```
// Schedule.cpp - SCHEDULING PROGRAM
//
// USAGE
// Schedule input-file
//
// input-file   is the name of the input file.
//
// MODULE INDEX
// NAME                          CONTENTS
// StringToDate                  Convert a date into an internal julian day
// DateToString                  Convert an internal format date into a string
// DayOfWeek                     Day-of-week (monday=0)
// TokenRcvr_c::TrNextChar       Read the next character
// TokenRcvr_c::TrOpen           Open the input file
// TokenRcvr_c::TrClose          Close the token receiver
// TokenRcvr_c::TrNextToken      Read the next token
// TokenRcvr_c::TrGetDateValue   Get a date value
// TokenRcvr_c::TrGetLongValue   Get a long value
// ReadInput                     Read the input file
// ScheduleTask                  Schedule a task
// ScheduleProj                  Schedule the project
// CreateFillers                 Create filler tasks
// DayToDate                     Convert project day number to calendar date
// RepGen_c::RgTitle             Display the report title
// RepGen_c::RgGenerateReport    Generate the report
// main                          Main line
//
// MAINTENANCE HISTORY
// DATE         PROGRAMMER AND DETAILS
// 31-08-05     JS      Original
// 07-09-05     ELT     Corrected error message format
// 07-09-05     ELT     Added filler task creation in the beginning of schedule
// 09-10-05     JS      Optimised coding of filler task creation
//
//-----------------------------------------------------------------------------

#include <iostream>             // C++ input/output streams
#include <iomanip>              // C++ input/output manipulators
#include <sstream>              // C++ string stream declarations
#include <fstream>              // C++ file streams
#include <string>               // C++ style strings
#include <vector>               // C++ vectors
#include <map>                  // C++ maps
#include <set>                  // C++ sets
#include <cstdlib>              // C-style standard library
#include <cstring>              // C-style string manipulation functions
#include <cctype>               // C-style character typing
#include <ctime>                // C-style system time functions
using namespace std;            // Expand standard namespace to global scope

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

// INTERNAL DATE TYPE (DAYS SINCE 01-01-1969)

typedef long Date_t;

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

// ERROR CLASS

struct Error_c {
    string              errMsg;         // Error message
    Error_c (const string msg) : errMsg(msg) {}
                                        // Message constructor
};

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

// TOKEN TYPES

enum TokenType_t {
    TT_WORD,                            // Identifier, date, number
    TT_STRING,                          // String
    TT_SEMI,                            // Semicolon
    TT_COMMA,                           // Comma
    TT_OPEN,                            // Open braces
    TT_CLOSE,                           // Close braces
    TT_EOF                              // End-of-file
};
```

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

// TOKEN RECEIVER CLASS

class TokenRcvr_c : private ifstream {

    // Class Instance Variables

    long                trLineNo;       // Input file line number
    char                trCh;           // Current input character
    TokenType_t         trTokenType;    // Current token type
    string              trTokenValue;   // Current token value

    // Private Methods

    void TrNextChar ();
                                        // Read the next character

    // Public Methods

public:
    void TrOpen (const char *fileName);
                                        // Open the input file
    void TrClose ();
                                        // Close the input file
    void TrNextToken ();
                                        // Read the next token
    const long &TrGetLineNo() { return trLineNo; }
                                        // Get the current input line number
    const TokenType_t &TrGetTokenType() { return trTokenType; }
                                        // Get the current token type
    const string &TrGetTokenValue() { return trTokenValue; }
                                        // Get the current token type
    Date_t TrGetDateValue ();
                                        // Get a date value
    long TrGetLongValue ();
                                        // Get a number value
};

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

// REPORT GENERATOR CLASS

class RepGen_c {

    // Class Instance Variables

    const char          *rgFileName;    // Input file name
    time_t              rgSysTime;      // System time
    size_t              rgPageNo;       // Current page number
    size_t              rgLnsRem;       // Lines remaining on the current page

    // Private Methods

    void RgTitle ();
                                        // Display a page title
    // Public Methods
public:
    void RgGenerateReport (const char *fileName);
                                        // Generate the report
};

