ECE 220 – Computer Programming for Engineering – Winter 2017

Laboratory No. 1: "Conversation with a machine"

Objective

The goal of this lab is make you familiar with basic C program concepts regarding interaction with the user (asking questions and accepting answers), and program structure (logic related to repetitions and selections). **This lab should be done individually, not in groups.**
The lab will also 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 variables. The program called debugger is a part of XCode, Visual Studio, or Eclipse

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 no. 2.
NOTE: The Lab 2 has a pre-lab. Your pre-lab will be checked after the demo of the Lab 1.
If you are not able to finish your program and/or demonstrate it at the beginning of the 2nd 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 the lab instructors, i.e., Jinbo (jinbo@ualberta.ca), Ning (ncao@ualberta.ca), or Tauhid (zuhori@ualberta.ca).

Problem Specification

You are asked to write a C program that allows the user to interact with a machine. In particular, the program should ask the user a number of questions and stores his/her answers in program variables.

Imagine that the ultimate goal is to build a program that allows its user to manage his/her expenses. It means, that based on the information provided by the user regarding his/her spending over a number of weeks, the program will estimate a weekly budget for the user. This program is a subject of two labs: 1 and 2.

During this lab, you are asked to write a program that asks the user a number of questions that determine a "spending behavior" of the user. A program you about to write should "interact" with the user and ask him/her different questions, and "remember" his/her answers.

The specification of the program is following:

- the program should ask the user for his/her name; any time the program interacts with the user it should use this name
  NOTE #1: even if the user enters name all lowercase, the program should display his/her name with the first letter uppercase

NOTE #2: you as a designer of the program should make it sure that the length of user's name is not longer that the length of an array you use to store it, if it happens that user's name is longer, you should exit the program asking him/her to provide a shorter version of his/her name when he/she runs the program again

- the program has to ask the user for details regarding two spending categories (see below); the program should give the user an option which category he/she wants to "deal with" at a given point of time, in other words the program should have a simple (command line) menu
    - GROCERIES
    - ENTERTAINMENT
    - EXIT

  that allows the user to select a spending category or exit the program

- if the category "GROCERIES" is selected, the program should ask the user to enter a cost for each subcategory:
    - FRUITS/VEGETABLES
    - OTHERS
    - back to main menu

  only numerical (positive) value is allowed, the user HAS to provide a value how much money per week he/she spends in each subcategory (fruit/vegetables, others), even if it is zero; if one of the options has been not selected and the user wants to go back to the main menu, the program should display a message "not all subcategories filled out", and display the above menu again;

- if the category "ENTERTAINMENT" is selected, the program should display the following menu:
    - ENTERTAINMENT COST
    - back to main menu

  only numerical (positive) value is allowed, the user has to provide a value per week, it cannot be left unanswered

- if "EXIT" is selected, the program should display entered values and exit (stop)

- each of the options has to be selected at least once before the user is allowed to select EXIT

- the program should have a mechanism preventing introduction of wrong/incorrect information, i.e., the program should be able to prevent accepting a negative number; the program should detect that a negative number has been entered and then ask the user again for entering a number again immediately, assume that the user will NOT enter any letter.

- all the values/answers should be stored in variables in your program. Please use variables with meaningful names. At the very end of your program, when the user selects "EXIT", the program should print all entered values.

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 class; therefore, all variables can be simple ones (like in the examples in the course slides).

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

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: the first step can be just asking for a name, then the second step – menu, then the third step one of the groups … 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/functions

Control structures

```
while(condition) {
…
}
```

This instruction works in the same way as `while` in MATLAB©. It is a loop that is executed as long as the condition is evaluated to true. Please note usage of brackets – both `( )` and `{ }`.

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
  int value = 0;

  …
  while (value < 5) {
    …
    value = value + 1;
  }
  …
}
```

```
if(condition) {
  … // part1
}
else {
  … // part2
}
```

As above, this instruction works as the one in MATLAB©. If condition is true `part1` is executed, if condition is not true the `part2` is executed. Please note usage of brackets – both `( )` and `{ }`.

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
  int value;

  …
  if (value == 5) {
    … // part1
  }
  else {
    … // part2
  }
  …
}
```

Input functions

`scanf()`

The function is "equivalent" of the function `printf()`, but instead of writing the text and values to the screen, it reads an input provided by the user via a keyboard.
In its simplest form, its "control-string" (Lecture 01-Introduction, slide 25) contains just one conversion specifier: `%d` if you want to read an integer, `%f` for float, `%s` for string.
IMPORTANT: in the case of `%d` and `%f`, you have to put `&` in front of the variable's name, but not in the front of the string (array) name. See example below. The explanation will come later, when we cover pointers.

