Learning to Use the Development Environment

Obviously you cannot program unless you understand how to create a file containing your C++ language source code, how to compile and link the source code to produce a program, how to execute (run) the program, test its behavior, fix errors, etc. For the first programming project, you will be given C++ source for a program that meets the specification given below. Your task is to type in the given source code, without modification, and verify that it does indeed perform as specified. Along the way, you'll be exposed to quite a bit of C++ language that we haven't even begun to cover, so don't get paranoid. Try to understand the given source code (there are lots of comments included to help), but realize that this program uses elements of the C++ language from later topics in the course notes.

What to turn in and how:

This project will be submitted to the enhanced automated C Grader. Instructions for submitting to the C Grader are linked from the CS1044 Web home page. Be sure to follow those instructions carefully. In particular, do not alter the labels used in the sample output file below, and do not produce additional lines of output for aesthetic reasons. No hard-copy submission is required for this project.

The Program Specification: Arithmetic Expressions and File I/O: Map Mileage

You are to write a very simple map distance calculation program, which will read data from a file, perform some arithmetic calculations, and write results in a nice tabular form to a file. The forms of the input and output files are described below, with examples. Be sure to pay careful attention to both.

Input file description and sample:

Your program **must** read its input from a file named `INMAP.DAT` — use of another input file name will result in massive deductions. The first line of the input file will contain two values, separated by whitespace:

- Number of places to visit (integer between 1 and 100)
- Map scale factor (real number between 0.0 and 1000.0)

The first value represents the total number of locations on a map to which one wishes to travel in sequence. The second value represents the number of miles per inch on the map.

Each following line of the input file will contain one value representing the map distance in inches between each sequential pair of locations to be visited:

- Distance between next two places (real number between 0.0 and 1000.0)

You may assume that all input values will be logically correct (no negatives, for instance), and that all values will be in the specified ranges. For instance:

```
4  0.25
1.5
2.3
5.9
4.0
```

Note that you must not make **any** assumptions about the number of lines of data in the input file. Your program must be written so that it will detect when it's out of input and terminate correctly, (a technique for this will be presented at a later time in the course).

What to Calculate:

You will write a program that computes the number of miles between two locations on a map given their distance in inches and the scale factor of the map. The map distances in inches and the corresponding computed mileage distances between each two locations must be output to the nearest tenth of an inch and mile. All of the distances are to be output in a labeled tabular format as shown below. In addition, the total mileage traveled between all of the distances on the map will be output to the nearest tenth of a mile.

Output description and sample:

Your program must write output data to a file named `OUTMAP.DAT` — use of any other output file name will result in a massive deduction of points.

The sample output file shown below corresponds to the input data given above:

```
Dwight Barnette
Simple Map Distance Computations

Map Scale Factor:    0.25 miles per inch

      Map       Mileage
      Measure   Distance
============================================================
#  1    1.5       0.4
#  2    2.3       0.6
#  3    5.9       1.5
#  4    4.0       1.0
============================================================
Total Distance:    3.5 miles
```

You are not required to use this exact spacing, but your output must satisfy the following requirements:

- All distances (inches and miles) must be printed to show one place after the decimal.
  The map scale factor is to be printed to two decimal places.
- You must use the specified header, line and column labels, and include your name as shown.
- You must arrange your output in neatly aligned columns, with a label identifying the contents of each column.
- You must show the required total distance below the corresponding columns.
- You must use the same ordering of the columns as shown here.

Programming Standards:

You'll be expected to observe good programming/documentation standards. All the discussions in class about formatting, structure, and commenting your code will be enforced. A copy of *Elements of Programming Style* is included with the course notes — if you don't have a copy I strongly suggest you read the on-line edition. Some specifics:

- You must include header comments specifying the compiler and operating system used and the date completed.
- You must include a comment explaining the purpose of every variable you use in your program.
- You must use meaningful, suggestive (of function or purpose!) variable names.
- Use constants instead of variables where appropriate.
- Precede every major block of your code with a comment explaining its purpose. You don't have to describe how it works unless you do something so sneaky it deserves special recognition.
- You must use indentation to make control structures like loops and if-else statements more readable.

Hints:

This program requires that you know how to manage file-oriented input/output operations --- the slides and the text provide good examples and guidelines. You'll have to use **manipulators** to manage the formatting of your code. (Use of `printf` is strictly forbidden.) Carefully read the discussion on pages 123-130 of your textbook, there are important clues there.

You'll need to use if-then control structures extensively --- be sure to read Chapter 5. You'll also need to use some sort of loop control structure to manage reading and processing of the input, this is covered in Chapter 6.

In order to execute and test the program with the sample input file given above, you will need to create a text file, enter the input data lines as given above, and save it as "INMAP.DAT" in the project folder with your source code. When you execute your project, the output file "OUTMAP.DAT" will be created in your project folder. Compare it to the sample output above and if it is different carefully check the lines you have typed to be sure they match the lines in the source code that follows.

