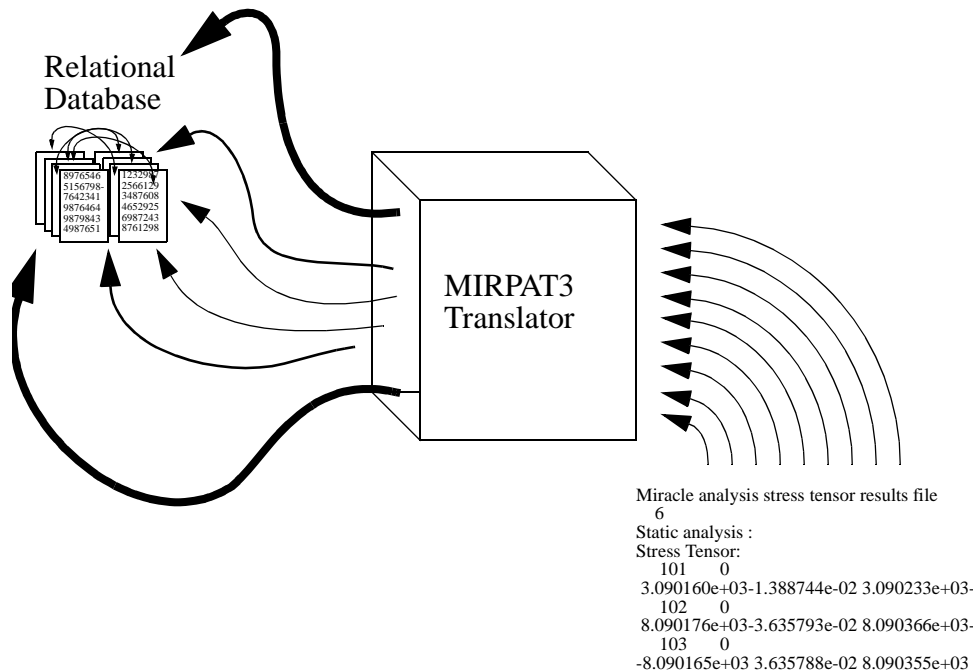


## EXERCISE 15

# *Write a Portion of the MirPat3 Translator*



### Objectives:

- Write or modify a PCL, FORTRAN or a C program which will read the Miracle “.els” file and import stress tensor results into P3/PATRAN database
- Compile the functions using a shell script
- Verify that the function works



## Problem Description:

In this Exercise, you will use P3/PATRAN database access, dbaccess, calls to write finite element stress tensor results to P3 database. This code may be used as a portion of the miracle reverse translator: MirPat3.

You may choose to write the program in PCL, FORTRAN or C. However, only the “C” sample solution is provided should you decide to only program the “P3” related calls.

We have also provided a Makefile which will automatically compile all the necessary routines and libraries to create an executable.

## Suggested Exercise Steps:

- Write or modify three functions using the on-line editor. The sample functions are “write\_tensor\_results.c”, “main\_write\_tensor\_results.c” and “read\_tensor\_results\_from\_els\_file.c”.

The main\_write\_tensor\_results\_driver.c is the main function for the program which will open up P3/PATRAN database, open the user specified “.els” file and handle subroutine dispatching. The read\_rensor\_results\_from\_els\_file.c is a function which will read the tensor results from the user specified file and write it into the JOBINFO “C” structure, which is defined in mirpat3.h header file. Finally, write\_transor\_results.c is a function which takes the JOBINFO structure as input and makes P3 database calls to input tensor results into P3.

If you choose to use our template functions to fill in the missing function arguments, you will be editing files with the same prefixes but a “.template” suffix.

- Compile the function using link\_load\_stress
- Verify the functions by running load\_stress

## Exercise Procedure:

### Write a Function

1. You may either write your program independently by using the sample “C” solution as a reference or edit the templates provided for filling in P3 related calls. The template files are called:

```
write_tensor_results.template
main_write_tensor_results_driver.template
read_tensor_results_from_els_file.template
```

when you complete filling in the templates, you must replace the .template suffix with .c before compilation.

- If you have used the vi editor before and would like to refer to the QuickReference card, turn to the back of your exercise book and look at Appendix A.

We have provided a “C” function prototype summary template in the Appendix of this exercise for your reference. This will help you identify the required data types for all of the “Results” related Dbaccess calls.

A sample solution is given at the end of the exercise.

