Introduction to C Programming in a Unix (Linux 32 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: add.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 theReading 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 ~
```

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!!!

Writing a simple program

Write a simple echo program named my_echo:

NAME
    my_echo - echoes text.
SYNOPSIS
    my_echo
DESCRIPTION
    my_echo prints out the text given in the command line by the user.
EXAMPLES

```
#> my_echo aa b c
aa b c
```

Mandatory requirements

- Create a proper makefile as described in the reading material.
- Test your program to see that it actually works…

The tasks below are to be done only during the lab session!
Students coming with ready code "from home" will be assigned low priority **and** will have to demonstrate re-writing all the code again **from scratch**. Additionally, you are expected (of course) to understand your code completely.

Task 1: The encoder program

In this task we will write a program that encodes specific characters from the input text.
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, 1b, and 1c, each building on top of the previous subtask.

Task 1a: A restricted encoder version - conversion to lowercase

The encoder program should be implemented as follows:

NAME
encoder - encodes the input text as lowercase letters
SYNOPSIS
encoder
DESCRIPTION
encoder receives text characters from standard input and prints the corresponding lowercase characters to the standard output. Non uppercase letters remain unchanged.
EXAMPLES

```
#> encoder
Hi, my name is Noah
hi, my name is noah
^D
#>
```

Information

- stdin and stdout are FILE* constants 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* constant and in response your program should terminate itself "normally".
- Refer to ASCII table for more information on how to convert characters to lower-case or upper-case.

Mandatory requirements

- You must read the input **character by character**, there is no need to store the data read at all.
- Important - you cannot make any assumption about the line length.
- Check whether a character is an uppercase letter by using a single "if" statement with two conditions. How?
- You are **not** allowed to use any library function for the purpose of recognizing whether a character is a letter, and its case.

Task 1b: Extending the encoder to support encryption

NAME

encoder - encodes the input text as lowercase encrypted letters.

SYNOPSIS

encoder [OPTION]…

DESCRIPTION

encoder receives text characters from standard input and prints the corresponding lowercase encrypted characters to the standard output. The encryption key is given as an argument.
If **no** argument is supplied, the encoder only converts characters to lowercase. Characters that are not English letters must remain unchanged (they should not be converted to lowercase nor encrypted).

Encryption key is given with a '+' or a '-' sign (see example below).

Note that you may assume that the encryption key is made of a single digit.

EXAMPLES

```
#> encoder +1
Hi, my name is Noah
ij, nz obnf jt opbi
^D
#> encoder -1
Hi, my name is Noah
gh, lx mzld hr mnzg
^D
```

Mandatory requirements

- You are **not** allowed to use any library function for the purpose of recognizing whether a character is a letter and its case.

- **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 1c: Supporting input from a file

NAME
encoder - encoders the input text as lowercase encrypted letters.
SYNOPSIS
encoder [OPTION]…
DESCRIPTION
encoder receives text characters from standard input or from a file and prints the corresponding lowercase encrypted characters to the standard output. The encryption key is given as an argument.
If **no** encryption key argument is supplied, the encoder only converts characters to lowercase.
Of course, characters that are not English letters must still be supported.
OPTIONS
-i FILE
Input file. Read list of characters to be encoded from a file, instead of from standard input.

ERRORS
If FILE cannot be opened for reading, print an error message to standard error and exit.
EXAMPLES

```
#> echo 'Hi, my name is Noah' > input
#> encoder +1 -i input
ij, nz obnf jt opbi
^D
```

Task 2: Supporting output to a file

NAME
encoder - encodes the input text as lowercase encrypted letters.
SYNOPSIS
encoder [OPTION]…
DESCRIPTION
encoder receives text characters from standard input or from a file and prints the corresponding lowercase encrypted characters to the standard output or the given files. The encryption key is given as an argument.
If **no** encryption key argument is supplied, the encoder only converts characters to lowercase.
Of course, characters that are not English letters must still be supported.
OPTIONS

-i FILE
Input file. Read list of characters to be encoded from a file, instead of from standard input.
-o
Output file. Prints output to a file instead of the standard output.

EXAMPLES

```
#> encoder +1 -o
Enter output file:
output
Enter text to encode:
Hi, my name is Noah
^D
#> more output
ij, nz obnf jt opbi
```

- You may assume the output file name is no longer than 30 characters.

Mandatory requirements

- Program arguments may arrive in an **arbitrary** order. Your program must support this feature.
- You must use *fgets* to read the output file name.

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 day.
You must submit source files for task1C 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):

+ task1C
  - makefile
  - encoder.c
+ task2
  - makefile
  - encoder.c