```
/* ========================================================================== */
/*  Test Grade Analysis Program

    AUTHOR: COP 3014 Teaching Staff
    FSU MAIL NAME: <your FSU user ID here>
    RECITATION SECTION NUMBER: <your section number here>
    TA NAME: <your TA's name here>
    COP 3014 - Spring 2010
    PROJECT NUMBER: 5
    PLATFORM: Windows Vista OS / MS Visual C++ Express 2008 IDE
    DUE DATE: Wednesday 4/7/2010

SUMMARY

This program grades a True/False exam.  Given a data file containing an answer
key and a list of student names and answers, it will produce a table of student
scores and a histogram depicting the frequency of scores.

INPUT

Input is taken from the data file "xfile.txt."  The first line consists of the
answer key for the exam, occupying columns 1 through 30.  Following the answer
key are the student answers:  the student's name appears on one line, then that
student's answers appear in columns 1 through 30 of the next line.  There are
25 students represented in the data file.

BAD DATA CHECKING

Student answers are checked to make sure they are either an upper-case 'T' or
an upper-case 'F'.  Bad answers are counted as wrong when computing the
students' scores, and an error message is printed following the score on the
appropriate line of the output.

OUTPUT

First, the answer key is echoprinted.  Then a table is generated, where each
line contains a student name, that student's answers, the score on the exam,
and (if necessary) an error message indicating an invalid answer.  Following
the table is a histogram of the scores.  The possible scores are divided into
the subranges 0-5, 6-10, 11-15, 16-20, 21-25, and 26-30.  Each line of the
histogram lists the subrange, the number of scores that fell in that subrange,
and a line of stars (asterisks) providing a graphical representation of the
number of scores:

    Score       Obtained By         5   10   15   20   25   30
    -----       -----------     ----+----+----+----+----+----+

   0 ... 5          4           ****

Finally, the mean score is output.

DATA STRUCTURES

One-dimensional arrays are used to store the answer key, the student exam
answers, and the frequency counts for the histogram.  The string class is
used to store a student name.

ASSUMPTIONS

The answer key and student names are assumed to be valid.

*/

// HEADER FILES
#include <iostream>
#include <iomanip>
```

```
#include <fstream>
#include <string>
using namespace std;

// GLOBAL CONSTANTS
const int NUM_QUESTIONS = 30;       // number of questions on exam
const int NUM_STUDENTS = 25;        // number of students taking exam
const int NUM_BINS = 6;             // number of frequency count categories
const int MAX_BIN = NUM_QUESTIONS / NUM_BINS ;
                                    // maximum index into freg count array
const char* BLANK = " ";            // blank for formatting output
const char EOLN = '\n';             // end of line marker

// TYPE DEFINITIONS
typedef char AnswerList [NUM_QUESTIONS];    // array to hold answers for
                                            // one exam
typedef int FrequencyList [NUM_BINS];       // array to hold frequency
                                            // counts

// FUNCTION PROTOTYPES
void StartUp ();
void GetAnswers (AnswerList key, ifstream& inFile);
void PrintTableHeader ();
void ProcessStudent (const AnswerList key, int& score, bool& errorFlag,
    ifstream& inFile);
void ProcessScore (int score, int& sum, FrequencyList freqCount);
void PrintHisto (const FrequencyList freqCount);
void PrintStars (int numStars);


//---------------------------------------------------------------------
int main ()
//---------------------------------------------------------------------
{
    const char* inFileName = "xfile.txt";   // name of input file

    AnswerList key;                         // array holding exam answer key
    FrequencyList freqCount = {0};          // array holding frequency counts
    int score,                              // score of one exam
        lcv,                                // loop control variable
        total = 0;                          // total of all exam scores
    double meanScore;                       // the mean score of all students
    bool error;                             // flags invalid data in student
                                            // answers
                                            // true if one or more answers is
                                            // invalid
    ifstream inFile;                        // data file

    StartUp ();                             // initialize program output

    // open the input file read only; exit program if open fails
    inFile.open (inFileName);
    if (!inFile)
