S22. LAB 1

Introduction to C, Function Declaration vs. Definition, Basic I/O (scanf/printf, getchar/putchar, input/output redirection)

Due: May 18 (Wed), 11:00 pm. Total mark: 100 pts

Pre-lab exercise:

Review the videos and instructions regarding coding environments for C, and decide your coding environment, e.g., sit in the LAS1006 lab physically, or, connect to the lab environment remotely, or, work locally and then transfer to the lab. If you cannot decide, contact the instructor for a tailored solution.

Problem 0 printf() in Java (5 pts)

Specification

While this is a course on C programming, let's start our very first lab with a JAVA exercise. Download the partially implemented Java program `Hello.java` . This program reads a name and an integer from the user, then outputs the double and triple value of the entered integer.

Implementation

- First, complete the `println()` statement, so that it produces the output as shown below. Don't add new lines in the program. Do all the formatting in `println()`. Probably you need to use string concatenations + multiple times in `println()`.
- Next, complete the `printf()` statement so that it produces the same output, as shown below. Do not use string concatenations in printf, i.e., don't use + . Again, don't add new lines of code. Do the printing in one call to `printf()` .

Objectives

The purpose of this exercise is for you to:

- be aware that `printf` also exists in JAVA (actually JAVA borrows the idea from C)
- learn the syntax of `printf` in JAVA and C (the formatting syntax are the same)
- observe that sometimes using `printf` is easier than `println`, the code is cleaner (IMHO)
- be aware that Java program can be compiled and run in command-line (as shown in class)

Sample Inputs/Outputs: (Don't use an IDE such as Eclipse) In a terminal, navigate to the directory where the Java file is located. E.g., if you have created a directory **`2031`** on your lab home directory, and a sub-directory **`lab1`** in directory **`2031`** and develop the programs there, then navigate to the directory by issuing **`cd 2031/lab1`**

red xxx % is the terminal prompt if you login to the **red** server using ssh. If you sit in the lab, or use the Remote lab connection, your prompt could be **eaYY xxx%** where YY is the terminal id that you chose.

```
red 300 % javac Hello.java
red 301 % java Hello
Please enter your name: Judy
Please enter a number: 22
Hi Judy, input is 22. Double and triple of 22 is 44 and 66, respectively
Hi Judy, input is 22. Double and triple of 22 is 44 and 66, respectively
red 302 % java Hello
Please enter your name: Joe
```

compile java program in command_line

run java program in command_line

```
Please enter a number: 100
Hi Joe, input is 100. Double and triple of 100 is 200 and 300, respectively
Hi Joe, input is 100. Double and triple of 100 is 200 and 300, respectively
red 303 %
```

Submission (from lab):

In a lab terminal, navigate to the directory where your file is located, and then issue the command `red 310 % submit 2031A lab1 Hello.java`

Problem 0 scanf, printf() in C (5 pts)

Specification

Write a C version of the above Hello program which, similar to the Java program, asks for a number and then outputs the double and triple of the number. Note that this C program does not ask for the name (we haven't learnt how to store strings.)
Name the program `hello.c` Since this may be the very first C program you write, if you have difficulties getting started, see the provided c files and instructions for other questions for hints.

Sample Inputs/Outputs:

```
red 331 % gcc hello.c
red 332 % a.out
Please enter a number: 23
Hi, input is 23. Double and triple of 23 is 46 and 69, respectively.
red 333 %
```

You may need **./a.out** if you work outside the lab environment. E.g., if you work locally on your MAC machine.

Submission (from lab):

In a lab terminal, navigate to the directory where your file is located, and then issue the command `red 310 % submit 2031A lab1 hello.c`

1 problem A scanf(), printf() in C

Specification

Download program `scanf2.c`, which takes two inputs from *standard input* (default: keyboard), and outputs the sum of the two numbers to *standard output* (default: screen). (This is the program shown in class and slides). Read the code and observe that,

- we specify what the input should "look like" in the formatting string of `scanf`. Here the formatting string `"%d %d"` indicates that `scanf` expects two integers separated by blank.
- in order to store the input values into variable `a` and `b`, we need to use `&a` and `&b` as additional arguments of `scanf`. We will explain later why `&` is needed.
- `printf` also uses `%d` for integer conversion. A conversion specification `%d` in the formatting string will be replaced/filled by a value of type `int`, which can be either an integer constant such as 32, or an `int` variable, or an arithmetic expression, or a function call which returns an `int`. Note that unlike in `scanf`, no `&` is used in front of variable names in `printf`.
- there is a `\n` at the end of the formatting string of the last `printf` statement, whereas there is no `\n` at the end of the formatting string of the first print statement.

