```
// TeamRep.cpp - GENERATE THE TEAM SELECTION REPORT
//
// MODULE INDEX
// NAME                         CONTENTS
// SelectTeam                   Select a team given contended proj assignments
// ExpandContended              Expand contended projects
// Title                        Emit a page title
// EmitReport                   Emit the report
// main                         Main line
//
// MAINTENANCE HISTORY
// DATE         PROGRAMMER AND DETAILS
// 02-10-06     JS      Original
// 19-10-06     ELT     Include math library
// 26-10-06     ELT     Fixed problem with Microsoft compiler
// 27-10-06     ELT     Updated module index
//
//-----------------------------------------------------------------------------

#include <iostream>             // C++ input/output streams
#include <iomanip>              // C++ input/output manipulators
#include <sstream>              // C++ string stream declarations
#include <vector>               // C++ vector declarations
#include <algorithm>            // C++ standard algorithms
#include <cstdlib>              // C-style standard library
#include <cstring>              // C-style string manipulation functions
#include <ctime>                // C-style time functions
#include <cmath>                // C-style math functions
using namespace std;            // Expand standard namespace to global scope
exec sql include sqlca;         // Include SQL communications area

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

// DEFINITIONS

static const int        LINES_PER_PAGE = 55;
                                // Useable lines per page
static const size_t     MAX_PROJ_PER_RATING = 5;
                                // Maximum projects per individual rating
static const double     Y = 1.1;
                                // The factor `Y' specified by NDS

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

// PROJECT STRUCTURE

struct Proj_t {
        long            projNum;        // Project number
        char            projName[21];   // Project name
        double          projValue;      // Project value
        vector<size_t>  projProgVec;    // Project programmer vector
        Proj_t () : projNum(0), projValue(0) { projName[0] = '\0'; }
                                        // Default constructor
        Proj_t (const Proj_t &proj) : projNum(proj.projNum),
                projValue(proj.projValue), projProgVec(proj.projProgVec)
                        { strcpy (projName, proj.projName); }
                                        // Copy constructor
};

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

// PROGRAMMER STRUCTURE

struct Prog_t {
        long            progNum;        // Programmer number
        char            progName[21];   // Programmer name
```

```
};

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

// PROJECT ALLOCATION STRUCTURE

struct Alloc_t {
        size_t          allocProgInd;   // Programmer index
        vector<size_t>  allocProjVec;   // Project index vector
        double          allocRating;    // Programmer's rating
        Alloc_t () : allocProgInd(0), allocRating(0) {}
                                        // Default constructor
        Alloc_t (const Alloc_t &alloc) : allocProgInd (alloc.allocProgInd),
                allocProjVec (alloc.allocProjVec),
                        allocRating (alloc.allocRating) {}
                                        // Copy constructor
};

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

// TEAM STRUCTURE

struct Team_t {
        vector<Alloc_t> teamAllocVec;   // Project allocation vector
        double          teamRating;     // Team's rating
        Team_t () : teamRating(0) {}
                                        // Default constructor
        Team_t (const Team_t &team) : teamAllocVec (team.teamAllocVec),
                teamRating (team.teamRating) {}
                                        // Copy constructor
};

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

// GLOBAL DATA

int             pageNo;         // Page number
int             linesRem;       // Lines remaining on the current page

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

// COMPARE PROJECT INDEXES BY DESCENDING RATING

class CmpProjInd_c {
        const vector<Proj_t> *cmpProjVec;
public:
        CmpProjInd_c (const vector<Proj_t> *projVec)
                { cmpProjVec = projVec; }
        bool operator () (const size_t &projInd1, const size_t &projInd2) const
                { return (*cmpProjVec)[projInd1].projValue
                        > (*cmpProjVec)[projInd2].projValue; }
};

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

// COMPARE ALLOCATIONS BY DESCENDING RATING

class CmpAllocRating_c {
public:
        bool operator () (const Alloc_t &alloc1, const Alloc_t &alloc2) const
                { return alloc1.allocRating > alloc2.allocRating; }
};

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

// SELECT A TEAM GIVEN CONTENDED PROJECT ASSIGNMENTS
```

```
void
SelectTeam (
        Team_t          *team,          // Selected team
        const vector<Proj_t> *projVec,  // Project vector
        const vector<Prog_t> *progVec,  // Programmer vector
        const vector<size_t> *progSelVec) // Programmer selection vector
{
        vector<Alloc_t> allocVec;       // Project allocation vector
        size_t          i, j;           // General purpose indices
        Alloc_t         *alloc;         // Allocation pointer
        const Proj_t    *proj;          // Project pointer
        size_t          projCnt;        // Number of projects in prog's rating
        size_t          teamSize;       // Team size
        double          progRating;     // Programmer rating
        double          sumOfRatings;   // Sum of ratings
        double          newTeamRating;  // New team rating

        // Generate the allocations

        team->teamAllocVec.clear ();
        team->teamAllocVec.resize (progVec->size());
        for (i = 0; i < projVec->size(); i++) {
                proj = &(*projVec)[i];
                j = (*progSelVec)[i];
                if (j < proj->projProgVec.size())
                        team->teamAllocVec[proj->projProgVec[j]].allocProjVec
                                .push_back (i);
        }

        // Determine individual programmer ratings

        for (i = 0; i < progVec->size(); i++) {
                alloc = &team->teamAllocVec[i];
                alloc->allocProgInd = i;
                sort (alloc->allocProjVec.begin(), alloc->allocProjVec.end(),
                        CmpProjInd_c (projVec));
                projCnt = alloc->allocProjVec.size();
                if (projCnt > MAX_PROJ_PER_RATING)
                        projCnt = MAX_PROJ_PER_RATING;
                alloc->allocRating = 0;
                for (j = 0; j < projCnt; j ++)
                        alloc->allocRating +=
                                (*projVec)[alloc->allocProjVec[j]].projValue;
        }

        // Sort the programmers by rating

        sort (team->teamAllocVec.begin(), team->teamAllocVec.end(),
                CmpAllocRating_c());

        // Add programmers to the team until either all the programmers
        // are in the team or the next programmer reduces the overall rating
        // of the team.