2. Compile the functions as follows:

```
% link_load_stress
```

If you do not have execute permission, type:

```
chmod +x link_load_stress
```

and retry. The Contents of the link\_load\_stress file are as follows:

```
#!/bin/sh

# This script is delivered to the user in order to demonstrate how the
# delivered libraries should be linked with any user written C
# programs.

# This script is run by simply typing in its name, e.g.
# Prompt> link_load_stress

# If this script is successful, it should produce an executable called
# "load_stress" in the current directory without any error
# messages from the linker.

# Note that the user may have to explicitly specify the location
# of the FORTRAN system libraries if it differs from what is listed
# below.

# In general the format of the compilation/link should be as follows -

# cc -o <name_of_resulting_executable> \
#     <user_source_files> \
```

## Exercise 15

---

```
# <user_object_files> \  
# <user_libraries> \  
# "$P3_HOME"/customization/dbaccess_stubs.o \  
# "$P3_HOME"/customization/dbaccess.a \  
# /usr/interbase/lib/gds_b.a \  
# <needed_FORTRAN_system_libraries>  
  
# Fetch value of $P3_HOME or set to default value of "/patran/patran3"  
# if this environment variable is not set  
  
if [ -z "$P3_HOME" ] ; then  
    p3_home="/patran/patran3"  
else  
    p3_home="$P3_HOME"  
fi  
  
# Identify the necessary system libraries  
  
SYSDEFS=$p3_home/customization/SYS_DEFS  
if [ -f $SYSDEFS ] ; then  
    . $SYSDEFS  
else  
    echo "$0: can't locate $SYSDEFS!"  
    exit 1  
fi  
  
# Link the write_nodes C program  
  
cc -o ./load_stress \  
    $CCOPTIONS \  
    main_write_tensor_results_driver.c  
read_tensor_results_from_els_file.c write_tensor_results.c \  
    "$p3_home"/customization/dbaccess_stubs.o \  
    "$p3_home"/customization/dbaccess.a \  
    /usr/interbase/lib/gds_b.a \  
    $LIBFORT $LIBMATH $LIBDLOADER
```

3. Test the function.

To test the function type the following command on the UNIX shell prompt

```
% load_stress clevis.db clevis
```

where;

clevis.db        the P3 database to load the results into

clevis            the name of the miracle job.

4.    Verify the function:

Type:

```
% load_stress
```

*load\_stress* should respond with:

```
wrong usage!!
```

```
try load_stress database_name jobname
```

```
example: load_stress test.db test
```

```
% load_stress clevis.db clevis
```

*load\_stress* should respond with:

```
Database version 1.5 created by Release 1.1B-1    successfully opened.
```

```
  InterBase/sun4 (access method), version "S4-V3.2H"
```

```
  InterBase/sun4 (remote server), version "S4-V3.2H/tcp (irvine)"
```

```
  InterBase/sun4 (remote interface), version "S4-V3.2H/tcp (ani)"
```

```
  on disk structure version 7.0
```

```
mirpat3: Found the following Tensor results in "clevis.els"
```

```
mirpat3: There are a total of:
```

```
  10 Beams
```

```
  100 quads
```

```
  1000 Hexs
```



**Exercise 15**



---

## Sample Solution

```

:::::::::::::
mirpat3.h
:::::::::::::
#define      SUCCESS      1
#define      FAILURE      0
#define      FALSE        0
#define      TRUE          1
#define      STATIC        1
#define      TENSOR        3
#define      REAL          1
#define      PARAMETRIC    1

#define      BAR_ELEMENT   0
#define      QUAD_ELEMENT  1
#define      HEX_ELEMENT   2

typedef struct job_information *JOBINFO;

struct job_information          /* define a structure called node */
{                               /* to store the employee data      */
char   jobname [20];
char   load_case_name[20];
int    number_of_beams;
int    number_of_quads;
int    number_of_hexs;
int    *element_ids;
int    *element_types;
float  *results_array;
};

:::::::::::::
main_write_tensor_results_driver.template
:::::::::::::
/* main (argc, argv)
 *
 * Purpose: This is the main function for the imaginary code miracle.
 *
 *          This code is written to demonstrate the use of Patran 3's
 *          database Access calls, to write a portion of a reverse translator.
 *
 *          This translator only supports a limited set of elements and options.
 *          In particular, it only supports plate, and HEX elements for the
 *          Element connectivity. In addition, only a handfull of element props
 *          are translated!!
 *
 * Input: fp          FILE*      file pointer to the jobname.inp file
 *
 * Output: none
 *
 * Side Effects: none
 *
 * Errors:
 *   Return 0, no error
 *   Return 1, any error

