

CMPT 300 D100

Assignment 1

Makefile tutorial

https://www.cs.swarthmore.edu/~newhall/unixhelp/howto_makefiles.html#creating

Notes:

1. **Failure to follow the instructions and rules, may lead to failed test cases and/or a final grade of 0.**
2. You can do this assignment individually or in a team of two. If you are doing it in a group, only one submission per group is required.
3. You may submit multiple times until the deadline. Grade penalties will be imposed for late submissions (see the course outline for the details).
4. Always plan before coding.
5. All the codes in this lab must be done using C language only. No other languages should be used.
6. Make sure that there should be **no infinite loop** in any of the programs.
7. Use function-level and inline comments throughout your code. We will not be specifically grading documentation. However, remember that we cannot comment on your code unless sufficiently documented. Take the time to document your code as you develop it properly.
8. We will carefully analyze the code submitted to look for plagiarism signs, so please do not do it! Please talk to an instructor or a TA if you are unsure what is allowed.

Coding Rules

- You have to follow the file name as specified in the instructions.
- **Makefile:** the Makefile provides the following functionality:
 - **All:** Compile your program (this is the default behaviour), producing an executable file named the same as the C file.
 - **Clean:** deletes the executable file and any intermediate files (.o, specifically)
- Create **one Makefile** for all three problems.
 - Using your text editor, create a Makefile (it must be called Makefile, with no extension). A Makefile defines how the code will be built (compiled). When creating a Makefile, be aware that it is very sensitive to tabs and spaces.
 - You must name the output file the same as the filename (without .c). For example:

```
gcc -g -Wall -o my_userid my_userid.c
```
- You will **receive 0** if your Makefile fails.
 - Check your build to ensure that there are no errors.
 - Visit TA's programming office hours to get help.

Question 1. Write a program to display the userID of the given username.

Each user has a unique userID to identify in different situations (such as logs). A user ID is an integer usually ranging from 0 to 4294967295 because they are 32-bit integers. In this part, you must write a program that receives a username and outputs the corresponding userID.

Hint: To solve this, first find the current user id of the given username. "You can test it by `id -u <username>`".

Steps:

1. Using a text editor (such as gedit or vi), create a new C file called "my_userid.c". This program will contain the main function, which will display the username's user ID.
 - a. Get a username from standard input and check whether it is valid, then get the corresponding userid for that username.
 - b. The output should contain only a single number indicating the username's userID. If a user inputs an invalid username, print "*no such user found*".
2. Run '`./my_userid`' to run the program.

Output sample:



```
615614
```

Question 2. Write a C program that will print the average partition usage in your system.

Storage usage is one of the important factors in a computing system. Storage is one of the important factors in a computing system. In this part, you are to calculate the average usage of each partition mounted on your system. In this part, you are to calculate the average usage of each partition mounted on your system.

In order to do so, you need to list all the partitions and their usage and then calculate a simple average. In order to do so, you need to list all the partitions and their usage and

then calculate a simple average. "hint: you can use `df -h` to verify the input information for your program."

Steps:

1. Using a text editor (such as gedit or vi), create a new C file called `'my_storage_use.c'`.
2. Write the necessary instructions to print file system usage information **to a .txt file**. It should contain: 1) name of the Filesystem, 2) the actual size of the filesystem, 3) Used and Avail storage, 4) where it is mounted on. Here's a sample of what should be inside the .txt file:

```
Filesystem      Size  Used Avail Use% Mounted on
udev            16G   0    16G   0% /dev
tmpfs           3.2G  2.3M  3.2G   1% /run
/dev/mapper/ROOTDISK-root 127G  40G   81G  33% /
tmpfs           16G   0    16G   0% /dev/shm
tmpfs           5.0M  4.0K  5.0M   1% /run/lock
tmpfs           16G   0    16G   0% /sys/fs/cgroup
/dev/loop0      56M   56M    0 100% /snap/core18/2284
/dev/loop3      71M   71M    0 100% /snap/lxd/21029
/dev/loop5      56M   56M    0 100% /snap/core18/2128
/dev/loop4      68M   68M    0 100% /snap/lxd/21835
/dev/loop2      44M   44M    0 100% /snap/snapd/14978
/dev/loop1      62M   62M    0 100% /snap/core20/1328
```

3. Calculate the average partition usage. You have to use the use% column for the calculations.
4. Print the result **on the console screen** up to two decimal points only, followed by newline (`\n`). For example, `60.02`. **Do not put a % sign or any other characters** with the numbers. Below is an example solution:

```
2.75
```


Question 3. Write a C program to display the installed shell programs in your system.

Linux is a family of operating systems based on the Unix operating system. For example, Ubuntu, Debian, RedHat, Fedora, etc., are Linux-based operating systems. Using a shell is a very useful tool to interact with the operating system and run commands. A shell is a CLI (command line interface) that does command interpretation and execution for users. In this task, you must print out the available shells in your system.

Hint: The name and path of each shell are stored at `/etc/shells`

Steps:

1. Using a text editor (such as gedit or vi), create a new C file called `'my_shells.c'`.
2. Read and print the output on the console. The output should only contain the name of the shell(s) and not their paths. Avoid duplicate names as well. Below is a sample of the output.



```
bash
ksh2020
```

Submission Instructions:

Submit .zip files on Canvas. Your submission should be composed of the following files, named exactly as mentioned in the instructions:

- **Required files**
 - my_userid.c
 - my_storage_use.c
 - my_shells.c
 - makefile

Rubric (Out of 40)

Check canvas.

FAQ

1. Are we allowed to use grep? yes
2. Are we allowed to use standard library functions? Yes
3. Are we allowed to use the pwd library? No
4. Can we use popen() , system()?

It's acceptable to utilize **standard library functions** such as popen() and system(). Popen() is a wrapper for fork and exec, but it's unnecessary for this assignment. You can choose not to use it, as this assignment doesn't require concepts like fork() and exec(). If you decide to use it, ensure you understand its functionality.

5. Can we use string.h? Yes
- 6.