ECE 220 – Computer Programming for Engineering – Winter 2019

Laboratory No. 4:
"Modular and Tailored"

Objective

The goal of this lab is to make you familiar with functions in C programs. Again, the lab will give you a chance to learn and use debugging – a special process that allows you to execute program line by line and see values of all variables. The program called debugger is a part of Visual Studio, CLion and Xcode.

Submission

Demonstrate your program to a TA of your section. It can happen during the following lab (your section). **The first hour of the next lab** will be dedicated to this purpose.

Problem Specification

You are asked to write a C program that manages a number of accounts in a bank. The program will create accounts, make deposits and withdraws, as well as calculate an interest. The program reads content of an input file that tells what banking operations have to be performed. It stores the results in an output file.

The program should be written as a set of modules (each module consists of two files .c and .h) as presented in Fig 1. Their descriptions are below:

- **Memory (.c/h)**: contains two functions to allocate and free memory, as well as to keep track of number of bytes allocated to the program.
- **Io (.c/h)**: contains functions to read from an input file, write results to an output file, and to test if the content of the output file generated by your program is the same as content of the expected output file (provided).
- **Bank (.c/h)**: contains all functions needed for handling account operations in a bank, these functions are called from the main.c (see Fig 2).
- **Main (.c/h)**: uses io.c functions to read line by line from the input file (provided); for each input line an appropriate function (from bank.c) is called from bank.c; and after it, calls io.c functions to write a result of the operation into an output file;
  - at the end, the program compares the output file with the expected output file (provided).

Please read carefully this manual, it contains many important information regarding components and behavior of the program.

Additionally, make sure you do PRE-LAB Quiz available on the eClass. It helps you evaluate your understanding of how pointers and memory allocation works.

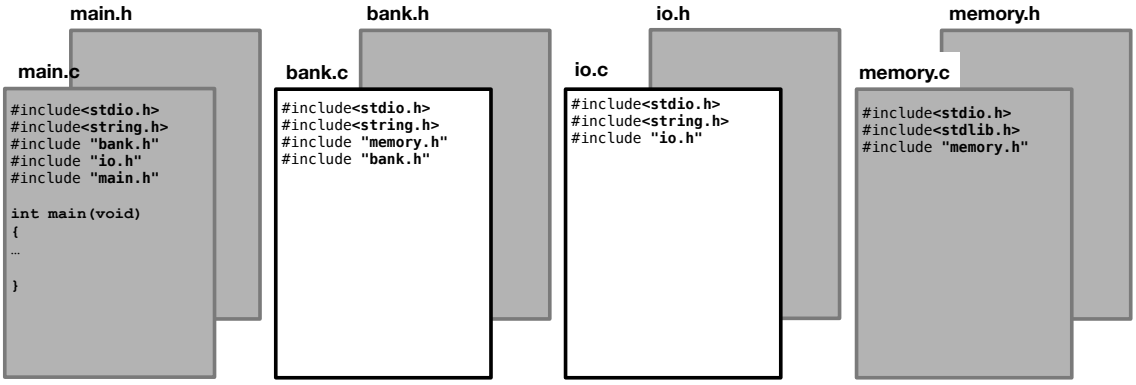

Fig 1. Overview of modules (files) of the program; shaded files are provided; the white ones are for you to complete (to write code for the functions).

The dependencies between modules, i..e, which modules call function calls from other different modules, are presented in Fig. 2.

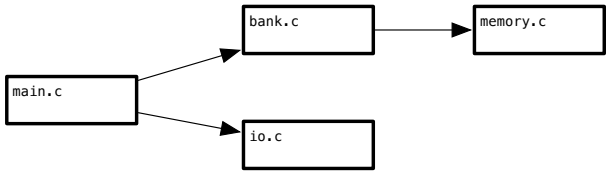

Fig 2. Overview of the interactions between different modules.

