```c
/*--------------------------------------------------------------------------*/
/*  File: admipex5.c                                                        */
/*  Version 9.1                                                             */
/*                                                                          */
/*  Copyright (C) 1997-2005 by ILOG.                                        */
/*  All Rights Reserved.                                                    */
/*  Permission is expressly granted to use this example in the              */
/*  course of developing applications that use ILOG products.               */
/*--------------------------------------------------------------------------*/

/* Examples admipex4.c and admipex5.c both solve the MIPLIB
   3.0 model noswot.mps by adding user cuts.  admipex4.c adds
   these cuts to the cut table before the branch-and-cut
   process begins, while admipex5.c adds them through the
   user cut callback during the branch-and-cut process.

   To run this example, no command line arguments are required.
   This program reads a problem from a file named "noswot.mps" */

/* Bring in the CPLEX function declarations and the C library
   header file stdio.h with the following single include */

#include <ilcplex/cplex.h>

/* Bring in the declarations for the string and character functions,
   malloc, and fabs */

#include <ctype.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

/* The following structure will hold the information we need to
   pass to the cut callback function */

struct cutinfo {
   CPXLPptr lp;
   int      numcols;
   int      numtoadd;
   int      num;
   double   *x;
   int      *add;
   int      *beg;
   int      *ind;
   double   *val;
   double   *rhs;
};
typedef struct cutinfo CUTINFO, *CUTINFOptr;

/* Declarations for functions in this program */


static int CPXPUBLIC
   mycutcallback (CPXCENVptr env, void *cbdata, int wherefrom,
                  void *cbhandle, int *useraction_p);

static int
   makeusercuts  (CPXENVptr env, CUTINFOptr cutinfo);

static void
   free_and_null (char **ptr);


int
main ()
{
   int status = 0;

   /* Declare and allocate space for the variables and arrays where
      we will store the optimization results, including the status,
      objective value, and variable values */

   int    solstat;
   double objval;
   double *x = NULL;

   CPXENVptr env = NULL;
   CPXLPptr  lp = NULL;

   int j;
   int cur_numcols;

   CUTINFO cutinfo;

   cutinfo.x   = NULL;
   cutinfo.add = NULL;
   cutinfo.beg = NULL;
```

```
   cutinfo.ind = NULL;
   cutinfo.val = NULL;
   cutinfo.rhs = NULL;

   /* Initialize the CPLEX environment */

   env = CPXopenCPLEX (&status);

   /* If an error occurs, the status value indicates the reason for
      failure.  A call to CPXgeterrorstring will produce the text of
      the error message.  Note that CPXopenCPLEX produces no
      output, so the only way to see the cause of the error is to use
      CPXgeterrorstring.  For other CPLEX routines, the errors will
      be seen if the CPX_PARAM_SCRIND parameter is set to CPX_ON */

   if ( env == NULL ) {
      char errmsg[1024];
      fprintf (stderr, "Could not open CPLEX environment.\n");
      CPXgeterrorstring (env, status, errmsg);
      fprintf (stderr, "%s", errmsg);
      goto TERMINATE;
   }

   /* Turn on output to the screen */

   status = CPXsetintparam (env, CPX_PARAM_SCRIND, CPX_ON);
   if ( status != 0 ) {
      fprintf (stderr,
