

VendRep.cpp

```

// VendRep.cpp - GENERATE THE VENDING MACHINE RECONCILIATION REPORT
//
// USAGE
// VendRep -D from-date to-date
// from-date      is the reporting period from-date
// to-date        is the reporting period to-date
//
// MODULE INDEX
// NAME           CONTENTS
// FormatNum      Format a number
// main           Main line
//
// MAINTENANCE HISTORY
// DATE          PROGRAMMER AND DETAILS
// 29-08-10      JS      Original
// 06-10-10      VIK      Added fromDate and toDate comparison
// 07-10-10      VIK      Corrected "Meter Mismatch" and added "Account-to-
//                          machine transfer" title
//
//-----

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

//-----

// TRANSACTION TYPE CODES

exec sql begin declare section;
    static const short TX_TYPE_CARD_IN = 1;
        // Transfers from a customer's account to a
        // machine interface
    static const short TX_TYPE_COLLECT_CARD = 2;
        // Transfers from a machine interface to a
        // customer's account
    static const short TX_TYPE_BILL_IN = 3;
        // Insertion of a banknote into the vending
        // machine
    static const short TX_TYPE_SALE = 4;
        // Sale of a product
    static const short TX_TYPE_REVERSAL = 5;
        // Reversal because of communications failure
exec sql end declare section;

//-----

// METER IDENTIFIERS

exec sql begin declare section;
    static const short METER_ID_MONEY_IN = 0;
        // Money transferred in
    static const short METER_ID_MONEY_OUT = 1;
        // Money transferred out

```

VendRep.cpp

```

static const short METER_ID_BILLS_IN = 2;
                        // Bills inserted
static const short METER_ID_SALES = 3;
                        // Sales
exec sql end declare section;

//-----

// GLOBAL DATA

Dtime_t      fromDate;          // From date
Dtime_t      toDate;           // To date

//-----

// 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);
}

```

```
//-----  
  
// MAIN LINE  
  
int  
main (  
    int          argc,          // Argument count  
    char         *argv[]       // Argument value pointers  
{  
    int          c;             // Argument character  
    bool         periodDefined; // Reporting period defined flag  
    char         dateBuf[10];   // Date formatting buffer  
    double       subTotal;      // Sub-total  
  
    exec sql begin declare section;  
        char         fromSqlDate[10+1]; // From SQL date  
        char         toSqlDate[10+1];  // To SQL date  
        double       custBal;          // Customer a/c balance  
        short        custBalInd;       // Indicator for custBal  
        double       prevDay;          // Tx in previous machine day  
        short        prevDayInd;       // Indicator for prevDay  
        double       nextDay;          // Tx in next machine day  
        short        nextDayInd;       // Indicator for nextDay  
        double       reversals;        // Reversals  
        short        reversalsInd;     // Indicator for reversals  
        double       meteredValue;     // Metered value  
        short        meteredValueInd;  // Indicator for meteredValue  
    exec sql end declare section;  
  
    // Connect to the database  
  
    exec sql connect to venddb;  
  
    // Jump to DbError whenever an SQL error occurs  
  
    exec sql whenever sqlerror goto DbError;  
  
    // Decode arguments  
  
    periodDefined = 0;  
    while ((c = getopt (argc, argv, "D:")) != -1) {  
        switch (c) {  
            case 'D':  
                fromDate = DtimeFromUserDate (optarg);  
                if (fromDate == NULL_DTIME) {  
                    cerr << "Error: bad from-date\n";  
                    exit (1);  
                }  
                SqlDateFromDtime (fromSqlDate, fromDate);  
                if (optind >= argc) {  
                    cerr << "Error: missing to-date\n";  
                    exit (1);  
                }  
                toDate = DtimeFromUserDate (argv[optind]);  
                if (toDate == NULL_DTIME) {  
                    cerr << "Error: bad to-date\n";  
                    exit (1);  
                }  
                if (fromDate > toDate) {  
                    cerr << "Error: from-date greater than"  
                        << " to-date\n";  
                    exit (1);  
                }  
            }  
        }  
    }  
}
```

