```
// MytelRep.cpp - MYTEL INTER-CITY TRIP ANALYSIS REPORT
//
// USAGE
// MytelRep from-date to-date
//
// from-date    is the from-date of the reporting period in DD-MM-YY format
// to-date      is the to-date of the reporting period in DD-MM-YY format
//
// 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
// LoadSubMap                   Load subscriber map
// RepRow_t::RepRow_t           Report row constructor
// RepRow_t::op +=              Report row summation
// LoadRepRowArr                Load the report row array
// FormatNum                    Format a number
// ShowRepRow                   Display a report line
// GenRep                       Generate the report
// main                         Main line
//
// MAINTENANCE HISTORY
// DATE         PROGRAMMER AND DETAILS
// 29-08-05     JS      Original
// 09-09-05     PYL     Modified the System Time in GenRep
// 12-09-05     PYL     Initialised fromTime and toTime in LoadSubMap
// 23-09-05     PYL     Corrected the return value in ConvertDate
//
//-----------------------------------------------------------------------------

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

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

// DEFINITIONS

static const size_t     CITY_CODE_LEN = 3;
                                // City code length

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

// CITY CODES

static const char CITY_CODE_ARR [] [CITY_CODE_LEN+1] = {
        "KUL", "SIN", "BKK"
};
static const size_t CITY_CODE_CNT =
        sizeof(CITY_CODE_ARR) / sizeof(CITY_CODE_ARR[0]);

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

// POINT BALANCE RANGES

static const long POINT_RANGE_ARR [] = {
        0, 100000, 300000
};
static const size_t POINT_RANGE_CNT =
        sizeof(POINT_RANGE_ARR) / sizeof(POINT_RANGE_ARR[0]);

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

// SUBSCRIBER MAP TYPE DEFINITION

// Subscriber Map Record Structure

struct SubRec_t {
        size_t          srLastCity;     // Last city index
        long            srCallFreq[CITY_CODE_CNT];
```

```
                                        MytelRep
                                        // Call frequencies
        long            srTripFreq[CITY_CODE_CNT];
                                        // Trip frequencies
};

// Subscriber Map Type

typedef map<long,SubRec_t> SubMap_t;

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

// REPORT ROW STRUCTURE

struct RepRow_t {
        size_t          rrSubCnt;       // Subscriber count
        size_t          rrTripFreq[CITY_CODE_CNT][CITY_CODE_CNT];
                                        // Trip frequencies
        RepRow_t ();
                                        // Constructor
        RepRow_t & operator += (const RepRow_t &otherRow);
                                        // Sumation
};

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

// ERROR EXCEPTION CLASS

struct Error_c {
        // Empty
};

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

// GLOBAL DATA

SubMap_t        subMap;
                                // Subscriber map
RepRow_t        repRowArr[POINT_RANGE_CNT];
                                // Report row array

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

// TEST FOR A LEAP YEAR

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

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

// CONVERT YEAR NUMBER INTO AN ABSTRACT DAY NUMBER

inline long
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

long
ConvertDate (
        const char      *sDate)         // String date
{
        int             year;           // Year
        int             month;          // Month
        int             day;            // Day
        const char      *p;             // General purpose pointer
        long            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;
        day = 0;
        if ( ! isdigit(*p)) throw Error_c ();
        while (isdigit(*p)) {
                day = day * 10 + *p - '0';
                p ++ ;
        }
        if (*p != '-') throw Error_c ();
        p ++ ;
        month = 0;
        if ( ! isdigit(*p)) throw Error_c ();
        while (isdigit(*p)) {
                month = month * 10 + *p - '0';
                p ++ ;
        }
        if (*p != '-') throw Error_c ();
        p ++ ;
        year = 0;
        if ( ! isdigit(*p)) throw Error_c ();
        while (isdigit(*p)) {
                year = year * 10 + *p - '0';
                p ++ ;
        }
        if (*p != '\0') throw Error_c ();

        // Convert the year into the 4-digit format

        if (year < 70)
                year = 2000 + year;
        else if (year < 100)
                year = 1900 + year;
        else
                throw Error_c ();

        // Select the appropriate day map

        dayMap = LeapYear(year) ? LEAP_MAP : ORD_MAP;

        // Validate the month

        if (month < 1 || month > 12) throw Error_c ();

        // Validate the day of the month

        if (day < 1 || day > dayMap[month]-dayMap[month-1]) throw Error_c ();

        // Do the conversion

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

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

// LOAD SUBSCRIBER MAP

void
LoadSubMap (
        long            xFromTime,      // From-time
        long            xToTime)        // To-time
{
        size_t          i;              // General purpose index
        SubMap_t::iterator subIt;       // Subscriber iterator
        size_t          cityNo;         // City number
        exec sql begin declare section;
                long            fromTime;       // From-time
                long            toTime;         // To-time
                long            subId;          // Subscriber identifier
                char            fromCity[3+1];  // From-city
                long            connectTime;    // Connection time
        exec sql end declare section;
```

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

        exec sql whenever sqlerror goto DbError;