        teamSize = 0;
        sumOfRatings = 0;
        while (teamSize < team->teamAllocVec.size()) {
                progRating = team->teamAllocVec[teamSize].allocRating;
                newTeamRating = (sumOfRatings + progRating)
                        / pow (Y, static_cast<double>(teamSize));
                if (newTeamRating < team->teamRating) break;
                team->teamRating = newTeamRating;
                sumOfRatings += progRating;
                teamSize ++ ;
        }
```

```
        // Truncate the team to the selected size

        team->teamAllocVec.resize (teamSize);
}

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

// EXPAND CONTENDED PROJECTS

void
ExpandContended (
        Team_t          *bestTeam,      // Best team
        const vector<Proj_t> *projVec,  // Project vector
        const vector<Prog_t> *progVec,  // Programmer vector
        vector<size_t>  *progSelVec,    // Programmer selection vector
        vector<size_t>  *contendProj)   // Contended projects
{
        Team_t          thisTeam;       // This team
        size_t          i;              // General purpose index
        size_t          projInd;        // Project index
        const Proj_t    *proj;          // Project pointer

        if (contendProj->size() == 0) {
                SelectTeam (&thisTeam, projVec, progVec, progSelVec);
                if (thisTeam.teamRating > bestTeam->teamRating)
                        *bestTeam = thisTeam;
        } else {
                projInd = contendProj->back();
                contendProj->pop_back();
                proj = &(*projVec)[projInd];
                for (i = 0; i < proj->projProgVec.size(); i++) {
                        (*progSelVec)[projInd] = i;
                        ExpandContended (bestTeam, projVec, progVec,
                                progSelVec, contendProj);
                }
                contendProj->push_back (projInd);
        }
}

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

// 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 << setfill(' ') << setw(15) << ' ';
        cout << "TEAM SELECTION REPORT";
        cout << setw(22) << ' ';
        cout << "PAGE " << setw(3) << pageNo << "\n\n";

        // Display the column titles

        cout << "Programmer" << setw(20) << ' ' << "Project\n";
        cout << "Number  Name" << setw(18) << ' ' << "Number  Name"
                << setw(17) << ' ' << "Value\n";

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

        linesRem = LINES_PER_PAGE;
}

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

// EMIT THE REPORT

void
EmitReport (
        const Team_t    *bestTeam,      // Best team
        const vector<Proj_t> *projVec,  // Project vector
        const vector<Prog_t> *progVec)  // Programmer vector
{
        size_t          i, j;           // General purpose indexes
        const Prog_t    *prog;          // Pointer to programmer structure
        const Proj_t    *proj;          // Pointer to project structure
        const Alloc_t   *alloc;         // Pointer to project allocation struct
        ostringstream   oss;            // Output string stream
        string          s;              // String temporary

        // Display the first page title

        pageNo = 0;
        Title ();

        // Emit the data for each programmer

        for (i = 0; i < bestTeam->teamAllocVec.size(); i++) {
                alloc = &bestTeam->teamAllocVec[i];
                prog = &(*progVec)[ alloc->allocProgInd ];

                // Start a new page if needed

                if (linesRem < static_cast<int>(alloc->allocProjVec.size() + 3))
                        Title ();

                // Emit the programmer number and name

                cout << '\n';
                cout << setfill('0') << setw(6) << prog->progNum;
                cout << "  " << setfill(' ') << left << setw(20)
                        << prog->progName << right;

                // Emit the data for each project

                for (j = 0; j < alloc->allocProjVec.size(); j++) {
                        proj = &(*projVec)[ alloc->allocProjVec[j] ];
```

```
                        if (j == 0)
                                cout << "  ";
                        else
                                cout << setw(30) << " ";
                        cout << setfill('0') << setw(6) << proj->projNum;
                        cout << "  " << setfill(' ') << left << setw(20)
                                << proj->projName << right;
                        cout << "  " << fixed << setprecision(1) << setw(4)
                                << proj->projValue << '\n';
                        linesRem -- ;
                }

                // Emit the programmer total

                cout << setw(60) << " " << "----\n";
                cout << setw(30) << " " << "Programmer total"
                        << setw(14) << " " << fixed << setprecision(1)
