```
// CashierGen.cpp - CASHIER ACTIVITY GENERATOR
//
// MODULE INDEX
// NAME                         CONTENTS
// LeapYear                     Test for a leap year
// YearToDayNo                  Convert year number into an abstract day number
// ConvertDate                  Convert a date string into the internal format
// LocEmulator_c::LocEmit       Emit a transaction
// LocEmulator_c::LocInit       Initialise the location emulator
// LocEmulator_c::LocCycle      Cycle the location emulator
// main                         Main line
//
// MAINTENANCE HISTORY
// DATE         PROGRAMMER AND DETAILS
// 24-09-06     JS      Original
// 31-10-06     KWP     Added IniRand, Drand and Lrand
// 31-10-06     KWP     Decrease credits and sales values
// 09-11-06     JS      Use rand for both Windows and Linux
//
//-----------------------------------------------------------------------------

#include <iostream>             // C++ input/output streams
#include <iomanip>              // C++ input/output manipulators
#include <sstream>              // C++ string stream declarations
#include <cstdlib>              // C-style standard library
#include <cstring>              // C-style string manipulation functions
#include <ctime>                // C-style time functions
using namespace std;            // Expand standard namespace to global scope
exec sql include sqlca;         // Include SQL communications area

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

// DEFINITIONS

static const long       FIRST_TRANS_NO = 85179232;
                                // First transaction number
static const char       START_DATE[] = "2005-01-01";
                                // Start date
static const char       END_DATE[] = "2006-11-11";
                                // End date
static const size_t     DATE_LEN = 10;
                                // Length of the date string
static const size_t     LOC_CODE_LEN = 6;
                                // Length of a cashier location code
static const double     PROB_SKIPPED_SHIFT = 0.33;
                                // Probability of a shift being skipped
static const long       MIN_INIT_BAL = 10;
                                // Minimum initial balance
static const long       MAX_INIT_BAL = 200;
                                // Maximum initial balance
static const long       MIN_INTER_TRANS_TIME = 1 * 60;
                                // Minimum inter-transaction time
static const long       MAX_INTER_TRANS_TIME = 10 * 60;
                                // Maximum inter-transaction time
static const long       MIN_SALE_VAL = 20;
                                // Minimum sale value
static const long       MAX_SALE_VAL = 30;
                                // Maximum sale value
static const long       MIN_CREDIT_VAL = 10;
                                // Minimum credit value
static const long       MAX_CREDIT_VAL = 100;
                                // Maximum credit value

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

// TRANSACTION TYPE CODES
```

```
static const short      TT_OPENING_BALANCE = 1;
static const short      TT_FILL = 2;
static const short      TT_CREDIT = 3;
static const short      TT_SALE = 4;
static const short      TT_CLOSING_BALANCE = 5;

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

// CASHIER LOCATION CODES

static const char *LOC_CODE_ARR[] = {
        "FF-001", "FF-002", "FF-003", "FF-004", "FF-005",
        "FF-006", "FF-007", "FF-008", "FF-009", "FF-010",
        "GB-001", "GB-002", "GB-003", "GB-004", "GB-005",
        "GB-006", "GB-007", "GB-008", "GB-009", "GB-010",
        "CP-001", "CP-002", "CP-003", "CP-004", "CP-005",
        "CP-006", "CP-007", "CP-008", "CP-009", "CP-010",
};
static const size_t LOC_CODE_CNT =
        sizeof(LOC_CODE_ARR) / sizeof(LOC_CODE_ARR[0]);

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

// SHIFT PARAMETERS

struct ShiftParms_t {
        time_t          spMinOpenTime;  // Minimum open time
        time_t          spMaxOpenTime;  // Maximum open time
};
static const ShiftParms_t SHIFT_PARMS_ARR[] = {
        {5L*3600L,      7L*3600L},
        {13L*3600L,     15L*3600L},
        {21L*3600L,     23L*3600L}
};
static const size_t SHIFT_PARMS_CNT =
        sizeof(SHIFT_PARMS_ARR) / sizeof(SHIFT_PARMS_ARR[0]);

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

// LOCATION EMULATOR CLASS

class LocEmulator_c {

        // Location State

        enum LocState_t {
                LOC_WAIT_OPEN,  // Waiting for the cashier location to open
                LOC_WAIT_TRANS, // Waiting for the next transaction time
                LOC_WAIT_CLOSE, // Waiting for the cashier location to close
                LOC_WAIT_SKIP,  // Waiting to skip a shift
