APPENDIX L

C PROGRAM FOR TIN-10 APPLICATION

The C program that distributes the rain gage information to a grid cell coverage used in the SFWMM is presented below.

```
static char *SCCSID = "@(#)gr_thsn.cc     1.7 09/11/02 SFWMD Hydrologic
Systems Modeling Division";

#include "hhmodels.h"
#include "timeseries.h"
#include <string.h>
#include <math.h>
#include "grid_io.h"
#include "Tessel.h"

static char *Usage = "Usage: %s input-file";
extern int input_errors;

class Grid_definition {
 public:

  int
    maxy,               /* number of rows in grid */
    maxxt,              /* right-most column number */
    nnodes,             /* number of nodes in model grid */
    *maxx,              /* -> array of right-most column numbers */
    *minx,              /* -> array of left-most column numbers */
    *isum;              /* -> array of the accumulated number of cells
in each row */

  float
    xorigin,                 /* origin in x-direction */
    yorigin,                 /* origin in y-direction */
    spacing,                 /* grid spacing (ft) */
    garea,              /* grid area (ft^2) */
    *value;             /* -> array of values assigned to each cell */

  Grid_definition() { maxy = -1; maxxt = -1; nnodes = -1; maxx = NULL;
minx = NULL; isum = NULL;
                  spacing = -901; garea = -901; value = NULL; }

  ~Grid_definition() { delete [] maxx; delete [] minx; delete [] isum;
delete [] value; }
};


struct float_element {
  float* value_ptr;
  struct float_element* next;
};

class Incell_stations {
 public:
  int no;

  struct float_item {
    float *value_ptr;
```

```
    struct float_item *next;
  } *value_list;

  Incell_stations() { no = 0; }

  ~Incell_stations() {
    float_item* ptr = value_list;
    while (ptr) {
      float_item* next_ptr = ptr->next;
      delete ptr;
      ptr = next_ptr;
    }
  }
};


class Node_specifications {
 public:
  int
    col,
    row;
  float
    xcoord,
    ycoord,
    *value;

  //Incell_stations incell_sta;

  float_element* value_list;

  Node_specifications
    *next;
  Node_specifications() { col = -1; row = -1;
                   xcoord = -901; ycoord = -901;
                   value = NULL; next = NULL; value_list = 0; }
  Node_specifications(int Col, int Row) { col = Col; row = Row;
                                xcoord = -901; ycoord = -901;
                                value = NULL; next = NULL; ; value_list
= 0; }
  ~Node_specifications() {
    float_element* ptr = value_list;
    while (ptr) {
      float_element* next_ptr = ptr->next;
      delete ptr;
      ptr = next_ptr;
    }
 }
  void compute_xy(float xorigin, float yorigin, float spacing) { xcoord
= xorigin + col * spacing;
                                                        ycoord = yorigin + row
* spacing;
                                                        }
  int incell(Sta_element *, float);
  float compute_dist(Sta_element *);
  float X() { return xcoord; }
  float Y() { return ycoord; }
```

```
};


int get_random(int, int, int);

class Nodes {
 public:
  int nnodes;                /* number of nodes in model grid */

  Node_specifications
    *list;              /* -> array of Node_specifications structures
*/

  Nodes() { nnodes = 0; list = 0; }
  ~Nodes() {
    Node_specifications* ptr = list;
    while (ptr) {
      Node_specifications* next_ptr = ptr->next;
      delete ptr;
      ptr = next_ptr;
    }
  }
};


char *month_name [] = {
  "January",
  "February",
  "March",
  "April",
  "May",
  "June",
  "July",
  "August",
  "September",
  "October",
  "November",
  "December"
  };

static int cum_days_by_month [][12] = {{ 0, 31, 59, 90, 120, 151, 181,
212, 243, 273, 304, 334},
                            { 0, 31, 60, 91, 121, 152, 182, 213, 244,
274, 305, 335}};

static int days_in_month [][12] = {{ 31, 28, 31, 30, 31, 30, 31, 31,
30, 31, 30, 31},
                        { 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30,
31}};

extern "C" int grid_write_header(FILE *, GRID *);
