Program 1

Due Jan 19 by 11:59pm **Points** 60 **Submitting** a file upload **File Types** tar
Available Jan 7 at 12am - Jan 24 at 11:59pm 18 days

This assignment was locked Jan 24 at 11:59pm.

Budget Buddy

Problem Statement

Every month it feels like money just keeps disappearing from your bank account. As one of your New Year's resolutions, you decide to start tracking your spending and categorizing it so you can see where all the money is going. Your friends and family think this is a great idea, so they decide to join you and use the same system.

To make all this tracking and organizing of the financial data secure and easier to manage, you need to create a program that will let your users log in to see their account, as well as import transactions and allow the users to filter them based on category, cost, etc.

Requirements

Command Line Arguments

When starting the program, the user will provide two command line arguments. The first command line argument will be the name of the file that holds the information about the users. The second command line argument will be the name of the file that holds the transactions that need to be imported and categorized.

User Login

Before the user can get information from the program they must login. You must prompt the user for an ID and password. The ID of the user must be all integers. If it isn't, you must re-prompt the user for an ID that is only ints. The password will be a string that can contain any amount of characters and numbers.

If the user fails to login after 3 tries, you must display an error message and quit the program. After a successful login, you must display their corresponding name, ID, and their updated account balance.

Sorting and Printing

Once logged in, the user should be prompted with a list of different ways to display the sorted transaction information. After they have chosen an option, they should be asked if they want the information printed to the screen or written to a file. If they choose to write to a file, they should be prompted for a file name. If the filename does not exist, the file should be created and the sorted information should be written to it.

Available Options (after login):

- Sort transactions by category
- Sort transactions by date
- Sort transactions by dollar amount
- Quit the program

Required Structs

The following structs are required in your program. They will help organize the information that will be read in (or derived) from the files. You cannot change the structs.

The user struct will be used when you read in the member information from the users.txt file.

```
struct user {
   string name;
   int id;
   string password;
}
```

The transaction and budget structs will be used to hold information from the budgets.txt file.

```
struct budget {
   int id;
   float balance;
   int num_transactions;
   struct transaction *t;
}
```

```
struct transaction {
   float amount;
   string date;
   string category;
   string description;
}
```

There are 5 options for category: Housing, Food, Entertainment, Personal Care, and Miscellaneous

Required Functions

You must implement the following functions in your program. You cannot change the function declarations in any way. You can add additional functions if you need to.

This function will dynamically allocate an array of budgets (of the requested size):

```
budget* create_budgets(int);
```

This function will fill a budget struct with information that is read in from budgets.txt. Hint: "ifstream &" is a reference to a filestream object. You will need to create one and pass it into this function to read from the budgets.txt file.

```
void get_budget_data(budget*, int, ifstream &);
```

This function will dynamically allocate an array of transactions (of the requested size):

```
transaction* create_transactions(int);
```

This function will fill a transaction struct with information that is read in from budgets.txt.

```
void get_transaction_data(transaction*, int, ifstream &);
```

You need to implement a function that will delete all the memory that was dynamically allocated. You can choose the prototype. Possible examples include the following:

```
void delete_info(user**, int, budget**, int);
void delete_info(user**, budget**, int);
```

Required Input File Format

Your program must accommodate the file formats as specified in this section. The users.txt file will have the following pattern:

```
<number of users in file>
<user name> <user ID#> <user password>
<user name> <user ID#> <user password>
<user name> <user ID#> <user password>
...
```

*Note: An example users.txt file can be found here: **users.txt**. This is not the same file that will be used to test your code, so you should also create your own covering any possible test cases.*

The budgets.txt file has a slightly different structure. The file information will be provided in sets (corresponding to each budget). Each budget will be accompanied by a listing of the transactions.

```
<number of budgets in file>
<budget ID#> <budget balance> <number of transactions>
<date of transaction> <transaction amount> <transaction description> <transaction category>
<date of transaction> <transaction amount> <transaction description> <transaction category>
<date of transaction> <transaction amount> <transaction description> <transaction category>
...
<second budget ID#> <budget balance> <number of transactions>
<date of transaction> <transaction amount> <transaction description> <transaction category>
<date of transaction> <transaction amount> <transaction description> <transaction category>
...
<third budget ID#> <budget balance> <number of transactions>
<date of transaction> <transaction amount> <transaction description> <transaction category>
...
```

