Functions, Functional Decomposition, Selection Control, and Nested Control Statements

NOTE: For this assignment, there are several additional requirements you will be graded on in addition to producing the correct output. They are: (1) you should write an outline of your program design that reflects your top-down functional/modular decomposition – this should be included in your comments at the beginning of your program, (2) your program should contain good documentation, (3) you must write and use two additional functions in your program. The GTAs will be HAND-CHECKING your code on this assignment to grade you on these three items. You should be sure to read and apply the documentation standards as referenced in the Programming Standards section of this specification.

For the outline of your design:
You must include, below the header comments, an outline program design. The design must reflect your top-down functional/modular decomposition of the problem. The design should follow the layout of the payroll design example in appendix 4 of the course notes.

This programming assignment uses many of the ideas presented in sections 5, 6, and 7 of the course notes, so you are advised to read these notes and the following program specification carefully.

The Program Specification: PhoneBill

You may have heard about all the recent merger activity within the telecommunications industry. Recently MCI WorldCom announced that it plans to merge with Sprint. Other recent and pending mergers include TCI with AT&T, Ameritech with SBC, NYNEX with Bell Atlantic, and WorldCom with MCI. With all these mergers, many telephone billing systems will have to be modified or combined.

For this assignment, you will write a program that produces a telephone bill based on data that is recorded by the telephone network. The data will be fictitious, but it will be similar to what actual telephone systems use.

The rate plan that will be used to calculate the cost of the calls is the little known but aptly named "Four Cent Fridays" plan:

a. There is a minimum total charge of $4.95 per month plus tax.
b. Any call started between 6:00 AM and 6:00 PM inclusive, Monday through Thursday, is billed at a rate of $0.30 per minute. These are called "Daytime" calls and will be abbreviated with a 'D'.
c. Any call started before 6:00 AM or after 6:00 PM, Monday through Thursday, is billed at a rate of $0.10 per minute. These are called "Evening" calls and will be abbreviated with an 'E'.
d. Any call started on Friday is billed at a rate of $0.04 per minute. These are called "Friday" calls and will be abbreviated with an 'F'.
e. Any call started on Saturday or Sunday is billed at a rate of $0.05 per minute. These are called "Weekend" calls and will be abbreviated with a 'W'.
f. All calls are billed in whole minute increments.

The data that is recorded by the telephone network for each call consists of the date the call was started, the phone number called, day of the week the call was started, the time the call was started, and the time the call ended. This data will be provided to you in an input file, one call per line. There will be two spaces in between each data element on the line. The format of the data will be as indicated below. Underscores (_) have been used to indicate where spaces occur on the line. There is a newline character at the end of each line.

`10/05/1999`	`_ _`	`540_555_2345`	`_ _`	`Tu`	`_ _`	`13:30`	`_ _`	`14:05`	↵
date the call was started	two spaces	telephone number called	two spaces	day of the week the call was started	two spaces	time the call was started	two spaces	time the call ended	newline character

Each of these formats will be explained in greater detail below.

Date

The date will be in the format `MM/DD/YYYY` where:

MM indicates the month (01=January, 02=February, 03=March, ... 12=December).
DD indicates the day of the month (01 is the 1st, 02 is the 2nd, ... 31 is the 31st).
YYYY indicates the year (i.e. 1999)

Telephone number

The telephone number will be a ten digit number specified in three parts. The first three digits are the area code. The next three are the exchange. The final four digits are the station number. There will be a single space in between each part. For example, in the phone number `540 555 1234`:

`540` is the area code
`555` is the exchange
`1234` is the station number

Day of the week

The day of the week the call was started on will be represented by two-letter abbreviations for the days as shown below:

`Mo` = Monday, `Tu` = Tuesday, `We` = Wednesday, `Th` = Thursday, `Fr` = Friday, `Sa` = Saturday, `Su` = Sunday

Start and End Times

The start and end times for the calls will be given in 24-hour notation with the hour first, then a colon (:), and then the minutes. Here are some examples:

	AM/PM Notation	24-hour Notation
midnight →	`12:00am`	`00:00`
	`12:01am`	`00:01`
	`3:13am`	`03:13`
	`1:30pm`	`13:30`
	`11:59pm`	`23:59`
noon →	`12:00pm`	`12:00`

You will use this data to compute the total number of minutes for each phone call, the cost for each call, and the total cost for all the calls. This information will then be printed in a format as explained in the section on the output file description.