               "Failure to turn on screen indicator, error %d.\n",
               status);
      goto TERMINATE;
   }
   CPXsetintparam (env, CPX_PARAM_MIPINTERVAL, 1000);

   /* Create the problem, using the filename as the problem name */

   lp = CPXcreateprob (env, &status, "noswot");

   /* A returned pointer of NULL may mean that not enough memory
      was available or there was some other problem.  In the case of
      failure, an error message will have been written to the error
      channel from inside CPLEX.  In this example, the setting of
      the parameter CPX_PARAM_SCRIND causes the error message to
      appear on stdout.  Note that most CPLEX routines return
      an error code to indicate the reason for failure */

   if ( lp == NULL ) {
      fprintf (stderr, "Failed to create LP.\n");
      goto TERMINATE;
   }

   /* Now read the file, and copy the data into the created lp */

   status = CPXreadcopyprob (env, lp, "../../data/noswot.mps", NULL);
   if ( status ) {
      fprintf (stderr,
               "Failed to read and copy the problem data.\n");
      goto TERMINATE;
   }

   /* Set parameters */

   /* Assure linear mappings between the presolved and original
      models */

   status = CPXsetintparam (env, CPX_PARAM_PRELINEAR, 0);
   if ( status )  goto TERMINATE;


   /* Let MIP callbacks work on the original model */

   status = CPXsetintparam (env, CPX_PARAM_MIPCBREDLP, CPX_OFF);
   if ( status )  goto TERMINATE;

   cur_numcols = CPXgetnumcols (env, lp);

   cutinfo.lp = lp;
   cutinfo.numcols = cur_numcols;

   cutinfo.x = (double *) malloc (cur_numcols * sizeof (double));
   if ( cutinfo.x == NULL ) {
      fprintf (stderr, "No memory for solution values.\n");
      goto TERMINATE;
   }

   /* Create user cuts for noswot problem */

```

```
   status = makeusercuts (env, &cutinfo);
   if ( status )  goto TERMINATE;

   /* Set up to use MIP callback */

   status = CPXsetcutcallbackfunc (env, mycutcallback, &cutinfo);
   if ( status )  goto TERMINATE;

   /* Optimize the problem and obtain solution */

   status = CPXmipopt (env, lp);
   if ( status ) {
      fprintf (stderr, "Failed to optimize MIP.\n");
      goto TERMINATE;
   }

   solstat = CPXgetstat (env, lp);
   printf ("Solution status %d.\n", solstat);

   status = CPXgetmipobjval (env, lp, &objval);
   if ( status ) {
      fprintf (stderr,"Failed to obtain objective value.\n");
      goto TERMINATE;
   }

   printf ("Objective value %.10g\n", objval);

   /* Allocate space for solution */

   x = (double *) malloc (cur_numcols * sizeof (double));

   if ( x == NULL ) {
      fprintf (stderr, "No memory for solution values.\n");
      goto TERMINATE;
   }

   status = CPXgetmipx (env, lp, x, 0, cur_numcols-1);
   if ( status ) {
      fprintf (stderr, "Failed to obtain solution.\n");
      goto TERMINATE;
   }

   /* Write out the solution */

   for (j = 0; j < cur_numcols; j++) {
      if ( fabs (x[j]) > 1e-10 ) {
         printf ( "Column %d:  Value = %17.10g\n", j, x[j]);
      }
   }


TERMINATE:

   /* Free the allocated vectors */

   free_and_null ((char **) &x);
   free_and_null ((char **) &cutinfo.x);
   free_and_null ((char **) &cutinfo.beg);
   free_and_null ((char **) &cutinfo.ind);
   free_and_null ((char **) &cutinfo.val);
   free_and_null ((char **) &cutinfo.rhs);

   /* Free the problem as allocated by CPXcreateprob and
      CPXreadcopyprob, if necessary */

   if ( lp != NULL ) {
      status = CPXfreeprob (env, &lp);
      if ( status ) {
         fprintf (stderr, "CPXfreeprob failed, error code %d.\n",