                        << setw(4) << alloc->allocRating << '\n';
                linesRem -= 2;
        }

        // Emit the team total

        if (linesRem < 2) Title ();
        cout << setw(59) << " " << "-----\n";
        oss.str("");
        oss << "Team total (" << bestTeam->teamAllocVec.size() << " members)";
        s = oss.str();
        cout << left << setw(59) << s << fixed << setprecision(1)
                << setw(5) << bestTeam->teamRating << '\n';
        linesRem -= 2;
}

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

// MAIN LINE

int
main (
        int             argc,           // Argument count
        char            *argv[])        // Argument value vector
{
        Proj_t          projRec;        // Project record
        vector<Proj_t>  projVec;        // Project vector
        size_t          projInd;        // Project index
        Prog_t          progRec;        // Programmer record
        vector<Prog_t>  progVec;        // Programmer vector
        size_t          progInd;        // Programmer index
        vector<size_t>  contendProj;    // Contended projects
        vector<size_t>  progSelVec;     // Programmer selection vector
        size_t          i;              // General purpose index
        Team_t          bestTeam;       // Best team
        exec sql begin declare section;
                long            projNum;        // Project number
                char            projName[21];   // Project name
                double          projValue;      // Project value
                long            progNum;        // Programmer number
                char            progName[21];   // Programmer name
        exec sql end declare section;

        // Jump to DbError whenever an SQL error occurs

        exec sql whenever sqlerror goto DbError;

        // Connect to the database
```

```
exec sql connect to teamdb;

// Load the projects table

exec sql declare projCur cursor for
        select  projNum, projName, projValue
        from    projects;
exec sql open projCur;
for (;;) {
        exec sql fetch  projCur
                into    :projNum, :projName, :projValue;
        if (SQLCODE != 0) break;
        projRec.projNum = projNum;
        strcpy (projRec.projName, projName);
        projRec.projValue = projValue;
        projRec.projProgVec.clear ();
        projVec.push_back (projRec);
}
exec sql close projCur;

// Load the programmers table

exec sql declare progCur cursor for
        select  progNum, progName
        from    programmers;
exec sql open progCur;
for (;;) {
        exec sql fetch  progCur
                into    :progNum, :progName;
        if (SQLCODE != 0) break;
        progRec.progNum = progNum;
        strcpy (progRec.progName, progName);
        progVec.push_back (progRec);
}
exec sql close progCur;

// Load the assignments

exec sql declare asnCur cursor for
        select  asnProgNum, asnProjNum
        from    assignments;
exec sql open asnCur;
for (;;) {
        exec sql fetch  asnCur
                into    :progNum, :projNum;
        if (SQLCODE != 0) break;

        // Determine the programmer index

        progInd = 0;
        while (
                progInd < progVec.size() &&
                progVec[progInd].progNum != progNum
        ) progInd ++ ;
        if (progInd >= progVec.size()) {
                cerr << "Error: invalid programmer number\n";
                exit (1);
        }

        // Determine the project index

        projInd = 0;
        while (
                projInd < projVec.size() &&
                projVec[projInd].projNum != projNum
        ) projInd ++ ;
```

```
                if (projInd >= projVec.size()) {
                        cerr << "Error: invalid project number\n";
                        exit (1);
                }

                // Append the programmer to the programmer vector
                // in the project structure.

                projVec[projInd].projProgVec.push_back (progInd);
        }
        exec sql close asnCur;

        // Load a vector of contended projects.  The contended projects
        // are all the projects with more than one programmer.

        contendProj.clear ();
        for (i = 0; i < projVec.size(); i++)
                if (projVec[i].projProgVec.size() > 1)
                        contendProj.push_back (i);

        // Initialise the programmer selection vector

        progSelVec.resize (projVec.size());
        for (i = 0; i < projVec.size(); i++)
                progSelVec[i] = 0;

        // Set the best team to the empty team and search for a better team.

        bestTeam.teamAllocVec.clear();
        bestTeam.teamRating = 0;
        ExpandContended (&bestTeam, &projVec, &progVec, &progSelVec,
                &contendProj);

        // Emit the report

        EmitReport (&bestTeam, &projVec, &progVec);

        // And that's all

        exec sql commit work;

        return 0;

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

        return 1;
}
```