Input file description and sample:

Your program **must** read its input from a file named `calldata.txt` — use of another input file name will result in a score of zero. The input file will consist of lines of call data as described in the section above. There will be one line for each call. You will not be given the number of calls in the input file, so your program will need to recognize when there are no more calls to be processed and terminate correctly.

You may assume that all the input values will be logically correct (no negatives, for instance). You may also assume that all calls will be at least one minute and will be less than 24 hours in duration.

Here is a sample input file:

```
10/05/1999  540 555 1234  Tu  09:55  10:28
10/05/1999  540 555 7890  Tu  20:15  20:44
10/08/1999  212 555 4567  Fr  07:16  08:32
10/09/1999  312 555 2345  Sa  23:11  00:12
10/10/1999  540 555 1234  Su  17:18  17:19
```

The items on each line will be separated by two space characters (i.e. there will be two spaces between the date and the phone number; there will be two spaces between the phone number and day of the week, etc.). However, the parts of the phone number will only be separated by one space. For example, in the sample input file shown above, there are two spaces in between the "1999" and the "540", but there is only one space between the "540" and the "555".

There are some tricks to reading in the input file. Initially, you might think about reading in the date like this:

```
int month, daynum, year;
char ch;

ifstream infile;
infile >> month >> ch >> daynum >> ch >> year;
```

However, there is a problem with this approach when reading values that might have a leading zero. It will not read a date such as `10/09/1999` correctly because of the leading zero in the `"09"` part. When C++ tries to read the `"09"` into the variable day, the zero in front makes it think that the `"09"` is an octal number (base eight). Since `"9"` is not a valid digit in octal, it only reads in the `"0"` into day. A similar problem occurs when trying to read in time values such as `09:23`.

The way around this is to read in these two-digit values as characters and convert them to integers. Here are the steps:

1. Read in the first character
2. Read in the second character
3. Convert these characters to the corresponding integer values
4. To get the overall result, multiply the first integer by 10 and add the second to it

If you try this out on some examples, you can see that it will work for numbers like "09" as well as ones like "15".

At the end of this specification is a function called `GetTwoCharsAsInt` that will help you read in two-digit numbers from the input file. GetTwoCharsAsInt takes an input file stream as a parameter, reads in the next two characters from that input stream, and returns an integer that corresponds to the value of these two characters interpreted as a single integer (using the steps 1-4 outlined above).

- **You are strongly encouraged to use this function in your program to read in the month, day number, and the hours and minutes for the start and end times**. Remember that you should write a function prototype at the beginning of your code in order to make use of the `GetTwoCharsAsInt` function (see slides 7.6-7.7 of the class notes).

- **To make the reading of the input file easier, you may assume that the year, area code, exchange, and station number will not start with a zero. They can be safely read in as integers using the extraction operator (>>).**

The result is that some values from the input file should be read using the `GetTwoCharsAsInt` function but that other values can be read as integers using the extraction operator (>>). To read in spaces and separator characters (such as ':' and '/') you can use the `get` function. For example, to read in the date you might do this:

```
int month, daynum, year;
char ch;
ifstream infile;

month = GetTwoCharsAsInt(infile);         // read in the month number
infile.get(ch);                           // read the slash
daynum = GetTwoCharsAsInt(infile);  // read in the day number
infile.get(ch);                           // read in the next slash
infile >> year;                           // read in the year
```

What to Calculate and How to Calculate it:

You will write a program to read the input file and use the given values to compute:

- the total number of minutes for each call
- the cost for each call based on the starting day of the week and starting time, as indicated in the rate plan
- the total charges for all the calls in the input file
- the tax on the total charges
- the total bill amount (charge + tax)

To receive full credit for this assignment, you must write and use at least two **functions** in your program. That is, two in addition to the `GetTwoCharsAsInt` function provided. Your functional decomposition should help you determine possible functions. The GTAs will be hand-checking your code to see if you have written and used at least two functions in your program. However, you are encouraged to use more functions if you think they are helpful. As a point of reference, the EAGS solution to this assignment uses over six functions.

For more information about functions, see section 7 of the class notes.

Output description and sample:

Your program must create an output file called `phbill.txt`. A sample output file produced from the sample input file above is shown below. The first line of your output should identify you by name, in a manner similar to the line shown. The second line should be blank.