extern "C" int grid_write(FILE *, GRID *, char *, float *);
extern "C" int rand(void);
//extern "C" int srand(int seed);

extern "C" void MAIN_();
```

```
static const int THIESSEN = 0;
static const int RANDOM = 1;
static const int TIN = 3;
static const int REFTIN = 4;

main(int argc, char **argv) {
  char *program_name = strdup(argv [0]);
  char *filename, *outfile, *title;
  int i, row, col, index, method, rbase, leap, day, seed, counter,
    incell_no, incell_ave;

  FILE *infile, *bin_out;
  Nodes* Node = new Nodes;
  Grid_definition* Grid = new Grid_definition;
  Node_specifications *node_ptr;
  Sta_element *sta_ptr;

  if (argc == 2)
    filename = strdup(argv [1]);
  else {
    printf(Usage, program_name);
    printf("\n\n");
    exit(-1);
  }

  if ((infile = fopen(filename, "r")) == NULL) {
    printf("Error: Unable to open %s.\n\n", filename);
    exit(-1);
  }

  extern char *Y_outfile;
  extern char *Y_title;
  extern char *Y_method;
  extern STATION_STRUC *Y_station_list;
  extern GRID_STRUC Y_grid;
  extern int Y_mlevel;
  extern int Y_incellave;
  Y_outfile = NULL;
  Y_title = NULL;
  Y_method = NULL;
  Y_station_list = NULL;
  Y_incellave = 1;
  input_errors = 0;
  incell_ave = 0;
  int denom = 1;

  extern FILE *yyin;

  /* read in time series parameters */
  TS_parameters* TS = new TS_parameters;

  TS->initialize(filename);

  if (input_errors) {
    printf("Error: %d errors found in %s\n\n", input_errors, infile);
    exit(-2);
  }
```

```
int mlevel = Y_mlevel;
zset_("MLEVEL", " ", &mlevel, 6, 1);

if (Y_title != NULL)
  title = strdup(Y_title);
else
  title = strdup("gr_thsn model");

rbase = (int) pow((double)2.0, (double)31.0);
if (Y_method == NULL || !strcasecmp(Y_method, "THIESSEN"))
  method = THIESSEN;
else if (!strcasecmp(Y_method, "RANDOM")) {
  method = RANDOM;
  seed = 1;
}
else if (!strcasecmp(Y_method, "TIN")) {
  method = TIN;
}
// check only the second letter for REFINEDTIN-#, since # is variable
else if (Y_method[1] == 'E' || Y_method[1] == 'e') {
  method = REFTIN;
  int len = strlen(Y_method);
  char* cptr = strstr(Y_method, "-");
  char cbuf [50];
  strcpy(cbuf, cptr);

  denom = atoi(cbuf+1);
  printf("resolution: %d\n", denom);
}
else {
  printf("Error: method \"%s\" is undefined.\n", Y_method);
  exit(-2);
}


if (Y_incellave)
  incell_ave = 1;

/* transfer information from scanner structures */
outfile = NULL;
if (Y_outfile != NULL)
  outfile = strdup(Y_outfile);
Grid->maxy = Y_grid.nrows;
Grid->maxx = new int [Grid->maxy];
for (i=0; i<Grid->maxy; i++)
  Grid->maxx [i] = Y_grid.rt_extent [i];
Grid->minx = new int [Grid->maxy];
for (i=0; i<Grid->maxy; i++)
  Grid->minx [i] = Y_grid.lt_extent [i];

Grid->xorigin = Y_grid.xorigin;
Grid->yorigin = Y_grid.yorigin;
Grid->spacing = Y_grid.spacing;
Grid->garea = Grid->spacing * Grid->spacing;

int yr, mo, da;
```

```
  Tw_point *indx;
  indx = TS->timewindow.get_overall_start();
  indx->get_ymd(&yr, &mo, &da);

  printf("\n\nSetup data structures ...\n\n");
  Station* in_stations = new Station;
  int no_stations = 0;
  for (STATION_STRUC *Sta_ptr = Y_station_list; Sta_ptr != NULL;
Sta_ptr = Sta_ptr->next) {
    if (Sta_ptr->pn == NULL)
      if (Sta_ptr->cb) {
      char ca [PART_LEN], cb [PART_LEN], cc [PART_LEN], cd [PART_LEN],
ce [PART_LEN], cf [PART_LEN];
      blkpad(ca, Sta_ptr->ca, PART_LEN);
      blkpad(cb, Sta_ptr->cb, PART_LEN);
      blkpad(cc, Sta_ptr->cc, PART_LEN);
      char buffer [BUF_LEN];
      blkpad(cd, "", PART_LEN);
      if (Sta_ptr->ce == NULL)
        Sta_ptr->ce = strdup("1DAY");
      blkpad(ce, Sta_ptr->ce, PART_LEN);
      if (Sta_ptr->dbkey > 0)
        sprintf(buffer, "%d-%s", Sta_ptr->dbkey, Sta_ptr->cf);
      else
        sprintf(buffer, "%s", Sta_ptr->cf);
      blkpad(cf, buffer, PART_LEN);
      Sta_ptr->pn = new char [PN_LEN];
      blkpad(Sta_ptr->pn, Sta_ptr->pn, PART_LEN);
      int npath;
      zpath_(ca, cb, cc, cd, ce, cf, Sta_ptr->pn, &npath,
             PART_LEN-1, PART_LEN-1, PART_LEN-1, PART_LEN-1, PART_LEN-