Compile and run the program with different inputs. Try valid inputs such as

```
3 45
50 125
```

Also try some invalid inputs such as

```
3 4.5
4.7 3
4 A
```

and observe that variable `a` or `b` or both cannot get correct number from the user. Also try to delete the `\n` in the formatting string of the last `printf` statement, see what happens. Try to add `\n` to the first `printf` statement, see what happens.
Next, modify the program so that it takes as input two integers separated by <><><>.

Sample input and outputs

You may need **./a.out** if you work outside the lab environment.

```
red 306 % gcc scanf2.c
red 307 % a.out
Enter two integers separated by <><><>: 4<><><>32
Entered 4 and 32, Sum is 36.
red 308 % a.out
Enter two integers separated by <><><>: 40<><><>302
Entered 40 and 302, Sum is 342.
```

Try some invalid inputs such as

```
4<><>32
4 32
4<><><>32.56
43.2<><><>3
```

observe that `a` or `b` or both cannot get the correct number from the user.
Finally, remove the `&` before `a` or `b`, then compile and run the program again. The program compiles BUT when you run, it crashes, generating error message `Segmentation fault (core dumped)`. We will explain this later.

You don't need to submit anything for this question but doing this exercise gets you better prepared for the following questions.

2 Problem B. scanf, printf in C (10 pts)

Specification

Write an ANSI-C program that reads input from the *standard input* (keyboard), and then outputs the reformatted versions of the input to *standard output* (screen).

Implementation

- name your program `lab1B.c`
- use `scanf` to read input (from Standard Input), which are in the form of `Month Day Year` (i.e., three integers separated by white spaces).
- use `printf` to generate output in the form of `Year/month/day` and `Year-month-day`
- display the following prompt (leave a white space after the colon) `Enter month, day and year separated by spaces:`
- display output as shown in the sample output
- Note: you should do the reformatting only within `printf`. Specifically, you should use at most three variables, and feed them into `printf` judiciously. There should be only one `printf` statement for the reformatted output. (So the program has two printf statements)

Sample Inputs/Outputs:

```
red 306 % gcc lab1B.c
red 307 % a.out
Enter month, day and year separated by spaces: 3 20 2022
The input '3 20 2022' is reformatted as 2022/3/20 and 2022-3-20
red 308 % a.out
Enter month, day and year separated by spaces: 9 16 2021
The input '9 16 2021' is reformatted as 2021/9/16 and 2021-9-16
red 309 %
```

Submit your program by issuing **`submit 2031A lab1 lab1B.c`**

3 Problem C. Functions in C (10 pts)

Download the program `lab1C.c`, compile it using **`gcc lab1C.c`**
Observe that the compilation process fails (why?), and consequently, `a.out` is not generated.

Modify the program to make it compile. Note that you should not modify or move the existing code. That is, **do not modify the code of main() and sum(), and also do not move the functions.** Instead, add something to make the program compile.

Next, modify the function `greet()`, so that it prints Hello X! where X is the value of parameter i, as shown in the same output.

Sample Inputs/Outputs:

```
red 306 % gcc lab1C.c
red 307 % a.out
Hello 2031!
Hello 1012!
2.200000 + 3.300000 = 5.500000
```

Submit your program by issuing **`submit 2031A lab1 lab1C.c`**

4 Problem D. Functions, scanf(), printf(), floats (10 pts)

Specification

Improve program `lab1C`, so that it can read two float numbers for the Standard Input, separated by two pound (#) signs, and then output the sum of the two float numbers to standard output.

Implementation

- name your program `lab1D.c`
- use `scanf` to read inputs (from Standard Input), which are in the form of `float1##float2` (i.e., two float numbers separated by two pound signs).
- use `printf` to generate output. Note that by default `printf` displays six digits after decimal points of a floating point number.