```

## Exercise 15

---

```
*
*/

#include <stdio.h>
#include <strings.h>
#include <malloc.h>
#include <fcntl.h>

#include "mirpat3.h"

main (argc, argv)

int argc;
char *argv[];

{

    char Errbuf[256];
    char els_filename[256];
    char message_string[ 256 ];

    int debug = FALSE,
        status;

    int fd;

    FILE *fp;

    JOBINFO current_job;

    extern int write_tensor_results ();
    extern int read_tensor_results_from_els_file ();

    if (argc <= 2)
    {
        fprintf (stdout, " wrong usage!!\n");
        fprintf (stdout, " try %s database_name jobname\n", argv[0]);
        fprintf (stdout, " example: %s test.db test\n\n", argv[0]);
        exit ( 1 );
    }

    /*
    open the data base
    */

    status = DbOpenDatabase (argv[1]);

    if (status != 0)
    {
        MsgGetString( status, 256, message_string );
        fprintf (stderr, " Error occured in application Mirpat3\nreason:%s",
                message_string);

        exit ( 1 );
    }
}
```

```

/* create the string for the .els file */

strcpy( els_filename, argv[ 2 ] );
strcat( els_filename, ".els" );

/* open the file to check for permissions & existence*/

fd = open ( els_filename, O_RDONLY );

if (fd < 0)
{
    sprintf (Errbuf, "Could not open the input file.\n reason:");
    perror (Errbuf);
    exit ( 1 );
}

/* convert the file descriptor to a file pointer */

fp = fdopen (fd, "r");

/* allocate memory for jobinformation structure */

current_job = ( JOBINFO ) malloc( sizeof( struct job_information ) );

/* stuff the information with known information */

strcpy( current_job->jobname, argv[ 2 ] );
strcpy( current_job->load_case_name, "Static" );

/* count the number of elements and the read results into an array */

if ( !read_tensor_results_from_els_file( fp, &current_job) ){
    fprintf (stdout, " mirpat3: Found the following Tensor results in '%s'\n",
            els_filename );
    fprintf (stdout, " mirpat3: There are a total of:\n%d Beams\n%d quads\n
            %d Hexs\n",
            current_job->number_of_beams,
            current_job->number_of_quads,
            current_job->number_of_hexs);
}

/* call the function to write the results to p3 database */

status = write_tensor_results ( current_job );

if( !status )
    fprintf(stdout,"mirpat3: Finished writing the results to the
            database.\n");

status = DbCloseDatabase ();

if (status != 0)
{
    MsgGetString( status, 256, message_string );
    fprintf (stderr, " Error occured in application Mirpat3\nreason:%s", m

```

## Exercise 15

---

```
message_string);
    exit ( 1 );
}

fclose (fp);

if (status == 0)
    fprintf (stdout, " mirpat3: Closed %s successfully\n", argv[ 1 ]);

}                                     /* END OF MAIN */
::::::::::::::::::
read_tensor_results_from_els_file.template
::::::::::::::::::
/* int read_tensor_results_from_els_file( fp, current_job_ptr)
 *
 * Purpose: This is the main function for the imaginary code miracle.
 *
 *         This code is written to demonstrate the use of Patran 3's
 *         database Access calls, to write the input deck of an analysis code.
 *
 *         This translator only supports a limited set of elements and options.
 *         In particular, it only supports plate, and HEX elements for the
 *         Element connectivity. In addition, only a handfull of element props
 *         are translated!!
 *
 * Input: fp          FILE*      file pointer to the jobname.els file
 *        current_job_ptr  *JOBINFO pointer to job information structure
 *
 * Output: none
 *
 * Side Effects: none
 *
 * Errors:
 *     Return 0, no error
 *     Return 1, any error
 */

#include <stdio.h>
#include <strings.h>
#include <malloc.h>
#include <fcntl.h>

#include "mirpat3.h"

read_tensor_results_from_els_file( fp, current_job_ptr)

FILE *fp;
JOBINFO *current_job_ptr;

{

    char buffer[256];
```

```

int i;
int debug = 0;
int element_id, element_type;

int beam_count = 0,
quad_count = 0,
hex_count = 0,
maximum_elements = 0;

int *element_ids, *element_ids_save;
int *element_types, *element_types_save;

float value1, value2, value3,
value4, value5, value6;

float *results_array, *results_array_save;

JOBINFO current_job_info = *current_job_ptr;

/* skip the first four records */

for( i=0; i<4; i++ )
    fgets( buffer, 255, fp );

/* read two records at a time to count the number of result sets*/

while( fgets( buffer, 255, fp ) ){

    sscanf( buffer, "%d %d ", &element_id, &element_type );
    fgets( buffer, 255, fp ); /* skip the next record */

    switch (element_type){
        case BAR_ELEMENT: beam_count++;
            break;
        case QUAD_ELEMENT: quad_count++;
            break;
        case HEX_ELEMENT: hex_count++;
            break;
        default: fprintf(stderr,"skipping unknown element type in count\n");
    }
    ;
    break;
}

}

/* size the results and element ID arrays */

maximum_elements = beam_count + quad_count + hex_count;

if( maximum_elements ){

    if ((results_array = (float *) malloc (maximum_elements * 6 * sizeof
(float))) == NULL){
        fprintf (stderr, " sorry cannot get free memory\n");
        return 1;
    }
}

