```
// Round.cpp - ROUND AN ARRAY TO EQUAL A ROUNDED TOTAL
//
// MODULE INDEX
// NAME                          CONTENTS
//
// MAINTENANCE HISTORY
// DATE         PROGRAMMER AND DETAILS
// 06-10-04     JS      Original
//
//-----------------------------------------------------------------------------

#include <string.h>             // String manipulation functions
#include <errno.h>              // Operating system error codes
#include <stdlib.h>             // C standard library
#include <iostream>             // C++ input/output streams
#include <iomanip>              // C++ input/output manipulators
#include <sstream>              // C++ string stream declarations
#include <vector>               // C++ vectors
#include <string>               // C++ strings
using namespace std;            // Expand standard namespace to global scope

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

// DEFINITIONS

const long              PRECISION = 100000;
                // Rounding precision

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

// EXECUTE A SIMPLE ROUND

inline long
SimpleRound (
    long                raw)            // Raw statistic
{
    long                rnd;            // Rounded statistic

    if (raw >= 0)
        rnd = (raw + PRECISION/2) / PRECISION;
    else
        rnd = (raw - PRECISION/2) / PRECISION;
    return rnd;
}

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

// ROUND AN ARRAY OF NUMBERS TO EQUAL A ROUNDED TOTAL

void
Round (
    long                *rndArr,        // Rounded array
    const long          *rawArr,        // Raw array
    size_t              cnt)            // Array item count
{
    int                 i;              // General purpose index
    long                rawTot;         // Raw total
    long                rndTot;         // Rounded total
    long                rndSum;         // Sum of the rounded items
    long                errVal;         // Error value
    size_t              minErrInd;      // Minimum error index
    long                minErrVal;      // Minimum error value

    // Calculate the raw total and the unadjusted, rounded
    // elements.

    rawTot = 0;
    for (i = 0; i < cnt; i++) {
        rawTot += rawArr[i];
        rndArr[i] = SimpleRound(rawArr[i]);
    }

    // Calculate the rounded total

    rndTot = SimpleRound(rawTot);

    // Calculate the sum of the rounded elements

    rndSum = 0;
    for (i = 0; i < cnt; i++) rndSum += rndArr[i];

    // Loop through adjustment sequences until the sum of the
```

```
    // rounded elements equals the rounded sum of the unrounded
    // elements.

    while (rndSum > rndTot) {
        minErrInd = 0;
        minErrVal = rawArr[0] - (rndArr[0] - 1)*PRECISION;
        for (i = 1; i < cnt; i++) {
            errVal = rawArr[i] - (rndArr[i] - 1)*PRECISION;
            if (errVal < minErrVal) {
                minErrInd = i;
                minErrVal = errVal;
            }
        }
        rndArr[minErrInd] -- ;
        rndSum -- ;
    }
    while (rndSum < rndTot) {
        minErrInd = 0;
        minErrVal = (rndArr[0] + 1)*PRECISION - rawArr[i];
        for (i = 1; i < cnt; i++) {
            errVal = (rndArr[i] + 1)*PRECISION - rawArr[i];
            if (errVal < minErrVal) {
                minErrInd = i;
                minErrVal = errVal;
            }
        }
        rndArr[minErrInd] ++ ;
        rndSum ++ ;
    }
}

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

// TESTING MAIN LINE

int
main ()
{
    const size_t        CNT = 10;       // Number of elements
    long                rawArr[CNT];    // Raw array
    long                rndArr[CNT];    // Rounded array
    long                rawTot;         // Raw total
    long                rndTot;         // Rounded total
    double              err2;           // Sum of the square of the errors
    double              altErr2;        // Alternate sum of sqare of errors
    size_t              i, j, k;        // General purpose index

    // Execute 10,000 tests

    for (int l = 0; l < 10000; l++) {

        // Generate and round some raw data

        for (i = 0; i < CNT; i++)
            rawArr[i] = lrand48() % 100000000 - 50000000;
        Round (rndArr, rawArr, CNT);

        // Check that the sum of the rounded elements equals the rounded
        // sum of the unrounded elements.

        rawTot = 0;
        rndTot = 0;
        for (i = 0; i < CNT; i++) {
            rawTot += rawArr[i];
            rndTot += rndArr[i];
        }
        if (rndTot != SimpleRound(rawTot)) {
            cerr << "Error: rndTot != SimpleRound(rawTot)\n";
            exit (1);
        }

        // Check that moving a thousand dollars from any rounded
        // element to any other rounded element does not reduce the
        // sum of the square of the errors.

        err2 = 0;
        for (i = 0; i < CNT; i++) {
            err2 += static_cast<double>(rawArr[i] - rndArr[i]*PRECISION)
                * static_cast<double>(rawArr[i] - rndArr[i]*PRECISION);
        }
        for (i = 0; i < CNT; i++) {
            for (j = 0; j < CNT; j++) {
```

```
                rndArr[i] -- ;
                rndArr[j] ++ ;
                for (k = 0; k < CNT; k++) {
                    altErr2 += static_cast<double>
                            (rawArr[k] - rndArr[k]*PRECISION)
                        * static_cast<double>
                            (rawArr[k] - rndArr[k]*PRECISION);
                }
                if (altErr2 < err2) {
                    cerr << "Error: better round exists\n";
                    exit (1);
                }
                rndArr[i] ++ ;
                rndArr[j] -- ;
            }
        }
    }

    cout << "All tests satisfactorily completed\n";
    return 0;
}
```