Description of input file

The **input file** contains a list of actions that needs to be performed. The actions span over a duration of a single year. Each line of the file has three or five values separated by white spaces:

<day> <name> <action> <type> <amount>

where:

<day> is a number from 1 to 365 inclusive indicating a day of a year;

<name> is series of characters without any white space in it representing a name of a person;

<action> can have one of the values *Open*, *Report*, *Deposit*, *Withdraw*;

<type> is either *Chequing* or *Savings*;

<amount> is a dollar value with two digits after the decimal point showing the cents. Note: Never store financial information as floating point values since they are rounded automatically when you do mathematical operations on them and money might be lost. Always convert values to cents and store it as *int* or *long long*.

An example of a file with a few action lines is shown below:

```
1   Marek  Open     Chequing 34.99
3   Antal  Open     Savings  300.00
7   Majed  Open     Savings  97.11
7   Antal  Report
…
```

Description of banking operations

Open: operation of opening a new account

Each customer is allowed to have only one *Chequing* account and one *Savings* account. If the account type already exists for a given customer you need to log the following message as output:

[<day>] <name> failed to open a new <type> account since it already exists

Otherwise you log:

[<day>] <name> opened a <type> account with an initial value of $<Amount>

Report: operation of storing, in the output file, information about all counts of a given customer

Whenever a customer asks for a report of their accounts, you need to save the information about all their accounts into the output file. Possible messages are presented below:

[<day>] <name> has two accounts: {Chequing account has a balance of $100.00 | Savings account has a balance of $100.00}

[<day>] <name> has one account: {Savings account has a balance of $100.00}

[<day>] <name> has no accounts

Deposit: operation of putting money into an account

When a customer tries to deposit some money into an account that does not exist, you need to output:

[<day>] <name> failed to deposit money into their <type> account since they do not have one.

Otherwise you put into the output file:

[<day>] <name> deposited $<Amount> into their <type> account.

Withdraw: operation of withdrawing some money from an account

If a customer tries to withdraw money from an account, there are three things that might happen: 1) either the account does not exist:

[<day>] <name> failed to withdraw money from their <type> account since they do not have one.

or 2) there is not enough money in the account:

[<day>] <name> failed to withdraw money from their <type> account since they are $<Missing_Amount> short

or 3) the withdraw is successful:

[<day>] <name> withdrew $<Amount> from their <type> account

Each type of account has specific characteristics.

Chequing account has a fee of $2.99 per 30 days. This fee will be subtracted from the account at end of the days 30, 60, 90, … of the year independent of when the account has been opened. If there is no enough money in the account to pay the fee, the balance of account should become zero.

Savings account gives customers daily interest of %0.01 on their money. This interest is deposited into their account at the end of each day.

At the end of the year, as your last line in the output file, you should store a line reporting if the bank has made or lost money due to these fees and interests. For example:

[365] The bank lost $162.08

or

[365] The bank made $3.60

Validation of the results of your program

To check the results of your program, use the function `compare()` (from the file io.c) to compare your results with the content of the output file provided to you (eClass). They should be exactly the same.

Description of a data structure storing information about accounts

The program uses functions to allocate and free memory (file memory.c). They are used to construct a data structure that contains information about all accounts created during execution of the program. An illustration of this structure is presented in Fig 3. Please, analyze it and make sure you understand how it is created.