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

// TASK STRUCTURE

struct Task_t {
    string              taskName;       // Task name
    string              taskDescr;      // Task description
    string              taskPerson;     // Person who is to do the task
    long                taskDuration;   // Task duration
    vector<size_t>      taskNeeds;      // Task names of `needs' list
    long                taskStartDay;   // Start day number
    long                taskFinishDay;  // Finish day number
    bool                taskScheduled;  // Task scheduled flag
};

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

```
// FINISH DAY KEY STRUCTURE

struct FinishDayKey_t {
    long                fdkFinishDay;   // Finish day number
    size_t              fdkTaskNo;      // Task number

    bool operator < (const FinishDayKey_t &fdk)
        const { return fdkFinishDay < fdk.fdkFinishDay ||
            (fdkFinishDay == fdk.fdkFinishDay && fdkTaskNo < fdk.fdkTaskNo); }
                                        // Less-than operator for sorting
};

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

// MAP MONTH TO NUMBER OF DAYS BEFORE THE FIRST OF THE MONTH

const static int ORD_MAP[13] =
    {0, 31, 59, 90, 120, 151, 181, 212, 243, 273, 304, 334, 365};
const static int LEAP_MAP[13] =
    {0, 31, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335, 366};

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

// GLOBAL VARIABLES

static const size_t     LINES_PER_PAGE = 55; // Lines per page
Date_t                  startDate;      // Start date
set<Date_t>             holidays;       // Holidays
vector<Task_t>          taskTable;      // Task table
map<string,size_t>      taskNameMap;    // Task name map
map<string,size_t>      lastTaskMap;    // Map initials to last task

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

// CONVERT A DATE INTO AN INTERNAL JULIAN DAY

Date_t
StringToDate (
    const string        &s)             // String to convert into a date
{
    const char          *p;             // Decoding pointer
    int                 year;           // Year component of date
    int                 month;          // Month component of date
    int                 day;            // Day component of date
    const int           *dayMap;        // Day map pointer

    // Load the decoding pointer

    p = s.c_str();

    // Decode the day

    if (!isdigit(*p)) goto BadDate;
    day = 0;
    while (isdigit(*p)) {
        day = day * 10 + *p - '0';
        p ++ ;
    }
    if (*p != '-') goto BadDate;
    p ++ ;

    // Decode the month

    if (!isdigit(*p)) goto BadDate;
    month = 0;
    while (isdigit(*p)) {
        month = month * 10 + *p - '0';
        p ++ ;
    }
    if (*p != '-') goto BadDate;
    p ++ ;

    // Decode the year

    if (!isdigit(*p)) goto BadDate;
    year = 0;
    while (isdigit(*p)) {
        year = year * 10 + *p - '0';
```

```
        p ++ ;
    }
    if (*p != '\0') goto BadDate;

    // Convert the year into a 4-digit year

    if (year < 70)
        year += 2000;
    else if (year < 100)
        year += 1900;
    else
        goto BadDate;

    // Select the day map depending on whether it is a leap year or not

    if (year % 4 == 0)
        dayMap = LEAP_MAP;
    else
        dayMap = ORD_MAP;

    // Validate the month

    if (month < 1 || month > 12) goto BadDate;

    // Validate the day

    if (day < 1 || day > dayMap[month] - dayMap[month-1]) goto BadDate;

    // Convert the year, month and day into the internal format

    return (year - 1969) * 365 + (year - 1969) / 4
        + dayMap[month-1] + day - 1;

BadDate:
    throw Error_c ("bad date");
}

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

// CONVERT AN INTERNAL FORMAT DATE INTO A STRING

const string
DateToString (
    Date_t              date)           // Internal format date
{
    int                 quadYears;      // Number of quad-years
    int                 year;           // Year component of date
    int                 month;          // Month component of date
    int                 day;            // Day component of date
    const int           *dayMap;        // Day map pointer
    ostringstream       oss;            // Output string stream

    // Deduct the number of 4-year periods from the date

    quadYears = date / (365L*4L+1L);
    date -= quadYears * (365L*4L+1L);