Sample Inputs/Outputs:

Use -o to specify executable name, in place of a.out. you can specify other name

```
red 338 % gcc lab1D.c  -o lab1D
red 339 % lab1D
Hello 2031!
Hello 1012!
Enter two float numbers separated by ##: 2.35##5.64
2.350000 + 5.640000 = 7.990000
```

```
red 340 % lab1D
Hello 2031!
Hello 1012!
Enter two float numbers separated by ##: 1.2345##6.783
1.234500 + 6.783000 = 8.017500
```

Submit your program by issuing `submit 2031A lab1 lab1D.c`

5 Problem E. Simple loops in C (15 pts)

Specification

Extend program `lab1D.c` above, in such a way that it first prompts the user to enter an integer number, which indicates how many times the user wants to interact with the program. Then the program interacts with the user accordingly.

Implementation

- name your program `lab1E.c`
- use a loop (`for` or `while`) to interact (i.e., read input and generate output) `n` times, where `n` is entered by the user.
- display the two inputs with 3 decimal points, and the sum with the default format followed by 2 decimal points version

Sample Inputs/Outputs: (ONE blank line between each interaction/iteration):

```
red 338 % gcc lab1E.c  -o lab1E
red 339 % lab1E
Hello 2031!
Hello 1012!
Enter the number of interactions: 4

Enter two float numbers separated by ##: 2.35##5.64
2.350 + 5.640 = 7.990000 (7.99)

Enter two float numbers separated by ##: 1.1##2.2
1.100 + 2.200 = 3.300000 (3.30)

Enter two float numbers separated by ##: 1.2343##6.789
1.234 + 6.789 = 8.023300 (8.02)

Enter two float numbers separated by ##: 1.2345##6.78994
1.235 + 6.790 = 8.024440 (8.02)
red 340 %
```

Submit your program by issuing `submit 2031A lab1 lab1E.c`

6. Problem F0. getchar, putchar, input/output redirection

6.1 Specification

Download the provided program `countChar.c`, which uses function `getchar()` to read user input from *standard input* (default: keyboard) and counts the total number of characters in the input, and then outputs to *standard output* (default: screen). (This program is also in lecture slides.)

Play with the program and make sure you understand the program. In particular, observe a few things about `getchar`:

- `getchar()` reads characters from standard input (stdin), which by default is the keyboard. But standard input can be redirected (substituted) from an input file using `< filename.` In the latter case `getchar()` will read from the input file instead.
- `getchar()` returns **EOF** (which is a special negative integer number defined in C) when the "*end of file*" is reached.
  - If the program reads from a text file (redirected using `<`), then the end of the text file is "*end of file*";
  - If the program reads from default standard in (i.e., keyboard) , then in Unix, `ctrl D` indicates "*end of file*" (in Windows, it is `ctrl Z`)
- Instead of a `char`, function `getchar` returns an `int`. This will be explained in class.

6.2 Sample Inputs/Outputs (from Standard input - keyboard):

```
red 309 % gcc countChar.c
red 310 % a.out
hello
how are you
I am good
^D                    (press Ctrl and D)
# of chars: 28
red 311 % a.out
hello
how are you?
I am good and Thanks!
^D
# of chars: 41
red 312 %
```

6.3 Sample Inputs/Outputs (use redirected input/output files):

All the program run so far take inputs from standard in (stdin) which by default is the keyboard, and write output to standard out (stdout) which by default is the screen.

You can always redirect the Standard in (keyboard) from an input file using <
You can always redirect the Standard out (screen) to an output file using >

Download (**don't copy and paste**) file `greetings.txt`, whose content is

```
hello
how are you
I am good
```

```
red 313 % a.out < greetings.txt
# of chars: 28
```

This time the program does not ask you to enter anything, because standard input is redirected/substituted from a text file, so program reads inputs from file **`greetings.txt`**

```
red 314 % a.out > output.txt
hello
how are you
```

```
I am good
^D
red 315
```