                LOC_DONE        // All data is generated for the location
        };

        // Class Instance Variables

        const char      *locCode;       // Location code
        LocState_t      locState;       // Location state
        time_t          locNextTime;    // Next activity time
        time_t          locCloseTime;   // Next shift closure time
        time_t          locDate;        // Current date
        size_t          locShiftNo;     // Shift number
        long            locBalance;     // Current balance at the location

        // Private Methods
```

```
        void LocEmit (long transNo, short transType, long transVal);
                                // Emit a transaction

        // Public Methods
public:
        void LocInit (const char *locCode);
                                // Initialise the location emulator
        void LocCycle ();
                                // Process the next activity (time is
                                // locNextTime)
        bool LocIsDone () { return locState == LOC_DONE; }
                                // Test whether the location is finished
        time_t LocGetNextTime() { return locNextTime; };
                                // Get the time of the next activity
};

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

// GLOBAL DATA

long            nextTransNo;    // Next transaction number
time_t          endDate;        // End date

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

// TEST FOR A LEAP YEAR

inline bool
LeapYear (
        long            y)              // Year number (e.g. 2004)
{
        return y % 4 == 0 && (y % 100 != 0 || y % 400 == 0);
}

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

// CONVERT YEAR NUMBER INTO AN ABSTRACT DAY NUMBER

inline time_t
YearToDayNo (
        long            y)              // Year number (e.g. 2004)
{
        return ((y-1L)*365L + (y-1)/4 - (y-1)/100 + (y-1)/400);
}

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

// CONVERT A DATE STRING INTO THE INTERNAL FORMAT

time_t
ConvertDate (
        const char      *sDate)         // String date (YYYY-MM-DD format)
{
        int             year;           // Year
        int             month;          // Month
        int             day;            // Day
        const char      *p;             // General purpose pointer
        time_t          jd;             // Julian day
        const int       *dayMap;        // Pointer to ORD_MAP or LEAP_MAP

        // Map month to number of days before the first of the month

        const static int ORD_MAP[13] =
                {0, 31, 59, 90, 120, 151, 181, 212, 243, 273, 304, 334, 365};
        const static int LEAP_MAP[13] =
                {0, 31, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335, 366};
```

```
        // Parse and decode the date string

        p = sDate;
        year = 0;
        if ( ! isdigit(*p)) goto BadDate;
        while (isdigit(*p)) {
                year = year * 10 + *p - '0';
                p ++ ;
        }
        if (*p != '-') goto BadDate;
        p ++ ;
        month = 0;
        if ( ! isdigit(*p)) goto BadDate;
        while (isdigit(*p)) {
                month = month * 10 + *p - '0';
                p ++ ;
        }
        if (*p != '-') goto BadDate;
        p ++ ;
        day = 0;
        if ( ! isdigit(*p)) goto BadDate;
        while (isdigit(*p)) {
                day = day * 10 + *p - '0';
                p ++ ;
        }
        if (*p != '\0') goto BadDate;

        // Select the appropriate day map

        dayMap = LeapYear(year) ? LEAP_MAP : ORD_MAP;

        // Validate the month

        if (month < 1 || month > 12) goto BadDate;

        // Validate the day of the month

        if (day < 1 || day > dayMap[month]-dayMap[month-1]) goto BadDate;

        // Do the conversion

        jd = dayMap[month-1] + day - 1;
        return (YearToDayNo(year) + jd - YearToDayNo(1970L)) * 24L*60L*60L;

        // Process invalid date strings
BadDate:
        cerr << "Error: invalid date string: " << sDate << '\n';
        exit (1);
}

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

// EMIT A TRANSACTION

void
LocEmulator_c::LocEmit (
        long            xTransNo,       // Transaction number
        short           xTransType,     // Transaction type code
        long            xTransVal)      // Transaction value
{
        struct tm       *tmPnt;         // Time pointer
        ostringstream   oss;            // Output string stream
        string          str;            // String
        exec sql begin declare section;
                long            transNo;        // Transaction number
```

```
                char            tradingDate[11]; // Trading day date
