Lab 11
2 Dimensional Arrays and File I/O

Grading:

Lab11.cpp

Documentation and Style	5 points
Correct use of 2-D Arrays	5 points
Correct use of File I/O	5 points
Correct Totals (Rows & Columns)	5 points
Correct Formatting of Output	5 points

Multidimensional Arrays

*** If you are already an expert at working with 2-dimensional arrays in C++, you can skip this section. If not, you should read carefully. Textbook reference: Section 6.9.

2-dimensional arrays represent tables of values. The first subscript identifies the ROW and the second identifies the COLUMN.

```
// Declare a 3 x 4 array of integers

int a[3][4];
```

	Column 0	Column 1	Column 2	Column 3
Row 0	a[0] [0]	a[0] [1]	a[0] [2]	a[0] [3]
Row 1	a[1] [0]	a[1] [1]	a[1] [2]	a[1] [3]
Row 2	a[2] [0]	a[2] [1]	a[2] [2]	a[2] [3]

Every element in array a is identified by an element name of the form `a[i][j]` where i is the row number and j is the column number.

A multidimensional array can be initialized when it is defined. For example:

```
int b[ 2 ][ 2 ] = { { 1, 2 } , { 3, 4 } };
```

If there are not enough initializers in the list, the remaining elements are initialized to 0.

To print a 2-dimensional array, it is best to use two for loops. The outer loop is for the rows. The inner loop is for the columns. Assuming that the array is declared as shown above:

```
int rows = 3, columns = 4;

// loop through rows
for ( int i = 0; i < rows; i++ ) {

   // loop through columns
   for (int j = 0; j < columns; j++)
       cout << setw(5) << a[ i ] [ j ];
   cout << endl;
}
```

To calculate Row totals for the array, first declare a 1-dimensional array using the number of rows and another 1-dimensional array using the number of columns.

```
int rowTotal[ rows ];
int columnTotal[ columns ];
```

Then initialize the array elements to 0. (Remember to always initialize total variables to 0.)

```
for (int i = 0; i < rows; i++)
   rowTotal[ i ] = 0;
for (int j = 0; j < columns; j++)
   columnTotal[ j ] = 0;
```

Now use two for loops (nested just like we used for printing) to add up the row totals and column totals.

```
// loop through rows
for ( int i = 0; i < rows; i++ ) {

   // loop through columns
   for (int j = 0; j < columns; j++) {
       rowTotal[ i ] = rowTotal [ i ] + a[ i ][ j ];
       columnTotal[ j ] = columnTotal[ j ] + a[ i ][ j ];
   }

}
```

File I/O in C++ (from chapter 23)

C++ input and output (I/O) occurs in **streams**. A stream is a sequence of bytes.

Input: bytes flow from an input device (keyboard, disk drive, etc) to main memory
Output: bytes flow from main memory to an output device (screen, printer, disk drive)

The <iostream> library declares basic services for stream I/O operations. It defines cin (standard input stream) and cout (standard output stream).

The typical way to read from an input stream is using >> the stream extraction operator. Here are some of the member functions defined in <iostream>

eof ( )	returns 0 (false) if the input stream still has data in it returns 1 (true) if the input stream has reached the end of file
get( )	returns one character from the input stream, even if the character is a white-space character
ignore(int)	reads and discards a designated number of characters (the default is one)
peek( )	reads the next character from the input stream but does not remove it from the stream
getline (char * c, int size)	reads characters from the input stream until a newline is reached or until size – 1 characters have been read, whichever comes first. Stores the characters in the array c. Inserts a null character after the last character that was stored in the array.

The <iomanip> library declares services useful for formatting I/O.