```

## Exercise 15

```
if ((element_ids = (int *) malloc (maximum_elements * sizeof ( int )))
    == NULL){
    fprintf (stderr, " sorry cannot get free memory\n");
    return 1;
}
if ((element_types = (int *) malloc (maximum_elements * sizeof ( int )))
    == NULL){
    fprintf (stderr, " sorry cannot get free memory\n");
    return 1;
}
}
else
    return( 1 ); /* return error status for no results */

/* rewind the file pointer to the top of file */

rewind( fp );
for( i=0; i<4; i++ )
    fgets( buffer, 255, fp ); /* skip four lines from the top of the file*/

element_ids_save = element_ids;
element_types_save = element_types;
results_array_save = results_array;

while( fgets( buffer, 255, fp ) ){

    sscanf( buffer, "%d %d ", &element_id, &element_type );
    fgets( buffer, 255, fp ); /* read the data record */
    sscanf( buffer, "%13e%13e%13e%13e%13e%13e",
            &value1, &value2, &value3,
            &value4, &value5, &value6 );

    if( debug )
        printf( "%#13.6e%#13.6e%#13.6e%#13.6e%#13.6e%#13.6e\n",
                value1, value2, value3,
                value4, value5, value6 );

    *element_ids++ = element_id;
    *element_types++ = element_type;

    *(results_array)      = value1;
    *(results_array + 1) = value2;
    *(results_array + 2) = value3;
    *(results_array + 3) = value4;
    *(results_array + 4) = value5;
    *(results_array + 5) = value6;

    results_array +=6; /* increment the pointer by six */

}

/* fill out the job information structure */

current_job_info->number_of_beams = beam_count;
```

```

    current_job_info->number_of_quads = quad_count;
    current_job_info->number_of_hexs = hex_count;

    current_job_info->results_array = results_array_save;
    current_job_info->element_ids = element_ids_save;
    current_job_info->element_types = element_types_save;

    return ( 0 );
} /* end of the read_tensor_results_from_els_file() function */

:::::::::::::
write_tensor_results.template
:::::::::::::
/* int write_tensor_results ( current_analysis )
 *
 * Purpose: This function writes tensor results fro Beams, Quads and
 *          Hexs into the P3 database. This is a portion of the mirpat3
 *          translator for the imaginary analysis code "Miracle".
 *
 *
 * Input: current_analysis  JOBINFO  atructure containing job information
 *
 *
 * Output: none
 *
 * Side Effects: none
 *
 * Errors:
 *   Return 0, no error
 *   Return 1, any other error
 *   Return 2, Memory allocation error
 *
 */
/*
Partial Reverse translator for imaginary code MIRACLE
*/

#include <stdio.h>
#include <strings.h>
#include <malloc.h>

#include "mirpat3.h"

int write_tensor_results ( current_analysis )

JOBINFO current_analysis;

{

    char message_string[ 256 ];

    int i,
        load_case_id,

