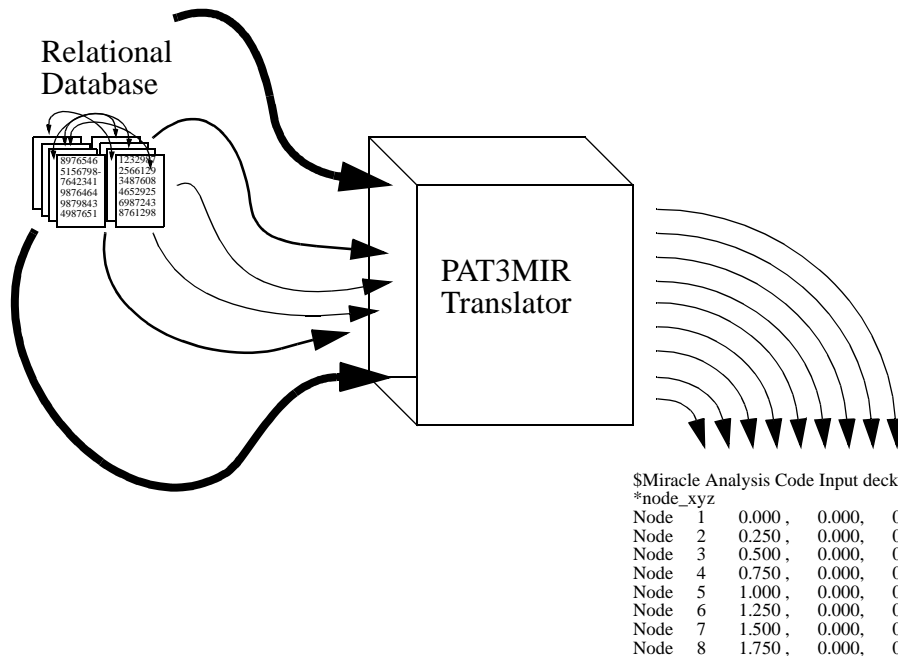


## EXERCISE 14

# *Write a Portion of the Pat3mir Translator*



### Objectives:

- Write or modify a PCL, FORTRAN or a C program which will access a p3 database and write nodes into the miracle input format
- 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 retrieve finite element nodes specified in a P3 model and write them in “Miracle’s” input deck format. This code may be used as a portion of the miracle forward translator: Pat3mir.

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 the write\_nodes executable.

## Suggested Exercise Steps:

- Write or modify two functions using the on-line editor. The functions are “write\_nodes.c” and “main\_node\_driver.c”. The write\_nodes.c function accesses the p3 database and writes the nodes in a user specified input file. The main\_write\_node\_driver.c is the main function for the executable.
- Compile the function using link\_write\_nodes shell script
- Verify the functions by running write\_nodes

## Exercise Procedure:

### Write a Function

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

write\_nodes.template  
main\_node\_write\_driver.template

when you complete filling in the templates, you must rename the files write\_nodes.c and main\_node\_write\_driver.c.

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

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

2. Compile the functions as follows:

```
% link_write_nodes
```

If you do not have execute permission, type:

```
chmod +x link_write_nodes
```

and retry. The Contents of the link\_write\_nodes shell script 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_write_nodes

# If this script is successful, it should produce an executable called
# "write_nodes" 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> \
#   <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
```

## Exercise 14

---

```
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 ./write_nodes \
  $CCOPTIONS \
  main_node_write_driver.c write_nodes.c \
  "$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

```
% write_nodes clevis.db clevis.inp
```

where;

clevis.db      the database with finite element nodes

clevis.inp     the name of the miracle input deck.

4. Verify the result:

Type:

```
% write_nodes
```

*write\_nodes* should respond with:

```
wrong usage!!
```

```
try write_nodes database_name miracle_input_filename
```

```
example: write_nodes test.db test.inp
```

```
% write_nodes clevis.db clevis.inp
```

*write\_nodes* 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
there are a total of 1331 nodes in the clevis.db database

pat3mir: Wrote the nodes successfully
pat3mir:closed clevis.db successfully
```

The translated nodes will be written to "clevis.inp" file as follows:

```
$Miracle Analysis Code Input deck
*node_xyz
Node    1      0.000 ,      0.000,      0.000
Node    2      0.100 ,      0.000,      0.000
Node    3      0.200 ,      0.000,      0.000
Node    4      0.300 ,      0.000,      0.000
Node    5      0.400 ,      0.000,      0.000
Node    6      0.500 ,      0.000,      0.000
Node    7      0.600 ,      0.000,      0.000
Node    8      0.700 ,      0.000,      0.000
Node    9      0.800 ,      0.000,      0.000
Node   10      0.900 ,      0.000,      0.000
Node   11      1.000 ,      0.000,      0.000
...
```



**Exercise 14**



---



## Exercise 14

---