This time the program reads inputs from user (keyboard), but nothing was generated on the standard out (screen), because all outputs are redirected to a text file using >
Now a new file `output.txt` should be generated (in the current directory). Use command `ls` or `ls -l` to confirm this. Then use command `cat` or `more` to view the content of `output.txt` (If you don't know what is happening here, you may want to review the CSE1020 Guided Tour or the Unix tutorial posted on the lab page on eClass.)

```
red 316 % ls -l
red 317 % cat output.txt
# of chars: 28
```

Finally, issue

```
red 318 % a.out < greetings.txt > output2.txt
red 319 %
```

This time both the standard input and standard out are redirected. Thus the program reads inputs from `greetings.txt`, and write outputs to `output2.txt`. Check the content of file `output2.txt`. Also in the terminal, issue `cal` or `date` to view the output, and then issue `cal > temp.txt` or `date > temp.txt` to see how the output is redirected to a file .

You don't need to submit anything for this question but doing this exercise gets you prepared for the next questions.

7. Problem F2 getchar, character comparison (15 pts)

Specification

The provided program uses `getchar()` to read input character by character, counting the number of characters from the standard input (keyboard or redirected from an input file). Modify the program so that it also counts the number of characters '`a`' in the input.

Implementation

- Name your program **`countChar2.c`**
- Hint: you might need to compare every character `getchar` reads in against the character 'a'. In Java or C, how to compare two characters?

Sample Inputs/Outputs

```
red 307 % gcc countChar2.c -o cc2
red 308 % cc2
hello
how are you?
I am good and Thanks!
^D
# of chars:  41
# of char 'a': 4
red 309 % cc2
hello
how are you
```

```
I am good
^D
# of chars:  28
# of char 'a': 2
red 310 % cc2 < greetings.txt
# of chars:  28
# of char 'a': 2
red 311 %
```

Submit your program by issuing **submit 2031A lab1 countChar2.c**

8. Prob F3 getchar, character comparison, special chars (15 pts)

Specification

Each input line to the above programs, either from keyboard or from the text file, ends with an invisible new line character. The new line characters are counted in the above programs. Modify the program `countChar` so that new line characters are not counted.

Implementation

- Name your program **`countChar3.c`**
- Hint: you might need to compare every character `getchar` reads in against the new line character. In Java or C, how is new line character represented?

Sample Inputs/Outputs

```
red 318 % gcc countChar3.c -o cc3
red 319 % cc3
hello
how are you?
I am good and Thanks!
^D
# of chars:  38
red 320 % cc3
hello
how are you
I am good
^D
# of chars:  25
red 320 % cc3 < greetings.txt
# of chars:  25
```

Submit your program by issuing **submit 2031A lab1 countChar3.c**

9. Prob F4 getchar, character comparison, special chars (15 pts)

Modify the program **`countChar3.c`** so that it also counts the number of lines and number of blank (space) in the input.
Name your program **`countChar4.c`**

```
red 308 % gcc countChar4.c -o cc4
red 309 % cc4
hello
how are you
I am good
^D
# of chars:  25 (# of blanks:  4)
# of lines:  3
red 319 % cc4
hello
how are you?
I am good and Thanks!
^D
# of chars:  38 (# of blanks: 6)
# of lines: 3
red 310 % cc4 < greetings.txt
# of chars:  25 (# of blanks: 4)
# of lines:  3
red 311 % cc4 < greetings.txt > output4.txt
red 312 % cat output4.txt
# of chars:  25 (# of blanks: 4)
# of lines:  3
```

Submit your program by issuing `submit 2031A lab1 countChar4.c`

Make sure your program compiles in the lab environment. A program that does not compile, or, crashes with "segmentation fault" in the lab will get 0.

All submissions need to be done from the lab, using command line.

In summary, for this lab you should submit the following files:

`Hello.java` `hello.c` `lab1B.c` `lab1C.c` `lab1D.c` `lab1E.c`

`countChar2.c` `countChar3.c` `countChar4.c`

From any directory, you can issue `submit -l 2031A lab1` to get a list of files that you have submitted for lab1.

lower case L

Also note that you can submit the same file multiple times. Then the latest file will overwrite the old one.

Common Notes

All submitted files should contain the following header:

```
/***************************************
* 22S - Lab01 *
* Author: Last name, first name *
* Email: Your email address *
* eecs_username: Your eecs login user name *
* York Student #: Your student number
****************************************/
```