```

## Exercise 15

---

```
    debug = FALSE,
    status;

int tensor_result_type;

int section_id;
int layer_id;
int load_subcase_id;
int results_case_id;

static int beam_position_coordinates[] = { 0.50, 0.0, 0.0, 0.0 };
static int quad_position_coordinates[] = { 0.50, 0.50 , 0.0, 0.0 };
static int hex_position_coordinates[] = { 0.50, 0.50, 0.50 , 0.0 };
static int section_position_coordinates[] = { 0.0, 0.0 }; /* correct the
pe docs for this */

int beam_element_position;
int quad_element_position;
int hex_element_position;
int maximum_elements;

int numform, units, datatype;

int layer_position_id;

int coord_type;

int element_type;

int *elem_position_ids, *elem_position_ids_save,
    *elem_layer_ids, *elem_layer_ids_save,
    *coord_type_ids, *coord_type_ids_save,
    *coord_ids, *coord_ids_save;

int *tmp_iptr;

/* stubbed variables needed for the DbFCreateLoadCase */

static char dlcname = '\0';
int nbrlc = 0, lbc, priority[1];
float mv = 0.0;

float *fptr;

char *malloc ();

priority[0] = 0;

/* drop the results indexing */
DbFDropResIndex( *****1*****);

/* create or use the loadcase available in the database */
```

```

status = DbFFindLoadCaseId(
    /* analysis jobname */      current_analysis->jobname,
    /* Integer job sequence */  1,
    /* returned load case ID */ &load_case_id      );

/* if the load case does not exist, create one */

if (status != 0)
{
    status = DbFCreateLoadCase(
        /* load case name */      *****2***** ,
        /* load case Type */      *****2***** ,
        /* load case desc. */     *****2***** ,
        /* num. of load cases */  *****2***** ,
        /* lbc's not used */      *****2***** ,
        /* stubbed variable */    &dlcname,
        /* not used */            mv,
        /* not used */            priority,
        /* load case Id */        *****2***** );
}

/* Create the results sub-case for this load case */

status = DbFCreateSubCase(
    /* Load case ID */          *****3***** ,
    /* Sub-case title */       *****3***** ,
    /* returned sub_case ID */  *****3***** ,
    /* results case database ID */ *****3***** );

if (status != 0)
{
    MsgGetString( status, 256, message_string );
    fprintf (stderr, " Error occured in application Mirpat3\nreason:%s",
            message_string);

    return 1;
}

datatype = TENSOR;
numform = REAL;
units = 0;

status = DbFCreateResTypes(
    /* dummy results index */    1,
    /* primary tensor */        *****4***** ,
    /* secondar primay */       *****4***** ,
    /* results data type */     *****4***** ,
    /* data numeric form */     *****4***** ,
    /* analysis code */         "Miracle",
    /* dummy index (not used) */ *****4***** ,
    /* returned result type ID */ *****4***** );

if (status != 0)

```

## Exercise 15

---

```
{
    MsgGetString( status, 256, message_string );
    fprintf (stderr, " Error occured in application Mirpat3\nreason:%s",
             message_string);
    return 1;
}

/* create Element positions */

status = DbFCreateElemPositions(
    /* dummy index (not used ) */    1,
    /* coordinate refrence Frame */  PARAMETRIC,
    /* position coord ( center )*/  *****5*****
    /* returned element pos. ID */  *****5*****);

if (status != 0)
{
    MsgGetString( status, 256, message_string );
    fprintf (stderr, " Error occured in application Mirpat3\nreason:%s",
             message_string);
    return 1;
}

status = DbFCreateElemPositions(
    /* dummy index (not used ) */    1,
    /* coordinate refrence Frame */  PARAMETRIC,
    /* position coordinates */       *****6*****
    /* returned element pos. ID */  *****6*****);

if (status != 0)
{
    MsgGetString( status, 256, message_string );
    fprintf (stderr, " Error occured in application Mirpat3\nreason:%s",
             message_string);
    return 1;
}

status = DbFCreateElemPositions(
    /* dummy index (not used ) */    1,
    /* coordinate refrence Frame */  PARAMETRIC,
    /* position coordinates */       *****7*****
    /* returned element pos. ID */  *****7*****);

if (status != 0)
{
    MsgGetString( status, 256, message_string );
    fprintf (stderr, " Error occured in application Mirpat3\nreason:%s",
             message_string);
    return 1;
}
}
```

```

/* Create Section position information */

coord_type = PARAMETRIC;

status = DbFCreateSectPos(
    /* dummy index (not used ) */      1,
    /* label for Section pos */      *****8***** ,
    /* coordinate reference Frame */  *****8***** ,
    /* position coordinates */      *****8***** ,
    /* returned element pos. ID */   *****8***** );

if (status != 0)
{
    MsgGetString( status, 256, message_string );
    fprintf (stderr, " Error occured in application Mirpat3\nreason:%s",
            message_string);
    return 1;
}

/* Create layer position information */

layer_id = 0;

status = DbFCreateLayers(
    /* dummy index (not used ) */      1,
    /* composite layer ID */          *****9***** ,
    /* section ID */                  *****9***** ,
    /* returned layer ID */           *****9***** );

if (status != 0)
{
    MsgGetString( status, 256, message_string );
    fprintf (stderr, " Error occured in application Mirpat3\nreason:%s",
            message_string);
    return 1;
}

/* allocate memory for the arrays */

maximum_elements = current_analysis->number_of_beams +
    current_analysis->number_of_quads +
    current_analysis->number_of_hexs;

if( maximum_elements ){

    if ((elem_position_ids = (int *) malloc (maximum_elements * sizeof ( int )))
        == NULL){
        fprintf (stderr, " sorry cannot get free memory\n");
        return 2;
    }

    if ((elem_layer_ids = (int *) malloc (maximum_elements * sizeof ( int
))) == NULL){
        fprintf (stderr, " sorry cannot get free memory\n");
        return 2;
    }
}