                char            sqlLocCode[7];  // SQL location code
                short           shiftNo;        // Shift number
                short           transType;      // Transaction type code
                long            transVal;       // Transaction value
        exec sql end declare section;

        // Jump to DbError whenever an SQL error occurs

        exec sql whenever sqlerror goto DbError;

        // Load SQL variables

        transNo = xTransNo;
        strcpy (sqlLocCode, locCode);
        shiftNo = locShiftNo;
        transType = xTransType;
        transVal = xTransVal;

        // Load the trading date

        tmPnt = gmtime (&locDate);
        if (tmPnt == 0) {
                cerr << "Error: gmTime failed: " << strerror(errno) << '\n';
                exit (1);
        }
        oss.str ("");
        oss << setfill('0');
        oss << setw(4) << tmPnt->tm_year + 1900;
        oss << '-';
        oss << setw(2) << tmPnt->tm_mon + 1;
        oss << '-';
        oss << setw(2) << tmPnt->tm_mday;
        str = oss.str();
        strcpy (tradingDate, str.c_str());

        // Insert the database row

        exec sql insert into transactions (
                transNo, tradingDate, locCode, shiftNo,
                transType, transVal
        ) values (
                :transNo, :tradingDate, :sqlLocCode, :shiftNo,
                :transType, :transVal
        );
        return;

        // Process a database error

DbError:
        cerr << "Error: SQLCODE=" << SQLCODE << '\n';
        exit (1);
}

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

// INITIALISE THE LOCATION EMULATOR

void
LocEmulator_c::LocInit (
        const char      *code)          // Location code
{
        const ShiftParms_t *sp;         // Shift parameters pointer

        // Initialise the class instance variables
```

```
        locCode = code;
        locDate = ConvertDate (START_DATE);
        locShiftNo = 1;

        // Load start and end times for the first shift

        sp = &SHIFT_PARMS_ARR[0];
        locNextTime = locDate + rand() % (sp->spMaxOpenTime
                - sp->spMinOpenTime) + sp->spMinOpenTime;
        sp = &SHIFT_PARMS_ARR[1];
        locCloseTime = locDate + rand() % (sp->spMaxOpenTime
                - sp->spMinOpenTime) + sp->spMinOpenTime;

        // Initialise for shifts that are skipped

        if (rand() / (RAND_MAX+1.0) < PROB_SKIPPED_SHIFT) {
                locNextTime = locCloseTime;
                locState = LOC_WAIT_SKIP;
        }

        // Initialise for shifts that are not skipped

        else {
                locState = LOC_WAIT_OPEN;
        }

        // Load the initial balance

        locBalance = rand() % (MAX_INIT_BAL-MIN_INIT_BAL+1) + MIN_INIT_BAL;
}

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

// CYCLE THE LOCATION EMULATOR

void
LocEmulator_c::LocCycle ()
{
        const ShiftParms_t *sp;         // Shift parameters pointer
        long            interTransTime; // Inter-transaction time
        long            r;              // A pseudo-random variable
        short           transType;      // Transaction type
        long            transVal;       // Transaction value

        // Process the cycle according to the current state

        switch (locState) {

        // Waiting for the shift to open

        case LOC_WAIT_OPEN:

                // Emit an opening balance row

                LocEmit (nextTransNo++, TT_OPENING_BALANCE, locBalance);

                // Schedule the first transaction

                goto ScheduleTrans;

        // Waiting for the next transaction

        case LOC_WAIT_TRANS:

                // Select the transaction type and value
```

```
        r = rand() % 1000;
        if (r < 980) {
                transType = TT_SALE;
                transVal = rand() % (MAX_SALE_VAL-MIN_SALE_VAL+1)
                        + MIN_SALE_VAL;
                locBalance += transVal;
        } else if (r < 995) {
                transType = TT_CREDIT;
                transVal = locBalance * 90 / 100;
                locBalance -= transVal;
        } else {
                transType = TT_FILL;
                transVal = rand() % (MAX_CREDIT_VAL-MIN_CREDIT_VAL+1)
                        + MIN_CREDIT_VAL;
                locBalance += transVal;
        }

        // Emit the transaction row

        LocEmit (nextTransNo++, transType, transVal);
        // FALLTHROUGH

// Schedule the next transaction

ScheduleTrans:

        // Calculate the inter-transaction time

        interTransTime = rand() % (MAX_INTER_TRANS_TIME
                - MIN_INTER_TRANS_TIME + 1) + MIN_INTER_TRANS_TIME;

        // If the inter-transaction time is before the end of the
        // shift, wait for the inter-transaction time to elapse