                  status);
      }
   }

   /* Free the CPLEX environment, if necessary */

   if ( env != NULL ) {
      status = CPXcloseCPLEX (&env);

      /* Note that CPXcloseCPLEX produces no output, so the only
         way to see the cause of the error is to use
         CPXgeterrorstring.  For other CPLEX routines, the errors
         will be seen if the CPX_PARAM_SCRIND parameter is set to
         CPX_ON */

      if ( status ) {
         char errmsg[1024];
         fprintf (stderr, "Could not close CPLEX environment.\n");
```

```
262           CPXgeterrorstring (env, status, errmsg);
263           fprintf (stderr, "%s", errmsg);
264        }
265     }
266
267     return (status);
268
269  } /* END main */
270
271
272  /* This simple routine frees up the pointer *ptr, and sets *ptr
273     to NULL */
274
275  static void
276  free_and_null (char **ptr)
277  {
278     if ( *ptr != NULL ) {
279        free (*ptr);
280        *ptr = NULL;
281     }
282  } /* END free_and_null */
283
284
285  static int CPXPUBLIC
286  mycutcallback (CPXCENVptr env,
287                 void       *cbdata,
288                 int        wherefrom,
289                 void       *cbhandle,
290                 int        *useraction_p)
291  {
292     int status = 0;
293
294     CUTINFOptr cutinfo = (CUTINFOptr) cbhandle;
295
296     int      numcols  = cutinfo->numcols;
297     int      numtoadd = cutinfo->numtoadd;
298     int      numcuts  = cutinfo->num;
299     double   *x       = cutinfo->x;
300     int      *add     = cutinfo->add;
301     int      *beg     = cutinfo->beg;
302     int      *ind     = cutinfo->ind;
303     double   *val     = cutinfo->val;
304     double   *rhs     = cutinfo->rhs;
305     int      *cutind  = NULL;
306     double   *cutval  = NULL;
307     double   cutvio;
308     int      addcuts = 0;
309     int      i, j, k, cutnz;
310
311     *useraction_p = CPX_CALLBACK_DEFAULT;
312
313     if ( numtoadd <= 0 )  goto TERMINATE;
314
315     status = CPXgetcallbacknodex (env, cbdata, wherefrom, x,
316                                   0, numcols-1);
317     if ( status ) {
318        fprintf(stderr, "Failed to get node solution.\n");
319        goto TERMINATE;
320     }
321
322     for (i = 0; i < numcuts; i++) {
323        if ( add[i] )  continue;
324        cutvio = -rhs[i];
325        k = beg[i];
326        cutnz = beg[i+1] - k;
327        cutind = ind + k;
328        cutval = val + k;
329        for (j = 0; j < cutnz; j++) {
330           cutvio += x[cutind[j]] * cutval[j];
331        }
332
333        /* Use a cut violation tolerance of 0.01 */
334
335        if ( cutvio > 0.01 ) {
336           status = CPXcutcallbackadd (env, cbdata, wherefrom,
337                                       cutnz, rhs[i], 'L',
338                                       cutind, cutval);
339           if ( status ) {
340              fprintf (stderr, "Failed to add cut.\n");
341              goto TERMINATE;
342           }
343           add[i] = 1;
344           (cutinfo->numtoadd)--;
345           addcuts++;
346        }
347     }
348
```

```
   /* Tell CPLEX that cuts have been created */

   *useraction_p = CPX_CALLBACK_SET;

TERMINATE:

   return (status);

} /* END mycutcallback */


/* Valid cuts for noswot
   cut1: X21 - X22 <= 0
   cut2: X22 - X23 <= 0
   cut3: X23 - X24 <= 0
   cut4: 2.08 X11 + 2.98 X21 + 3.47 X31 + 2.24 X41 + 2.08 X51
         + 0.25 W11 + 0.25 W21 + 0.25 W31 + 0.25 W41 + 0.25 W51
         <= 20.25
