Computer Science 1631 - Assignment 4, Part 1 (Web Billing)

Date Given: Wednesday, October 21, 2020 (Part 1 only)
Due Date:
Part 1: Monday, November 2, 2020 before 11:59 p.m. sharp
NO LATE assignments will be accepted for Part 1!

Part 2: Monday, November 16, 2020 before 11:59 p.m. sharp
The *normal* course policy for late assignments will apply for Part 2.

The entire assignment will be marked once both parts are submitted. This means that Part 1 **will not** be marked & returned before Part 2 is due. Should you have any concerns with Part 1 you should contact your professor BEFORE Part 2 is due!

Notes

1) Part 1 **cannot be submitted late** because a solution will be placed in the course directory at 12:00 a.m. on Monday, November 2. You **must use this solution** for Part 2.

2) The total value of this assignment is 7% of the course grade, almost twice as much as previous assignments (Part 1 is worth `3.5`%). The final program will be much larger than any you have worked on so far in the course, but none of its individual components are difficult. The keys to developing a larger program are *top-down design* and *incremental development*. A top-down design breaks a problem down into a number of smaller problems (that will be implemented as functions) with clearly defined interfaces (the IPO specifications for the functions). This simplifies solving the problem because small problems are easier to solve than big problems. It also supports incremental development because lower-level functions can be stubbed out initially and worked on one at a time later. As you already know, a function stub is a shell that can be called and that returns any results required to test the caller. Because top-down design and incremental development skills can only be developed through practice, this assignment is split into two phases to guide you through the process. In Part 1, you will write the final version of `three` functions (these will **not** be changed in any way in the second phase) and stubbed versions of main and one other function.

Objectives

- To continue to build problem solving and algorithm development skills.
- To gain more practice with incremental development.
- To learn how to use the pass by reference mechanism effectively, not only for arguments that are strictly output arguments, but also for arguments that are for both input & output.
- To gain more experience with functions, decisions (and loops in Part 2).
- To continue to develop your C++ coding, documenting and debugging skills.

The Task

wCubed Web Hosting is a web hosting service. Its customers range from individuals who use wCubed to host a personal home page up to companies that use the service to run an e-commerce site. The amount of disk space used on a wCubed web server is a major factor (actually, the **only** factor in this assignment) in determining a customer's monthly bill. wCubed offers its customers a variety of plans, each with a set amount of disk storage space.

You are to write a program that will run at the end of a month to process disk transactions on web hosting accounts and to print out customer bills. ***Part 1 is the first step towards achieving this goal – for this part, the program will only process a <u>single</u> customer***.

Input

All the required information is stored in a file. You don't have to use any specific file input commands – simply redirect the input into your program when you run it. For example, if your executable program is called `a.out`, and your input file is `part1.dat`, then type

```
            a.out < part1.dat
```

to execute the program and read all data from `part1.dat`. If you want to test from the keyboard, just type

```
            a.out
```

(but there won't be any prompts! You will have to know the order of the inputs to enter).

Format of the input file:

For Part 1, the file will consist of two lines of data for the customer. Lines in a data file are referred to as *records.* The first line is a ***header record*** that contains the account number, the base monthly charge for the customer's plan, the disk usage limit (in MB) for this account, and the amount of disk space in MB that was in use at the beginning of the month. The second line is the ***monthly transaction record*** that has the maximum amount of disk space used during that month, the amount in use at the end of that month, plus a value which indicates whether the transaction data is valid (we'll talk more about this value in a bit).

An example `part1.dat` could hold the following two lines:

```
951482 6.25 50 35.733
55.5 42.125 T
```

The header record indicates this customer has account number `951482`, their plan has a base monthly charge of `$6.25`, a disk usage limit of `50` MB, and were using `35.733` MB of disk space at the beginning of the month. The monthly transaction record tells us that `55.5` MB was the maximum amount of disk space used during the month and that `42.125` MB was in use at the end of the month.