    // Calculate the number of remaining non-leap years

    year = date / 365;
    if (year > 3) year = 3;
    date -= year * 365;
    year += 1969 + quadYears * 4;

    // Select the day map depending on whether it is a leap year or not

    if (year % 4 == 0)
        dayMap = LEAP_MAP;
    else
        dayMap = ORD_MAP;

    // Calculate the number of months

    month = 1;
    while (dayMap[month] <= date) month ++ ;
    date -= dayMap[month-1];

    // What remains can only be days
```

```
    day = date + 1;

    // Convert the date components into a string

    oss << setfill('0') << setw(2) << day << '-' << setw(2) << month << '-'
        << setw(2) << year % 100;
    return oss.str();
}

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

// DAY-OF-WEEK (MONDAY=0)

inline int
DayOfWeek (
    Date_t              date)           // Internal format date
{
    return (date + 2) % 7;
}

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

// READ THE NEXT CHARACTER

void
TokenRcvr_c::TrNextChar ()
{
    if (trCh == '\n') trLineNo ++ ;
    get (trCh);
    if ( ! *this && ! eof())
        throw Error_c ("input i/o fault");
}

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

// OPEN THE INPUT FILE

void
TokenRcvr_c::TrOpen (
    const char          *fileName)      // Input file name
{
    open (fileName);
    if ( ! *this)
        throw Error_c (string("cannot open ") + string(fileName));
    trCh = '\n';
    trLineNo = 0;
    TrNextChar ();
}

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

// CLOSE THE TOKEN RECEIVER

void
TokenRcvr_c::TrClose ()
{
    close ();
}

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

// READ THE NEXT TOKEN

void
TokenRcvr_c::TrNextToken ()
{
    // Skip white space and comments

    while (
        !eof() &&
        (trCh == ' ' || trCh == '\t' || trCh == '\n' || trCh == '#')
    ) {
        if (trCh == '#') {
            do TrNextChar(); while (!eof() && trCh != '\n');
            if (!eof()) TrNextChar();
        } else {
            TrNextChar();
        }
    }
```

```
    // Process end-of-file

    if (eof()) {
        trTokenType = TT_EOF;
    }

    // Process words of various kinds

    else if (isalpha(trCh) || isdigit(trCh) || trCh == '_') {
        trTokenType = TT_WORD;
        trTokenValue = "";
        do {
            trTokenValue += trCh;
            TrNextChar();
        } while (
            ! eof() &&
            (isalpha(trCh) || isdigit(trCh) || trCh == '_' || trCh == '-')
        );
    }

    // Process strings

    else if (trCh == '"') {
        trTokenType = TT_STRING;
        TrNextChar();
        trTokenValue = "";
        while ( ! eof() && trCh != '"' && trCh != '\n') {
            if (trCh == '\\') {
                TrNextChar();
                if (eof())
                    throw Error_c ("eof in string");
                if (trCh == '\n')
                    throw Error_c ("new line in string");
                if (trCh != '\\' && trCh != '"')
                    trTokenValue += '\\';
            }
            trTokenValue += trCh;
            TrNextChar();
        }
        if (eof())
            throw Error_c ("eof in string");
        if (trCh == '\n')
            throw Error_c ("new line in string");
        TrNextChar();
    }

    // Process special characters

    else if (trCh == ';') {
        trTokenType = TT_SEMI;
        TrNextChar();
    }
    else if (trCh == '{') {
        trTokenType = TT_OPEN;
        TrNextChar();
    }
    else if (trCh == '}') {
        trTokenType = TT_CLOSE;
        TrNextChar();
    }
    else if (trCh == ',') {
        trTokenType = TT_COMMA;
        TrNextChar();
    }

    // Other characters are unacceptable

    else
        throw Error_c ("bad input char");
}

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

// GET A DATE VALUE

Date_t
TokenRcvr_c::TrGetDateValue ()
{
```

```
    if (trTokenType != TT_WORD)
        throw Error_c ("bad date");
    return StringToDate(trTokenValue);
}

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

// GET A LONG VALUE

long
TokenRcvr_c::TrGetLongValue ()
{
    long                l;              // Long value to create
    const char          *p;             // Decoding pointer

    p = trTokenValue.c_str();
    if (!isdigit(*p)) goto BadNum;
    l = 0;
    while (isdigit(*p)) {
        l = l * 10 + *p - '0';
        p ++ ;
    }
    if (*p != '\0') goto BadNum;
