Introduction to C Programming in Unix 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 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

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 filter program

In this task we will write a program that filters 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 filter version - filtering preset characters

The filter program should be implemented, as follows:

NAME
filter - filters out the characters 'h' and 'H' from the input text
SYNOPSIS
filter
DESCRIPTION
filter receives text characters from standard input and prints the characters that passed the filter to the standard output.
EXAMPLES

```
#> filter
Hi, my name is Noah
i, my name is Noa
```

```
^D
#>
```

Mandatory requirements

- In Task 1a (and only in this task), you must use a **switch** to check if a char should be filtered. Click here for more info.
- You must read the input **character by character**, there is no need to store the data read at all.
- 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.
- Important - you cannot make any assumption about the line length.

Task 1b: Extending the filter to support different filter characters

NAME
  filter - filters given characters from the input text.
SYNOPSIS
  filter [OPTION]...
DESCRIPTION
  filter gets text characters from the standard input and prints those that pass the filter to standard output. The characters to filter are given as an argument.
  filter operates in a case-insensitive manner (i.e., lower-case and upper-case letters, like 'a' and 'A', are considered the same).
  If **no** characters are given as argument, the filter operates on the letters 'h' and 'H'.
  Of course, characters that are not English letters must still be supported.

EXAMPLES

```
#> filter n,
Hi, my name is Noah
Hi my ame is oah
^D
```

Note that 'n','N', and ',' were filtered.

Mandatory requirements

- Refer to ASCII table for more information on how to convert characters to lower-case or upper-case. Check whether a character is a lower-case or upper-case letter by using a single "if" command 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.
- 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.

Task 1c: Supporting input from a file

NAME
  filter - filters given characters from given text.
SYNOPSIS

filter [OPTION]…

DESCRIPTION

filter gets text characters from the standard input and prints those that pass the filter to standard output. The characters to filter are given as an argument or from a file.
filter operates in a case-insensitive manner (i.e., lower-case and upper-case letters, like 'a' and 'A', are considered the same).
If **no** file is given as argument, the filter operates on the letters 'h' and 'H'.
Of course, characters that are not English letters must still be supported.

OPTIONS

-i FILE
Input file. Read list of characters to be filtered from a file, instead of an argument.

ERRORS

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

EXAMPLES

```
#> echo n,! > input
#> filter -i input
Hi, my name is Noah
Hi my ame is oah
^D
```

- The text file will contain no more than 30 characters

Task 2: Supporting output to a file

NAME

filter - filters given characters from given text.

SYNOPSIS

filter [OPTION]…

DESCRIPTION

filter gets text characters from the standard input and prints those that pass the filter to standard output or to a file. The characters to filter are given as an argument or from a file.
filter operates in a case-insensitive manner (i.e., lower-case and upper-case letters, like 'a' and 'A', are considered the same).
If **no** file is given as argument, the filter operates on the letters 'h' and 'H'.
Of course, characters that are not English letters must still be supported.

OPTIONS

-i FILE
Input file. Read characters to filter from a file, instead of an argument.

-o
Output file. Prints output to a file instead of to standard output.

EXAMPLES

```
#> filter -o n
Enter output file:
output
Enter text to filter:
Hi, my name is Noah
^D
```

```
#> more output
Hi, my ame is oah
```

Mandatory requirements

- Program arguments may arrive in an **arbitrary** order. Your program must support this feature.
- You may assume the output file name is no longer than 30 characters.
- You must use *gets* 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 and also a makefile that compiles them. The source files must be named task1c.c and task2.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.