```
// PokerRep.cpp - GENERATE THE POKER WIN COMPARISON REPORT
//
// USAGE
// PokerRep     path fromRound toRound
// path         is the path name of the folder with the game history
// fromRound    is the first round
// toRound      is the last round
//
// MODULE INDEX
// NAME         CONTENTS
// DcdNum       Decode a number
// main         Main line
//
// MAINTENANCE HISTORY
// DATE     PROGRAMMER AND DETAILS
// 29-09-16 MPF Original
// 01-11-16 SYW Add folder name parameter
//          (WARNING: path in Linux format)
// 05-12-16 WLL Corrected invalid input rejection
//
//-----------------------------------------------------------------------------

#include <vector>       // C++ vector declaration
#include <set>          // C++ set declaration
#include <map>          // C++ map declaration
#include <iostream>     // C++ I/O stream declaration
#include <fstream>      // C++ file stream declaration
#include <iomanip>      // C++ manipulation declaration
using namespace std;

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

// CONSTANTS

const int THREE_OF_KIND_FLUSH = 0;
const int STRAIGHT_FLUSH = 1;
const int THREE_OF_KIND = 2;
const int STRAIGHT = 3;
const int FLUSH = 4;
const int PAIR = 5;
const int HIGH_CARD = 6;
const int HAND_CNT = 7;

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

// DECODE NUMBER

bool
DcdNum (
    register char   *p,     // Pointer to string
    long            *val)   // Value
{
    *val = 0;

    while (*p != '\0') {
        if (*p < '0' || *p > '9') return 0;
        *val *= 10;
        *val += *p - '0';
        p++;
    }

    return 1;
```

```
}

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

// DECODE SUIT

int
DcdSuit (
    int     ch)             // Character
{
    if (ch == 'd')
        return 0;
    else if (ch == 'c')
        return 1;
    else if (ch == 'h')
        return 2;
    else if (ch == 's')
        return 3;
    else
        return -1;
}

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

// DECODE RANK

int
DcdRank (
    int     ch)             // Character
{
    if (ch >= '2' && ch <= '9')
        return ch - '2';
    else if (ch == 'X')
        return 8;
    else if (ch == 'J')
        return 9;
    else if (ch == 'Q')
        return 10;
    else if (ch == 'K')
        return 11;
    else if (ch == 'A')
        return 12;
    else
        return -1;
}

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

// ENCODE NUMBER

char *
EncNum (
    char            *buf,   // Output string
    int             len,    // Field length
    long            num)    // Number
{
    register char   *p, *q; // General purpose pointers
    long            base;   // Base number
    bool            first;  // First digit flag
    bool            negative;   // Negative flag

    if (num <= -1000 || num >= 1000) {
```

```
        p = EncNum (buf, len - 4, num / 1000);
        if (num < 0) num *= -1;
        if (p - buf <= len - 4) {
            sprintf (p, ",%03d", num % 1000);
            p += 4;
        } else {
            sprintf (p, "%03d", num % 1000);
            p += 3;
        }
    } else if (num == 0) {
        p = buf;
        while (len > 1) {
            *p++ = ' ';
            len--;
        }
        *p++ = '0';
    } else {
        negative = 0;
        if (num < 0) {
            negative = 1;
            num *= -1;
        }
        first = 1;
        p = buf;
        for (base = 100; base > 0; base /= 10) {
            if (! first)
                *p++ = static_cast<char>('0' + (num / base % 10));
            else if (num / base > 0) {
                if (negative)
                    *p++ = '-';
                *p++ = static_cast<char>('0' + (num / base % 10));
                first = 0;
            }
        }
        while (p - buf < len) {
            q = p;
            for (q = p; q != buf; q--)
                q[0] = q[-1];
            *q = ' ';
            p++;
        }
    }
    return p;
}

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

// MAIN LINE

int
main (
    int         argc,           // Argument count
    char        *argv[])        // Argument value
{
    long        fromRound;      // First round
    long        toRound;        // Last round
    long        round;          // Current round
    string      folderName;     // Folder name
    char        fileName[64];   // File name
    string      filePath;       // File path
    int         ch;             // Read ahead character
    int         rank[3];        // Rank
```

```
int         suit[3];        // Suit
int         i;              // General purpose index
int         j;              // General purpose index
int         hands[HAND_CNT]; // Number of hands
int         temp;           // Temporary variable
long        sums[HAND_CNT]; // Sums of higher hands
long        totalHands;     // Total hands
long        distValue;      // Value distributed to each
long        curValue[HAND_CNT]; // Current value
long        curSum;         // Current sum
long        newValue[HAND_CNT]; // New value
long        newSum;         // New sum