    return l;

BadNum:
    throw Error_c ("bad number");
}

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

// READ THE INPUT FILE

void
ReadInput (
    const char          *fileName)      // Input file name
{
    TokenRcvr_c         tr;             // Token receiver instance
    Task_t              taskData;       // Task data
    ostringstream       oss;            // Output string stream
    map<string,size_t>::iterator lti;   // Last task iterator
    map<string,size_t>::iterator tnmi;  // Task name map iterator

    // Open the input file and receive the first token

    tr.TrOpen (fileName);

    // Append the input line number to any error message

    try {

        // Read the look-ahead token

        tr.TrNextToken ();

        // Process the start date

        if (tr.TrGetTokenType() != TT_WORD || tr.TrGetTokenValue() != "start")
            throw Error_c ("no start keyword");
        tr.TrNextToken ();
        startDate = tr.TrGetDateValue();
        tr.TrNextToken ();
        if (tr.TrGetTokenType() != TT_SEMI)
            throw Error_c ("missing semicolon");
        tr.TrNextToken ();

        // Process the holidays

        if (tr.TrGetTokenType()!=TT_WORD || tr.TrGetTokenValue()!="holidays")
            throw Error_c ("no holidays keyword");
        tr.TrNextToken ();
        if (tr.TrGetTokenType() != TT_OPEN)
            throw Error_c ("no open brace");
        tr.TrNextToken ();
        if (tr.TrGetTokenType() != TT_CLOSE) {
            for (;;) {
                if (tr.TrGetTokenType() != TT_WORD)
                    throw Error_c ("date expected");
                holidays.insert (tr.TrGetDateValue());
```

```
        tr.TrNextToken ();
        if (tr.TrGetTokenType() != TT_COMMA) break;
        tr.TrNextToken ();
    }
    if (tr.TrGetTokenType() != TT_CLOSE)
        throw Error_c ("no close brace");
}
tr.TrNextToken ();

// Process the task list

if (tr.TrGetTokenType() != TT_WORD || tr.TrGetTokenValue() != "tasks")
    throw Error_c ("no tasks keyword");
tr.TrNextToken ();
if (tr.TrGetTokenType() != TT_OPEN)
    throw Error_c ("no open brace");
tr.TrNextToken ();
while (tr.TrGetTokenType() == TT_WORD) {

    // Read the task name

    taskData.taskName = tr.TrGetTokenValue();
    tr.TrNextToken ();

    // Read the task description

    if (tr.TrGetTokenType() != TT_STRING)
        throw Error_c ("task description expected");
    taskData.taskDescr = tr.TrGetTokenValue();
    tr.TrNextToken ();

    // Read the initials of the person who will do the task

    if (tr.TrGetTokenType() != TT_WORD)
        throw Error_c ("person's initials expected");
    taskData.taskPerson = tr.TrGetTokenValue();
    tr.TrNextToken ();

    // Read the task duration

    if (tr.TrGetTokenType() != TT_WORD)
        throw Error_c ("person's initials expected");
    taskData.taskDuration = tr.TrGetLongValue();
    tr.TrNextToken ();

    // Initialise the needs list

    taskData.taskNeeds.clear();

    // If the person is known, add the person's last task
    // to the needs list.

    lti = lastTaskMap.find(taskData.taskPerson);
    if (lti != lastTaskMap.end())
        taskData.taskNeeds.push_back (lti->second);

    // Add any explicit needs to the needs list

    if (
        tr.TrGetTokenType() == TT_WORD &&
        tr.TrGetTokenValue() == "needs"
    ) {
        do {
            tr.TrNextToken ();
            if (tr.TrGetTokenType() != TT_WORD)
                throw Error_c ("needed task name expected");
            tnmi = taskNameMap.find (tr.TrGetTokenValue());
            if (tnmi == taskNameMap.end())
                throw Error_c ("unrecognised needed task");
            taskData.taskNeeds.push_back (tnmi->second);
            tr.TrNextToken ();
        } while (tr.TrGetTokenType() == TT_COMMA);
    }

    // Reset the task scheduled flag

    taskData.taskScheduled = 0;

    // Add the task to the task table and task name map
```

```
            taskNameMap[taskData.taskName] = taskTable.size();
            lastTaskMap[taskData.taskPerson] = taskTable.size();
            taskTable.push_back (taskData);

            // Validate the semicolon at the end of the task definition

            if (tr.TrGetTokenType() != TT_SEMI)
                throw Error_c ("semicolon expected");
            tr.TrNextToken ();
        }
        if (tr.TrGetTokenType() != TT_CLOSE)
            throw Error_c ("task name expected");