The third value of T indicates that 'the transaction data is valid' – if it was a `F`, it would indicate 'the transaction data is invalid'. Character values are being used because Booleans can't be written or read. In Part 1 the monthly transaction record is a summary of possibly many transactions and it is possible that one or more of these transactions could contain errors. Thus, for the time being a character error flag is being used. In Part 2 this will be modified. The following is an example where a transaction error has been detected:

```
420143 12.50 100 95.4
250.5 74.225 F
```

It will be possible for an account number to be invalid (discussed in the Account Number Validation section). Account number errors must be handled; otherwise, no further error handling is required, i.e. you can assume that the remaining input is valid.

Processing

Bill Calculations

A customer's monthly bill will be based on the maximum amount of disk space used during the month *and* the amount of disk usage at the end of the month.

Customers are always charged the base monthly charge.

If a customer's *maximum* disk usage for the month exceeds their limit, they are also charged a surcharge for disk space in excess of the limit. This *over-limit surcharge* is incrementally calculated using the following table:

Amount over-limit (MB)	Per MB charge
<= 50.0	$0.15
> 50.0 but <= 250.0	$0.26
> 250.0 but <= 500.0	$0.37
> 500.0	$0.48

Read through the example following this table carefully – most students misunderstand this table when they first read it!

Thus, if an account is 250.82MB over during the month they would be charged $7.50 (50 * 0.15) for the first 50MB and $52.00 (200 * 0.26) for the next 200MB and then $0.3034 (0.82 * 0.37) for the last 0.82MB – for a total over-limit surcharge of $59.8034, which will be rounded to $59.80.

An **additional** surcharge (wCubed *loooooves* surcharges!) will be added if the customer's disk usage at the *end* of the month exceeds the disk usage limit. This *end-of-month penalty surcharge* will be either $0.05 per MB in excess of the limit (regardless of the amount in excess) or $5.00, whichever is greater.

Lastly, GST of 5% is charged to the total amount owed (base monthly charge + over-limit surcharge + end-of-month penalty surcharge).

Account Number Validation

A valid account number must meet the following criteria:

- A valid account number must be exactly six digits long.
- We will say the rightmost of the 6 digits is digit 1. The leftmost digit is digit 6. We will call digit 6 the *check digit*: it is compared to the units digit of the result of summing each of the preceding 5 digits multiplied by their respective position, i.e. $\sum i * d_i$. If the check digit does not equal the units digit of the sum, then the account number is invalid.

For example, account 951482 is valid since
`(5 * 5) + (4 * 1) + (3 * 4) + (2 * 8) + (1 * 2) =` 59 and the units digit of 59 (the 9) equals the check digit.

Output

The program will output either a month-end bill or an error message.

The month-end bill contains:

- the account number,
- the disk usage limit for the plan,
- the amount of disk space in use at the beginning of the month,
- the maximum amount of space used during the month,
- the amount of space in use at the end of the month,
- the base monthly charge for the plan,
- the over-limit surcharge (if any),
- the end-of-month penalty surcharge (if any),
- GST for the total amount,
- and the total amount due (including GST).

The statement should be neatly formatted **exactly as shown below**, with one item per line and with disk values being formatted to three decimal places and dollar values being formatted to two decimal places. Each numeric value is labelled and right-justified in a column. How to do this will be covered in class.

Extended Examples

The following are 4 runs of the program each showing the input file contents and the resulting program output.

```
Input 1
951482 6.25 50 35.733
55.5 42.125 T

Output 1
Account Number:       951482

Disk Allotment:         50.000 MB
Beginning of Month:     35.733 MB
Maximum during Month:   55.500 MB
End of Month:           42.125 MB

Base Charge:              6.25
Over-limit Surcharge:     0.82
GST:                      0.35
Amount Due:           $   7.43
```