        // Process each row in the call register

        fromTime = xFromTime;
        toTime = xToTime;

        exec sql declare callRegCur cursor for
                select  crSubId, crFromCity, crConnectTime
                from    callRegister
                where   crConnectTime >= :fromTime and
                        crConnectTime < :toTime
                order   by crConnectTime;
        exec sql open callRegCur;
        for (;;) {
                exec sql fetch  callRegCur
                        into    :subId, :fromCity, :connectTime;
                if (SQLCODE != 0) break;

                // Look-up the city number

                cityNo = 0;
                while (
                        cityNo < CITY_CODE_CNT &&
                        strcmp (CITY_CODE_ARR[cityNo], fromCity) != 0
                ) cityNo ++ ;

                // If the city code is not recognised, quitely ignore the
                // call register row.

                if (cityNo >= CITY_CODE_CNT) continue;

                // Look-up any Subscriber Map row

                subIt = subMap.find (subId);

                // If no Subscriber Map row exists, create a new one.

                if (subIt == subMap.end()) {
                        subMap[subId] = SubRec_t ();
                        subIt = subMap.find (subId);
                        for (i = 0; i < CITY_CODE_CNT; i++) {
                                subIt->second.srCallFreq[i] = 0;
                                subIt->second.srTripFreq[i] = 0;
                        }
                }

                // If a Subscriber Map row exists, check for new trips

                else {
                        if (cityNo != subIt->second.srLastCity)
                                subIt->second.srTripFreq[cityNo] ++ ;
                }

                // Increment the call frequency for the city and
                // set the last city number for the next call made by this
                // customer.

                subIt->second.srCallFreq[cityNo] ++ ;
                subIt->second.srLastCity = cityNo;
        }
        exec sql close callRegCur;
        return;

        // Process database errors

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

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

// REPORT ROW CONSTRUCTOR

RepRow_t::RepRow_t ()
{
```

```
        size_t          j, k;           // General purpose indices

        rrSubCnt = 0;
        for (j = 0; j < CITY_CODE_CNT; j++)
                for (k = 0; k < CITY_CODE_CNT; k++)
                        rrTripFreq[j][k] = 0;
}

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

// REPORT ROW SUMMATION

RepRow_t &
RepRow_t::operator += (
        const RepRow_t  &otherRow)      // Row to be added to this row
{
        size_t          j, k;           // General purpose indices

        rrSubCnt += otherRow.rrSubCnt;
        for (j = 0; j < CITY_CODE_CNT; j++)
                for (k = 0; k < CITY_CODE_CNT; k++)
                        rrTripFreq[j][k] += otherRow.rrTripFreq[j][k];
        return *this;
}

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

// LOAD THE REPORT ROW ARRAY

void
LoadRepRowArr ()
{
        size_t          i;              // General purpose indices
        SubMap_t::iterator subIt;       // Subscriber iterator
        size_t          rangeNo;        // Point range number
        size_t          baseCityNo;     // Base city no
        SubRec_t        *sr;            // Subscriber record pointer
        RepRow_t        *rr;            // Report row pointer
        exec sql begin declare section;
                long            rrSubId;        // Subscriber identifier
                long            pointBal;       // Point balance
        exec sql end declare section;

        // Vector to DbError whenever an SQL error occurs

        exec sql whenever sqlerror goto DbError;

        // Reset the report row array

        for (i = 0; i < POINT_RANGE_CNT; i++)
                repRowArr[i] = RepRow_t();

        // Process each Subscriber Map relation

        for (subIt = subMap.begin(); subIt != subMap.end(); subIt++) {

                // Load the subscriber identifier and the subscriber
                // record pointer.

                rrSubId = subIt->first;
                sr = &subIt->second;

                // Look up the subscriber's point balance

                exec sql select subBal
                        into    :pointBal
                        from    subscribers
                        where   subId = :rrSubId;
                if (SQLCODE != 0) goto DbError;

                // Convert the point balance into a range number

                rangeNo = 0;
                while (
                        rangeNo < POINT_RANGE_CNT-1 &&
                        pointBal >= POINT_RANGE_ARR[rangeNo+1]
                ) rangeNo ++ ;
                rr = repRowArr + rangeNo;
```

```
                // Determine the subscriber's base city

                baseCityNo = 0;
                for (i = 1; i < CITY_CODE_CNT; i++)
                        if (sr->srCallFreq[i] > sr->srCallFreq[baseCityNo])
                                baseCityNo = i;

                // Add the subscriber to the report row structure

                rr->rrSubCnt ++ ;
                for (i = 0; i < CITY_CODE_CNT; i++)
                        rr->rrTripFreq[baseCityNo][i] += sr->srTripFreq[i];
        }
        return;

        // Process database errors

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

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

// FORMAT A NUMBER

string
FormatNum (
        long            n)              // Number to format
{
        size_t          grpCnt;         // Group count
        char            buf[20], *p;    // Formatting variables

        p = buf + sizeof(buf);
        *--p = '\0';
        grpCnt = 0;
        do {
                if (grpCnt >= 3) {
                        *--p = ',';
                        grpCnt = 0;
                }
                *--p = static_cast <char> (n % 10 + '0');
                grpCnt ++ ;
                n /= 10;
        } while (n != 0);
        return string(p);
}

