```
// TrioRep.cpp - FINANCIAL STATEMENTS
//
// USAGE
// TrioRep date
//
// date         is the date of the report in DD-MM-YY format.
//
// MODULE INDEX
// NAME                 CONTENTS
// LeapYear             Test for a leap year
// UserToDbDate         Convert a user date to a database date
// Title                Emit a page title
// FormatNum            Format a number
// CalcBackBal          Calculate the backdated account balances
// CalcLineBal          Calculate the balance for a particular line number
// main                 Main line
//
// MAINTENANCE HISTORY
// DATE         PROGRAMMER AND DETAILS
// 17-08-07     JS      Original
//
//-----------------------------------------------------------------------------

#include <iostream>             // C++ input/output streams
#include <iomanip>              // C++ input/output manipulators
#include <sstream>              // C++ string stream declarations
#include <map>                  // C++ map declarations
#include <set>                  // C++ set declarations
#include <vector>               // C++ vector declarations
#include <cstdlib>              // C-style standard library
#include <cstring>              // C-style string manipulation functions
#include <ctime>                // C-style time functions
#include <cctype>               // C-style character typing functions
#include <cmath>                // C-style mathematical functions
using namespace std;            // Expand the standard namespace
exec sql include sqlca;         // Include the SQL communications area

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

// DEFINITIONS

static const size_t     USER_DATE_LEN = 8;
                                // User-format date string length
typedef char            UserDate_t[USER_DATE_LEN+1];
                                // User-format date type
static const int        LINES_PER_PAGE = 55;
                                // Useable lines per page

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

// LINE TYPE CODES

static const short      LINE_TYPE_BALANCE = 0;
static const short      LINE_TYPE_BLANK = 1;
static const short      LINE_TYPE_SINGLE = 2;
static const short      LINE_TYPE_DOUBLE = 3;

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

// ACCOUNT BALANCE MAP

typedef map<string,double> AccBalMap_t;
typedef AccBalMap_t::iterator AccBalIter_t;
```

```
//----------------------------------------------------------------------------

// LINE NUMBER SET

typedef set<short> LineNoSet_t;
typedef LineNoSet_t::iterator LineNoIter_t;

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

// GLOBAL DATA

const char      *userDate;      // User format date of report
int             pageNo;         // Page number
int             linesRem;       // Lines remaining on the current page
AccBalMap_t     accBalMap;      // Account balance map
exec sql begin declare section;
        char            dbDate[10+1];   // Database format date of report
exec sql end declare section;

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

// TEST FOR A LEAP YEAR

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

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

// CONVERT A USER DATE TO A DATABASE DATE

void
UserToDbDate (
        char            *dbDate,        // Database date string
        const char      *userDate)      // User date string
{
        int             yr, mo, dy;     // Date components
        const char      *p;             // Decoding pointer
        int             daysInThisMonth; // Days in this month
        ostringstream   oss;            // Output string stream
        string          ostr;           // Output string

        // Days per month

        static const int DAYS_PER_MONTH[] =
                {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

        // Decode the user date

        p = userDate;
        dy = 0;
        while (isdigit(*p)) dy = dy * 10 + *p++ - '0';
        if (*p++ != '-') goto BadDate;
        mo = 0;
        while (isdigit(*p)) mo = mo * 10 + *p++ - '0';
        if (*p++ != '-') goto BadDate;
        yr = 0;
        while (isdigit(*p)) yr = yr * 10 + *p++ - '0';
        if (*p != '\0') goto BadDate;

        // Validate the date
```

```
        if (yr < 0 || yr > 99) goto BadDate;
        if (mo < 1 || mo > 12) goto BadDate;
        if (LeapYear(yr) && mo == 2)
                daysInThisMonth = 29;
        else
                daysInThisMonth = DAYS_PER_MONTH[mo-1];
        if (dy < 1 || dy > daysInThisMonth) goto BadDate;

        // Convert to a 4-digit year

        if (yr < 70)
                yr += 2000;
        else
                yr += 1900;

        // Reformat the date in the internal format

        oss << setfill('0');
        oss << setw(4) << yr << '-' << setw(2) << mo << '-' << setw(2) << dy;
        ostr = oss.str();
        strcpy (dbDate, ostr.c_str());
        return;

        // Report an invalid date
BadDate:
        cerr << "Error: invalid date\n";