```

## Exercise 15

```
if ((coord_type_ids=(int *)malloc (maximum_elements*sizeof (int)))==NULL){
    fprintf (stderr, " sorry cannot get free memory\n");
    return 2;
}
if (( coord_ids=(int *) malloc (maximum_elements * sizeof ( int )))== NULL){
    fprintf (stderr, " sorry cannot get free memory\n");
    return 2;
}
}
else
    return( 1 ); /* return error status for no results */

/* stuff the arrays with results */

elem_position_ids_save = elem_position_ids;
elem_layer_ids_save = elem_layer_ids;
coord_type_ids_save = coord_type_ids;
coord_ids_save = coord_ids;

for( i = 0; i< maximum_elements ; i++ ){

    *elem_layer_ids++ = layer_position_id;
    *coord_type_ids++ = 0;
    *coord_ids++ = 0;

    element_type = *(current_analysis->element_types);
    current_analysis->element_types++;

    switch ( element_type ){

        case BAR_ELEMENT: *elem_position_ids++ = beam_element_position;
        break;
        case QUAD_ELEMENT: *elem_position_ids++ = quad_element_position;
        break;
        case HEX_ELEMENT: *elem_position_ids++ = hex_element_position;
        break;
        default: fprintf(stderr,"skipping unknown element type in count\n")
;
        break;
    }
}

/* Write the tensor results to the database */

status = DbFAddTElemResByType(
    /* number of Elements */          *****10*****
    /* result type ID */              *****10*****
    /* result case ID */              *****10*****
    /* array of element pos. */       *****10*****
    /* array of element layer id */   *****10*****
    /* array of element ID */         current_analysis->element_ids,
    /* array or coordinate type */     *****10*****
    /* array of CID's */              */* *****10*****
```

---

```
        /* results array (tensor) */    current_analysis->results_array );

    if (status != 0)
    {
        MsgGetString( status, 256, message_string );
        fprintf (stderr, " Error occured in application Mirpat3\nreason:%s", m
message_string);
        return 1;
    }

    /* free malloced memory */

    free ( elem_position_ids_save );
    free ( elem_layer_ids_save );
    free ( coord_type_ids_save );
    free ( coord_ids_save );

    return 0;
}                                     /* END FUNCTION write_tensor_results() */
```

```

*1* DbFDropResIndex( )
*2* status = DbFCreateLoadCase(
      /* load case name */      current_analysis->jobname,
      /* load case Type */      STATIC,
      /* load case desc. */     "Static analysis job",
      /* num. of load cases */  nbrlc,
      /* lbc's not used */      &lbc,
      /* stubbed variable */    &dlcname,
      /* not used */            mv,
      /* not used */            priority,
      /* load case Id */        &load_case_id );
*3* status = DbFCreateSubCase(
      /* Load case ID */        load_case_id,
      /* Sub-case title */      "Static Analysis",
      /* returned sub_case ID */ &load_subcase_id,
      /* results case database ID */ &results_case_id );
*4* status = DbFCreateResTypes(
      /* dummy results index */ 1,
      /* primary tensor */      "Stress Tensor",
      /* secondar primay */     " At the centroid",
      /* results data type */    &datatype,
      /* data numeric form */    &numform,
      /* analysis code */        "Miracle",
      /* dummy index (not used) */ &units,
      /* returned result type ID */ &tensor_result_type );
*5* status = DbFCreateElemPositions(
      /* dummy index (not used) */ 1,
      /* coordinate refrence Frame */ PARAMETRIC,
      /* position coord ( center)*/ beam_position_coordinates,
      /* returned element pos. ID */ &beam_element_position);

```

```

*6* status = DbFCreateElemPositions(
    /* dummy index (not used) */ 1,
    /* coordinate reference Frame */ PARAMETRIC,
    /* position coordinates */ quad_position_coordinates,
    /* returned element pos. ID */ &quad_element_position);
*7* status = DbFCreateElemPositions(
    /* dummy index (not used) */ 1,
    /* coordinate reference Frame */ PARAMETRIC,
    /* position coordinates */ hex_position_coordinates,
    /* returned element pos. ID */ &hex_element_position);
*8* status = DbFCreateSectPos(
    /* dummy index (not used) */ 1,
    /* label for Section pos */ "Center",
    /* coordinate reference Frame */ &coord_type,
    /* position coordinates */ section_position_coordinates,
    /* returned element pos. ID */ &section_id);
*9* status = DbFCreateLayers(
    /* dummy index (not used) */ 1,
    /* composite layer ID */ &layer_id,
    /* section ID */ &section_id,
    /* returned layer ID */ &layer_position_id );
*10* status = DbFAddTElemResByType(
    /* number of Elements */ maximum_elements,
    /* result type ID */ tensor_result_type,
    /* result case ID */ results_case_id,
    /* array of element pos. */ elem_position_ids_save,
    /* array of element layer id */ elem_layer_ids_save,
    /* array of element ID */ current_analysis->element_ids,
    /* array of coordinate type */ coord_type_ids_save,
    /* array of CID's */ coord_ids_save,
    /* results array (tensor) */ current_analysis->results_array
);