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

// DISPLAY A REPORT LINE

void
ShowRepRow (
        const string    &rangeDescr,    // Range description
        const RepRow_t  *repRow)        // Report row structure
{
        size_t          i, j;           // General purpose indices
        long            totalTrips;     // Total trips

        // Calculate the total number of trips

        totalTrips = 0;
        for (i = 0; i < CITY_CODE_CNT; i++)
                for (j = 0; j < CITY_CODE_CNT; j++)
                        if (i != j)
                                totalTrips += repRow->rrTripFreq[i][j];

        // Process each base city line

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

                // If this is the first line, display the point range

                if (i == 0) {
                        cout << setw(20) << left << rangeDescr;
                        cout << setw(9) << right;
                        cout << FormatNum(repRow->rrSubCnt);
                        cout << "    ";
```

```
                } else {
                        cout << setw(32) << " ";
                }

                // Display the city code of the base city

                cout << setw(3) << CITY_CODE_ARR[i];

                // Display the trip frequencies to each other city

                for (j = 0; j < CITY_CODE_CNT; j++) {
                        cout << "  ";
                        cout << setw(9) << right;
                        if (j != i)
                                cout << FormatNum(repRow->rrTripFreq[i][j]);
                        else
                                cout << "0";
                }

                // If this is the first line, display the total number of
                // trips.

                if (i == 0) {
                        cout << "  ";
                        cout << setw(9) << right;
                        cout << FormatNum(totalTrips);
                }
                cout << '\n';
        }
}

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

// GENERATE THE REPORT

void
GenRep (
        const char      *fromDate,      // From date
        const char      *toDate)        // To date
{
        size_t          i;              // General purpose indices
        time_t          sysTime;        // Current system time
        struct tm       localTm;        // Local time
        ostringstream   rangeDescr;     // Range description
        RepRow_t        totalRow;       // Total row

        // Display the title

        sysTime = time(0);
        localTm = *localtime(&sysTime);
        cout << right << setfill('0');
        cout << setw(2) << localTm.tm_mday;
        cout << '-';
        cout << setw(2) << localTm.tm_mon + 1;
        cout << '-';
        cout << setw(2) << localTm.tm_year % 100;
        cout << ' ';
        cout << setw(2) << localTm.tm_hour;
        cout << ':';
        cout << setw(2) << localTm.tm_min;
        cout << setfill(' ') << setw(10) << ' ';
        cout << "INTER-CITY TRIP ANALYSIS REPORT";
        cout << setw(19) << ' ';
        cout << "PAGE 1\n";

        // Display the date range

        cout << setw(27) << " " << "for " << fromDate;
        cout << " to " << toDate << '\n' << '\n';

        // Display the column titles

        cout << setw(20) << " " << "Number of   Base        Number of trips to";
        cout << "         Total no\n";
        cout << setw(18) << left << "Point range";
        cout << "subscribers   city";
        for (i = 0; i < CITY_CODE_CNT; i++)
                cout << setw(10) << right << CITY_CODE_ARR[i] << ' ';
        cout << "  of trips\n\n";
```

```
        // Display the report lines

        for (i = 0; i < POINT_RANGE_CNT; i++) {
                rangeDescr.str("");
                rangeDescr << FormatNum(POINT_RANGE_ARR[i]);
                if (i != POINT_RANGE_CNT-1)
                        rangeDescr << " to " << FormatNum(POINT_RANGE_ARR[i+1]);
                else
                        rangeDescr << " or more";
                ShowRepRow (rangeDescr.str(), &repRowArr[i]);
                cout << '\n';

                // Accumulate the total row

                totalRow += repRowArr[i];
        }

        // Display the total row

        ShowRepRow ("Total", &totalRow);
}

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

// MAIN LINE

int
main (
        int             argc,           // Argument count
        char            *argv[])        // Argument value pointers
{
        long            fromTime;       // From-time
        long            toTime;         // To-time

        // Validate the argument count

        if (argc != 3) {
                cerr << "Usage: MytelRep from-date to-date\n";
                exit (1);
        }

        // Decode the from-date

        try {
                fromTime = ConvertDate (argv[1]);
        } catch (Error_c) {
                cerr << "Error: invalid from-date '" << argv[1] << "'\n";
                exit (1);
        }

        // Decode the to-date

        try {
                toTime = ConvertDate (argv[2]) + 24L*60L*60L;
        } catch (Error_c) {
                cerr << "Error: invalid to-date '" << argv[2] << "'\n";
                exit (1);
        }

        // Vector to DbError whenever an SQL error occurs

        exec sql whenever sqlerror goto DbError;

        // Connect to the Customer Database

        exec sql connect to mytel;

        // Load the Subscriber Map

        LoadSubMap (fromTime, toTime);

        // Load the Report Row Array

        LoadRepRowArr ();

        // Generate the report

        GenRep (argv[1], argv[2]);
```

```
        // And that's all

        return 0;

        // Process database errors

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