COP290
Semester 2 2022-23
Lab 1

Part 1: Typing

Why? The faster you type, the faster you code. We do not want to spend any conscious energy on typing. All your conscious energy can be spent on thinking about the code that you are writing.

Learn touch typing from: https://www.typingstudy.com

Do one lesson everyday. With regular practice, you shall be able to come to 60 to 70 words per minute. You may regularly measure your typing speed here: https://www.typingstudy.com/speedtest

Part 2: Editing

Why?

- You will need to code in a variety of environments such as by logging into a remote server or into an embedded device. GUI-based editors may not be available there.
- Touching the trackpad/mouse and clicking buttons is very slow.

Learn vim. (If you already know some Emacs, then you can just sharpen your Emacs skill instead).

Basic: run vimtutor in a Linux terminal.
Advanced (optional):

- You should try learning "." and writing basic macros.
- https://github.com/iggredible/Learn-Vim

Part 3: Shell scripting

Why? You can save a lot of time writing one-off programs if you are comfortable with basic shell scripting.

Note: It's best to work in a Linux environment with bash. Same commands in other shells like zsh (default on Mac) may have different semantics. You can know your shell type by running:

```
echo "$SHELL"
```

You can also force Mac to use bash by running your script like `bash ./script.sh` instead of like `./script.sh`

We will use a linux machine for all the evaluations. Thus we recommend testing your scripts on a linux based machine at least once. If you do not have a Linux machine, you may request a virtual machine from https://baadal.iitd.ac.in/user/request_vm. Once allotted, you can access it using ssh.

Commands to learn: ls, cd, rm, mv, cp, touch, top, pwd, mkdir, ln, cat, echo, grep, awk, sed, parallel, cut, sort, uniq, wc, head, less, xargs, ping, nmap, ssh, find, tree

You can run `man <command>` to learn about a command.

Some resources:

1. http://linuxcommand.org/lc3_learning_the_shell.php
2. https://ryanstutorials.net/bash-scripting-tutorial/
3. https://www.geeksforgeeks.org/bash-script-arithmetic-operators/

Some practice problems:

Question 1 (Count Files)

Count the number of text files starting with a given prefix within a given directory.

Write a script **count_files.sh** that takes 2 arguments

1. Path of the directory: This is where you will search for the files
2. Prefix: Only files whose names begin with the specified prefix shall be counted
3. Recursive (true or false): This parameter will only take values "true" or "false". If recursive is set to true then you also need to search for the subdirectories in the path mentioned. Otherwise you will only report the files present in the root of the directory mentioned

Note: Script has to give the output to stdout. If the path is invalid (not a directory) or passed parameters are not valid then your script should output relevant error messages and exit with error code -1.

Examples:

Directory structure of /Users/vishal
. ├── 1.txt ├── 2.txt

```
├── abc1.txt
├── abc2.txt
└── subdir
    ├── 4.txt
    └── abc3.txt
```

```
bash ./count_files.sh . abc true
3
```

Recursive search is enabled thus sub directory is included. For the directory path I have passed "." which means current directory. Files counted are abc1.txt, abc2.txt and abc3.txt.

```
bash ./count_files.sh /Users/vishal abc true
3
```

Recursive search is enabled thus sub directory is included. Files counted are abc1.txt, abc2.txt and abc3.txt

```
bash ./count_files.sh /Users/vishal abc false
2
```

Recursive search is disabled thus sub directory is excluded. Files counted are abc1.txt and abc2.txt/

```
bash ./count_files.sh /Users/vishal "" false
6
```

Prefix passed is empty thus the script will count all the txt files.

```
bash ./count_files.sh /Users/vishal/popo "" false (TODO Check)
/Users/vishal/popo is not a directory
```

and exit with error code -1

```
bash ./count_files.sh . "" true2
recursive should take only true or false values
```

and exit with error code -1

Question 2 (Arithmetic Operations)

Write a script **evaluation.sh** that takes input file as an argument and gives final value after all the arithmetic operations.

You only need to handle +, -, *, / and %. All of these are integer operations. Look at the following examples for better understanding.

Note: Script has to give the output to stdout. If the path is invalid (not a file or invalid file) or passed parameters are not valid then your script should output relevant error messages and exit with error code -1.

Example 1

input.txt

```
10 +
20 -
2 *
4 /
10 +
2 %
```

```
bash evaluation.sh input.txt
1
```

```
0 + 10 = 10
10 - 20 = -10
-10 * 2 = -20
-20 / 4 = -5
-5 + 10 = 5
5 % 2 = 1
```

Example 2

input.txt

```
10 +
20 -
2 *
6 /
10 +
```

```
bash ./evaluation.sh input.txt
7
```

```
0 + 10 = 10
10 - 20 = -10
-10 * 2 = -20
-20 / 6 = -3 (Integer Division)
-3 + 10 = 7
```

Part 4: make

Why? You can automate any step by step process such as compiling your project by using make.

Resource: https://makefiletutorial.com/

All the required files are in starter code. Check the submission instructions for the link.