        exit (1);
}

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

// EMIT A PAGE TITLE

void
Title ()
{
        time_t          sysTime;        // Current system time
        struct tm       localTm;        // Local time

        // Increment the page number

        pageNo ++ ;

        // If not the first page, separate the pages with a form-feed

        if (pageNo != 1) cout << '\f';

        // 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 << "     ";
        cout << "FINANCIAL STATEMENTS";
```

```
        cout << setfill(' ') << setw(13) << ' ';
        cout << "PAGE " << setw(3) << pageNo << '\n';

        // Display the statement date

        cout << setw(22) << " " << "as at " << userDate << '\n' << '\n';

        // Display the column titles

        cout << "Line\n";
        cout << "No" << setw(5) << ' ' << "Description" << setw(34) << ' '
                << "Balance" << '\n' << '\n';

        // Reset the number of data lines remaining on the page

        linesRem = LINES_PER_PAGE;
}

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

// FORMAT A NUMBER

string
FormatNum (
        double          n)              // Number to format
{
        double          prevN;          // Previous value of n
        size_t          grpCnt;         // Group count
        char            buf[20], *p;    // Formatting variables
        bool            negative;       // Number is negative flag

        if (n > 1.0e10) {
                cerr << "Error: numeric overflow\n";
                exit (0);
        }

        p = buf + sizeof(buf);
        *--p = '\0';
        negative = 0;
        if (n < 0) {
                negative = 1;
                n = -n;
        }
        prevN = n;
        modf (n/10, &n);
        *--p = static_cast <char> (prevN - 10*n + '0');
        prevN = n;
        modf (n/10, &n);
        *--p = static_cast <char> (prevN - 10*n + '0');
        *--p = '.';
        grpCnt = 0;
        do {
                if (grpCnt >= 3) {
                        *--p = ',';
                        grpCnt = 0;
                }
                prevN = n;
                modf (n/10, &n);
                if (p == buf) {
                        cerr << "Error: number too big\n";
                        exit (1);
                }
                *--p = static_cast <char> (prevN - 10*n + '0');
                grpCnt ++ ;
```

```
        } while (n != 0);
        if (negative) *--p = '-';
        return string(p);
}

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

// CALCULATE THE BACKDATED ACCOUNT BALANCES

void
CalcBackBal ()
{
        exec sql begin declare section;
                double          balance;                // Account balance
                char            accId[10+1];            // Account id
                char            fromAccId[10+1];        // From account id
                char            toAccId[10+1];          // To account id
                double          value;                  // Transaction value
        exec sql end declare section;

        // Jump to DbError whenever an SQL error occurs

        exec sql whenever sqlerror goto DbError;

        // Load the current account balances into the account
        // balance map

        accBalMap.clear();
        exec sql declare accCur cursor for
                select  accId, accBalance
                from    accounts;
        exec sql open accCur;
        for (;;) {
                exec sql fetch  accCur
                        into    :accId, :balance;
                if (SQLCODE != 0) break;
                accBalMap[accId] = balance;
        }
        exec sql close accCur;

        // Roll back the effect of the transactions between the
        // statement date and now

        exec sql declare txCur cursor for
                select  txFromAccId, txToAccId, txValue
                from    transactions
                where   txDate > :dbDate;
        exec sql open txCur;
        for (;;) {
                exec sql fetch  txCur
                        into    :fromAccId, :toAccId, :value;
                if (SQLCODE != 0) break;
                accBalMap[fromAccId] += value;
                accBalMap[toAccId] -= value;
        }
        exec sql close txCur;
        return;

        // Process database errors
DbError:
        cerr << "Error: SQLCODE=" << SQLCODE << '\n';
        exit (1);
}
```

```
//-----------------------------------------------------------------------------

// CALCULATE THE BALANCE FOR A PARTICULAR LINE NUMBER

double
CalcLineBal (
        short           lineNo,         // Line number
        LineNoSet_t     *prevLineSet)   // Set of previous line numbers
{
        double          lineBal;        // Line balance
        vector<short>   otherLineVec;   // Other line vector
        vector<short>   otherSignVec;   // Other line sign vector
        size_t          i;              // General purpose index
        exec sql begin declare section;