```



**Exercise 15**



---

---

## Appendix A: Results C Prototype

```

$
$  FORTRAN to C templates:
$
>From Fortran: DB_POST_RESULTS_LOAD
To C:          DbFPostResultsLoad
Syntax C:
return:       integer

>From Fortran: DB_CREATE_RES_INDEX
To C:          DbFCreateResIndex
Syntax C:
return:       integer

>From Fortran: DB_DROP_RES_INDEX
To C:          DbFDropResIndex
Syntax C:
return:       integer

>From Fortran: DB_FIND_LOAD_CASE_ID
To C:          DbFFindLoadCaseId
Syntax C:
Arg:          jobname  input  char[]
Arg:          lcseq    input  int
Arg:          lcid     output int
return:       integer

>From Fortran: DB_CREATE_SUB_CASE
To C:          DbFCreateSubCase
Syntax C:
Arg:          lcid     input  int
Arg:          sctitle  input  char[]
Arg:          scid     output int
Arg:          rcid     output int
return:       integer

>From Fortran: DB_CREATE_GLOBAL_VARIABLES
To C:          DbFCreateGlobalVariables
Syntax C:
Arg:          num      input  int
Arg:          rcid     input  int
Arg:          alabel   input  cstring[]
Arg:          num_form input  int[]
Arg:          cid_type input  int[]

```

Exercise 15

---

```
Arg:      cid          input  int[]
Arg:      crby         input  char[]
Arg:      gvid         output int[]
Arg:      avalue       input  float[][6]
return:   integer
```

>From Fortran: DB\_CREATE\_ELEM\_POSITIONS

To C: DbFCreateElemPositions

Syntax C:

```
Arg:      num          input  int
Arg:      ctype        input  int
Arg:      coords       input  float[][4]
Arg:      elem_pos_id  output  int[]
return:   integer
```

>From Fortran: DB\_CREATE\_SECT\_POS

To C: DbFCreateSectPos

Syntax C:

```
Arg:      num          input  int
Arg:      alabel       input  cstring[]
Arg:      coord_type   input  int[]
Arg:      sect_coords  input  float[][2]
Arg:      sect_pos_id  output  int[]
return:   integer
```

>From Fortran: DB\_CREATE\_LAYERS

To C: DbFCreateLayers

Syntax C:

```
Arg:      num          input  int
Arg:      layerid      input  int[]
Arg:      sect_pos_id  input  int[]
Arg:      layerposid   output  int[]
return:   integer
```

>From Fortran: DB\_ASSOC\_RES\_TYPES

To C: DbFAssocResTypes

Syntax C:

```
Arg:      num          input  int
Arg:      list1        input  int[]
Arg:      list2        input  int[]
return:   integer
```

>From Fortran: DB\_ASSOC\_GLOBAL\_VARIABLES

---

```

To C:          DbFAssocGlobalVariables
Syntax C:
Arg:          num          input   int
Arg:          rcid         input   int
Arg:          list1        input   int[]
Arg:          list2        input   int[]
return:       integer

```

```
>From Fortran: DB_CREATE_RES_TYPES
```

```

To C:          DbFCreateResTypes
Syntax C:
Arg:          num          input   int
Arg:          prilbl       input   cstring[]
Arg:          seclbl       input   cstring[]
Arg:          datatype     input   int[]
Arg:          numform      input   int[]
Arg:          crby         input   char[]
Arg:          unitedd      input   int[]
Arg:          rtid         output  int[]
return:       integer

```

```
>From Fortran: DB_ADD_S_ELEM_RES_BY_POS
```

```

To C:          DbFAddSElemResByPos
Syntax C:
Arg:          num          input   int
Arg:          rtid         input   int[]
Arg:          rcid         input   int
Arg:          epid         input   int
Arg:          lpid         input   int
Arg:          elid         input   int
Arg:          value        input   float[]
return:       integer