Here are some stream manipulators:

hex	Sets the base for printing to hexadecimal (sticky)
oct	Sets the base for printing to octal (sticky)
setw(int)	Sets the field width for printing (not sticky)
fixed	Sets the format for floating point numbers to fixed (sticky). Used with setprecision, this will set the number of digits to the right of the decimal point.
setprecision(int)	Sets the precision of floating point numbers (sticky). If neither fixed nor scientific has been used, this sets the total number of significant digits to display. If fixed has been used, this sets the number of digits to the right of the decimal point.
showpoint	Forces a floating point number to be output with its decimal point and trailing zeros (as specified by the current precision)
left	Sets alignment to left-justify (sticky)
right	Sets alignment to right-justify (sticky)

The <fstream> library declares services for file processing.

To open a file for reading and/or writing, simply instantiate an object of the appropriate file I/O class, with the name of the file as a parameter. Then use the insertion (<<) or extraction (>>) operator to read/write to the file. Once you are done, there are several ways to close a file: explicitly call the close() function, or just let the file I/O variable go out of scope (the file I/O class destructor will close the file for you). Here's an example program that writes to a file. (Thanks to: http://www.learncpp.com/cpp-tutorial/136-basic-file-io/)

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

int main()
{

    // ofstream is used for writing files
    // We'll make a file called Sample.dat
    ofstream outf("Sample.dat");

    // If we couldn't open the output file stream for writing
    if (!outf)
    {
        // Print an error and exit
        cerr << "Uh oh, Sample.dat could not be opened for writing!" << endl;
        return 1;
    }

    // We'll write two lines into this file
    outf << "This is line 1" << endl;
    outf << "This is line 2" << endl;

    return 0;

    // When outf goes out of scope, the ofstream
    // destructor will close the file
}
```

Here's an example program that gets input from a file:

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

int main()
{
    string strInput;
    // ifstream is used for reading files
    // We'll read from a file called Sample.dat
    ifstream inf("Sample.dat");

    // If we couldn't open the output file stream for reading
    if (!inf)
    {
        // Print an error and exit
        cerr << "Uh oh, Sample.dat could not be opened for reading!" << endl;
        return 1;
    }

    // While there's still stuff left to read
    while (inf)
    {
        // read stuff from the file into a string and print it
        inf >> strInput;

        // check to see if the input was successful
        if (inf) {
           cout << strInput << endl;
        }
    }
    return 0;

    // When inf goes out of scope, the ifstream
    // destructor will close the file
}
```

The Assignment (Lab11.cpp)

Use C++ file input to read data from a file called Sales.dat (a sample file has been included on the assignment page).

Use C++ file output to send the output of your program to a file called SalesReport.txt.

Use a double-subscripted (2-dimensional) array called ***sales*** to solve the following problem:

A company has 5 sales people (salesperson numbers 1 to 5) who sell 4 different products (product numbers 1 to 4).

At the end of each day, each salesperson turns in a slip with the following information for each product they have sold that day:

a) salesperson number
b) product number
c) the dollar value of that product sold that day

Write a program that will read the information (one slip per line in the input file) and summarize the total sales by salesperson by product. All totals should be stored in the double-subscripted array ***<u>sales</u>***.

After processing all of the information in the file, print the results in tabular (neatly aligned) format with each row representing a particular salesperson and each column representing a particular product.

Cross total each row to get the total sales for that salesperson. Cross total each column to get the total sales for that product. Your tabular printout should include these cross totals to the right of the totaled rows and to the bottom of the totaled columns.

Output is shown below for the Sales.dat provided with the assignment. You should test your program with different data to make sure it works correctly.

```
                   Product #1  Product #2  Product #3  Product #4     Totals
Salesperson #1         887.02        0.00        0.00      589.53    1476.55
Salesperson #2        1448.90      898.70        0.00      557.82    2905.42
Salesperson #3           0.00      331.74      936.27     1550.97    2818.98
Salesperson #4           0.00     1745.11       35.13      730.49    2510.73
Salesperson #5         892.67      162.87        0.00        0.00    1055.54
        Totals        3228.59     3138.42      971.40     3428.81   10767.22
```

Submit Lab11.cpp on Canvas