The Source Code:

```
//  Mapmiles.cpp: A Simple Map Mileage Program
//
//  You should substitute the correct information in the following
//  lines:
//
//  Programmers:        N.D., C.W. & M.H.
//  Modifier:           Dwight Barnette
//  Section:            CS1044: #5228 & 9696
//  Compiler:           Visual C++ version 6.0
//  Platform:           PentiumII 450 / Windows NT Server 4.00.1381 SR5
//  Date:               August 30, 1999
//
//*****************************************************************
// Walk program
// This program computes the mileage (rounded to tenths of a mile)
// for distances between points in a city, given
// the measurements on a map with the scale the number of points
// and the map distance between each.
//*****************************************************************
//
//  Purpose of the program:
//
//  This program reads information about a a map and an arbitrary number of
//  distances between locations from an input file named "inmap.dat":
//
//          Line 1:
//          - number of locations
//          - map scale factor (miles per inch)
//          Line x: (all following lines)
//          - distance between locations (inches)
//
//  and computes the correct mileage for:
//
//          - map distances
//          - Total distance traveled.
//
//  It then prints out a labeled table of results, showing for
//  each location distance the actual mileage to reach it.
//  In addition, the total distance to be traveled is computed and
//  printed.
//
//  This is the #include section.  These statements tell the compiler
//  to read and use variables and functions declared in the specified
//  files.
//

#include <fstream.h>
#include <iomanip.h>    // For setprecision()

//  Every C++ program must have a function named main.  This is the
//  beginning of the definition of the main function for this
//  simple program.  In this case the whole program consists of just
//  the main function --- usually programs involve more than one
//  function.
//

int main( ) {

//  This section is where named constants are declared and given
```

```
//  their values.  We could just use the numbers themselves in the
//  code that follows, but the names make the code more readable;
//  the name DEFSCALE carries meaning where the number 0.25 doesn't.
//
  const double DEFSCALE = 0.25;     // Default map scale (miles per inch)

//  This section is where the variables used in main() are declared.
//  Each variable has a type (integer, float, etc) but no particular
//  value at this point.  The variable names like the constant names
//  used above are supposed to be meaningful, and each declared
//  variable is also given a short comment explaining what it will
//  be used for.
//
  ifstream inMap;                   // input file stream
  ofstream outMap;                  // output file stream


  double mapScale,         // Map scaling factor (miles per inch)
         distance,         // Map distance between locations (inches)
         totMiles,         // Total of rounded mileages
         miles;            // An individual rounded mileage


  int   numPoints,        // Number of locations on map to visit
        visited;          // Number of locations' distances converted

//  This is the initialization section of main().  We must make sure that
//  every variable is given a value before it is used in a calculation
//  or printed.  Some variables get their values by being read from the
//  input file, or assigned so they don't need to be initialized here.

  visited      =   0;          // Initialize counters and running totals
  totMiles     = 0.0;          // to zero.
  mapScale     = DEFSCALE;     // Initialize map scale factor to default value

//  The following four statements prepare the input and output files
//  for use.  Don't worry too much about what's going on here just yet,
//  we'll cover the details shortly.
//
  inMap.open("inmap.dat");
  outMap.open("outmap.dat");
  outMap.setf(ios::fixed, ios::floatfield);
  outMap.setf(ios::showpoint);

//  The following statments print the header info to output file.  It's
//  rather cryptic at this point, but you'll see output is really very
//  simple in C++.

  outMap << "Dwight Barnette" << endl; //*** substitute your own name ***
  outMap << "Simple Map Distance Computations" << endl << endl;

//  This statement tries to read first line of data from the input file.  If
//  something goes wrong, the variable inMap will be false (small lie) and
//  that will prevent us from reading further from the file and generating
//  nonsense.

  inMap >> numPoints >> mapScale;
  inMap.ignore(200, '\n');               // Skip to next line of input.

//  The next statments complete the printing of the header info
//  to the output file.
```

```
  outMap << "Map Scale Factor: " << setw(7) << setprecision(2) << mapScale;
  outMap << " miles per inch" << endl;
  outMap << endl;
  outMap << "      Map       Mileage" << endl;
  outMap << "      Measure   Distance" << endl;
  outMap << "============================================================";
  outMap << endl;

//  This statement tries to read the second line of data from the input file.  If
//  something goes wrong, the variable inMap will become false (another lie) and
//  that will prevent us from going into the loop that follows and generating
//  nonsense.

  if ( (inMap) && (numPoints > 0) )
        inMap >> distance;

//  What comes next is a while loop.  The body of the loop gets
//  executed over and over again until the expressions that follows
//  the while (inMap  and visted  less than numPoints in this case) becomes false.
//  Therefore, we'd better be sure that inMap will eventually become false,
//  or that visted will eventually become greater than or equal to numPoints,
//  otherwise the program will continue to execute the body of this
//  loop forever.

  while ( (inMap) && (visited < numPoints) ) {

        //  The first part of the loop body (down to the comment with
        //  all the hyphens) processes the data that was just read in.

        visited++;                              // Count number of locations.
        inMap.ignore(200, '\n');                // Skip to next line of input.

        // Compute miles for each distance on the map
        miles = double(int(distance * mapScale * 10.0 + 0.5)) / 10.0;

        totMiles = totMiles + miles;                  // Update total


        // -------------------------------- End of Processing Section

        //  This section of the loop body prints the results calculated
        //  above into the output file.

        outMap << "#" <<setw(3) << visited;
        outMap << setw(7) << setprecision(1) << distance;
        outMap << setw(10) << setprecision(1) << miles;
        outMap << endl;

        //  This statement tries to read the next line of input.  This
        //  is the last statement in the loop body, so the next thing
        //  that will happen is that the loop control expressions (inMap
        //  && visited in this case) are tested.

        inMap >> distance;

  }  // End of while loop.

//  Once we've exited the loop, we're done reading and processing data for
//  each location.  It's time to sum things up.  The following statements
//  prints out the total distance in a nice form.  First print a line to mark
//  the end of the table body:
```

```
  outMap << "============================================================";
  outMap << endl;
  outMap << "Total Distance:" ;
  outMap << setw(7) << setprecision(1) ;
  outMap << totMiles << " miles" <<endl;


//  These two statements just close the input and output files; this
//  tells the operating system that we're done with them and (hopefully)
//  that the files are properly saved.

  inMap.close();
  outMap.close();

//  The return statement sends a value back to whoever called this
//  function.  If you're using Visual C++, you may see this value
//  displayed in the window at the bottom of the screen when you run
//  the program.

  return visited;
} // End of main()
```