Usage:

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
  int int_value;
  float float_value;
  char str_value[10];
  …
  printf("Enter an integer value: ");
  scanf("%d", &int_value);
  printf("Enter a float value: ");
  scanf("%f", &float_value);
  printf("Enter a string: ");
  scanf("%s", str_value);
  …
}
```

`scanf_s()`

In Visual Studio the function `scanf()` has been replaced with the function `scanf_s()`. It requires the buffer size to be specified for input parameters of type s (string), or c (character).

For example:

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{

  char str_value[10];
  char char_value;
  …
  scanf_s("%9s",str_value, _countof(s));
  …
  scanf_s("%c",&char_value, 1);
  …
}
```

`gets()`
It is another function, that reads the whole line from the keyboard (till you press enter) and treat this as a string no matter what you entered, even numbers.
NOTE: when you use it, you will see the warning: `…uses gets(), which is not safe`
The explanation will come later, when we cover pointers.
Usage:

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
  int int_value;
  float float_value;
  char str_value[10];
  …
  printf("Enter an integer value: ");
  gets(str_value);
  // conversion to integer
  printf("Enter an float value: ");
  gets(str_value);
  // conversion to float
  printf("Enter a string: ");
  gets(str_value);
  …
}
```

Conversion functions
There are a number of functions converting a string to different data types:
to `integer` – function `atoi()`,
to `double` – function `atof()`.

Function `atoi()` takes the C string `str` interprets its content as an integral number, and returns it as an `int` value. Function `atof()` takes the C string `str` interprets its content as an floating point number, and returns it as an `double` value. If `str` does not contains numbers (digits) – no conversion is done and zero is return.

Usage:

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
  int int_value;
  float float_value;
  char str_value[10];

  printf("Enter an integer value: ");
  gets(str_value);
  int_value = atoi(str_value);

  printf("value: %i\n", int_value);
  printf("Enter an float value: ");
  gets(str_value);
  float_value = atof(str_value);
  printf("value: %f\n", float_value);

  printf("Enter a string: ");
  gets(str_value);
  printf("value: %s\n", str_value);
}
```

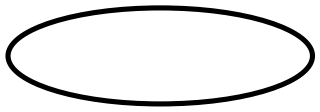
Function `toupper()`
The function returns the corresponding uppercase letter if one exists; otherwise, it returns the original character. It needs a header file `ctype.h`

Usage:

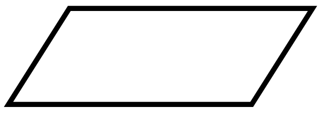
```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>

int main(void)
{
  char str_name[10];
  …
  name[0] = toupper(str_name[0]);
  …
}
```

Program Exit
In order to exit a program in any place, you simply use `exit(1);`

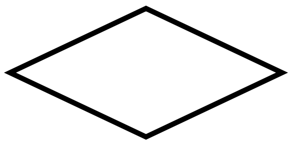
Flowchart
It is a graphical representation of a program. Built using the following elements:



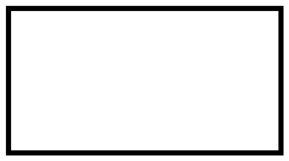
an oval indicates the beginning of a section of code