VendRep.cpp

```

        }
        SqlDateFromDtime (toSqlDate, toDate);
        optind ++ ;
        periodDefined = 1;
        break;
    default:
        cerr << "Error: invalid option\n";
        exit (1);
        // NOTREACHED
    }
}
if (optind < argc) {
    cerr << "Error: superfluous arguments\n";
    exit (1);
}

// If no reporting period was defined, it's a problem
if ( ! periodDefined) {
    cerr << "Error: no reporting period\n";
    exit (1);
}

// Show report title
cout << "VENDING MACHINE RECONCILIATION REPORT\n";
cout << "FOR " << UserDateFromDtime (dateBuf, fromDate);
cout << " TO " << UserDateFromDtime (dateBuf, toDate) << "\n\n";
cout << "Account-to-machine transfers" << "\n";

// Show customer account balance
exec sql select sum(txValue)
    into      :custBal indicator :custBalInd
    from      custTx, txData
    where     custDate between :fromSqlDate and :toSqlDate and
            txData.txNo = custTx.txNo and
            txType = :TX_TYPE_CARD_IN;
if (custBalInd < 0) custBal = 0;
cout << left << setw(45) << "      As shown in customer accounts"
    << right << setw(12) << FormatNum (custBal) << '\n';

// Show transactions occurring in the previous machine day
exec sql select sum(txValue)
    into      :prevDay indicator :prevDayInd
    from      custTx, vendTx, txData
    where     custDate between :fromSqlDate and :toSqlDate and
            machDate < :fromSqlDate and
            txData.txNo = custTx.txNo and
            txData.txNo = vendTx.txNo and
            txType = :TX_TYPE_CARD_IN;
if (prevDayInd < 0) prevDay = 0;
cout << left << setw(45) << "      Less transactions occurring in the"
    << right << setw(12) << FormatNum (prevDay) << '\n';
cout << "      previous machine day\n";

// Show transactions logged in the next customer day that
// occurred in the current machine day
exec sql select sum(txValue)
    into      :nextDay indicator :nextDayInd
    from      custTx, vendTx, txData

```

VendRep.cpp

```

        where  custDate > :toSqlDate and
               machDate between :fromSqlDate and :toSqlDate and
               txData.txNo = custTx.txNo and
               txData.txNo = vendTx.txNo and
               txType = :TX_TYPE_CARD_IN;
if (nextDayInd < 0) nextDay = 0;
cout << left << setw(45) << "      Plus transactions logged in the next"
     << right << setw(12) << FormatNum (nextDay) << '\n';
cout << "      customer day that occurred in the\n";
cout << "      current machine day\n";

// Show reversals

exec sql select sum(txValue)
into      :reversals indicator :reversalsInd
from      vendTx, txData
where     machDate between :fromSqlDate and :toSqlDate and
         txData.txNo = vendTx.txNo and
         txType = :TX_TYPE_REVERSAL;
if (reversalsInd < 0) reversals = 0;
cout << left << setw(45) << "      Plus reversed transfers"
     << right << setw(12) << FormatNum (reversals) << '\n';

// Show sub-total

subTotal = custBal - prevDay + nextDay + reversals;
cout << left << setw(45) << " "
     << right << setw(12) << "-----" << '\n';
cout << left << setw(45) << "      Sub-total"
     << right << setw(12) << FormatNum (subTotal) << '\n';

// Show metered value

exec sql select sum(meteredValue)
into      :meteredValue indicator :meteredValueInd
from      vendMeters
where     machDate between :fromSqlDate and :toSqlDate and
         meterId = :METER_ID_MONEY_IN;
if (meteredValueInd < 0) meteredValue = 0;
cout << left << setw(45) << "      Vending machine metered value"
     << right << setw(12) << FormatNum (meteredValue) << '\n';

// Show meter mismatch

if (meteredValue != subTotal) {
    cout << left << setw(45) << " "
         << right << setw(12) << "-----" << '\n';
    cout << left << setw(45) << "      Meter Mismatch"
         << right << setw(12)
         << FormatNum (subTotal-meteredValue) << '\n';
}

return 0;

// Process database errors

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

```