*Note: An example budgets.txt file can be found here: **budgets.txt**. This is not the same file that will be used to test your code, so you should also create your own covering any possible test cases.*

Programming Style/Comments

In your implementation, make sure that you include a program header. Also ensure that you use proper indentation/spacing and include comments! Below is an example header to include. Make sure you review the style guidelines for this class, and begin trying to follow them, i.e. don't align everything on the left or put everything on one line!

```
/******************************************************
** Program: budget_buddy.cpp
** Author: Your Name
** Date: 01/01/2020
** Description:
** Input:
** Output:
******************************************************/
```

When you compile your code, it is acceptable to use C++11 functionality in your program. In order to support this, change your Makefile to include the proper flag.
For example, consider the following approach (note the inclusion of **-std=c++11**):

```
g++ -std=c++11 <other flags and parameters>
```

In order to submit your homework assignment, you must create a tarred archive that contains your .h, .cpp, and Makefile files. This tar file will be submitted to Canvas. In order to create the tar file, use the following command:

```
tar –cvf assign1.tar budget_buddy.h budget_buddy.cpp prog.cpp Makefile
```

Note that you are expected to modularize your code into a header file (.h), an implementation file (.cpp), and a driver file (.cpp).

You do not need to submit the txt files. The TA's will have their own for grading.

Program 1 Rubric

Criteria	Ratings	Pts
Program Header / Good indentation & Use of whitespace / Function Documentation At a minimum, header should contain author's name (1pt) and a description of the program. (1pt) Is code easy for the TA to read? Conditional blocks of code should always be indented. (3pts) Every function contains it's own initial block comment (or multiple lines of comments prior to the function definition) that provides the reader with an explanation of the function's purpose (3pts). -1 pt for each function that is missing a header (up to 3 point penalty).		8.0 pts
All functions designed in a modular fashion / No global variables -3 pts if there are any global variables used in the code. If functions are exceptionally long (greater than about 20 lines of actual code) this is a potential indication of poor modularity. -2 pts for each function that the TA concludes is poorly modularized.		7.0 pts
Command Line Functionality (2 pts) Program displays error and terminates gracefully if exactly two command line parameters are not provided. (2pts) Program displays error and terminates gracefully if 2 non-existent files are specified.		4.0 pts
Struct Usage Code defines/uses the user, budget, and transaction structs exactly as specified. -1 pts for any incorrect struct. It's okay if the code uses additional structs for other purposes.		3.0 pts
Memory Cleanup The Valgrind LEAK SUMMARY should report: "definitely lost: 0 bytes in 0 blocks". Each "new" operation should have a corresponding delete operation.		2.0 pts
Program Functionality Program does not crash during testing (e.g. segmentation fault or similar abrupt halts)		2.0 pts
Correct TAR file Program was submitted as a single TAR file that contains a working Makefile, an application .cpp file, a header .h file, and an implementation .cpp file. Exact filenames do not matter for the source code.		4.0 pts
Dynamic arrays of structs Dynamically allocated arrays of structs are used for both the budgets and the transactions.		4.0 pts
User authentication Program authenticates user against the credentials in file. -1pt if program does not provide explanatory message when user supplies invalid credentials. -1pt if program does not terminate after 3 incorrect password attempts. -2pts if the program allows access with the wrong credentials.		4.0 pts
Sort transactions by category Program sorts and displays the transactions by category.		4.0 pts
Sort transactions by date Program sorts and displays transactions by date		4.0 pts

Criteria	Ratings	Pts
Sort transactions by amount Program sorts and displays transactions by amount		4.0 pts
Program writes requested output to file The user is able to request that sorted output be redirected to a user-specified filename. The correct information is appended to that particular file.		4.0 pts
Program repeats Program repeats until the user chooses to exit.		2.0 pts
Function Usage Code defines/uses the function prototypes exactly as specified. -1 pt for any incorrect function prototype. It's okay if the code uses additional functions for other purposes.		4.0 pts
Total Points: 60.0		