```
// CashierRep.cpp - CASHIER ACTIVITY REPORT
//
// USAGE
// CashierRep from-date to-date
//
// from-date    is the from-date in DD-MM-YY format.
// to-date      is the to-date in DD-MM-YY format.
//
// MODULE INDEX
// NAME                          CONTENTS
// Date_c::DateDaysInMonth      Return the number of days in a month
// Date_c::DateIsValid          Test whether the date components are valid
// Date_c::DateFromISO          Load a date from an iso-8601 format string
// Date_c::DateFromUser         Load a date from a user format string
// Date_c::DateToUser           Convert a date into a user-format string
// Date_c::op <                 Compare dates
// Date_c::op --                Postfix decrement operator
// Date_c::op ++                Postfix increment operator
// RowKey_t::op <               Compare report row keys
// Title                        Emit a page title
// FormatNum                    Format a number
// main                         Main line
//
// MAINTENANCE HISTORY
// DATE         PROGRAMMER AND DETAILS
// 24-09-06     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 <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 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

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

// 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;

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

// DATE CLASS

class Date_c {
```

```
        // Date components

        int             dateYear;       // Year
        int             dateMonth;      // Month
        int             dateDay;        // Day

        // Private Methods

        int DateDaysInMonth (int year, int month);
                                // Return the number of days in a month
        bool DateIsValid ();
                                // Test whether the date components are valid
        // Public methods
public:
        bool DateFromISO (const char *isoDate);
                                // Load the date from an ISO-8601 format
                                // date string (YYYY-MM-DD).  Return 0
                                // if the string is invalid.
        bool DateFromUser (const UserDate_t userDate);
                                // Load the date from a user format date
                                // string (DD-MM-YY).  Return 0 if the string
                                // is invalid.
        char *DateToISO (char *isoDate);
                                // Convert the date to an ISO format string.
        char *DateToUser (UserDate_t userDate) const;
                                // Convert the date to a user format string.
        bool operator < (const Date_c &date) const;
                                // Compare dates
        Date_c &operator -- (int);
                                // Postfix decrement operator
        Date_c &operator ++ (int);
                                // Postfix increment operator
};

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

// REPORT ROW MAP DECLARATIONS

// Key Structure

struct RowKey_t {
        char            rowLocCode[9];  // Location code
        Date_c          rowDate;        // Trading day date
        short           rowShiftNo;     // Shift number

        bool operator < (const RowKey_t &row) const;
                                        // Comparison operator
};

// Value Structure

struct RowVal_t {
        long            rowOpenBal;     // Opening balance
        long            rowFills;       // Total fills
        long            rowCredits;     // Total credits
        long            rowSales;       // Total sales
        long            rowCloseBal;    // Closing balance
        bool            rowClosed;      // Shift closed flag

        RowVal_t () : rowOpenBal(0), rowFills(0), rowCredits(0),
                rowSales(0), rowCloseBal(0), rowClosed(0) {}
                                        // Default constructor
};

// Type Definitions
```

```
typedef map<RowKey_t,RowVal_t> RowMap_t;
                                        // Row map type definition
typedef RowMap_t::iterator RowIter_t;
                                        // Row map iterator type definition

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

// GLOBAL DATA

Date_c          fromDate;       // From date
Date_c          toDate;         // To date
int             pageNo;         // Page number
int             linesRem;       // Lines remaining on the current page

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

// RETURN THE NUMBER OF DAYS IN A MONTH

int
Date_c::DateDaysInMonth (
        int             year,           // Year
        int             month)          // Month
{
        static const int DAYS_IN_MONTH[] =
                {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
                                        // Days in each month

        if (month == 2 && year%4 == 0 && (year%100 != 0 || year%400 == 0))
                return 29;
        return DAYS_IN_MONTH [ month - 1 ];
}

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

// TEST WHETHER THE DATE COMPONENTS ARE VALID

bool
Date_c::DateIsValid ()
{
        if (dateYear < 1000 | dateYear >= 3000) return 0;
        if (dateMonth < 1 || dateMonth > 12) return 0;
        if (dateDay < 1 || dateDay > DateDaysInMonth (dateYear, dateMonth))
                return 0;
        return 1;
}

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

// LOAD A DATE FROM AN ISO-8601 FORMAT STRING

bool
Date_c::DateFromISO (
        const char      *isoDate)
{
        istringstream   iss;            // Input string stream

        iss.str (isoDate);
        iss >> dateYear;
        if (iss.get() != '-') return 0;
        iss >> dateMonth;
        if (iss.get() != '-') return 0;
        iss >> dateDay;
        if (iss.bad() || ! iss.eof()) return 0;
        return DateIsValid ();
}
```

