Computer Science 1631 - Assignment 4, Part 2 (Web Billing)

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

You ***must*** use the Part 1 solution a4p1.cpp for Part 2. It will be placed in the asg4 subdirectory of the course directory at 12:10 a.m. on Tuesday, November 3. Copy this file to your home directory and add the code and documentation required for the second part of the assignment.

Overview

This assignment requires a program to create month-end bills for the customers of a web hosting company. It produces a bill for each customer, or, if there are errors in the data provided for a customer, it lists all the errors detected, along with the line in the file the error occurred. (This way, a data entry clerk can correct the data and re-run the program.) This builds on Part 1.

As previously stated you are to revise the provided solution to achieve the revised problem specifications. The provided code is ONLY to be changed in the ways specified in this document. *Re-writing the provided code because you don't like the style is NOT allowed and will result in a significant grade reduction.*

New Requirements (High-Level Overview)

The program is to be modified to deal with the following changes to the requirements:

1. rather than processing a single customer, the program is to handle all customers
2. in Part 1 the monthly transaction record contained totals of the customer's activities for the month. In reality, a customer's monthly activity is a series of individual transactions. The monthly transaction record is no longer in the file, but is instead replaced by all the transactions for a customer. The program will now need to read these transactions and determine the maximum amount of disk spaced used during the month, the amount of disk spaced used at the end of the month
3. data validation on an account number and each transaction record is required and detailed error messages are also needed

No other changes should be required to the provided code. However, you are allowed to replace the code in the `is_valid_account_number` function that decomposes the account number into individual digits with code that uses a loop – but there are no grades allocated for changing the `is_valid_account_number` function.

Detailed Specs

Unlimited Customers

The program needs to handle any number of customers (and ends when there is no more data in the file). Modifications to handle this should be limited to a few minor changes to the main program.

Transaction Processing

A customer's monthly transactions consist of a series of disk changes: increases and decreases. Therefore, the input file will contain the following lines for each customer:

- the customer header record (unchanged from Part 1), and
- all the transaction records for a customer.

For example the data for one customer might be:

```
951482 6.25 50 35.733
951482 C 5.250
951482 D 15.111
951482 C 29.628
951482 D 10.200
951482 D 3.175
-1 X -1
```

← header record

this customer's transaction records

Each transaction record consists of an account number, a transaction code and an amount. The account number should match the account number from the header record. The transaction code identifies the type of transaction with `'C'` being an increase in the amount of disk space being used and `'D'` being a decrease in the amount of disk space being used.

The disk transactions appear in the file in the order in which they occurred during the month. This means that the amount of disk space in use at every point during the month can be calculated from the amount in use at the beginning and the transaction amounts. For example, the account above was using `35.733` MB at the beginning of the month and then `5.250` MB was added, bringing the amount used to `40.983` MB. Adding and subtracting each of the disk transactions can be used to determine the maximum and the amount in use at the end of the month.

The `-1 X -1` record is a sentinel that marks the end of the transaction records for an account. There may be no transaction records for a customer, but it is guaranteed that there will be a sentinel record. **All three items must be present for this to be a valid sentinel**. (Countless students have misunderstood or neglected the previous statement over the years – don't be one of them!)

Data Validation and Reporting

Code to validate the account number in the account header was done in Part 1. In the event of an invalid account number in the header record, an error message should be output, along with the line number (in the file) of this bad header record. Also, the corresponding transaction records need to be read and ignored (since this is an invalid account, no further error checking is required). Write a function `read_and_ignore` to read and ignore transaction records.

It is also possible that an error can occur in a transaction record. Transaction records need to be checked for the following errors:

- an account number not matching the account number in the header record
- an invalid transaction code (`'C'` and `'D'` are the only valid codes)
- a negative transaction amount

In order to specify in which line an error occurs, a variable to maintain the line count must be used. This must be declared in the main program and passed to `read_and_process`. (If you set it up in `read_and_process`, it would be a local variable there. Since these disappear at the end of each call, it would not keep its value from call to call).

