ECE 220 – Computer Programming for Engineering – Winter 2017

Laboratory No. 2: "Let's Compute"

Objective

The goal of this lab is make you familiar with basic C program concepts regarding computations with numbers of different data types, and program structure (logic related to repetitions and selections). **This lab should be done individually, not in groups.**
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, Xcode and Eclipse.

Pre-lab

You are asked to prepare a flowchart of the program to write. This flowchart is a modification/a new version of the flowchart you did in Lab 1.

Submission

Demonstrate your program to a TA of your section. It can happen during the same lab or the following lab (your section). **The first hour of the next lab** will dedicated to this purpose. For the rest of the lab, you should focus on the Lab 2.

Lab Schedule:	Pre-lab (flowchart) checked on:	Demo the code from:
1st	Lab 1	
2nd	Lab 2	Lab 1
3rd	Lab 3	Lab 2
4th	Lab 4	Lab 3
5th	Lab 5	Lab 4 and Lab 5

NOTE: The Lab 3 has a pre-lab. Your pre-lab will be checked after the demo of the Lab 2.
In the case you are not able to finish your program and/or demonstrate it at the beginning of the 3rd lab, you are able to submit it during the lab on the following day(s) (any section). The penalty is 10% for each day of delay. If your section is on Thursday (and there is not lab on Friday, and of course Saturday and Sunday) then to avoid loosing too many points, you can send a code (only once) to both lab instructors, i.e., Jinbo (jinbo@ualberta.ca) or Ning (ncao@ualberta.ca) or Syed Tauhid Zuhori (zuhori@ualberta.ca)

Problem Specification

You are asked to write a C program that performs a number of operations based on answers obtained from the user, and the content of a file. This new program IS AN EXTENSION of the program from the Lab 1.

In particular, the new program reads from the file the user's living expenses over a number of weeks. This is a continuation of our ultimate goal, i.e., building a program that allows its user to monitor his/her budget. Once the budget is determined based on the values read from a file, the program should be able to accept current expenses and provide the user with the updates regarding his/her remaining funds.

In order to do so, your program has to read a single file that contains information about expenses related to a number of subcategories (as explained in Lab 1) for *N* weeks. A program should determine the weekly amounts spent in each category as an average over *N* weeks. These amounts will characterize weekly "allowances" for each category. After this is done, the program should "interact" with the user.

The specification of the program is following:

- the program should read the expenses from the file: **`cost.txt`** – this file should be downloaded from eClass, and put into the same directory where the sources files are; the first line of this file is a number – this number tells how many "weeks of data" are in the file; the rest of the file is a sequence of values where each line represents a value from a single category: fruits/vegetables, next line: others, next line: entertainment. The values for the next week follow.

  Example of `cost.txt` file

```
2
35.00
65.00
10.00
30.00
75.00
30.00
```

  the meaning of the lines in the file: `2` represents the number of weeks, `35.00` is a cost of fruits/vegetables (1st week), `65.00` is a cost of other expenses (1st week), `10.00` is a cost of entertainment (1st week), and `30.00` is a cost of fruits/vegetables (2nd week), `75.00` is a cost of other expenses (2nd week), `30.00` is a cost of entertainment (2nd week).

- you are asked to sum up these values for each item (fruits/vegetables, dairy products, others, and entertainment), and calculate average values per week, these average values would represent weekly budget values for each item; they should be displayed and the user should be asked if he/she is okay with the budget values or wants to modify them – the only allowed modification is a **percentage change**, the program should ask the user how many percent he/she want to increase/decrease the budget values; one percentage change – for example +10% - should be applied to **all** categories

- the program provides the user with a chance to enter his/her current expenses, the program should display a new menu:
  - enter expenses
  - show balance
  - exit

if "enter expenses" is selected, another menu is shown:
  - groceries (fruits, others)
  - entertainment
  - back to main menu

- the program should keep the current balance, where balance = budget-expense, in each category, and be able to display it to the user on demand

- if any of the categories goes below the budget, the program should display a message on the screen: "<put here a name of a suitable category> has negative balance", and should allow the user to modify a budget of **this category**.

Implementation Details

Design your program first !!!! Draw a diagram or a flowchart of the program. Show it to a TA before you proceed further.

The whole program is just one function `int main(void)`.

We have not covered structures in the lectures, therefore all variables can be simple ones (like in examples in the course slides).

The program should use loop and selection statements: `while`, `if-else`.

Use comments in your program. A comment should describe a goal/purpose of the following few lines of code. The comment should not "copy" the lines of codes, but enable a better understating of what the code suppose to do/accomplish.

Naming Convention: please follow the following rules when you name your variables, and functions (in the next labs).
1) use names that describe what the variable represents, or function does, **2)** decide a way of building a name and follow this rule in the whole code, for example, a function's name can be built using verb and object/result of the action – add_numbers (or addNumbers) – then be consistent, it means to not have names like numbers_multiplied but multNumbers; **3)** make the names short, you have to type them – so be reasonable.

A much more detailed description of coding conversion and program structure is contained in the document "Coding Convention" that is available on the eClass. Please download it, make yourself familiar with it, and follow it.

Hints

1. Start with a simple version of the program: used the program form the previous lab, modify it and add a section for reading a `cost.txt` file, then add a section for calculation of expenses in the category accommodation, 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.

