```
// GenMicroBank.cpp - GENERATE MICROBANK LOAN DATA
//
// MODULE INDEX
// NAME                 CONTENTS
// ErrFun               Database error handling function
// GenName              Generate executive or customer name
// Main                 Main line
//
// MAINTENANCE HISTORY
// DATE         PROGRAMMER AND DETAILS
// 03-11-03     PFT     Original
//
//-----------------------------------------------------------------------------

#include <stdio.h>              // Standard input/output declarations
#include <string.h>             // String manipulation functions
#include <stdlib.h>             // Standard library
#include <unistd.h>             // Unix standard library
#include <math.h>               // Mathematical functions

//-----------------------------------------------------------------------------

// GLOBAL DEFINITIONS

#define DB_NAME         "Microbank"     // Database name
#define MIN_AMT         100             // Minimum loan amount ($)
#define MAX_AMT         1000            // Maximum loan amount ($)
#define DLR_SCALE       100.0           // Dollar scale
#define N_LOANS         1000000         // Number of loans
#define N_EXEC          1000            // Number of executives
#define DEL_PROB        0.01            // Executive deletion probability
#define NP_LOAN_PROB    0.30            // Non-performing loan probability

//-----------------------------------------------------------------------------

// EXECUTIVE DATA STRUCTURE

typedef struct {
        int     execCode;               // Executive code
        short   execDel;                // Deleted flag
} Exec_t;

//-----------------------------------------------------------------------------

// DATABASE ERROR HANDLING FUNCTION

void
ErrFun ()
{
        exec sql rollback work;

        printf ("DB error SQLCODE=%d ", SQLCODE);
        printf ("errd[1]=%d ", sqlca.sqlerrd[1]);
        printf ("errm=%s\n", sqlca.sqlerrm);
        exit (1);
}

//-----------------------------------------------------------------------------

// GENERATE EXECUTIVE/CUSTOMER NAME

void
GenName (
```

```
        char            *name)          // Name buffer
{
        char    nam[21];                // Name
        int     namLen;                 // Name length
        int     i, j;                   // General purpose indices

        // Generate last name

        namLen = (int)(4 + lrand48() % 3);
        for (i=0; i<namLen; i++)
                nam[i] = (char)('A' + lrand48() % ('Z'-'A'+1));
        nam[i++] = ' ';

        // Generate middle name

        namLen = (int)(4 + lrand48() % 3);
        for (j=i; j<i+namLen; j++)
                nam[j] = (char)('A' + lrand48() % ('Z'-'A'+1));
        nam[j++] = ' ';

        // Generate last name

        namLen = (int)(4 + lrand48() % 3);
        for (i=j; i<j+namLen; i++)
                nam[i] = (char)('A' + lrand48() % ('Z'-'A'+1));
        nam[i] = '\0';

        // Load the name buffer

        sprintf (name, "%s", nam);
}

//---------------------------------------------------------------------------

// MAIN LINE

int
main ()
{
        Exec_t  execArr[N_EXEC];        // Executive array
        short   delFlag;                // Deleted flag
        int     fstExecInd;             // 1st executive index
        int     secExecInd;             // 2nd executive index
        int     i;                      // General purpose index
        double  v;                      // General purpose variable

        exec sql begin declare section;
                char    dbNam[10];              // Database name
                int     execCode;               // Executive code
                char    execName[21];           // Executive name
                int     loanId;                 // Loan Id
                char    loanCust[21];           // Customer name
                double  loanAmount;             // Loan amount in cents
                short   loanStatus;             // Loan status
                int     fstExecCode;            // 1st executive code
                short   fstExecAppType;         // 1st executive approver type
                int     secExecCode;            // 2nd executive code
                short   secExecAppType;         // 2nd executive approver type
        exec sql end declare section;

        // Initiate database error handling

        exec sql whenever sqlerror goto :db_error;
```

```
// Seed the random number generator

srand48 (121);

// Create Database

strcpy (dbNam, DB_NAME);
exec sql create database :dbNam with log mode ansi;

// Create SQL tables

exec sql create table executives (
        execCode        integer not null,
        execName        char(20) not null
);

exec sql create table loans (
        loanId          integer not null,
        loanCust        char(20) not null,
        loanAmount      double precision not null,
        loanStatus      smallint not null
);

exec sql create table approvals (
        appLoanId       integer not null,
        appExecCode     integer not null,
        appType         smallint not null
);
exec sql create index appLoanIdInd on approvals (appLoanId);

// Generate executives

for (i=0; i<N_EXEC; i++) {
        execArr[i].execCode = execCode = 100000 + i;
        GenName (execName);
        execArr[i].execDel = delFlag = drand48 () < DEL_PROB;

        // Store undeleted executives

        if (! delFlag) {
                exec sql insert into executives (
                        execCode, execName
                ) values (
                        :execCode, :execName

                );
        }
}

// Simulate loans

for (i=0; i<N_LOANS; i++) {
        loanId = 10000000 + i;
        GenName (loanCust);
        v = MIN_AMT + drand48 () * (MAX_AMT - MIN_AMT) * DLR_SCALE;
        modf (v, &loanAmount);
        loanStatus = drand48 () < NP_LOAN_PROB ? 0 : 1;

        // Store the loan data

        exec sql insert into loans (
                loanId, loanCust, loanAmount, loanStatus
```

```
                ) values (
                        :loanId, :loanCust, :loanAmount, :loanStatus
                );

                // Store approval data for undeleted executives

                fstExecInd = (int)(lrand48() % N_EXEC);
                fstExecCode = execArr[fstExecInd].execCode;
                fstExecAppType = drand48() < 0.5 ? 1 : 0;
                if (! execArr[fstExecInd].execDel) {
                        exec sql insert into approvals (
                                appLoanId, appExecCode, appType
                        ) values (
                                :loanId, :fstExecCode, :fstExecAppType
                        );
                }

                do {
                        secExecInd = (int)(lrand48() % N_EXEC);
                } while (secExecInd == fstExecInd);
                secExecCode = execArr[secExecInd].execCode;
                secExecAppType = fstExecAppType == 1 ? 0 : 1;
                if (! execArr[secExecInd].execDel) {
                        exec sql insert into approvals (
                                appLoanId, appExecCode, appType
                        ) values (
                                :loanId, :secExecCode, :secExecAppType
                        );
                }
        }

        // Commit database changes

        exec sql commit work;

        // Exit gracefully

        return 0;

db_error:
        ErrFun();
}
```