a parallelogram indicates an input or output

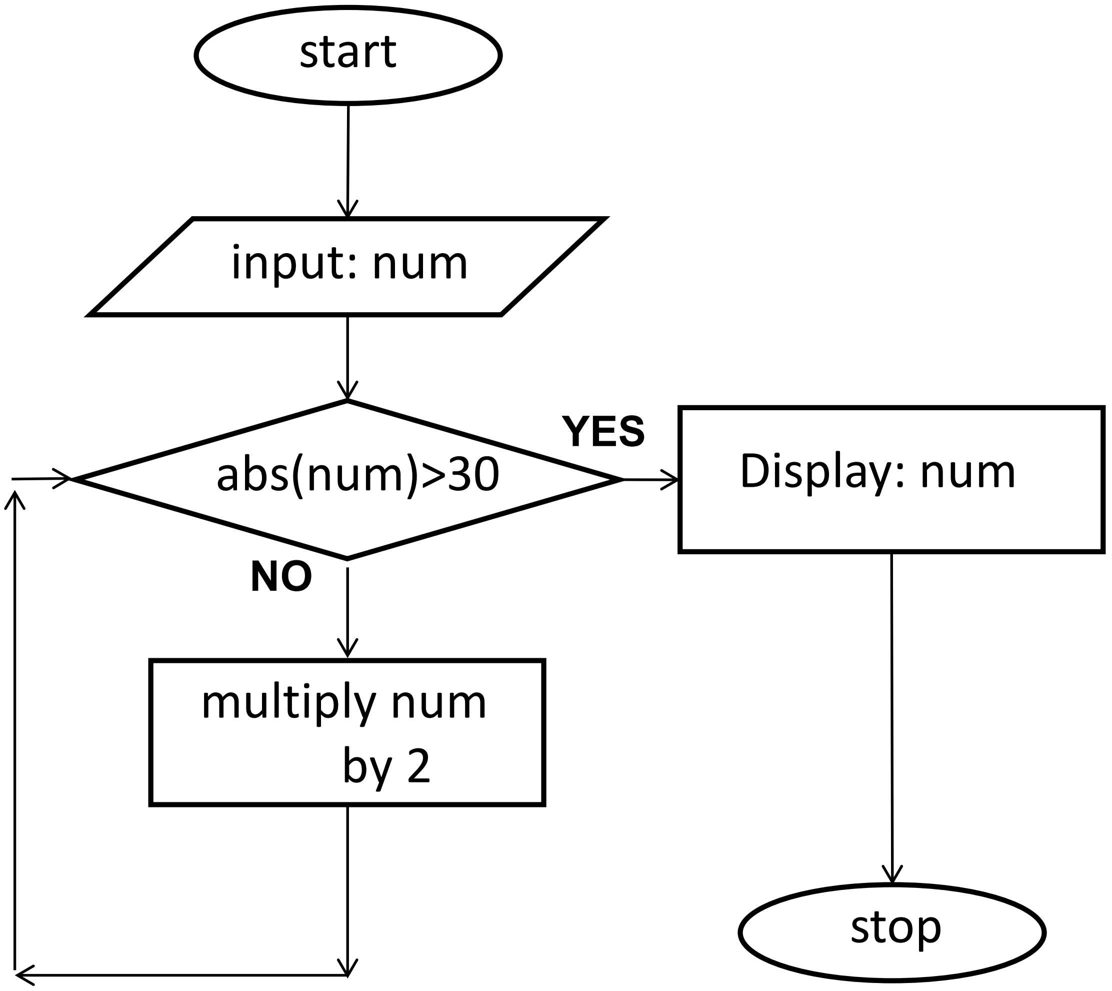


a diamond indicates a decision point



calculations are placed in rectangles

Example:


start
input: num
abs(num)>30
YES
NO
Display: num
multiply num by 2
stop

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	
Quality of flowchart (flowchart's ability to convey the main steps of the program, easiness of its understanding)	**/10 points**
Execution of your program	
Displaying user's name with the first letter uppercase (if entered lowercase)	/5
Preventing from too long user's name	/5
Display menu/submenu	/5
Preventing from entering negative values for any asked cost	/10
Forcing all answers in categories "GROCERIES" and "ENTERTAINMENT"	/10
Allowing the user to enter information about each group multiple times (until (s)he chooses "exit")	/10
Subtotal	**/45 points**
Easiness of interaction with your program	
Clear and well-structured messages	/5
Easy and straightforward selection of groups and "exit" option	/5
"Friendly" attitude of the program when talking to the user	/5
Subtotal	**/15 points**
Quality of code (easiness to read/comprehend your program)	
Naming and usage of variables	/10
Design (logic structure, layout: indentation/spacing)	/10
Documentation (commenting)	/10
Subtotal	**/30 points**
Total	**/100 points**