1, PART_LEN-1, PN_LEN-1);
      }

    in_stations->setdss(Sta_ptr->name, Sta_ptr->dssfile, Sta_ptr->pn,
yr);
    no_stations++;
  }

  in_stations->print("station.list");
  Sta_element *sta_list = in_stations->get_sta_list();
  TS->add_parameter(DSS_input, sta_list);

  float average, sum_value;
  TS->add_parameter(computed, &average, "average", sta_list->Intl(),
sta_list->Units(), "PER_AVER");

  /* determine maxxt (maximum number of nodes in any row) and calculate
isum array */
  Grid->maxxt = 0;
  Grid->isum = new int [Grid->maxy+1];
  for (i=0; i<Grid->maxy+1; i++)
    Grid->isum [i] = 0;
  for (i=0; i<Grid->maxy; i++) {
    if (Grid->maxx [i] > Grid->maxxt)
      Grid->maxxt = Grid->maxx [i];
    Grid->isum[i+1] = Grid->isum [i] + Grid->maxx [i] - Grid->minx [i]
```

```
+ 1;
  }

  Grid->nnodes = Grid->isum [Grid->maxy-1] + Grid->maxx [Grid->maxy-1]
- Grid->minx [Grid->maxy-1] + 1;
  Grid->value = new float [Grid->nnodes];
  Node->list = NULL;
  Node->nnodes = Grid->nnodes;

  for (row = 1, index=0; row <= Grid->maxy; row++)
    for (col = Grid->minx [row-1]; col <= Grid->maxx [row-1]; col++) {
      if (Node->list == NULL)
      node_ptr = Node->list = new Node_specifications(col, row);
      else {
      node_ptr->next = new Node_specifications(col, row);
      node_ptr = node_ptr->next;
      }
      node_ptr->value = &(Grid->value [index]);
      node_ptr->compute_xy(Grid->xorigin, Grid->yorigin, Grid-
>spacing);
      index++;
    }

  /* setup binary output file */
  if ((bin_out = fopen(outfile, "w")) == NULL) {
    printf("Error: Unable to open file \"%s\".  Exiting ...\n\n",
outfile);
    exit(-1);
  }

  GRID grid;
  strcpy(grid.header.title, title);
  grid.header.number_of_rows = Grid->maxy;
  grid.header.number_of_nodes = Grid->nnodes;
  grid.header.size.x = Grid->spacing;
  grid.header.size.y = Grid->spacing;

  int array_size;
  if (grid.header.number_of_rows < MAX_GRID_ROWS)
    array_size = MAX_GRID_ROWS;
  else
    array_size = grid.header.number_of_rows;

  grid.config.xstart = new int [array_size];
  grid.config.xend = new int [array_size];
  grid.config.cum_node_count = new int [array_size];

  for (i=0; i<Grid->maxy; i++) {
    grid.config.xstart [i] = Grid->minx [i];
    grid.config.xend [i] = Grid->maxx [i];
    grid.config.cum_node_count [i] = Grid->isum [i];
  }

  int result = grid_write_header(bin_out, &grid);

  /* check existence of station x-y coordinates */
  for (sta_ptr = sta_list; sta_ptr != NULL; sta_ptr = sta_ptr->next)
```

```
    input_errors += sta_ptr->coord_exist();

  if (input_errors) {
    printf("Error: coordinates for %d stations are missing\n\n",
input_errors);
    exit(-2);
  }


  /* identify "in-cell" stations */
  for (node_ptr = Node->list; node_ptr != NULL; node_ptr = node_ptr-
>next) {
    for (sta_ptr = sta_list; sta_ptr != NULL; sta_ptr = sta_ptr->next)
{
      if (node_ptr->incell(sta_ptr, Grid->spacing)) {
      if (!node_ptr->value_list) {
        node_ptr->value_list = new float_element;
        node_ptr->value_list->value_ptr = &sta_ptr->value;
        node_ptr->value_list->next = NULL;
      }
      else {
        float_element *item_ptr;
        for (item_ptr = node_ptr->value_list;
             item_ptr->next != NULL; item_ptr = item_ptr->next)
          ;
        item_ptr = item_ptr->next = new float_element;
        item_ptr->value_ptr = &sta_ptr->value;
        item_ptr->next = NULL;
      }
      }
    }
  }

  printf("\n\nProcessing ==>  yr/mo/da");
  while (TS->timewindow.incrm(525600)) {
    indx = TS->timewindow.get_time_indx();
    indx->get_ymd(&yr, &mo, &da);

    if (yr % 4 == 0 && yr % 100 != 0 || yr % 400 == 0)
      leap = 1;
    else
      leap = 0;

    while (TS->timewindow.incrm("1DAY")) {
      TS->update_para("1DAY");
      indx = TS->timewindow.get_time_indx();
      indx->get_ymd(&yr, &mo, &da);

      printf("\b\b\b\b\b\b\b\b\b%2d/%02d/%02d", yr-1900, mo, da);
      fflush(stdout);

      sum_value = 0;
      counter = incell_no = 0;