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

// LOAD A DATE FROM A USER FORMAT STRING

bool
Date_c::DateFromUser (
        const UserDate_t userDate)      // User format date string
{
        istringstream   iss;            // Input string stream

        iss.str (userDate);
        iss >> dateDay;
        if (iss.get() != '-') return 0;
        iss >> dateMonth;
        if (iss.get() != '-') return 0;
        iss >> dateYear;
        if (iss.bad() || ! iss.eof()) return 0;
        if (dateYear < 0 || dateYear >= 100) return 0;
        if (dateYear >= 70)
                dateYear += 1900;
        else
                dateYear += 2000;
        return DateIsValid ();
}

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

// CONVERT A DATE INTO A USER-FORMAT STRING

char *
Date_c::DateToUser (
        UserDate_t      userDate)       // User date buffer
const
{
        ostringstream   oss;            // Output string stream
        string          str;            // String instance

        oss << setfill('0');
        oss << setw(2) << dateDay << '-';
        oss << setw(2) << dateMonth << '-';
        oss << setw(2) << (dateYear % 100);
        str = oss.str();
        strcpy (userDate, str.c_str());
        return userDate;
}

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

// COMPARE DATES

bool
Date_c::operator < (
        const Date_c    &date)          // Date to compare to
const
{
        if (dateYear < date.dateYear) return 1;
        if (dateYear > date.dateYear) return 0;
        if (dateMonth < date.dateMonth) return 1;
        if (dateMonth > date.dateMonth) return 0;
        return dateDay < date.dateDay;
}

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

// POSTFIX DECREMENT OPERATOR
```

```
Date_c &
Date_c::operator -- (int)
{
        if (dateDay > 1) {
                dateDay -- ;
        } else {
                if (dateMonth > 1) {
                        dateMonth -- ;
                } else {
                        dateMonth = 12;
                        dateYear -- ;
                }
                dateDay = DateDaysInMonth (dateYear, dateMonth);
        }
        return *this;
}

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

// POSTFIX INCREMENT OPERATOR

Date_c &
Date_c::operator ++ (int)
{
        if (dateDay < DateDaysInMonth (dateYear, dateMonth)) {
                dateDay ++ ;
        } else {
                dateDay = 1;
                if (dateMonth < 12) {
                        dateMonth ++ ;
                } else {
                        dateMonth = 1;
                        dateYear ++ ;
                }
        }
        return *this;
}

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

// COMPARE REPORT ROW KEYS

bool
RowKey_t::operator < (
        const RowKey_t  &key)           // Key to compare to
const
{
        int             c;              // Result of comparison

        c = strcmp (rowLocCode, key.rowLocCode);
        if (c < 0) return 1;
        if (c > 0) return 0;
        if (rowDate < key.rowDate) return 1;
        if (key.rowDate < rowDate) return 0;
        return rowShiftNo < key.rowShiftNo;
}

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

// EMIT A PAGE TITLE

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

```
        UserDate_t      userFromDate;   // User-format from-date
        UserDate_t      userToDate;     // User-format to-date

        // 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 << setfill(' ') << setw(14) << ' ';
        cout << "CASHIER ACTIVITY REPORT";
        cout << setw(21) << ' ';
        cout << "PAGE " << setw(3) << pageNo << '\n';

        // Display the date range

        cout << setw(27) << " " << "for " << fromDate.DateToUser(userFromDate);
        cout << " to " << toDate.DateToUser(userToDate) << '\n' << '\n';

        // Display the column titles

        cout << "Cashier" << setw(20) << ' ' << "Opening"
                << setw(37) << ' ' << "Closing\n";
        cout << "Location Date     Shift    balance"
                "      Fills    Credits      Sales    balance\n\n";

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

        linesRem = LINES_PER_PAGE;
}

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

// FORMAT A NUMBER

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

        p = buf + sizeof(buf);
        *--p = '\0';
        negative = 0;
        if (n < 0) {
                negative = 1;
                n = -n;
        }
```