Notice that charges that are not applied must not be displayed – in this example the customer has not exceed the maximum limit at the end of the month so the end-of-month penalty surcharge does not apply and is not printed.

```
Input 2
420143 12.50 100 95.4
250.5 74.225 F

Output 2
Bad monthly transaction record
*** Account 420143 not processed due to errors ***
```

```
Input 3
751482 11.50 100 97.0
200.2 12.12 T

Output 3
Invalid account number 751482 found
*** Account 751482 not processed due to errors ***
```

```
Input 4
300143 1.10 50 17.0
123.42 100.01 F

Output 4
Invalid account number 300143 found
Bad monthly transaction record
*** Account 300143 not processed due to errors ***
```

```
Input 5
951482 6.25 50 35.733
155.5 150.200 T

Output 5
Account Number:       951482

Disk Allotment:         50.000 MB
Beginning of Month:     35.733 MB
Maximum during Month:  155.500 MB
End of Month:          150.200 MB

Base Charge:              6.25
Over-limit Surcharge:    21.93
Penalty Surcharge:        5.01
GST:                      1.66
Amount Due:           $  34.85
```

Detailed Specifications

A basic main function must be completed in the Part 1. It will read the customer header record and then call functions 1, 3 and 4 (function 1 calls function 2). In the main function only i**f the customer's records are error free then will the calculations and the printing of a statement be done; otherwise the main**

function will print the message indicating the account was not processed, i.e. the last line in each of the last three examples.

You are to implement the following four functions:

1. `read_and_process`, which reads the **monthly transaction record**. The function will receive the account number and the disk usage at the start of the month. The account number will be validated and if found to be invalid, an appropriate error message will be printed. *Regardless of whether the account number is valid or not, the monthly transaction record will then be read.* Also regardless of whether or not the account number is valid, if the transaction error item is an 'F', then an appropriate error message will be printed (see the Extended Example). The function will return the maximum disk usage, the disk usage at the end of the month and a Boolean value indicating whether either an invalid account number or invalid data has been encountered.

   It is important to read and follow the above description exactly, especially the order of processing and parameters being passed. While these items may not seem to make sense now, they will make sense once you get to Part 2. Failure to follow these instructions will result in grade deductions.

2. `is_valid_account_number`, which receives an account number. The function checks the account number according to the process specified in Account Number Validation and returns a Boolean true if the account number is valid and false otherwise. This function does not print an error message – that task is left to `read_and_process.`

3. `calculate_bill`, which receives the base charge, the disk usage limit, the maximum disk usage for the month and the disk usage at the end of the month. The function calculates and returns the over-limit surcharge, the end-of-month penalty surcharge , the amount of GST owed and the total amount of the bill. The method of calculating all of these charges has been previously described in the Bill Calculations section.

4. `print_bill`, which receives all of the values that are printed on a statement. This function prints out a statement to the screen in the format shown in the Output section.

Development Guidance

Get into the habit of carefully designing every program before starting to code. The pseudocode design for every program module (main program or any other function) should be a complete solution – essentially C++ code without syntax and details such as output formatting. It is easier to understand, develop, and modify the logic at the pseudocode level because it is not cluttered up with details which are not part of the algorithm's logical structure. All aspects of the problem should be solved in the algorithm design phase, so that you only need to think about syntax when coding. Carefully check the algorithm to ensure it is correct and meets the specifications of the assignment.
The steps in designing a program and the order in which they should generally be performed are listed below. The first step and most of the second have been done for you in this assignment. The fourth is not required until Part 2.

1) Sketch a design of the main program and decide what parts of the problem will be assigned to functions called by main.

2) Specify each function by describing in a sentence or two *what* it will do and by listing its parameters on an IPO diagram. Then forget about this part of the problem for now. Don't worry about *how* the function will do its job until later.
3) Complete a pseudocode design for the main program, including all arguments in function calls. Then code function main from it.
4) Design an algorithm for each function, looking at it as an independent subprogram. If necessary, push part of the processing down into other functions, in the same way as was done for the main program.

Documentation Required in Part 1

The normal ID block must precede the main function, but for this part NO block comment is required, because the main program will be altered in Part 2. Full documentation for the four functions is required.

Hand In

Specific submission instructions and test data will be provided a couple of days before the assignment is due.