        tr.TrNextToken ();

        // Check for end-of-file

        if (tr.TrGetTokenType() != TT_EOF)
            throw Error_c ("data after task list");
    }

    // Catch input errors and append the file name

    catch (Error_c err) {
        oss.clear();
        oss <<  "line " << tr.TrGetLineNo() << ": " << err.errMsg;
        throw Error_c (oss.str());
    }

    // Close the input file

    tr.TrClose ();
}

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

// SCHEDULE A TASK

void
ScheduleTask (
    size_t              taskNo)         // Task number
{
    long                startDay;       // Start day
    Task_t              *taskData;      // Task data pointer
    size_t              needTaskNo;     // Needed task number
    Task_t              *needData;      // Need data pointer
    size_t              i;              // General purpose index

    // Look up the task data

    taskData = & taskTable[taskNo];

    // Find the greatest finish day of each task that this task needs

    startDay = 0;
    for (i = 0; i < taskData->taskNeeds.size(); i++) {
        needTaskNo = taskData->taskNeeds[i];
        needData = & taskTable[needTaskNo];
        if ( ! needData->taskScheduled)
            ScheduleTask (needTaskNo);
        if (needData->taskFinishDay >= startDay)
            startDay = needData->taskFinishDay + 1;
    }

    // Load the start day and finish days and set the scheduled flag

    taskData->taskStartDay = startDay;
    taskData->taskFinishDay = startDay + taskData->taskDuration - 1;
    taskData->taskScheduled = 1;
}

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

// SCHEDULE THE PROJECT

void
ScheduleProj ()
{
    size_t              taskNo;         // Task number
```

```
    // Schedule each task

    for (taskNo = 0; taskNo < taskTable.size(); taskNo++)
        if ( ! taskTable[taskNo].taskScheduled) ScheduleTask (taskNo);
}

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

// CREATE FILLER TASKS

void
CreateFillers ()
{
    map<string,size_t>::iterator ltmi;  // Last task map iterator
    map<long,size_t>    startDayMap;    // Start day map
    map<long,size_t>::iterator sdmi;    // Start day map iterator
    size_t              i;              // General purpose indices
    Task_t              taskData;       // Task data structure
    const Task_t        *prevTask;      // Previous task
    const Task_t        *task;          // Current task
    long                prevFinishDay;  // Previous task finish day

    // Process each person from the last task map

    for (ltmi = lastTaskMap.begin(); ltmi != lastTaskMap.end(); ltmi ++) {

        // Construct a list of the person's tasks sorted by start day

        startDayMap.clear();
        for (i = 0; i < taskTable.size(); i++) {
            if (taskTable[i].taskPerson == ltmi->first)
                startDayMap[taskTable[i].taskStartDay] = i;
        }

        // Check for idle time and create filler tasks as needed

        prevTask = 0;
        for (sdmi = startDayMap.begin(); sdmi != startDayMap.end(); sdmi ++) {
            task = & taskTable[sdmi->second];
            prevFinishDay = prevTask == 0 ? -1 : prevTask->taskFinishDay;
            if (task->taskStartDay != prevFinishDay + 1) {
                taskData.taskName = "SPACE";
                taskData.taskDescr = "Waiting for dependency";
                taskData.taskPerson = ltmi->first;
                taskData.taskDuration = task->taskStartDay - prevFinishDay - 1;
                taskData.taskStartDay = prevFinishDay + 1;
                taskData.taskFinishDay = task->taskStartDay - 1;
                taskData.taskScheduled = 1;
                taskTable.push_back (taskData);
            }
            prevTask = task;
        }
    }
}

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

// CONVERT PROJECT DAY NUMBER TO CALENDAR DATE

Date_t
DayToDate (
    long                day)            // Project day number
{
    Date_t              date;           // Date corresponding to day number

    // This is not particularly efficient, but it is quite robust.
    // Start at the first day in the project.

    date = startDate;
    for (;;) {

        // Skip weekends and public holidays

        while (
            DayOfWeek(date) == 5 ||
            DayOfWeek(date) == 6 ||
            holidays.find(date) != holidays.end()
        ) date ++ ;
```

```