Given project directory structure

```
.
├── dependencies
│   └── header.h
├── input.txt
├── main.c
├── makefile (1) -> Work on this file
├── src1
│   ├── makefile (2) -> Work on this file
│   └── square.c
└── src2
    ├── add.c
    └── makefile (3) -> Work on this file
```

makefile in the root directory should have following targets

1. pre-build
    a. Should create obj and exe directories if not already present. Otherwise do nothing.
2. all
    a. Should call make pre-build
    b. Call make files of src1 and src2 recursively. Make files of src1 and src2 should create object files in the obj directory.
    c. Make the executable. Final executable *myApp* should be created in the exe directory.
3. run
    a. Should run the executable created.
4. clean
    a. Calls make clean recursively for src1 and src2 (Will make the obj directory empty)
    b. Then delete exe and obj directories.

makefiles in src1 and src2 will have following target

1. default
    a. Create the object file for .c file
2. clean
    a. Delete the object file or any other temporary files created.

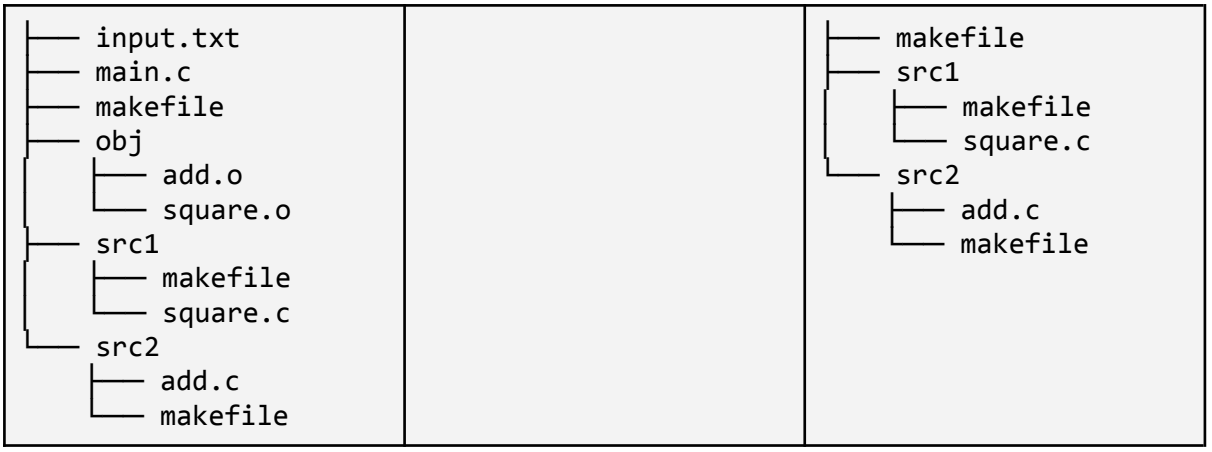
Examples

Before	Command	After
. ├── dependencies │ └── header.h ├── input.txt ├── main.c ├── makefile ├── src1 │ ├── makefile │ └── square.c └── src2 ├── add.c └── makefile	make pre-build	. ├── dependencies │ └── header.h ├── exe ├── input.txt ├── main.c ├── makefile ├── obj ├── src1 │ ├── makefile │ └── square.c └── src2 ├── add.c └── makefile

Before	Command	After
. ├── dependencies │ └── header.h ├── input.txt ├── main.c ├── makefile ├── src1 │ ├── makefile │ └── square.c └── src2 ├── add.c └── makefile	make all	. ├── dependencies │ └── header.h ├── exe │ └── myApp ├── input.txt ├── main.c ├── makefile ├── obj │ ├── add.o │ └── square.o ├── src1 │ ├── makefile │ └── square.c └── src2 ├── add.c └── makefile

```
$ make run
./exe/myApp
hello world
```

Before	Command	After
. ├── dependencies │ └── header.h ├── exe │ └── myApp	make clean	. ├── dependencies │ └── header.h ├── input.txt ├── main.c



Subdirectories src1 and src2 should also have proper make files

```
$ make clean
$ make pre-build
$ cd src1
$ make
$ ls ../obj
square.o
$ cd ../src2
$ make
$ ls ../obj
add.o    square.o
$ make clean
$ ls ../obj
square.o
```

Submission Instructions

1. Download the starter code from here and unzip the file
2. Modify the required files
    a. part3/q1/count_files.sh
    b. part3/q2/evaluation.sh
    c. part4/makefile
    d. part4/src1/makefile
    e. part4/src2/makefile
3. Change the directory name to <entry_no> and rezip the directory
4. Submit <entry_no>.zip

Zip file <entry_no>.zip on unzipping should give a folder which will contain all the relevant files.

```
(base) ➜  testing unzip 2018CS50426.zip
(base) ➜  testing tree
.
├── 2018CS50426
│   ├── part3
│   │   ├──
```

```
|   ...
└── 2018CS50426.zip

10 directories, 17 files
(base) ➜  testing ls
2018CS50426     2018CS50426.zip
```