    {
        cout << "Fatal Error, file will not open.";
        return (EXIT_FAILURE);
    }

    GetAnswers (key, inFile);               // load answer key

    PrintTableHeader ();                    // print headings for student data
                                            // table

    for (lcv = 0; lcv < NUM_STUDENTS; lcv++)   // print name, answers, and score
                                               // for each student
    {
        ProcessStudent (key, score, error, inFile);
        if (error)
            cout << endl << setw(21) << BLANK << " * Invalid Answer *";
        cout << endl;
        ProcessScore (score, total, freqCount);     // process each score by
                                                    // adding to running total
                                                    // and incrementing freq
                                                    // count
    }
    inFile.close ();                    // close the file - done with it now

    PrintHisto (freqCount);             // produce the score histogram

    // calculate mean and print it
    meanScore = static_cast<double> (total) / NUM_STUDENTS;
    cout << "<<< Mean score: " << meanScore << " >>>" << endl;

    // print closing message
    cout << endl << "Execution Terminated." << endl << endl;

    return (0);
}



//---------------------------------------------------------------------
void StartUp ()

// This function initializes the output by printing an introductory
// message and setting up real number formattting.
//---------------------------------------------------------------------

{   // StartUp

    // print an intro message
    cout << "<<< Welcome to the Fully Automated Examination Examiner! >>>"
         << endl;
    cout
        << "<<< To you will be revealed the results of a Xenobiology >>>"
        << endl;
    cout
        << "<<< examination administered to a group of reknowned "
        << " scientists. >>>";
    cout << endl << endl;

    // set up real output formatting
    cout << fixed << showpoint << setprecision (2);

}   // StartUp



//---------------------------------------------------------------------
void GetAnswers
    (AnswerList key,            // the answer key for the exam
    ifstream& inFile)           // data file

// This function loads the answer key from the first line of the
// data file. The answers are stored in a 30-element array of
// characters. The answer key is also printed here.
//-----------------------------------------------------------------
{   // GetAnswers

    int lcv;            // loop control for reading each element in the key

    cout << endl << "Test Answer Key:  ";

    for (lcv = 0; lcv < NUM_QUESTIONS; lcv++)   // for every question
    {
        inFile >> key[lcv];                     // read and echoprint
        cout << key[lcv];                       // each answer in the key
    }

    inFile.ignore (200, EOLN);                  // go to next line in data file
    cout << endl << endl;

}   // GetAnswers



//---------------------------------------------------------------------
void PrintTableHeader ( )

// This function prints the headings for the student data table.
//---------------------------------------------------------------------

{   // PrintTableHeader

    // print a brief title and the headings for the table
    cout << endl << "<<< Exam Results >>>" << endl << endl;
    cout << "NAME                          ANSWERS  SCORE"
            << endl;
    cout << "-----                         -------  -----"
            << endl << endl;

}   // PrintTableHeader



//---------------------------------------------------------------------
void ProcessStudent
    (const AnswerList key,          // exam answer key
    int& score,                     // exam score for one student
    bool& errorFlag,                // flags erroneus data in student answers
    ifstream& inFile)               // data file

// This function processes all data associated with one student.
// First, the student name is read and echoprinted.  Then, each
// student's answers are read and compared with the corresponding answer
// from the key.  If the answers are the same, the score is incremented;
// if not, the student answer is checked for validity.  When all answers
// have been processed, the score is output.
//---------------------------------------------------------------------

{   // ProcessStudent

    int lcv;                        // loop control for checking each
                                    // student answer

    string stdtName;                // name of one student
    AnswerList stdtAnswers;         // parallel array of student answers
                                    // parallel to the answer key!