        // If there are no more days, exit the loop

        if (day == 0) break;

        // Move to the next day in the project

        date ++ ;
        day -- ;
    }

    return date;
}

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

// DISPLAY THE REPORT TITLE

void
RepGen_c::RgTitle ()
{
    struct tm           localTm;        // Local time
    ostringstream       oss;            // Output string stream

    // Display the first line of the title

    localtime_r (&rgSysTime, &localTm);
    cout << right << setfill('0')
        << setw(2) << localTm.tm_mday
        << '-'
        << setw(2) << localTm.tm_mon
        << '-'
        << setw(2) << localTm.tm_year % 100
        << ' '
        << setw(2) << localTm.tm_hour
        << ':'
        << setw(2) << localTm.tm_min
        << setfill(' ') << setw(13) << ' '
        << "PROJECT SCHEDULE LISTING"
        << setw(23) << ' '
        << "PAGE " << ++rgPageNo << '\n';

    // Display `for <file-name>' centred

    oss.clear ();
    oss << "for " << rgFileName;
    cout << left << setw(39-oss.str().size()/2) << ' ' << oss.str() << "\n\n";

    // Display the column titles

    cout << "Task name  Description" << setw(28) << ' '
        << "Person Days  Start    Finish\n\n";

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

    rgLnsRem = LINES_PER_PAGE;
}

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

// GENERATE THE REPORT

void
RepGen_c::RgGenerateReport (
    const char          *fileName)      // Input file name
{
    FinishDayKey_t      finishDayKey;   // Finish day key record
    set<FinishDayKey_t> finishDayIndex; // Finish day index
    set<FinishDayKey_t>::iterator fdii; // Finish day index iterator
    size_t              i;              // General purpose index
    const Task_t        *task;          // Task pointer
    long                projDuration;   // Project duration

    // Initialise the class instance variables

    rgFileName = fileName;
    rgSysTime = time(0);
    rgPageNo = 0;
```

```
    // Load the finish day index to order the tasks by finish day
    // and calculate the project duration

    for (i = 0; i < taskTable.size(); i++) {
        finishDayKey.fdkFinishDay = taskTable[i].taskFinishDay;
        finishDayKey.fdkTaskNo = i;
        finishDayIndex.insert (finishDayKey);
    }

    // Display the first page title

    RgTitle ();

    // Display the report lines and determine the project duration

    projDuration = 0;
    for (fdii = finishDayIndex.begin(); fdii != finishDayIndex.end(); fdii++) {
        task = & taskTable[fdii->fdkTaskNo];

        // Start a new page if the last page is full

        if (rgLnsRem < 2) RgTitle ();

        // Display the task line

        cout << left << setw(10) << task->taskName
            << ' ' << setw(40) << task->taskDescr
            << ' ' << setw(3) << task->taskPerson
            << "   " << setw(3) << right << task->taskDuration
            << "  " << DateToString(DayToDate(task->taskStartDay))
            << ' ' << DateToString(DayToDate(task->taskFinishDay))
            << '\n';

        // Decrement the number of lines remaining on the page

        rgLnsRem -- ;

        // Update the project duration

        if (task->taskFinishDay + 1 > projDuration)
            projDuration = task->taskFinishDay + 1;
    }

    // Display the project duration

    cout
        << right << setw(61) << "----" << '\n'
        << left << setw(57) << "Project duration"
        << right << setw(4) << projDuration << '\n';
}

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

// MAIN LINE

int
main (
    int                 argc,           // Argument count
    char                *argv[])        // Argument value pointers
{
    RepGen_c            repGen;         // Report generator instance

    // Catch errors

    try {

        // Validate the command line and read the input file

        if (argc != 2) {
            cerr << "Usage: Scheduler input-file\n";
            exit (1);
        }
        ReadInput (argv[1]);

        // Schedule the project

        ScheduleProj ();

        // Create the filler tasks
```

```
        CreateFillers ();

        // Generate the report

        repGen.RgGenerateReport (argv[1]);
    }

    // Catch errors

    catch (Error_c err) {
        cerr << "Error: " << err.errMsg << '\n';
        exit (1);
    }

    // Exit gracefully

    return 0;
}
```