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

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

// MAIN LINE

int
main (
        int             argc,           // Argument count
        char            *argv[])        // Argument value vector
{
        Date_c          startDate;      // From date minus 1 days
        Date_c          endDate;        // To date plus 2 days
        Date_c          mDate;          // Middle date
        RowMap_t        rowMap;         // Row map
        RowKey_t        rowKey;         // Row key structure
        RowVal_t        *rowVal;        // Pointer to row value
        RowIter_t       rowIter;        // Row iterator
        UserDate_t      userDate;       // User-format date string
        long            closeBal;       // Closing balance
        RowVal_t        totals;         // Totals
        char            prevLocCode[9]; // Previous location code
        bool            prevLocDef;     // Previous location defined flag
        long            prevCloseBal;   // Previous closing balance
        exec sql begin declare section;
                long            transNo;        // Transaction number
                char            isoDate[11];    // ISO format date
                char            locCode[9];     // Location code
                short           shiftNo;        // Shift number
                short           transType;      // Transaction type
                long            transVal;       // Transaction value
                long            lTransNo;       // Low transaction number
                short           lTransNoInd;    // Indicator for lTransNo
                long            hTransNo;       // High transaction number
                long            mTransNo;       // Middle transaction number
                long            xTransNo;       // Test transaction number
                short           xTransNoInd;    // Indicator for xTransNo
        exec sql end declare section;

        // Jump to DbError whenever an SQL error occurs

        exec sql whenever sqlerror goto DbError;