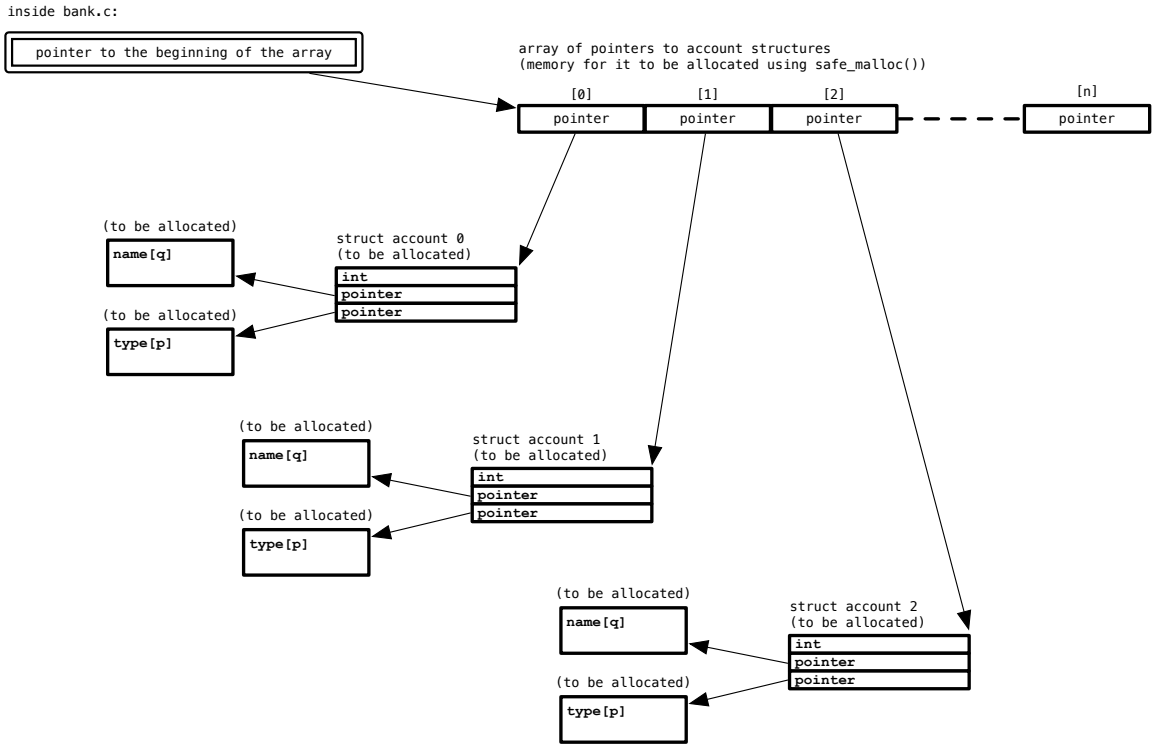

Fig 3. Overview of the data structure with a clarification of needed memory allocations (double border – a variable defined in bank.c; single border – allocated memory, and freed at the end of the program).

Important:
It is mandatory to follow the signature of the functions in the program template**.**

Implementation Details
Design your program first!!! Draw a diagram or a flowchart of the program. Use the provided flowchart as a starting point.

The template of the program and prototypes of the functions are below.

Hints
1. Start with a simple version of the program: at the beginning just reading lines from the input file, and so on … Such an approach is called "incremental development".
2. DO NOT DO EVERYTHING with the FIRST ATTEMPT.
3. Compile your program after adding few statements. DO NOT wait till the end!!!
4. **Use debugger**!!! It is the best tool to verify correctness of your program.

Template for your program

Files of the program are on the eClass. They are all files shown in Fig. 1, as well as two files required to run and validate your program:

- `sampleInput.txt` (it contains list of operations that your program has to perform)
- `testOutput.txt` (it contains the results of operations – the results you obtained running your program should be compared against the file)

Marking Scheme

This assignment is worth 6% of your final mark. A total number of points you can obtain is 100. The marking of the lab is done according to the following schema:

TASK	POINTS
Execution of your program	
Reading/writing from/into a file	/15
Opening account	/15
Depositing into account	/10
Withdrawing from account	/20
Reporting	/10
Matching the result to the expected output	/20
Subtotal	**/90 points**
Quality of code (easiness to read/comprehend your program)	
Naming and usage of variables	/5
Documentation (commenting)	/5
Subtotal	**/10 points**
Total	**/100 points**