      Tessel* tin = 0;
      switch (method) {
      case REFTIN:
      case TIN:
```

```
      tin = new Tessel(sta_list, no_stations);
      //tin->Print();

      break;
      }

      node_ptr = Node->list;

      for (node_ptr = Node->list; node_ptr != NULL; node_ptr =
node_ptr->next) {
      float value;

      switch (method) {
      case TIN:
        *(node_ptr->value) = tin->Interpolate(node_ptr->X(), node_ptr-
>Y());
        break;
      case REFTIN:
        *(node_ptr->value) = tin->Interpolate(node_ptr->X(), node_ptr-
>Y(),
                                      denom, Grid->spacing);
        break;

      case THIESSEN:
      case RANDOM:
        float distance;
        float closest = 10e30;
        int found_one = 0;
        int no_incell = 0;
        float sum_incell = 0;

        /* are there incell stations with data? */
        if (node_ptr->value_list)
          for (float_element *item_ptr = node_ptr->value_list;
             item_ptr != NULL;
             item_ptr = item_ptr->next)

            if (*(item_ptr->value_ptr) != -901 && *(item_ptr-
>value_ptr) != -902) {
            found_one = 1;
            ++no_incell;
            sum_incell += *(item_ptr->value_ptr);
            }

        if (incell_ave && found_one) {
          *(node_ptr->value) = sum_incell / no_incell;
          incell_no++;
        }

        else {

          if (method == RANDOM)
            day = get_random(rbase, leap, mo);


          /* find closest station with data */
          for (sta_ptr = sta_list; sta_ptr != NULL; sta_ptr = sta_ptr-
```

```
>next) {

            switch (method) {
            case THIESSEN:
            value = sta_ptr->value;
            break;
            case RANDOM:
            value = sta_ptr->Data(day-1);
            break;
            }

            if (value != -902 && value != -901) {
            distance = node_ptr->compute_dist(sta_ptr);
            ++counter;
            if (distance < closest) {
              found_one = 1;
              closest = distance;
              *(node_ptr->value) = value;
            }
            }
          }

          if (!found_one) {
            int julday = indx->get_julian();
            int istyle = -1;
            int ndate;
            char date [DATE_LEN];
            blkpad(date, "", DATE_LEN);
            juldat_(&julday, &istyle, date, &ndate, DATE_LEN-1);
            printf("Warning: all stations report missing data on %s\n",
date);
          }
        }
        break;
      }

      sum_value += *(node_ptr->value);
      }

      /* compute overall grid average */
      average = sum_value / grid.header.number_of_nodes;
      TS->store_para("1DAY");

      /* dump output to binary file */
      char tag [GRID_TAG_LENGTH];
      sprintf(tag, "%s %2d, %d", month_name [mo-1], da, yr);
      grid_write(bin_out, &grid, tag, Grid->value);

      if (tin)
      delete tin;
    }
  }

  delete TS;
  delete [] program_name;
  delete [] title;
  delete [] outfile;
```

```
  delete [] filename;
  delete in_stations;
  delete Grid;
  delete Node;
  delete [] grid.config.xstart;
  delete [] grid.config.xend;
  delete [] grid.config.cum_node_count;

  printf("\n\nFinished.\n\n");
}


float Node_specifications::compute_dist(Sta_element *sta)
{
  float dx = sta->xcoord - xcoord;
  float dy = sta->ycoord - ycoord;

  float dist2 = dx * dx + dy * dy;
  return (float) sqrt((double) dist2);
}


int Node_specifications::incell(Sta_element *sta, float spacing)
{
  if (xcoord >= sta->get_xcoord() - 0.5 * spacing && xcoord <= sta-
>get_xcoord() + 0.5 * spacing &&
      ycoord >= sta->get_ycoord() - 0.5 * spacing && ycoord <= sta-
>get_ycoord() + 0.5 * spacing)
    return 1;
  else
    return 0;
}

int get_random(int rbase, int leap, int mo) {
  int random_num = rand();
  float random_frac = (float) random_num / (float) rbase;
  int random_day = (int) (random_frac * (float) days_in_month
[leap][mo-1]) + 1;
  return cum_days_by_month [leap][mo-1] + random_day;
}


void MAIN_()
{
}
```