long        poolBal;        // Pool balance
char        *p, buf[512];   // Formatting buffer
static const char *handNames[] = {
    "HIGHEST ",
    "SECOND  ",
    "THIRD   ",
    "FOURTH  ",
    "FIFTH   ",
    "SIXTH   ",
    "SEVENTH "
};

if (argc != 4) {
    cerr << "USAGE: PokerRep path fromRound toRound\n";
    return 1;
}
folderName = argv[1];
if (! DcdNum (argv[2], &fromRound)) {
    cerr << "Bad fromRound\n";
    return 1;
}
if (! DcdNum (argv[3], &toRound)) {
    cerr << "Bad toRound\n";
    return 1;
}

for (round = fromRound; round <= toRound; round++) {
    memset (hands, 0, sizeof(hands));

    sprintf (fileName, "%ld", round);
    filePath = folderName + "/" + fileName;
    ifstream ifs (filePath.c_str());
    while (! ifs.eof()) {

        // Retrieve the 3 cards

        for (i = 0; i < 3; i++) {
            rank[i] = DcdRank (ifs.get());
            suit[i] = DcdSuit (ifs.get());
            ifs.get();
        }
        if (
            rank[0] == -1 || suit[0] == -1 ||
            rank[1] == -1 || suit[1] == -1 ||
            rank[2] == -1 || suit[2] == -1
        ) {
            cerr << "bad data\n";
            return 1;
            break;
        }
```

```
    // Sort the ranks

    if (rank[1] < rank[2]) {
        temp = rank[1];
        rank[1] = rank[2];
        rank[2] = temp;
    }
    if (rank[0] < rank[1]) {
        temp = rank[0];
        rank[0] = rank[1];
        rank[1] = temp;
    }
    if (rank[1] < rank[2]) {
        temp = rank[1];
        rank[1] = rank[2];
        rank[2] = temp;
    }

    // Determine the hand

    if (rank[0] == rank[1] && rank[1] == rank[2]) {
        if (suit[0] == suit[1] && suit[1] == suit[2])
            hands[THREE_OF_KIND_FLUSH] ++ ;
        else
            hands[THREE_OF_KIND] ++ ;
    } else if (
        (
            (rank[0] == rank[1] + 1) &&
            (rank[1] == rank[2] + 1)
        ) || (
            rank[0] == 12 &&
            rank[1] == 1 &&
            rank[2] == 0
        )
    ) {
        if (suit[0] == suit[1] && suit[1] == suit[2])
            hands[STRAIGHT_FLUSH] ++ ;
        else
            hands[STRAIGHT] ++ ;
    } else {
        if (suit[0] == suit[1] && suit[1] == suit[2])
            hands[FLUSH] ++ ;
        else if (
            rank[0] == rank[1] ||
            rank[1] == rank[2]
        )
            hands[PAIR] ++ ;
        else
            hands[HIGH_CARD] ++ ;
    }
}
ifs.close();

// Calculate number of hands

totalHands = 0;
for (i = 0; i < HAND_CNT; i++) {
    sums[i] = totalHands;
    totalHands += hands[i];
}
```

```
// Calculate values under current scheme

curSum = 0;
memset (curValue, 0, sizeof(curValue));
for (i = HAND_CNT - 1; i > 0; i--) {
    if (sums[i] == 0)
        distValue = 0;
    else
        distValue = ( hands[i] * 10 + sums[i] / 2 )
                        / sums[i];
    for (j = i - 1; j >= 0; j--)
        curValue[j] += distValue;
    curSum += distValue * sums[i];
}

// Calculate values under new scheme

poolBal = 0;
newSum = 0;
memset (newValue, 0, sizeof(newValue));
for (i = HAND_CNT - 1; i > 0; i--) {
    poolBal += hands[i] * 10;
    if (sums[i] == 0)
        distValue = 0;
    else
        distValue = poolBal / sums[i];
    for (j = i - 1; j >= 0; j--)
        newValue[j] += distValue;
    distValue *= sums[i];
    poolBal -= distValue;
    newSum += distValue;
}

// Display the report

cout << "                              "
    << "   CURRENT SCHEME   "
    << "     NEW SCHEME\n";
cout << "PRIORITY  PLAYERS      POOL"
    << "  / PLAYER     TOTAL"
    << "  / PLAYER     TOTAL"
    << "      DIFF\n";
cout << "------- --------- ---------"
    << " --------- ---------"
    << " --------- ---------"
    << " ---------\n";

for (i = 0; i < HAND_CNT; i++) {
    strcpy (buf, handNames[i]);
    p = buf + strlen (buf);
    p = EncNum (p, 9, hands[i]);
    *p++ = ' ';
    p = EncNum (p, 9, hands[i] * 10);
    *p++ = ' ';
    p = EncNum (p, 9, curValue[i]);
    *p++ = ' ';
    p = EncNum (p, 9, hands[i] * curValue[i]);
    *p++ = ' ';
    p = EncNum (p, 9, newValue[i]);
    *p++ = ' ';
    p = EncNum (p, 9, hands[i] * newValue[i]);
    *p++ = '\n';
```

```
            *p = '\0';
            cout << buf;
        }

        cout << "------- --------- ---------"
            << " --------- ---------"
            << " --------- ---------"
            << " ---------\n";

        strcpy (buf, "TOTAL   ");
        p = buf + 8;
        p = EncNum (p, 9, totalHands);
        *p++ = ' ';
        p = EncNum (p, 9, totalHands * 10);

        for (i = 0; i < 9; i++)
            *p++ = ' ';

        *p++ = '-';
        *p++ = ' ';
        p = EncNum (p, 9, curSum);

        for (i = 0; i < 9; i++)
            *p++ = ' ';

        *p++ = '-';
        *p++ = ' ';
        p = EncNum (p, 9, newSum);
        *p++ = ' ';
        p = EncNum (p, 9, newSum - curSum);
        *p++ = '\n';
        *p = '\0';

        cout << buf;
        cout << "------- --------- ---------"
            << " --------- ---------"
            << " --------- ---------"
            << " ---------\n";
    }

    return 0;
}
```