, 13] := 1; tapsTab [2, 14] := 1;
   tapsTab [4, 3] := 1; tapsTab [4, 7] := 1; tapsTab [4, 12] := 1; tapsTab [4, 13] := 1;
   tapsTab [5, 8] := 1; tapsTab [5, 9] := 1; tapsTab [5, 11] := 1; tapsTab [5, 12] := 1;
   tapsTab [6, 5] := 1; tapsTab [6, 7] := 1; tapsTab [6, 10] := 1; tapsTab [6, 11] := 1;
   tapsTab [7, 8] := 1; tapsTab [7, 10] := 1;
   tapsTab [8, 6] := 1; tapsTab [8, 9] := 1;
   tapsTab [9, 4] := 1; tapsTab [9, 8] := 1;
   tapsTab [10, 3] := 1; tapsTab [10, 4] := 1; tapsTab [10, 5] := 1; tapsTab [10, 7] := 1;
   tapsTab [11, 3] := 1; tapsTab [11, 6] := 1;
   tapsTab [12, 4] := 1; tapsTab [12, 5] := 1;
   tapsTab [13, 2] := 1; tapsTab [13, 4] := 1;
   tapsTab [14, 2] := 1; tapsTab [14, 3] := 1;
   tapsTab [15, 1] := 1; tapsTab [15, 2] := 1;

   NEW (taps, MaxNoTaps);
   NEW (delayLine, MaxNoTaps);

   FOR i := 0 TO n - 1 DO  (* copy the nth taps table *)

      taps [i] := tapsTab [MaxNoTaps - n, i];
      delayLine [i] := 1;

   END;

   FOR i := 0 TO p - 1 DO  (* Generate an MLS by summing the taps mod 2 *)

      sum := 0;

      FOR j := 0 TO n - 1 DO sum := sum + (taps [j] * delayLine [j]) END;
      sum := sum MOD 2;

      mls [i] := delayLine [n - 1];

      FOR j := n - 2 TO 0 BY -1 DO delayLine [j + 1] := delayLine [j] END;

      delayLine [0] := SHORT (SHORT (sum));

   END;

END GenerateMls;
```

```
PROCEDURE FastHadamard (VAR x: ARRAY OF REAL; p1, n: INTEGER);

VAR

   i, i1, j, k, k1, k2: INTEGER;
   temp: REAL;

BEGIN

   k1 := p1;
   FOR k := 0 TO n - 1 DO

      k2 := k1 DIV 2;
      FOR j := 0 TO k2 - 1 DO

         i := j;
         WHILE i < p1 DO

            i1 := i + k2;
            temp := x [i] + x [i1];
            x [i1] := x [i] - x [i1];
            x [i] := temp;

            INC (i, k1);

         END;

      END;

      k1 := k1 DIV 2;

   END;

END FastHadamard;


PROCEDURE PermuteSignal

   (IN signal: ARRAY OF REAL; VAR permutation: ARRAY OF REAL; tagS: ARRAY OF INTEGER; p: INTEGER);

VAR

   i: INTEGER;
   dc: REAL;

BEGIN

   dc := 0;
   FOR i := 0 TO p - 1 DO dc := dc + signal [i] END;

   (* Just a permutation of the measured signal *)

   permutation [0] := -dc;
   FOR i := 0 TO p - 1 DO  permutation [tagS [i]] := signal [i] END;

END PermuteSignal;
```

```
PROCEDURE PermuteResponse
  (IN permutation: ARRAY OF REAL; VAR response: ARRAY OF REAL; tagL: ARRAY OF INTEGER; p: INTEGER);
VAR
  fact: REAL;
  i: INTEGER;
BEGIN
  fact := 1 / (p + 1);
  (* Just a permutation of the impulse response *)
  FOR i := 0 TO p - 1 DO response [i] := permutation [tagL [i]] * fact END;
  response [p] := 0;
END PermuteResponse;


PROCEDURE GeneratetagL (IN mls: ARRAY OF BYTE; VAR tagL: ARRAY OF INTEGER; p, n: INTEGER);
VAR
   i, j: INTEGER;
  colSum, index: POINTER TO ARRAY OF INTEGER;
BEGIN
  NEW (colSum, p);
  NEW (index, n);
  FOR i := 0 TO p - 1 DO  (* Run through all the columns in the autocorrelation matrix *)
    colSum [i] := 0;
    FOR j := 0 TO n - 1 DO  (* Find colSum as the value of the first N elements regarded as a binary number *)
      colSum [i] := colSum [i] + (mls [(p + i - j) MOD p] * ORD ({n - 1 - j}));
    END;
    FOR j := 0 TO n - 1 DO  (* Figure out if colSum is a 2^j number and store the column as the jth index *)
      IF colSum [i] = ORD ({j}) THEN index [j] := i END;
    END;
  END;
  FOR i := 0 TO p - 1 DO  (* For each row in the L matrix *)
    tagL[i] := 0;
    FOR j := 0 TO n - 1 DO  (* Find the tagL as the value of the rows in the L matrix regarded as a binary number *)
      tagL[i] := tagL[i] + (mls [(p + index [j] - i) MOD p] * ORD ({j}));
    END;
  END;
END GeneratetagL;
```

```
PROCEDURE GeneratetagS (IN mls: ARRAY OF BYTE; VAR tagS: ARRAY OF INTEGER; p, n: INTEGER);

VAR i, j: INTEGER;

BEGIN

  FOR i := 0 TO p - 1 DO  (* For each column in the S matrix *)

    tagS [i] := 0;

    FOR j := 0 TO n - 1 DO  (* Find the tagS as the value of the columns in the S matrix regarded as a binary number *)

      tagS [i] := tagS [i] + (mls [(p + i - j) MOD p] * ORD ({n - 1 - j}));

    END;

  END;

END GeneratetagS;


PROCEDURE Main*; (* exported command Main *)

CONST

  N = 18;
  P = ORD ({N}) - 1;

VAR

  i: INTEGER;
  mls: POINTER TO ARRAY OF BYTE;
  tagL, tagS: POINTER TO ARRAY OF INTEGER;
  signal, permutation, response: POINTER TO ARRAY OF REAL;

BEGIN

  NEW (mls, P);
  NEW (tagL, P);
  NEW (tagS, P);
  NEW (signal, P);
  NEW (permutation, P + 1);
  NEW (response, P + 1);

  GenerateMls (mls, P, N);   (* generate MLS *)
  GeneratetagL (mls, tagL, P, N); (* generate tagL for the L matrix *)
  GeneratetagS (mls, tagS, P, N);   (* generate tagS for the S matrix *)

  GenerateSignal (mls, signal, P);   (* do a simulated measurement and get the signal *)
  PermuteSignal (signal, permutation, tagS, P);  (* permute the signal according to tagS *)

  FastHadamard (permutation, P + 1, N);   (* do a Hadamard transform in place *)

  PermuteResponse (permutation, response, tagL, P);  (* permute the impulse response according to tagL *)

  Out.String ("Impulse response:"); Out.Ln;

  FOR i := 0 TO 6 DO Out.Real (response [i], 20); Out.Ln; END;

END Main;


END MLS.
```