        if (locNextTime + interTransTime < locCloseTime) {
                locNextTime += interTransTime;
                locState = LOC_WAIT_TRANS;
        }

        // If the inter-transaction time is after the end of the
        // shift, wait for the end of the shift

        else {
                locNextTime = locCloseTime;
                locState = LOC_WAIT_CLOSE;
        }
        break;

// Waiting for closure of the shift

case LOC_WAIT_CLOSE:

        // Emit the closing balance row

        LocEmit (nextTransNo++, TT_CLOSING_BALANCE, locBalance);
        // FALLTHROUGH

// Waiting for skipping of a shift

case LOC_WAIT_SKIP:

        // Move to the next shift or day

        locShiftNo ++ ;
        if (locShiftNo > SHIFT_PARMS_CNT) {
                locShiftNo = 1;
```

```
                        locDate += 24L*60L*60L;
                }

                // If the emulation is not yet finished, select the close
                // time for the shift

                if (locDate < endDate) {

                        // Select the close time for the shift

                        sp = &SHIFT_PARMS_ARR[locShiftNo-1];
                        locCloseTime = locDate + rand() % (sp->spMaxOpenTime
                                - sp->spMinOpenTime) + sp->spMinOpenTime;

                        // Initialise for shifts that are skipped

                        if (rand() / (RAND_MAX+1.0) < PROB_SKIPPED_SHIFT) {
                                locNextTime = locCloseTime;
                                locState = LOC_WAIT_SKIP;
                        }

                        // Initialise for shifts that are not skipped

                        else {
                                locState = LOC_WAIT_OPEN;
                        }
                }

                // If the emulation is finished, switch to the finished
                // state

                else {
                        locState = LOC_DONE;
                }
                break;

        // Emulation finished

        case LOC_DONE:
                break;
        }
}

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

// MAIN LINE

int
main ()
{
        LocEmulator_c   locArr[LOC_CODE_CNT];   // Location emulator array
        size_t          i;                      // General purpose index
        bool            nextKnown;              // Next time known flag
        time_t          nextTime;               // Time of the next cycle
        size_t          nextLoc;                // Next location index

        // Connect to the database

        exec sql connect to cashdb;

        // Drop table

        exec sql whenever sqlerror continue;
        exec sql drop table transactions;
        exec sql commit work;
```

```
        // Jump to DbError whenever an SQL error occurs

        exec sql whenever sqlerror goto DbError;

        // Create the database tables

        // Transaction Table

        exec sql create table transactions (
                transNo         integer not null,
                tradingDate     char(10) not null,
                locCode         char(6) not null,
                shiftNo         smallint not null,
                transType       smallint not null,
                transVal        integer not null
        ) not logged initially;
        exec sql create unique index transNoInd on
                transactions (transNo);

        // Initialise the random number generator

        srand (20360L);

        // Initialise the next transaction number and end date

        nextTransNo = FIRST_TRANS_NO;
        endDate = ConvertDate (END_DATE);

        // Initialise each of the location emulators

        for (i = 0; i < LOC_CODE_CNT; i++)
                locArr[i].LocInit (LOC_CODE_ARR[i]);

        // Execute cycles until all the location emulators are finished

        for (;;) {
                nextKnown = 0;
                for (i = 0; i < LOC_CODE_CNT; i++) {
                        if (
                                ! locArr[i].LocIsDone() &&
                                (
                                        ! nextKnown ||
                                        locArr[i].LocGetNextTime() < nextTime
                                )
                        ) {
                                nextTime = locArr[i].LocGetNextTime();
                                nextLoc = i;
                                nextKnown = 1;
                        }
                }
                if ( ! nextKnown) break;
                locArr[nextLoc].LocCycle ();
        }

        // And that's all

        exec sql commit work;
        return 0;

        // Process database errors

DbError:
        cerr << "Error: SQLCODE=" << SQLCODE << '\n';
        return 1;
}
```