The third line should read, "Details of calls and charges:" The fourth line should be blank. The fifth line should contain headings, exactly as shown.

The next set of lines should contain detailed information for each call in the input file. Each line should contain the following items in order:

1. The rate code for the call. This will be 'D', 'E', 'F', or 'W', depending on the day of the week and the time the call was made. You will need to use the rate plan table given earlier in this specification to decide what rate to use for each call.
2. The time the call started. You should print the time in 24-hour format, as shown.
3. The date the call started. You should print the date in MM/DD/YYYY format, as shown. Be sure to include any leading zeros as needed (i.e. 02/04/1999).
4. The phone number called. You should print the number in the format shown, with the area code in parenthesis, then a space, then the exchange, then a dash (minus sign), and then the station number.
5. The length of the call in minutes.
6. The cost for the call in the format shown.

After the detailed table of calls and charges, the next line should be blank. On the next two lines, print the total charges as shown. The total charges will be the larger of (a) sum of all the costs for the calls, or (b) $4.95. On the next two lines print the tax as shown. Calculate the tax as 8.25% of the total charges. The next line should be blank. The last line should contain the amount of the total bill as shown. The total bill amount is calculated as the total charges plus the tax.

All dollar amounts should be printed with two decimal places (as shown) and should **NOT** have a dollar sign ($) in front.

```
Phone Bill generated by Rob Capra

Details of calls and charges:

Rate    Time    Date         Number called    Length    Cost
D       09:55   10/05/1999   (540) 555-1234       33      9.90
E       20:15   10/05/1999   (540) 555-7890       29      2.90
F       07:16   10/08/1999   (212) 555-4567       76      3.04
W       23:11   10/09/1999   (312) 555-2345       61      3.05
W       17:18   10/10/1999   (540) 555-1234        1      0.05

Total charges:
18.94

Tax:
1.56

Total bill:
20.50
```

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

- All decimal values must be printed to show the same number of decimal places as in this sample.
- You must use the specified header and column labels, and include your name in the first line as shown.
- You must arrange your output in neatly aligned columns, with a label identifying the contents of each column. Use spaces, not tabs to align your output.
- You must use the same ordering of the columns as shown here.
- You must print a newline at the end of each line.

Programming Standards:

The GTAs will be hand-checking your program to grade your documentation.

You'll be expected to observe good programming/documentation standards. All the discussions in class about formatting, structure, and commenting your code should be followed. A copy of *Elements of Programming Style* is included with the course notes — if you don't have a copy we strongly suggest you read the on-line edition (available from the course web page).

Some specifics:

- Your header comment must describe what your program does.
- You must include a comment explaining the purpose of every variable or named constant you use in your program.
- You must use meaningful identifier names that suggest the meaning or purpose of the constant, variable, function, etc.
- Use named constants instead of variables where appropriate.
- Precede every major block of your code with a comment explaining its purpose.
- You must use indentation and blank lines to make control structures (like loops and if-else statements) more readable.

Hints:

This program requires that you read, understand, and apply a number of detailed specifications. Be sure that you read the specification carefully.

You should do a functional decomposition prior to writing your program. This will help you break the program into parts and determine what functions to write.

It is often a good idea to write and test your functions separately from the main logic of your program.

Testing:

Obviously, you should be certain that your program produces the output given above when you use the given input file. However, verifying that your program produces correct results on a single test case does not constitute a satisfactory testing regimen. You should test your program on all the posted input/output examples given along with this specification.

Provided Functions:

Here is a function to help you read in two digit numbers from the input file:

```
//*********************************************************
Function      GetTwoCharsAsInt

PURPOSE:      reads in two characters from the input file stream
              and returns the integer value represented
              by the two characters
              (i.e. read in “09” and return the integer 9)

PARAMETERS:   infile: the ifstream to read from

SIDE EFFECTS: two characters from the ifstream are read
//*********************************************************)
int GetTwoCharsAsInt(ifstream& infile)
{
      int  i1, i2;
      char ch1, ch2;

      infile.get(ch1);        // read in the first character
      infile.get(ch2);        // read in the second character

      i1 = ch1 - '0';         // convert the characters to
      i2 = ch2 - '0';         // the corresponding integers

      return (i1*10+i2);      // return the value
}
```