ESPLab B 2022

Lab 1

Introduction to C Programming in a Unix (Linux 32 bits) Environment

- C primer
- Parsing command-line arguments
- Undestanding character (ASCII) and hexadecimal encodings
- Implementing a debug mode for your program
- Introduction to standard streams (stdin, stdout, stderr)
- Simple stream IO library functions

(This lab is to be done SOLO)

Task 0 (to be done before the lab session!): Maintaining a project using `make`

You should perform this task **before** attending the lab session. For this task, 3 files are provided in: https://moodle2.bgu.ac.il/moodle/mod/folder/view.php?id=2094949&forceview=1 The first file is assembly language code, and the other 2 are C source code.

1. Make sure you have a working Linux environment. You are provided with an image file that you can find on the announcements page. Run the image with VirtualBox and make sure it works.
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 thehttps://moodle2.bgu.ac.il/moodle/mod/page/view.php?id=2104189&forceview=1. 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, and make sure you **understand** it. This may entail writing simple code to exercise the use of library functions and running it, in addition to any other task 0 code you are **required** to write.
6. Read the puts(3) and printf(3) manuals. What is the difference between the functions? To read the manuals type man followed by the function name (e.g. `man puts`) in a "console".

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!!!

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...

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.

On this lab you can assume that the resulting encrypted char will always be in the range from 32 to 126 (inclusive).

(from here onwards, to be done during the lab session)

Task 1: The encoder program

In this task we will write a program that encodes characters from the input text.

As stated in task 0 and the reading material, you should already have consulted the man pages for **strncmp(3), fgetc(3), fputc(3), fopen(3), fclose(3)** before the lab.

Task 1 consists of three subtasks: 1a, 1b, 1c and 1d, each building on top of the previous subtask. Therefore, your program for each task should contain all the features from the previous tasks.

Task 1a: A restricted encoder version

The encoder program should be implemented as follows:

NAME

encoder - encodes the input text.

SYNOPSIS

encoder

DESCRIPTION

encoder reads the characters from standard input and prints it except the uppercase (capital letters) that it replaced with the character '.'.

EXAMPLES

```
#>
encoder
Hi, my name is Noah
.i, my name is .oah
 ^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 upper-case or lower-case.

Mandatory requirements

- You must read and process the input **character by character**, there is no need to store the characters you read at all.
- Important - you cannot make any assumption about the line length.
- Check whether a character is a 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 debug mode

As you develop a program, it is important to allow for easy debugging.

The debug mode which you introduce here explains this idea. Using this scheme, any program can be run in a debug mode that allows special debugging features for testing the program. As a minimum, implemented here, when in debug mode the program prints out important information to stderr. Printing out the command-line parameters allows for easy detection of errors in retrieving them. Henceforth, code you write in most labs will also require adding a debug mode, and it is a good idea to have this option in all programs you write, even if not required to do so!

NAME

encoder - encodes the input text.

SYNOPSIS

encoder [OPTION]...

DESCRIPTION

encoder receives text characters from standard input and prints it after encoding uppercase letters into '.' to the standard output.

The debug mode is activated via command-line argument (-D).

If the debug-mode is activated, print the command-line arguments to stderr and each character you receive from the input (decimal value) before and after the conversion. and print the number of letters you converted

regardless of the debug-mode, the encoder will convert characters into '.'.

EXAMPLES

```
#>
 encoder -D
Hi, my name is Noah
72 46
105 105
44 44
32 32
109 109
121 121
32 32
110 110
97 97
109 109
101 101
32 32
105 105
115 115
32 32
78 46
111 111
97 97
104 104
the number of letters: 2
.i, my name is .oah
^D
#>
```

Note: the left colomn is the decimal representation of the **input** characters whereas the right colomn is the decimal representation of the **modified** characters (in this case switched from upper-case to lower-case)

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 https://moodle.bgu.ac.il/moodle/pluginfile.php/3444254/mod_page/content/14/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: Extending the encoder to support encryption

In this task, make sure you follow the output format precisely. Programs which deviate from the instructions will not be accepted! Please make your output is exactly as the examples below.

NAME

encoder - encodes the input text using encryption key.

SYNOPSIS

encoder [OPTION]...

DESCRIPTION

encoder receives characters from standard input and prints the corresponding encrypted characters to the standard output. The encryption key is given as a command-line argument.

If no argument is supplied, the encoder converts uppercase characters into '.' as before.

The encryption key is of the following structure: +e{key}. The argument {key} stands for a digit in hexadecimal whose value will be the number of characters to add to the end of the input such that it repeats the first character.

You should support both addition and substruction, +e{key} is for addition and -e{key} is for substruction. On this task, you need to ignore the char '\n' which means new line after you press ENTER.

EXAMPLES

```
#>
encoder +e1
ABCDEF
ABCDEFA
ABCDEFAA
ABCDEFAAA
^D
#>
encoder +eA
ACRZ
ACRZAAAAAAAAAA
^D
#>
encoder -e5
hEllo56
56
-NONE-
^D
```

Mandatory requirements

You should convert the key character into a number to be added (or subtracted) using straightforward code without any macros or library functions. If the key is outside the permissible range, print an error message and exit the program.

Tips for thought

Note that the encoding number is only a single hexadecimal character in the range 0 to 9 or A to F, inclusive. It is therefore easier than a full multiple character number. Also, think about how to get its value from the argv, it is simple, no "string" functions are needed! If your code to compute the key value is more than 10 lines of C code, then you have probably done something the hard way, or wrong!

Task 1d: Supporting input from a file

NAME

encoder - encoders the input text as lowercase or encrypted letters.

SYNOPSIS

encoder [OPTION]...

DESCRIPTION

encoder reads characters from standard input and prints the corresponding lowercase characters or encrypted characters (depending on whether the encryption key was given) to the standard output.

If no encryption key argument is supplied, the encoder only converts upper-case characters into '.'.

OPTIONS

-iFILE

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 'hEllo56' > input
#> encoder -e5 -iinput
56
```

- Notice that there is no separation between the -i indicator and the file name (same as in the encryption key).

Task 2: Supporting output to a file

NAME

encoder - encodes the input text as lowercase or encrypted letters.

SYNOPSIS

encoder [OPTION]...

DESCRIPTION

encoder reads ASCII text characters from standard input or from a file and prints the corresponding lowercase characters or encrypted characters (depending on whether the encryption key was given) to the standard output or the given file. The encryption key is given as an argument.

If no encryption key argument is supplied, the encoder only converts upper-case characters into '.'.

OPTIONS

-oFILENAME

Output file. Prints output to a file named FILENAME instead of the standard output.

EXAMPLES

```
#> encoder -e5 -ooutput hEllo56
^D
#> more output
56
```

- Notice that there is no separation between the -o indicator and the file name (same as in the encryption key and the input file name).

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 day.

You must submit source files for task1D 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):

+ task1D
  - makefile
  - encoder.c
+ task2
  - makefile
  - encoder.c

Submission instructions

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

Last modified: Monday, 28 March 2022, 11:19 AM

Administration

› Course administration

◄ Lecture 7 recording

Jump to...

Files for Task 0 ►