```
char Errbuf[256];
char message_string[256];

int debug = FALSE,
    status;

int fd;

FILE *fp;

extern int write_nodes ();

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

/*
open the data base
*/

status = DbOpenDatabase (*****1*****);

if (status != 0)
{
    fprintf (stderr, " could not open the database\n");
    exit ( 1 );
}

/*
open and write the header information to the mail box
*/

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

fd = open ( argv[2], O_WRONLY | O_CREAT | O_APPEND, 0644);

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

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

fp = fdopen (fd, "a");
```

```

/*
write Nodes to the input file.
*/

if (!write_nodes (fp))
    fprintf (stderr, " pat3mir: Wrote the nodes successfully\n");

status = DbCloseDatabase (*****2*****);

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

    return 1;
}

fclose (fp);

if (status == 0)
    fprintf (stdout, " pat3mir:closed test.db successfully\n");

}                                     /* END OF MAIN */
:::::::::::::
write_nodes.c
:::::::::::::
/* int write_nodes ( fp )
*
* Purpose: This function writes the nodes of the model in the input
*          file for the imaginary code miracle.
*
*
* Input: fp          FILE*      file pointer to the jobname.inp file
*
* Output: none
*
* Side Effects: none
*
* Errors:
*   Return 0,  no error
*   Return 1,  any error
*   Return 2,  Memory allocation error
*
*/
/*
Forward translator for imaginary code MIRACLE
*/

#define SUCCESS 1
#define FAILURE 0
#define FALSE 0
#define TRUE 1

```

## Exercise 14

---

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

int write_nodes (fp)

FILE *fp;

{

    char message_string[256];

    int num_nodes,
        debug = FALSE,
        status;

    int *node_ids,
        i;
        /* node_id 1-D array */
    int *ref_cords;
    int *anal_codes;
    int *tmp_iptr;

    float *fptr;

    char *malloc ();

    /*
    Read the nodes in the database
    */

    status = DbFCountNodes (*****3*****);

    if ( status )
    {
        MsgGetString( status, 256, message_string );
        fprintf (stderr, " Error ocured in application pat3mir\n reason:%s\n",
            message_string);
        return 1;
    }

    fprintf (stdout, " there are a total of %d nodes in the test.db database\n\n",
        num_nodes);

    if (debug)
        fprintf (stdout, " allocating memory for the node array !!\n");
    if ((fptr = (float *) malloc (num_nodes * 3 * sizeof (float))) == NULL)
    {
        fprintf (stderr, " sorry cannot get free memory\n");
        return 2;
    }

    if ((node_ids = (int *) malloc (num_nodes * sizeof (int))) == NULL)
```

```

{
    fprintf (stderr, " sorry cannot get free memory\n");
    return 2;
}

status = DbFGetNodeIds (*****4*****);

if ( status )
{
    MsgGetString( status, 256, message_string );
    fprintf (stderr, " Error occured in application pat3mir\n reason:%s\n",
            message_string);
    return 1;
}
if (debug)
{
    for (i = 0, tmp_iptr = node_ids; i < num_nodes; i++, tmp_iptr++)
        fprintf (stdout, "Node %d \n", *tmp_iptr);
}

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

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

if (debug)
    fprintf (stdout, " getting XYZ coordinates for the nodes !!\n");

status = DbFGetNodes (*****5*****);

if ( status )
{
    MsgGetString( status, 256, message_string );
    fprintf (stderr, " Error occured in application pat3mir\n reason:%s\n",
            message_string);
    return 1;
}

/*
write the header information
*/
fprintf (fp, "$Miracle Analysis Code Input deck\n");
fprintf (fp, "**node_xyz\n");
for (i = 0; i < num_nodes; i++, node_ids++, fptr += 3)
    fprintf (fp, "Node %5d %10.3f , %10.3f, %10.3f \n", *node_ids, *fptr,
            *(fptr + 1), *(fptr + 2));
if (debug)
    fprintf (stdout, " freeing the malloced memory !!\n");
/*
free malloced memory

```

  
**Exercise 14**  


---

```
*/  
  
free (node_ids);  
free (fptr);  
free (ref_cords);  
free (anal_codes);  
  
return 0;  
}  
/* END FUNCTION write_nodes() */
```

```
*1*      argv[1]
*2*      status = DbCloseDatabase ();
*3*      knum_nodes
*4*      num_nodes, node_ids
*5*      num_nodes, node_ids, ref_cords, anal_codes, fptr
```