```
/* modify.ox

  A C programming example of interfacing with the
  LINDO API demonstrating adding new variables and
  constraints.

  Original problem:

     MAX = 20 * A + 30 * C
     S.T.       A +  2 * C  <= 120
                A           <=  60
                         C  <=  50

  Modified problem after adding variables:

     MAX = 20 * A + 30 * C  - 2 *D
     S.T.       A +  2 * C          <= 120
                A                   <=  60
                         C          <=  50

         [               C  + D     >=  50]  add later


   Solving such a problem with the LINDO API involves
   the following steps:

      1. Create a LINDO environment.
      2. Create a model in the environment.
      3. Specify the model.
      4. Add a new variable
      5. Perform the optimization.
      6. Retrieve the solution.
      7. Delete the LINDO environement.
*/

#include <oxstd.h>

/* LINDO API header file is located under lindoapi\ox */
#import <packages/lindoapi/ox/oxlindo>

/* main entry point */
main()
{
   decl nErrorCode;
/* Number of constraints */
   decl nM = 3;

/* Number of variables */
   decl nN = 2;

/* declare an instance of the LINDO environment object */
   decl pEnv;

/* declare an instance of the LINDO model object */
   decl pModel;

/* >>> Step 1 <<< Create a LINDO environment. */
   pEnv = OxLScreateEnv();

/* >>> Step 2 <<< Create a model in the environment. */
   pModel = LScreateModel ( pEnv, &nErrorCode);
   LSerrorCheck(pEnv, nErrorCode);

   {

/* >>> Step 3 <<< Specify the model.

 To specify our model, we make a call to LSloadLPData,
  passing it:

 - A pointer to the model which we are specifying(pModel)
 - The number of constraints in the model
 - The number of variables in the model
 - The direction of the optimization (i.e. minimize or
 -  maximize)
 - The value of the constant term in the objective (may
    be zero)
 - The coefficients of the objective function
 - The right-hand sides of the constraints
 - The types of the constraints
 - The number of nonzeros in the constraint matrix
 - The indices of the first nonzero in each column
 - The length of each column
 - The nonzero coefficients
 - The row indices of the nonzero coefficients
 - Simple upper and lower bounds on the variables
*/

/* The direction of optimization */
      decl nDir = LS_MAX;

/* The objective's constant term */
      decl dObjConst = 0.;

/* The coefficients of the objective function */
      decl adC = < 20., 30.>;

/* The right-hand sides of the constraints */
      decl adB = < 120., 60., 50.>;

/* The constraint types */
      decl acConTypes = "LLL";

/* The number of nonzeros in the constraint matrix */
      decl nNZ = 4;

/* The indices of the first nonzero in each column */
      decl anBegCol = 0 ~ 2 ~ nNZ;

/* The length of each column.  Since we aren't leaving
    any blanks in our matrix, we can set this to NULL */
      decl pnLenCol = <>;

/* The nonzero coefficients */
      decl adA = < 1., 1., 2., 1.>;

/* The row indices of the nonzero coefficients */
      decl anRowX = < 0, 1, 0, 2>;

/* Simple upper and lower bounds on the variables.
    By default, all variables have a lower bound of zero
    and an upper bound of infinity.  Therefore pass NULL
    pointers in order to use these default values. */
      decl pdLower = <>, pdUpper = <>;

/* We have now assembled a full description of the model.
    We pass this information to LSloadLPData with the
    following call. */
      nErrorCode = LSloadLPData( pModel, nM, nN, nDir,
       dObjConst, adC, adB, acConTypes, nNZ, anBegCol,
       pnLenCol, adA, anRowX, pdLower, pdUpper);
      LSerrorCheck(pEnv, nErrorCode);

   }

/* >>> Step 4 <<< Add a new variable */
```

```
   {
     decl nA = 1;
     decl achVtype = "C";
     decl pdLower = <>, pdUpper = <>;
     decl apszVname = {};
     decl ia=<>;
     decl ka=<>;
     decl cnta=<>;
     decl a=<>;
     decl c= -2.0;

     /* It is assumed that the new variable has no nonzeros in the existing
     constraints, therefore, sparse matrix data are set to <>.*/
     nErrorCode = LSaddVariables(pModel, nA,achVtype,apszVname,<>,<>,<>,
                                <>,c,<>, <>);

   }


 /* >>> Step 5 <<< Perform the optimization */
   decl nSolStatus;
   nErrorCode = LSoptimize( pModel, LS_METHOD_PSIMPLEX, &nSolStatus);
   LSerrorCheck(pEnv, nErrorCode);

   {

 /* >>> Step 6 <<< Retrieve the solution */
      decl i;
      decl adX, dObj;

 /* Get the value of the objective */
      nErrorCode = LSgetInfo(pModel,LS_DINFO_POBJ,&dObj);
      LSerrorCheck(pEnv, nErrorCode);

      println( "Objective Value = ", dObj);

 /* Get the variable values */
      nErrorCode = LSgetPrimalSolution ( pModel, &adX);
      LSerrorCheck(pEnv, nErrorCode);

      nErrorCode = LSgetInfo ( pModel, LS_IINFO_NUM_VARS, &nN);
      print ("Primal values = ", adX);
   }

 /* >>> Step 7 <<< Delete the LINDO environment */
   nErrorCode = LSdeleteEnv( &pEnv);
}
```