Every error must be reported with a meaningful message, as well as the line number where the error appeared. For example, if the first customer record in the data file was:

```
420143 12.50 100 95.4
420143 C 155.100
823456 Q 200.000
420143 D -99.999
420143 A 1001.75
420143 D 176.275
-1 X -1
```

then the error messages produced should look something like:

Error in line #3 - non-matching account number 8234566
Error in line #3 - invalid transaction code 'Q'
Error in line #4 - invalid transaction amount -99.999
Error in line #5 - invalid transaction code 'A'

Notice that:

- since the possible errors are unrelated, each transaction record is checked for all possible errors,
- all transaction records are checked even those after the first erroneous transaction record, and
- the error message from Part 1 ("Bad monthly transaction record") is no longer needed

In the event of a transaction record error, no processing to generate a statement for that customer is to be done. Remember that if either an invalid account number or one or more transaction record errors occur NO statement will be generated for this customer.

Line numbers are for the entire file NOT relative to each customer, i.e. if the previous customer's header record had been on the 10th line of the file then the first error would have been on line 12.

In the main program after all customers have been process and just before the program terminates the program should print statistics for the job. This should include the total number of lines processed, the number of accounts and the number of errors found in the data file. This information must be printed to the screen in the following fashion:

Processed # total lines. Processed # accounts. There were # errors.

Where # is the correct number of things in the data. The total lines is simply the line number variable whose addition has already been discussed. The other two will also require the addition of counter variables to the main program. It is left up to you to determine when and where these counters need to be updated and therefore, if they need to passed to other functions and which functions these might be. **Be careful as passing variables to a function that are not used or needed will result in a grade deduction!!**

Development Strategy:
It is best to break a problem into pieces and solve it one piece at a time. For this problem you could proceed as follows:

1. The `read_and_process` function needs to be modified in three ways: a) to process transaction records generating maximum disk use and end of month disk use, b) to handle errors in the transaction records and c) to read and ignore all transaction records in the event of an invalid customer account number. We will use a helper function - details to follow in step 2 - to do a) and b) for us.

2. Write a function `read_and_total` that will read a series of transaction records until the sentinel record is encountered. It will pass back the maximum monthly disk usage and the end of month disk usage. Assume at this point that all transaction records are valid – we'll worry about possible transaction errors next in step 3.

3. Once `read_and_total` correctly processes a series of valid transaction records, modify it to handle each of the possible transaction record errors. **Do this one error at a time.** Upon detecting an error, an appropriate message should be output. For now, ignore the line number. And remember: since an error occurred, this fact needs to be returned to the caller.

4. Once the `read_and_total` function completely processes all of a customer's transaction records, you can move on to handling an invalid account. There should be no need to change the `is_valid_account_number` function.

   In the event of an invalid account in a header record, the transaction records for this customer need to be read and ignored. This should be done by a function called `read_and_ignore`. (Again when first creating this function, you can ignore line numbers.)

   Once all of these modifications are complete you should have a program that will process a single customer with or without errors.

5. Now you should add the handling of line numbers to the program. The line number variable needs to be passed and returned to a number of functions. The key to correctly maintaining the line number is to *increment it whenever you read a complete line of data*.

6. The final modification is to change the main program to process multiple customers.

Documentation / Style

As always, your final submission must follow the requirements in the "Coding Standard" document.

Testing

Your instructor will supply official submission test data a day or two before the due date. However, prior to this, you should come up with your own input/output test values. In this way, you should attempt to determine, well in advance of the due date, whether or not your program works correctly.

Submission and Grading

As noted above, official test data and detailed submission instructions will be released 1-2 days before the due date.

Your work will be graded not only on program correctness, but also on quality of algorithm design, code style, readability, and quality of documentation.

Start the assignment right away. Do not wait until the official test data has been released. Instead, generate a few test cases of your own by hand before you even design and code the algorithm.