```
##########################################################################
#
# polyinterp: Constructs Lagrange Interpolating Polynomial
#
# Given n distinct "data points" (x_i,f_i) , i = 1 ... n, and a name,
# this procedure returns the unique polynomial (in name) of degree
# n - 1 which passes through (interpolates) all the points.
#
# Input parameters:
#
#    ldata:    list of lists, which defines (x_i,f_i)
#    var:      name, returned interpolating polynomial is
#              a polynomial in 'var'
#
# Usage example:
#
#    > polyinterp([ [0,1], [1,6], [2,4], [3,0] ], 'x' );
#
#                               3      2
#                          5/6 x  - 6 x  + 61/6 x + 1
#
# Implementation notes:
#
#    This routine converts the list of input pairs (each pair
#    itself a two-element list) to separate *sequences* of
#    the x_i and f_i.  You could also build up separate *lists*
#    but it is syntactically easier to build sequences in Maple.
#
##########################################################################
```

```
polyinterp := proc(ldata,var)
#------------------------------------------------------------------------
#  Local variables:
#
#     n:         number of data points
#     i,   j:    loop variables used in evaluation of Lagrange formula
#     sx, sf:    for building up sequences of x_i, f_i respectively
#     num, den: for building up the numerators and denominators of the
#                characteristic polynomials
#     p:         for building up the interpolating polynomial itself
#
#------------------------------------------------------------------------
   local  n,   i,   j,   sx,   sf,   num,   den,   p;
   if not type(ldata,'list(list)') then
      ERROR(`first argument must be a list of lists`);
   fi;
   if not type(var,'name') then
      ERROR(`second argument must be a name`);
   fi;
#------------------------------------------------------------------------
#  Determine number of data points
#------------------------------------------------------------------------
   n  := nops(ldata);
#------------------------------------------------------------------------
#  Initialize polynomial and x_i and f_i sequences
#------------------------------------------------------------------------
   p  := 0;
   sx := NULL;
   sf := NULL;
#------------------------------------------------------------------------
#  Convert input list-of-lists into separate sequences of x_i and f_i
#------------------------------------------------------------------------
   for i from 1 to n do;
      sx := sx , ldata[i][1];
      sf := sf , ldata[i][2];
   od;
```

```
#-----------------------------------------------------------------------
#  For each of the x_i ...
#-----------------------------------------------------------------------
   for i from 1 to n do;
#-----------------------------------------------------------------------
#     ... build up the numerators and denominators of the ith
#     characteristic polynomial.  First initialize the numerator
#     and denominator ...
#-----------------------------------------------------------------------
      num := 1;
      den := 1;
#-----------------------------------------------------------------------
#     ... and then build them up using the Lagrange formula.  Note that
#     both the numerator and denominator are products of n - 1
#     terms, one term for each j = 1..n such that j <> i.
#-----------------------------------------------------------------------
      for j from 1 to n do;
         if j <> i then
            num := num * (var   - sx[j]);
            den := den * (sx[i] - sx[j]);
         fi
      od;
#-----------------------------------------------------------------------
#     Update the polynomial
#-----------------------------------------------------------------------
      p := p + sf[i] * (num / den);
   od;
#-----------------------------------------------------------------------
#  Return the polynomial in expanded form
#-----------------------------------------------------------------------
   expand(p);
#-----------------------------------------------------------------------
#  End of procedure 'polyinterp'
#-----------------------------------------------------------------------
end:
```

```
########################################################################
# Usage examples
########################################################################

#-----------------------------------------------------------------------
# Same sample usage as in procedure documentation above
#-----------------------------------------------------------------------
res1 := polyinterp([ [0,1], [1,6], [2,4], [3,0] ],'x');

#-----------------------------------------------------------------------
# Generates error message since 'f(x)' is not a name
#-----------------------------------------------------------------------
res2:= polyinterp([ [0,1], [1,6], [2,4], [3,0] ],f(x));

#-----------------------------------------------------------------------
# Example using symbolic parameters ... useful to collect the f_i in
# this case.
#-----------------------------------------------------------------------
res3  := polyinterp([ [-h,f[-1]], [0,f[0]], [h,f[1]] ],'x');
res3c := collect(res3,{f[-1],f[0],f[1]});
```