```
/* Include statements. */

#include
EXEC SQL INCLUDE SQLCA;

/* Function prototypes */

void cleanUp(void);

/* ------ MAIN ------ */

/* Top level function */

void main(void)
{
EXEC SQL BEGIN DECLARE SECTION;
        char host_name[21];
        int host_emp_number;
EXEC SQL END DECLARE SECTION;

EXEC SQL DECLARE empcsr CURSOR FOR
        SELECT name, emp_number
        FROM employees
        WHERE emp_number = 10001;

/* An error when opening the frans database will cause
the error to be printed and the program to be aborted */

EXEC SQL WHENEVER SQLERROR STOP;

EXEC SQL CONNECT frans;

/* Errors from here on will cause the program to clean up */

EXEC SQL WHENEVER SQLERROR CALL cleanUp;
```

```
EXEC SQL OPEN empcsr;

printf("Some values from the /"employees/" table /n");

/* When ever no more rows are fetched, close the cursor */

EXEC SQL WHENEVER NOT FOUND GOTO close_empcsr;

/* The last executable SQL statement was OPEN so we know that
the value of "sqlcode" cannot be SQLERROR or NOT FOUND */

while(sqlca.sqlcode == 0) {     /* Loop is broken by NOT FOUND */
        EXEC SQL FETCH empcsr
                INTO :host_name, :host_emp_number;

/* This "printf" does not execute after the previous FETCH returns the
NOT FOUND condition */

        printf("%s %d/n",host_name,host_emp_number);
        }

/* From this point onwards the program ignore all errors.
Also turn off the NOT FOUND condition for consistency */

EXEC SQL WHENEVER SQLERROR CONTINUE;
EXEC SQL WHENEVER NOT FOUND CONTINUE;

close_empcsr:
        EXEC SQL CLOSE empcsr;

EXEC SQL DISCONNECT;
}
```

```
/* ------ CLEAN UP ------ */

/* Error handling procedure (print error and disconnect) */

void cleanUp(void)
{
EXEC SQL BEGIN DECLARE SECTION;
        char errmsg[10];
EXEC SQL END DECLARE SECTION;

EXEC SQL INQUIRE_SQL (:errmsg = ERRORTEXT); /* Get error message. */
/* Alternatively to get only the error number
EXEC SQL COPY SQLERROR INTO :errmsg WITH 256; */

printf("Aborting because of error: /n%s/n",errmsg);

EXEC SQL DISCONNECT;
exit(-1);
}
```

Discussion

1. EXEC SQL INCLUDE: The INCLUDE statement provides a means of including external SQL files in the source code. Here it is used to include the SQLCA structure.

2. EXEC SQL BEGIN .... END DECLARE SECTION: Host variables must be declared prior to their use. Host variables can be global or local.

3. EXEC SQL DECLARE CURSOR: Names a cursor for use with a specified set of retrieval criteria. Once declared the cursor can be opened, using an OPEN statement, which causes the select statement specified in DECLARE CURSOR to be executed. The run time retrieval actually occurs when a FETCH is subsequently performed. When all processing has been completed the cursor can be closed with the CLOSE statement.

4. EXEC SQL WHENEVER SQLERROR STOP: The WHENEVER statement stipulates that some action occurs when a given condition is satisfied. In this case the action is STOP and the condition SQLERROR. The SQLERROR condition is satisfied if sqlcode in the SQCLA structure is negative (this indicates that an error has occurred.) The STOP action simply terminates program execution after printing an error message.

5. EXEC SQL CONNECT: Connects the program to the named database. The statement must precede any statements using the database.

6. EXEC SQL WHENEVER SQLERROR CALL: Similar to 4. The CALL action is used to call a host language procedure (cleanUp in this case). No arguments can be passed to the procedure.

7. EXEC SQL OPEN: Opens the cursor (empcsr) for processing. A cursor must be opened before any data manipulation statements can be performed e.g. a FETCH statement.

8. EXEC SQL WHENEVER NOT FOUND GOTO: Similar to 4 and 6. The NOT FOUND condition becomes true when sqlcode in the SQLCA structure is set to a value of 100 thus indicating that (say) a FETCH statement affected no rows.

9. EXEC SQL FETCH: Used by the cursor (empcsr). It first advances the open cursor one row. Next it loads the values indicated in the SELECT clause of the DECLARE CURSOR statement into the host variables listed in its INTO clause. Once loaded into the host variables the values can be further processed.

10. EXEC SQL WHENEVER SQLERROR CONTINUE and EXEC SQL WHENEVER NOT FOUND CONTINUE: Similar to 4, 6 and 8. The CONTINUE action simply means that no action should be taken based on the associated condition, the program proceeds to the next statement.

11. EXEC SQL CLOSE: Close the open cursor (empcsr)

12. EXEC SQL DISCONNECT: Terminate access to the (frans) database.

13. EXEC SQL COPY SQLERROR: Allows access to text of error messages. Thus when an error occurs, i.e. sqlcode is negative, the text of the error message can be put into the name variable errmsg) and printed.