```

```
>From Fortran: DB_ADD_V_ELEM_RES_BY_POS
```

```

To C:          DbFAddVElemResByPos
Syntax C:
Arg:          num          input   int
Arg:          rtid         input   int[]
Arg:          rcid         input   int
Arg:          epid         input   int
Arg:          lpid         input   int
Arg:          elid         input   int
Arg:          cid_type     input   int[]
Arg:          cid          input   int[]

```

Exercise 15

---

Arg: value input float[][3]  
return: integer

>From Fortran: DB\_ADD\_T\_ELEM\_RES\_BY\_POS

To C: DbFAddTElemResByPos

Syntax C:

Arg: num input int  
Arg: rtid input int[]  
Arg: rcid input int  
Arg: epid input int  
Arg: lpid input int  
Arg: elid input int  
Arg: cid\_type input int[]  
Arg: cid input int[]  
Arg: value input float[][6]  
return: integer

>From Fortran: DB\_ADD\_S\_ELEM\_RES\_BY\_TYPE

To C: DbFAddSElemResByType

Syntax C:

Arg: num input int  
Arg: rtid input int  
Arg: rcid input int  
Arg: epid input int[]  
Arg: lpid input int[]  
Arg: elid input int[]  
Arg: value input float[]  
return: integer

>From Fortran: DB\_ADD\_V\_ELEM\_RES\_BY\_TYPE

To C: DbFAddVElemResByType

Syntax C:

Arg: num input int  
Arg: rtid input int  
Arg: rcid input int  
Arg: epid input int[]  
Arg: lpid input int[]  
Arg: elid input int[]  
Arg: cid\_type input int[]  
Arg: cid input int[]  
Arg: value input float[][3]  
return: integer

```

>From Fortran: DB_ADD_T_ELEM_RES_BY_TYPE
To C:          DbFAddTElemResByType
Syntax C:
Arg:          num          input   int
Arg:          rtid         input   int
Arg:          rcid         input   int
Arg:          epid         input   int[]
Arg:          lpid         input   int[]
Arg:          elid         input   int[]
Arg:          cid_type     input   int[]
Arg:          cid          input   int[]
Arg:          value        input   float[][][6]
return:       integer

```

```

>From Fortran: DB_ADD_S_NOD_RES_BY_POS
To C:          DbFAddSNodResByPos
Syntax C:
Arg:          num          input   int
Arg:          rtid         input   int[]
Arg:          rcid         input   int
Arg:          lpid         input   int
Arg:          nid          input   int
Arg:          value        input   float[]
return:       integer

```

```

>From Fortran: DB_ADD_V_NOD_RES_BY_POS
To C:          DbFAddVNodResByPos
Syntax C:
Arg:          num          input   int
Arg:          rtid         input   int[]
Arg:          rcid         input   int
Arg:          lpid         input   int
Arg:          nid          input   int
Arg:          cid_type     input   int[]
Arg:          cid          input   int[]
Arg:          value        input   float[][][3]
return:       integer

```

```

>From Fortran: DB_ADD_T_NOD_RES_BY_POS
To C:          DbFAddTNodResByPos
Syntax C:
Arg:          num          input   int
Arg:          rtid         input   int[]
Arg:          rcid         input   int
Arg:          lpid         input   int

```

Exercise 15

---

```
Arg:      nid      input  int
Arg:      cid_type input  int[]
Arg:      cid      input  int[]
Arg:      value   input  float[][6]
return:   integer
```

>From Fortran: DB\_ADD\_S\_NOD\_RES\_BY\_TYPE

To C: DbFAddSNodResByType

Syntax C:

```
Arg:      num      input  int
Arg:      rtid     input  int
Arg:      rcid     input  int
Arg:      lpid     input  int[]
Arg:      nid      input  int[]
Arg:      value   input  float[]
return:   integer
```

>From Fortran: DB\_ADD\_V\_NOD\_RES\_BY\_TYPE

To C: DbFAddVNodResByType

Syntax C:

```
Arg:      num      input  int
Arg:      rtid     input  int
Arg:      rcid     input  int
Arg:      lpid     input  int[]
Arg:      nid      input  int[]
Arg:      cid_type input  int[]
Arg:      cid      input  int[]
Arg:      value   input  float[][3]
return:   integer
```

>From Fortran: DB\_ADD\_T\_NOD\_RES\_BY\_TYPE

To C: DbFAddTNodResByType

Syntax C:

```
Arg:      num      input  int
Arg:      rtid     input  int
Arg:      rcid     input  int
Arg:      lpid     input  int[]
Arg:      nid      input  int[]
Arg:      cid_type input  int[]
Arg:      cid      input  int[]
Arg:      value   input  float[][6]
return:   integer
```

x