        // Decode arguments

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

```
if ( ! fromDate.DateFromUser (argv[1])) {
        cerr << "Error: invalid from-date\n";
        return 1;
}
if ( ! toDate.DateFromUser (argv[2])) {
        cerr << "Error: invalid to-date\n";
        return 1;
}

// Connect to the database

exec sql connect to cashdb;

// Select the highest transaction number in the table
// plus one.  It is amazing that this is quicker than
// select max(transNo), but it is, by quite a significant
// factor.

hTransNo = 0x7fffffff;
lTransNo = 0;
while (hTransNo > lTransNo) {
        mTransNo = (lTransNo + hTransNo) / 2;
        exec sql select min(transNo)
                into    :xTransNo indicator :xTransNoInd
                from    transactions
                where   transNo >= :mTransNo;
        if (xTransNoInd >= 0)
                lTransNo = xTransNo + 1;
        else
                hTransNo = mTransNo;
}

// Select the lowest transaction number in the table

exec sql select min(transNo)
        into    :lTransNo indicator :lTransNoInd
        from    transactions;
if (lTransNoInd < 0) lTransNo = 0;

// Use a binary search to find the starting point for scanning
// the transactions table.  When the binary search loop exits,
// lTransNo points to a position in the transactions table
// immediately after a transaction with a date less than the
// start date.

startDate = fromDate;
startDate -- ;
while (lTransNo < hTransNo) {
        mTransNo = (lTransNo + hTransNo) / 2;
        exec sql select tradingDate
                into    :isoDate
                from    transactions
                where   transNo = :mTransNo;
        if (SQLCODE != 0) {
                exec sql select tradingDate
                        into    :isoDate
                        from    transactions
                        where   transNo = (
                                        select  min(i.transNo)
                                        from    transactions i
                                        where   transNo > :mTransNo
                                );
                if (SQLCODE != 0) goto DbError;
        }
        mDate.DateFromISO (isoDate);
        if (mDate < startDate)
```

```
                lTransNo = mTransNo + 1;
        else
                hTransNo = mTransNo;
}

// Process transaction data starting from the transaction number
// discovered in the binary search

endDate = toDate;
endDate ++ ;
endDate ++ ;
exec sql declare transCur cursor for
        select  transNo, tradingDate, locCode, shiftNo,
                transType, transVal
        from    transactions
        where   transNo >= :lTransNo
        order   by transNo
        optimize for 20 rows;
exec sql open transCur;
for (;;) {
        exec sql fetch  transCur
                into    :transNo, :isoDate, :locCode, :shiftNo,
                        :transType, :transVal;
        if (SQLCODE != 0) break;

        // Exit the loop when a transaction has a date equal
        // to the to-date plus two days.

        rowKey.rowDate.DateFromISO (isoDate);
        if ( ! (rowKey.rowDate < endDate)) break;

        // If the transaction is in the date range, add it to
        // the row map.

        if (
                ! (rowKey.rowDate < fromDate) &&
                ! (toDate < rowKey.rowDate)
        ) {
                strcpy (rowKey.rowLocCode, locCode);
                rowKey.rowShiftNo = shiftNo;
                rowVal = &rowMap[rowKey];
                switch (transType) {
                case TT_OPENING_BALANCE:
                        rowVal->rowOpenBal = transVal;
                        break;
                case TT_FILL:
                        rowVal->rowFills += transVal;
                        break;
                case TT_CREDIT:
                        rowVal->rowCredits += transVal;
                        break;
                case TT_SALE:
                        rowVal->rowSales += transVal;
                        break;
                case TT_CLOSING_BALANCE:
                        rowVal->rowCloseBal = transVal;
                        rowVal->rowClosed = 1;
                        break;
                }
        }
}
exec sql close transCur;

// Emit the report

prevLocDef = 0;
```

```
pageNo = 0;
Title ();
for (rowIter = rowMap.begin(); rowIter != rowMap.end(); rowIter ++) {

        // Calculate the closing balance

        if (rowIter->second.rowClosed)
                closeBal = rowIter->second.rowCloseBal;
        else
                closeBal = rowIter->second.rowOpenBal
                        + rowIter->second.rowFills
                        - rowIter->second.rowCredits
                        + rowIter->second.rowSales;

        // Emit the report line

        if (linesRem < 1) Title ();
        cout << left << setw(8) << rowIter->first.rowLocCode
                << ' ' << right;
        cout << rowIter->first.rowDate.DateToUser(userDate) << "   ";
        cout << rowIter->first.rowShiftNo << "   ";
        cout << setw(10) << FormatNum(rowIter->second.rowOpenBal)
                << ' ';
        cout << setw(10) << FormatNum(rowIter->second.rowFills)
                << ' ';
        cout << setw(10) << FormatNum(rowIter->second.rowCredits)
                << ' ';
        cout << setw(10) << FormatNum(rowIter->second.rowSales)
                << ' ';
        cout << setw(10) << FormatNum(closeBal);
        cout << '\n';
        linesRem -- ;

        // Accumulate totals

        totals.rowFills += rowIter->second.rowFills;
        totals.rowCredits += rowIter->second.rowCredits;
        totals.rowSales += rowIter->second.rowSales;

        // Update the opening and closing balances

        if (
                ! prevLocDef ||
                strcmp (prevLocCode, rowIter->first.rowLocCode) != 0
        ) {
                totals.rowOpenBal += rowIter->second.rowOpenBal;
                if (prevLocDef)
                        totals.rowCloseBal += prevCloseBal;
                prevLocDef = 1;
                strcpy (prevLocCode, rowIter->first.rowLocCode);
        }
        prevCloseBal = closeBal;
}

// Post the last closing balance

if (prevLocDef)
        totals.rowCloseBal += prevCloseBal;

// Emit the totals

if (linesRem < 3) Title ();
cout << setw(24) << ' ';
cout << "---------- ---------- ---------- ---------- ----------\n";
cout << "TOTAL" << setw(19) << ' ';
cout << setw(10) << FormatNum(totals.rowOpenBal) << ' ';
```

```
        cout << setw(10) << FormatNum(totals.rowFills) << ' ';
        cout << setw(10) << FormatNum(totals.rowCredits) << ' ';
        cout << setw(10) << FormatNum(totals.rowSales) << ' ';
        cout << setw(10) << FormatNum(totals.rowCloseBal) << ' ';
        cout << '\n';
        cout << setw(24) << ' ';
        cout << "========== ========== ========== ========== ==========\n";

        // And that's all

        exec sql commit work;
        return 0;

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