   cut5: 2.08 X12 + 2.98 X22 + 3.47 X32 + 2.24 X42 + 2.08 X52
         + 0.25 W12 + 0.25 W22 + 0.25 W32 + 0.25 W42 + 0.25 W52
         <= 20.25
   cut6: 2.08 X13 + 2.98 X23 + 3.4722 X33 + 2.24 X43 + 2.08 X53
         + 0.25 W13 + 0.25 W23 + 0.25 W33 + 0.25 W43 + 0.25 W53
         <= 20.25
   cut7: 2.08 X14 + 2.98 X24 + 3.47 X34 + 2.24 X44 + 2.08 X54
         + 0.25 W14 + 0.25 W24 + 0.25 W34 + 0.25 W44 + 0.25 W54
         <= 20.25
   cut8: 2.08 X15 + 2.98 X25 + 3.47 X35 + 2.24 X45 + 2.08 X55
         + 0.25 W15 + 0.25 W25 + 0.25 W35 + 0.25 W45 + 0.25 W55
         <= 16.25
*/

static int
makeusercuts (CPXENVptr  env,
              CUTINFOptr cutinfo)
{
   int status = 0;

   int beg[] = {0, 2, 4, 6, 16, 26, 36, 46, 56};

   double val[] =
   {1, -1,
    1, -1,
    1, -1,
    2.08, 2.98, 3.47, 2.24, 2.08, 0.25, 0.25, 0.25, 0.25, 0.25,
    2.08, 2.98, 3.47, 2.24, 2.08, 0.25, 0.25, 0.25, 0.25, 0.25,
    2.08, 2.98, 3.47, 2.24, 2.08, 0.25, 0.25, 0.25, 0.25, 0.25,
    2.08, 2.98, 3.47, 2.24, 2.08, 0.25, 0.25, 0.25, 0.25, 0.25,
    2.08, 2.98, 3.47, 2.24, 2.08, 0.25, 0.25, 0.25, 0.25, 0.25};

   char *varname[] =
   {"X21", "X22",
    "X22", "X23",
    "X23", "X24",
    "X11", "X21", "X31", "X41", "X51",
    "W11", "W21", "W31", "W41", "W51",
    "X12", "X22", "X32", "X42", "X52",
    "W12", "W22", "W32", "W42", "W52",
    "X13", "X23", "X33", "X43", "X53",
    "W13", "W23", "W33", "W43", "W53",
    "X14", "X24", "X34", "X44", "X54",
    "W14", "W24", "W34", "W44", "W54",
    "X15", "X25", "X35", "X45", "X55",
    "W15", "W25", "W35", "W45", "W55"};

   double rhs[] = {0, 0, 0, 20.25, 20.25, 20.25, 20.25, 16.25};

   CPXLPptr lp = cutinfo->lp;

   int    *cutadd = NULL;
   int    *cutbeg = NULL;
   int    *cutind = NULL;
   double *cutval = NULL;
   double *cutrhs = NULL;

   int i, varind;
   int nz   = 56;
   int cuts = 8;

   cutadd = (int *)    malloc (cuts * sizeof (int));
   cutbeg = (int *)    malloc ((cuts+1) * sizeof (int));
   cutind = (int *)    malloc (nz * sizeof (int));
   cutval = (double *) malloc (nz * sizeof (double));
   cutrhs = (double *) malloc (cuts * sizeof (double));

   if ( cutadd == NULL ||
        cutbeg == NULL ||
```

```
436          cutind == NULL ||
437          cutval == NULL ||
438          cutrhs == NULL   ) {
439        fprintf (stderr, "No memory.\n");
440        status = CPXERR_NO_MEMORY;
441        goto TERMINATE;
442     }
443
444     for (i = 0; i < nz; i++) {
445        status = CPXgetcolindex (env, lp, varname[i], &varind);
446        if ( status )  {
447           fprintf (stderr,
448                    "Failed to get index from variable name.\n");
449           goto TERMINATE;
450        }
451        cutind[i] = varind;
452        cutval[i] = val[i];
453     }
454
455     for (i = 0; i < cuts; i++) {
456        cutadd[i] = 0;
457        cutbeg[i] = beg[i];
458        cutrhs[i] = rhs[i];
459     }
460     cutbeg[cuts] = beg[cuts];
461
462     cutinfo->numtoadd = cuts;
463     cutinfo->add      = cutadd;
464     cutinfo->num      = cuts;
465     cutinfo->beg      = cutbeg;
466     cutinfo->ind      = cutind;
467     cutinfo->val      = cutval;
468     cutinfo->rhs      = cutrhs;
469
470  TERMINATE:
471
472     if ( status ) {
473        free_and_null ((char **) &cutadd);
474        free_and_null ((char **) &cutbeg);
475        free_and_null ((char **) &cutind);
476        free_and_null ((char **) &cutval);
477        free_and_null ((char **) &cutrhs);
478     }
479
480     return (status);
481
482  } /* END makeusercuts */
```