                short           thisLineNo;     // This line number
                short           sign;           // Sign of account
                char            lnAccId[10+1];  // Account identifier
                short           otherLineNo;    // Other line number
        exec sql end declare section;

        // Jump to DbError whenever an SQL error occurs

        exec sql whenever sqlerror goto DbError;

        // Load SQL variables

        thisLineNo = lineNo;

        // Add the line number to the line number set

        prevLineSet->insert(thisLineNo);

        // Accumulate balances of accounts

        lineBal = 0;
        exec sql declare eleCur cursor for
                select  eleSign, eleAccId
                from    elements
                where   eleLineNo = :thisLineNo;
        exec sql open eleCur;
        for (;;) {
                exec sql fetch  eleCur
                        into    :sign, :lnAccId;
                if (SQLCODE != 0) break;
                lineBal += sign * accBalMap[lnAccId];
        }
        exec sql close eleCur;

        // Load references to other lines
        // References must be buffered in a vector because
        // if the reference was evaluated while refCur is open
        // a reference to a line containing a reference would
        // result in opening a cursor that is already open.

        exec sql declare refCur cursor for
                select  refSign, refOtherLineNo
                from    references
                where   refThisLineNo = :thisLineNo;
        exec sql open refCur;
        for (;;) {
                exec sql fetch  refCur
                        into    :sign, :otherLineNo;
                if (SQLCODE != 0) break;
                otherSignVec.push_back(sign);
```

```
                otherLineVec.push_back(otherLineNo);
        }
        exec sql close refCur;

        // Accumulate references to other lines

        for (i = 0; i < otherLineVec.size(); i++) {

                // Check for references to self

                if (prevLineSet->count(otherLineVec[i]) != 0) {
                        cerr << "Error: self-referencing line reference\n";
                        exit (1);
                }

                // Accumulate the value of the line

                lineBal += otherSignVec[i] *
                        CalcLineBal (otherLineVec[i], prevLineSet);
        }

        // Drop the line from the previous line set

        prevLineSet->erase (thisLineNo);

        // Return the calculated line balance

        return lineBal;

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

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

// MAIN LINE

int
main (
        int             argc,           // Argument count
        char            *argv[])        // Argument value vector
{
        LineNoSet_t     prevLineSet;    // Previous line set
        exec sql begin declare section;
                short           lineNo;         // Line number
                char            lineDescr[40+1]; // Line description
                short           lineType;       // Line type
        exec sql end declare section;

        // Jump to DbError whenever an SQL error occurs

        exec sql whenever sqlerror goto DbError;

        // Decode arguments

        if (argc != 2) {
                cerr << "Usage: TrioRep date\n";
                return 1;
        }
        userDate = argv[1];
        UserToDbDate (dbDate, userDate);
```

```
        // Connect to the database

        exec sql connect to triodb;

        // Calculate backdated balances

        CalcBackBal ();

        // Show the first page title

        Title ();

        // Emit the lines in the report

        exec sql declare lineCur cursor for
                select  lineNo, lineDescr, lineType
                from    lines
                order   by lineNo;
        exec sql open lineCur;
        for (;;) {
                exec sql fetch  lineCur
                        into    :lineNo, :lineDescr, :lineType;
                if (SQLCODE != 0) break;

                cout << setfill('0') << setw(4) << lineNo << "   ";
                cout << setfill(' ') << left << setw(42) << lineDescr << right;
                switch (lineType) {
                case LINE_TYPE_BALANCE:
                        cout << setw(10) <<
                                FormatNum (CalcLineBal (lineNo, &prevLineSet));
                        break;
                case LINE_TYPE_BLANK:
                        break;
                case LINE_TYPE_SINGLE:
                        cout << "----------";
                        break;
                case LINE_TYPE_DOUBLE:
                        cout << "==========";
                        break;
                }
                cout << '\n';
        }
        exec sql close lineCur;


        // And that's all

        exec sql rollback work;
        return 0;

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