Dynamic Memory, Makefile and Git with C

Due: November 25, 2019 on myCourses at 23:55

Labs G and H will provide some background help for this mini assignment.
Week 7, slides 15 and 16 will help you with the linked lists.

YOU CAN WORK IN A TEAM OF TWO, if you want, for this assignment. Both team mates will hand in the same files.

QUESTION

Assume we have an SSV file called `students.ssv`. An SSV file is different from a CSV file because each field is delimited by the space character instead of the comma. A limitation of SSV files is that each field can only be a single word or number. The advantage of an SSV file is that it can be easily parsed using default C functions.

For example:

Assume we have the line: "`bob mary 10`" in a file.

Assume arrays: `char line[100], word1[10], word2[10], word3[10]`

Then I can write the following code:

```
fgets(line, 99, file_ptr);
sscanf(line, "%s %s %d", word1, word2, word3);
int n = atoi(word3);
```

The structure of the SSV file is fixed and is represented as follows:
ACCOUNT_NO AMOUNT

Where ACCOUNT_NO is an integer and AMOUNT is a float.

For example:

```
10 100.0
20 -50.5
10 -20.0
```

The above file states that there are two accounts, account number 10 and 20. According to the file account 10 was deposited $100.00 and then withdrawn $20.00, and account 20 was withdrawn $50.50.

Assume we do not know the number of records in the file beforehand, therefore we must build a dynamic data structure to store all the accounts. Assume the following linked-list structure:

```
struct ACCOUNT {
    int accountNumber;
    float balance;
    struct ACCOUNT *next;
};
```

Notice that the ACCOUNT structure does not store the records of the SSV file but instead stores the balance of the accounts. Given the example SSV file, from above, having three records; It would result in a linked list of two nodes, one node for account 10 and one for account 20. The balance of account 10 would be $80.00 and the balance of account 20 would be $-50.50.

Write a project that compiles to an executable named `./bank` and is made from three source files: main.c, ssv.c, and linked.c. Your source files must be built as modules with a makefile for the project. You must determine the number of .h files, if any, you will need. Your project must use a private git. You will submit the git log. Your log must have a minimum of five entries on different dates. This means you should use git from the beginning of this assignment. IF YOU ARE WORKING IN A TEAM, then your git log MUST be different from your teammate's log, and you will NOT use a shared git (just email the files to each other).

Your project must do the following:

(A) The main() function opens the SSV file, read the records one at a time, and calls the parse() function within the ssv.c file. The parse() function will return the fields back to the main() function. The main() function will then call the findUpdate() function from the linked.c file that will updated the linked list. The main() function will repeat the above steps for all the records in the SSV file. The main() function terminates by calling the prettyPrint() function from the linked.c file to display to the screen all the values in your linked list. Then program then terminates.

(B) The parse() function's signature in ssv.c is:
`void parse(char record[], int *acct, float *amnt);`
The array is a single record from the SSV file. The function uses call-by-reference to return to the main() function the account number and the transaction amount.

(C) The findUpdate() function's signature in linked.c is:
`void findUpdate(int account, int amount);`
The function assumes a private global linked list exists within the linked.c module. The function scans the linked list to find the given account number. If it find a node with the account number it then updates that node's balance with the transaction amount. If it does not find a node with the account number, it then creates a new node, assigned the amount to the balance, and adds the node to the linked list. If an error occurs, the function simply terminates.

(D) The prettyPrint() function's signature in linked.c is:
`void prettyPrint();`
The function assumes a private linked list exists within the linked.c module. This function displays one line of text on the screen for each node in the linked list. Each line of text follows this format:
```
ACCOUNT ID: nnnnn  BALANCE: $ nnnnnn.nn
```
The nnn is the actual spaces you must leave for the values. Both numbers must be right justified.

(E) To make a private linked list within the linked.c module, make a global private structure, `struct ACCOUNT *head = NULL;` pointer in the linked.c file. Do not share that variable. All the functions, within linked.c, will use the head pointer as a global variable.

IMPORTANT

You must use **mimi.cs.mcgill.ca** to create the solutions to this assignment. You cannot use your Mac command-line, Windows command-line, nor a Linux distro installed locally on your laptop. You can `ssh` or `putty` from your laptop to **mimi**, or you can go to the third floor of Trottier and use any of those labs, to complete this assignment.

WHAT TO HAND IN

Everything must be submitted to My Courses before the due date. Remember that you can hand in your assignment up to two days late but there will be a penalty of 5% each day. After that, your assignment will not be accepted. Please hand in the following:

- The git.log file
- The makefile file
- The C files: main.c, ssv.c, and linked.c
- Any .h files you needed
- The TA will use your makefile to compile your project.
- **Do not hand in the executable**
- Zip all these files into a SINGLE zip file so that myCourses will receive them correctly.
- Make sure to add comments to your C program.
- **Add your name and student ID number as a comment.**

HOW IT WILL BE GRADED

THE TESTING SCRIPT

A testing script will not be provided, instead an SSV file will be provided for testing `mini6tester.ssv`. The TA will use this file for testing.

POINTS AWARDED

The assignment is worth a total of 20 points.

- Question
  - 2 points – git.log
  - 2 points – makefile
  - 3 points – correct modular programming
  - 2 points – main()
  - 3 points – parse()
  - 5 points – findUpdate()
  - 1 point – prettyPrint()
  - 1 point – private linked list
  - 1 points – correct SSV file format

GRADING RULES

The following rules are followed by the TA when grading assignments:

- A program must run in order to get a grade (even if it does not run well). If it does not run (does not compile) it will receive a zero. (Make sure to run your programs from mimi.cs.mcgill.ca).
- The TA will grade using the mimi.cs.mcgill.ca server.
- All questions are graded proportionally. This means that if 40% of the question is correct, you will receive 40% of the grade.