Introduction to C Programming in a Unix (Linux 64 bits) Environment

Lab goals: C primer, on simple stream IO library functions and parsing command-line arguments.

(This lab is to be done SOLO)

Task 0: Maintaining a project using `make`

You should perform this task **before** attending the lab session. For this task, 3 files are provided: adds.s, main.c, numbers.c. The first file is assembly language code, and the other 2 are C source code.

1. Log in to Linux.
2. Decide on an ASCII text editor of your choice (vi, emacs, kate, pico, nano, femto, or whatever). It is **your responsibility** to know how to operate the text editor well enough for all tasks in all labs.
3. Using the text editor that you decided to use, write a makefile for the given files (as explained in the introduction to GNU Make Manual, see the Reading Material). The Makefile should provide targets for compiling the program and cleaning up the working directory.
4. Compile the project by executing make in the console.
5. Read all of lab1 reading material before attending the lab.
6. Read puts(3) and printf(3) manuals. What is the difference between the functions?

Important

To protect your files from being viewed or copied by other people, thereby possibly earning you a disciplinary hearing, employ the Linux permission system by running:

```
chmod 700 ~ -R
```

In order to make sure you have sufficient space in your workspace, run the following command once you're logged in

```
du -a | sort -n
```

Then you can see a list of your files/directories and the amount of space each file/directory takes.

If you need space and KNOW which files to remove, you can do that by:

```
rm -f [filename]
```

Control+D, Control+C and Control+Z

- What does Control+D (^D) do?
  Control+D causes the Unix terminal driver to signal the EOF condition to the process running in this terminal foreground. You can read more about it here.
- What does Control+C (^C) do?
  Pressing Control+C in the terminal, causes the Unix terminal to send the SIGINT signal to the process running in the terminal foreground. This will usually terminate the process.
- What does Control+Z (^Z) do?
  Pressing Control+Z in the terminal, causes the Unix terminal to send the SIGTSTP signal to the process running in the terminal foreground. This will suspend the process (meaning the process will still live in background).

  Do not use Control+Z for terminating processes!!!

Task 1: The wordCounter program

In this task we will implement a program similar to “wc” program in linux.
As stated in task 0 and the reading material, you should already have consulted the man pages for **strcmp(3), fgetc(3), fputc(3), fopen(3), fclose(3)** before the lab.
Task 1 consists of three subtasks: 1a and 1b with each subsequent task building upon the previous one.

Task 1a: A restricted wc version

The wordCounter program should be implemented as follows:

NAME
    wordCounter - count and print the number of word in a sentence
SYNOPSIS
    wordCounter
DESCRIPTION
    wordCounter receives text characters from standard input and prints the number of words to the standard output
EXAMPLES

```
#> wordCounter
Hi, My     name is Noam.
^D
5
#>
```

Information

- stdin and stdout are FILE* than can be used with fgetc and fputc.
- Make sure you know how to recognize end of file (*EOF*).
- Control-D causes the Unix terminal driver to signal the EOF condition to the process running in this terminal foreground, using this key combination (shown in the above example as ^D) will cause *fgetc* to return an *EOF* and in response your program should terminate itself "normally".

- Important - make no assumption about the length of the line.
- Important- all character up to 20 hexa (in ascii table) are regarded as spaces.

- Important- Multiple successive spaces count as a single delimiter (so `"hello\n\r\n \tworld"` counts as just 2 words).

Task 1b: An extended wc version

NAME

wordCounter - count and print the number of word/characters in a sentence or number of characters in the longest word.

SYNOPSIS

wordCounter[OPTION]…

DESCRIPTION

wordCounter receives text characters from standard input and prints the number of words/characters or number of characters in the longest word to the standard output, according to the options specified by the user.

OPTIONS

-w

number of words. (If there is no option this is the default option)

-c

number of characters.

-l

number of characters in the longest word

EXAMPLES

```
#> wordCounter -w
Hi, My name is Noam.
^D
5
#>wordCounter -c
Hi, My name
^D
9
#>wordCounter -l
Hi, My name
^D
4
#>wordCounter -c -l
Hi, My name
^D
9
4
```

Mandatory requirements

- If there is no option, number of words is the default option
- Space is NOT computes as characters, but all other signs are characters like

(, >< @)

- **Read your program parameters in the manner of task0 main.c, first set default values to the variables holding the program configuration and then scan through *argv* to update those values. Points will be reduced for failing to do so.**

Task 2: Supporting input from a file

NAME

wordCounter - count and print the number of word/characters in a sentence or number of characters in the longest word or number of lines.

SYNOPSIS

wordCounter[OPTION]…

DESCRIPTION

wordCounter receives text characters from standard input or from a file and prints the number of words/characters or number of characters in the longest word or number of lines in the file to the standard output, by options.

OPTIONS

-w

number of words. (If there is no option this is the default option)

-c

number of characters.

-l

number of characters in the longest word

-n

number of lines in file OR in stdin.

-i FILE

Input file. Read list of characters to be encoded from a file, instead of from standard input.

EXAMPLES

```
#>cat input.txt
My name
Is Noam
I study CS
#> wordCounter -w -i input.txt
7
#> wordCounter -n -i input.txt
3
#>
```

Mandatory requirements

- Program arguments may arrive in an **arbitrary** order. Your program must support this feature.

Deliverables:

Task 1 must be completed during the regular lab. Task 2 may be done in a completion lab, but only if you run out of time during the regular lab. The deliverables must be submitted until the end of the lab.
You must submit source files for task1b and task2 in respective folders, and also a makefile that compiles them. The source files must be organized in the following tree structure (where '+' represents a folder and '-' represents a file):

+ task1b
  - makefile
  - wordCounter.c
+ task2
  - makefile
  - wordCounter.c

Submission instructions

- Create a zip file with the relevant files (only).
- Upload zip file to the submission system.
- Download the zip file from the submission system and extract its content to an empty folder.
- Compile and test the code to make sure that it still works.