```c
#include <stdio.h>            /* C input/output                           */
#include <stdlib.h>           /* C standard library                       */
#include <glpk.h>             /* GNU GLPK linear/mixed integer solver */

// This file is not enough to understand how to use GLPK. You need
// retrieve the glpk documentation in order to learn many useful
// functions.

#define STRINGLENGTH 30
#define NBROFCOEFS 10


// A small simili-object to manage the coefficients of the main matrix
struct matrix {
  double coef[1+NBROFCOEFS];
  int constraint_of_coef[1+NBROFCOEFS];
  int variable_of_coef[1+NBROFCOEFS];
  int next_coef;
};

struct matrix *new_matrix() {
  struct matrix *mat = malloc(sizeof(struct matrix));
  mat->next_coef = 1;
  return mat;
}

void matrix_add_coef(struct matrix *mat,
                     int constraint,
                     int variable, double value)
{
  int i = mat->next_coef;
  mat->coef[i] = value;
  mat->constraint_of_coef[i] = constraint;
  mat->variable_of_coef[i] = variable;
  mat->next_coef = i + 1;
}

void matrix_load_coefficients(struct matrix *mat, glp_prob *lp) {
  glp_load_matrix(lp,
                  mat->next_coef-1,
                  mat->constraint_of_coef,
                  mat->variable_of_coef,
                  mat->coef);
}

void matrix_free(struct matrix * mat) {
  free(mat);
}
// end of simili-object
```

```
// datas. Usually, datas should be read from a file or generated by a program.
// You should not have to store all the coefficients in a 2-dim matrix
// (as below), but instead immediately use [matrix_add_coef] each time you read
// a parameter from a file or when you generate it.
//    max 0.6 x1 + 0.5 x2
//    s.t.    x1 +   2 x2 <= 1
//    s.t.  3 x1 +     x2 <= 2
const double matrix_coef[2][2] = { {1, 2}, {3, 1}};
const double constraint_upper_bound[2] = { 1, 2 };
const double objective_coef[2] = { 0.6, 0.5 };
const int nbr_constraints = 2;
const int nbr_variables = 2;
const int nbr_coefs = 4;


char * get_variable_name(int variable) {
  static char variable_name[STRINGLENGTH];
  sprintf(variable_name, "x_%d", variable);
  return variable_name;
}

void add_variable(glp_prob *lp, int variable, double coef_in_objective) {
  char *variable_name = get_variable_name(variable);
  glp_set_col_name(lp, variable, variable_name);
  // glp_set_col_kind(lp,variable,GLP_IV); for an integer variable
  glp_set_col_bnds(lp, variable, GLP_LO, 0.0, 0.0); // 0 <= x
  glp_set_obj_coef(lp, variable, coef_in_objective);
}

void add_all_variables(glp_prob *lp) {
  glp_add_cols(lp, nbr_variables);
  for (int variable = 1; variable <= nbr_variables; variable++)
    add_variable(lp, variable, objective_coef[variable-1]);
}



void add_constraint_coefficients(struct matrix *mat, int constraint) {
  for (int variable = 1; variable <= nbr_variables; variable++)
    matrix_add_coef(mat,
                    constraint,
                    variable,
                    matrix_coef[constraint-1][variable-1]);
}
```

```
void add_constraint(glp_prob *lp,
                    struct matrix *mat,
                    int constraint,
                    double upper_bound)
{
  char constraint_name[STRINGLENGTH];
  sprintf(constraint_name, "contrainte %d", constraint);
  glp_set_row_name(lp, constraint, constraint_name);
  glp_set_row_bnds(lp, constraint, GLP_UP, 0.0, upper_bound);
  add_constraint_coefficients(mat,constraint);
}

void add_all_constraints(glp_prob *lp, struct matrix *mat) {
  glp_add_rows(lp, nbr_constraints);
  for (int constraint = 1; constraint <= nbr_constraints; constraint++)
    add_constraint(lp, mat, constraint, constraint_upper_bound[constraint-1]);
}



void show_variable_value(int variable, double variable_value) {
  char *variable_name = get_variable_name(variable);
  printf("%s = %lf\n", variable_name, variable_value);
}

void get_solution(glp_prob *lp, double *solution) {
  for (int variable = 1; variable <= nbr_variables; variable++)
    solution[variable] = glp_get_col_prim(lp, variable);
    //use glp_mip_col_val in mixed integer programming
}

void show_solution(glp_prob *lp) {
  double optimum = glp_get_obj_val(lp);
  double solution[1+nbr_variables];
  get_solution(lp,solution);

  printf("Best solution objective value z = %lf\n",optimum);
  for (int variable = 1; variable <= nbr_variables; variable++) {
    show_variable_value(variable, solution[variable]);
  }
}
```

```
void set_variables_and_constraints(glp_prob *lp) {
  struct matrix *mat = new_matrix();
  add_all_variables(lp);
  add_all_constraints(lp, mat);
  matrix_load_coefficients(mat, lp);
  matrix_free(mat);
}

glp_prob * create_linear_program() {
  glp_prob *lp = glp_create_prob();
  glp_set_prob_name(lp, "example");
  glp_set_obj_dir(lp, GLP_MAX);
  set_variables_and_constraints(lp);
  return lp;
}

void solve_problem(glp_prob *lp) {
  glp_simplex(lp, NULL);
}

void solve_mixed_integer_problem(glp_prob *lp) {
  glp_iocp param;
  glp_init_iocp(&param);
  param.presolve = GLP_ON;
  param.msg_lev = GLP_MSG_ALL; // choose among GLP_MSG_(OFF|ERR|ON|ALL)
  glp_intopt(lp,&param);
}

int main(void)
{
  glp_prob *lp = create_linear_program();
  solve_problem(lp);
  show_solution(lp);

  glp_delete_prob(lp);
  glp_free_env();
  exit(0);
}
```