Useful information

Opening a file for reading (Xcode):
You can read from and write to a file used two functions: `fprintf` and `fscanf` which are very similar to already known to you `printf` and `scanf`.
The first step is to open a file, and the last is to close it. See examples below. A special type is used here `FILE *`, the variable `file_id` plays the role of an identifier of the open file being open (its kind of like file ID – you use it anywhere you refer to the file). Two arguments of the function `fopen` are: a name of the file, and a mode in which should be open `"r"` means for reading, `"w"` – for writing.

NOTE: put `cost.txt` file into a directory of your choice and 'record' the path, for example `/Users/YourUserID/ece220/lab-02/cost.txt`
Then use this path in your code (see below).

`fscanf` (and also `fprintf`) has just one change – it has `file_id` at the beginning of arguments, the rest is as in the case of `scanf` (`printf`).

```
#include <stdio.h>
#define ERR -1
int main(void)
{
  FILE *file_id;
  int no_weeks;
  double fruit_vegetable;
  …
  file_id = fopen("/Users/YourUserID/ece220/lab-02/cost.txt","r");

  if (file_id == NULL) {
     printf("The file cannot be found\n");
     return ERR;
  }
  fscanf(file_id, "%d", & no_weeks);
  …
  fscanf(file_id, "%lf", & fruit_vegetable);
  …
```

```
  fclose(file_id);

  return 0;
}
```

Another way of reading a content of file is to use a function called `fgets`. The function reads in only characters and puts it into a `char` array. It takes three input parameters: a `char` array (it will be used to store read input), size of this array, and file ID. See the example below:

```
#include <stdio.h>
#include <stdlib.h>
#define ARR_SIZE 30
#define ERR -1
int main()
{
   FILE *file_id;
   char in_str[ARR_SIZE];
   int no_weeks;
   double fruit_vegetable;
   …
   file_id = fopen("/Users/YourUserID/ece220/lab-02/cost.txt " , "r");
   if(file_id == NULL) {
      perror("The file cannot be found\n");
      return ERR;
   }
   fgets (in_str, ARR_SIZE, file_id);
   no_weeks = atoi(in_str);
   …
   fgets (in_str, ARR_SIZE, file_id);
   fruit_vegetable = atof(in_str);
   …
   fclose(file_id);

   return(0);
}
```

Opening a file for reading (Visual Studio):
You can read from and write to a file used two functions: `fprintf` and `fscanf_s` which are very similar to already known to you `printf` and `scanf_s`.
The first step is to open a file, and the last is to close it. See examples below. A special type is used here `FILE *`, the variable `file_id` plays the role of an identifier of the open file being open (its kind of like file ID – you use it anywhere you refer to the file). Three arguments of the function `fopen_s` are: file identifier, a name of the file, and a mode in which should be open `"r"` means for reading, `"w"` – for writing.

NOTE: put `cost.txt` file into directory:
… `YourProjectDirectory\YourProject_name\YourProject_name`
(you can also apply the same approach as presented for Xcode)

`fscanf_s` (and also `fprintf`) has just one change – it has `file_id` at the beginning of arguments, the rest is as in the case of `scanf_s` (`printf`).

```
#include <stdio.h>
#define ERR -1

int main(void)
{
  FILE *file_id;
  errno_t err;
  int no_weeks;
  double fruit_vegetable;
  …
  err = fopen_s(&file_id, "cost.txt", "r");
  if (err) {
     printf("The file cannot be found\n");
     return ERR;
  }
  fscanf_s(file_id, "%d", & no_weeks);
  …
  fscanf_s(file_id, "%lf", & fruit_vegetable);
  …
  fclose(file_id);

  return 0;
}
```

You can also use a function called `fgets`. The function reads in only characters and puts it into a `char` array. It takes three input parameters: a `char` array (it will be used to store read input), size of this array, and file ID. See the example below:

```
#include <stdio.h>
#include <stdlib.h>
#define ARR_SIZE 30
#define ERR -1
int main()
{
   FILE *file_id;
   errno_t err;
   char in_str[ARR_SIZE];
   int no_weeks;
   double fruit_vegetable;
   …
   err = fopen_s(&file_id, "cost.txt", "r");
   if(err) {
      perror("The file cannot be found\n");
      return ERR;
   }
   fgets (in_str, ARR_SIZE, file_id);
   no_weeks = atoi(in_str);
   …
   fgets (in_str, ARR_SIZE, file_id);
   fruit_vegetable = atof(in_str);
   …
   fclose(file_id);

   return(0);
}
```

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
Flowchart	
Pre-lab Task: Quality of flowchart (flowchart's ability to convey the main steps of the program, easiness of its understanding)	**/10 points**
Execution of your program	
Opening a file and reading values from it	/10
Calculations of weekly budgets for each category	/5
Interaction with the user: modification of the budget values	/10
Interaction with the user: entering current expenses	/10
Checking and modifying a budget of category with negative balance	/5
Displaying final balance of each category.	/5
Subtotal	**/45 points**
Easiness of interaction with your program	
Clear and well structured messages	/5
"Friendly" attitude of the program when talking to the user	/5
Subtotal	**/10 points**
Quality of code (easiness to read/comprehend your program)	
Naming and usage of variables	/10
Design (logic structure)	/10
Documentation (commenting)	/5
Layout (indentation/spacing)	/10
Subtotal	**/35 points**
Total	**/100 points**