Introduction to C Programming in Unix Environment

Objectives

- Learn how to use Linux shell, together with some basic commands.
- Learn how to write basic programs in C.
- Number representation, and bit-wise operators.

Task 0:

You should read the Reading Material and accomplish this task **before** attending the lab session.

Using shell commands, do the following:

- Login into Linux.
- Open a shell (Konsole/terminal).
- Create a directory called espl under your home directory (`mkdir`).
- Go to the espl directory (using `cd`), and create a directory called lab1.

Maintaining a project using `make`

For this task, 3 files are provided: add.s, main.c, numbers.c.

1. Start up a 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.
2. Using the above editor of your choice, write a Makefile for the given files (as explained in the introduction to GNU Make Manual, see the Reading Material and Lecture 1). The Makefile should provide targets for compiling the program and cleaning up the working directory.
3. Compile the project by executing make in the console
4. Read all of lab 1 reading material before attending the lab.

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
```

Task 1: ASCII code printer

Each character in ASCII has an integer representation, see ASCII table for more information. For example, the ASCII code of the 's' character is 115, which is 1110011 in binary representation.

Mandatory requirements

- Create a separate file for each task.
- Your program must not have code repetition in it.
- Split your code into functions. Don't write everything in the main.
- Create a makefile to compile your code, required in all tasks, even if it's not mentioned.
- Backward compatibility - Each task must support the previous tasks unless otherwise mentioned.
- Use unsigned char to save each character you read (why?).
- Read command-line arguments in a loop as done in main.c of task0.
- Recommended: Converting the numbers to their binary representation must be done using bit-wise operations. You can read about bit-wise operations at this tutorial.
- Dividing by 2 is like doing >>.

Task 1a: Decimal representation

Write a program that receives characters from the standard input and prints their ASCII codes in decimal (space separated). The program runs in a loop until EOF is received.

NAME

char2ascii - text to ascii code.

SYNOPSIS

char2ascii

DESCRIPTION

char2ascii receives characters from the standard input , until EOF is received, and prints their ascii codes to the standard output.

EXAMPLES

```
$>./char2ascii
s
115
ab
97 98
```

Task 1b: Binary representation

Extend your previous program to print the ASCII code in binary if a -b flag is received, from the most significant bit to the least significant bit. Write a function that receives a char and an array of 8 cells, and returns the binary representation of the char in the array. The program must run like task 1a if not given any flags.

NAME

char2ascii - text to ascii code.

SYNOPSIS

char2ascii -b

DESCRIPTION

char2ascii receives characters from the standard input , until EOF is received, and prints their ascii codes in binary to the standard output.

-b receives characters from the standard input , until EOF is received, and prints their ascii codes in binary from most significant bit to the least significant.

EXAMPLES

```
$>./char2ascii -b
s
01110011
ab
01100001 01100010
```

Task 1c: Upper to lower case, lower to upper case

Extend your previous program and add the following feature if a -c flag is received: the program still receives characters from the standard input but now prints upper case letters if the input is lower case and vice versa, lower case letter if the input is upper case.

NAME

char2ascii - text to ascii code.

SYNOPSIS

char2ascii

DESCRIPTION

char2ascii receives characters from the standard input , until EOF is received, and prints their ascii codes in binary to the standard output.

-b receives characters from the standard input , until EOF is received, and prints their ascii codes in binary from least significant bit to the most significant.

-c receives characters from the standard input , until EOF is received, and prints upper case letters if the input is lower case and vice versa.

EXAMPLES

```
$>./char2ascii -c
s
S
aB
Ab
```

Task 2: Supporting output and input file

Extend the program from task1 to write the output to a file instead of stdout if the -o flag is given. Extend the program from task1 to read the input from a file instead of stdin if the -i flag is given.

NAME

char2ascii - text to ascii code.

SYNOPSIS

char2ascii [option] [option]

DESCRIPTION

char2ascii receives characters from the standard input , until EOF is received, and prints their ascii codes.

OPTIONS

-b receives characters from the standard input , until EOF is received, and prints their ascii codes in binary from least significant bit to the most significant.

-c receives characters from the standard input , until EOF is received, and prints upper case letters if the input is lower case and vice versa.

-i read the input from the given file instead of stdin.

-o print the output to the given file instead of stdout.

EXAMPLES

```
$>./char2ascii -b -o out
sa
$>cat out
01110011 01100001

$>./echo "Lab1" > in /*create the "in" input file*/
$>./char2ascii -c -i in
lab1

$>./echo "s" > in1
$>./char2ascii -i in1 -o out1
$>cat out1
115

$>./char2ascii -b
s
01110011
```

Submission

- SUBMIT a ZIP file at the end of the lab session to the submission system .
- Your zip file should include the following files: task1a.c, task1b.c, task1c.c, task2.c, makefile (which contains targets for creating each of the tasks and subtasks).

Good luck!