    score = 0;                      // initialize score
    errorFlag = false;              // and errorFlag

    getline (inFile, stdtName);         // read and echo student name
    cout << stdtName;                   // left-justified
    cout << setw (22 - static_cast<int>(stdtName.length())) << BLANK;

    // read the student answers into array and echoprint
    for (lcv = 0; lcv < NUM_QUESTIONS; lcv++)
    {
        inFile >> stdtAnswers[lcv];
        cout << stdtAnswers[lcv];
    }

    // now go through the student answers and compare to the key, scoring
    for (lcv = 0; lcv < NUM_QUESTIONS; lcv++)
    {
        if (stdtAnswers[lcv] == key[lcv])
            score++;                // if correct, increment score
                                    // if incorrect, check for bad data
        else                        // and set error flag
            errorFlag = errorFlag ||
                       ((stdtAnswers[lcv] != 'T') && (stdtAnswers[lcv] != 'F'));
    }

    cout << setw (14) << score;
    inFile.ignore (200, EOLN);      // go on to next student data line

}   //ProcessStudent



//---------------------------------------------------------------------
void ProcessScore
    (int score,                     // one student score
    int& sum,                       // running total of all scores
    FrequencyList freqCount)        // running count of score frequencies

// This function processes each student score.  First, each score is
// added to a running total, for the purpose of eventually computing
// the mean.  Then the appropriate 'bin' in the frequency counts is
// incremented.
//---------------------------------------------------------------------

{   // ProcessScore

    int bin;                // numbers 'bins' for score frequencies
    sum = sum + score;      // increment sum

    // For "bins" numbered 0 through MAX_BIN, the bin number for each score is
    // the score - 1 divided by MAX_BIN.  This clearly applies only to
    // scores > 1; scores <= 1 are put in bin 0.

    if (score > 1)                          // increment bin counter
    {                                       // for scores > 1
        bin = (score - 1) / MAX_BIN;
        freqCount[bin]++;
    }
    else
        freqCount[0] = freqCount[0] + 1;    // and for scores <= 1

}   // ProcessScore



//---------------------------------------------------------------------
void PrintHisto
    (const FrequencyList freqCount)             // count of score frequencies

// This function uses the final values of the frequency counts to
// generate a histogram of the scores.  The first bin is processed
// separately, since it is a special case (including 0).  Then the
// remaining bins are processed.  The range of each bin is output,
// followed by the contents, in both numeric and graphical format.
//---------------------------------------------------------------------

{   // PrintHisto

    int i;  // loop control variables
    // print the title and headings for the histogram
    cout << endl << endl << "<<< Histogram >>>" << endl;
    cout << "                                    Frequency" << endl;
    cout << "                                    ---------" << endl << endl;
    cout << "  Score        Obtained By              5    10   15   20   25"
         << endl;
    cout << "  -----        -----------          ----+----+----+----+----+"
        << endl << endl;

    // bin 0 is a special case, since it counts NUM_BINS possible scores
    cout << " 0 ...  " << MAX_BIN << setw (12) << freqCount[0] << setw(15)
         << BLANK;
    PrintStars (freqCount[0]);
    cout << endl;

    // the rest of the bins, 1 through MAX_BIN, range from MAX_BIN * i + 1
    // to MAX_BIN * (i + 1)
    for (i = 1; i <= MAX_BIN; i++)
    {
        cout << setw (2) << MAX_BIN * i + 1 << " ... " << setw (2)
             << (i + 1) * MAX_BIN
             << setw (12) << freqCount[i] << setw (15) << BLANK;
        PrintStars (freqCount[i]);
        cout << endl;
    }
    cout << endl << endl;

} // PrintHisto



//---------------------------------------------------------------------
void PrintStars
    (int numStars)              // number of stars to print

// This function simply prints a string of stars ('*') to be used in
// the histogram.
//---------------------------------------------------------------------

{   // PrintStars

    int lcv;            // loop control variable

    for (lcv = 0; lcv < numStars; lcv++)            // print numStars stars
        cout << '*';

}   // PrintStars



/* ========================================================================= */
/*                  E N D    O F    P R O G R A M                            */
/* ========================================================================= */
```