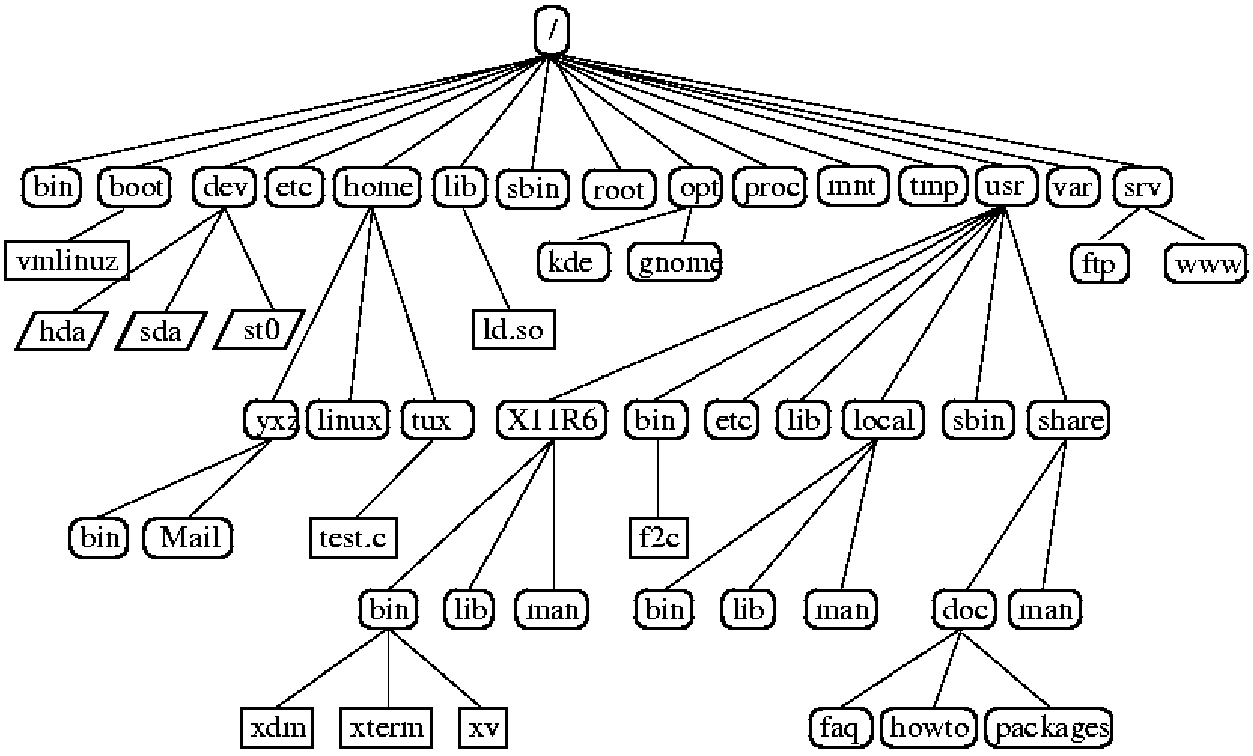
In this course, we will build 4 assignments, which together make up a project. The assignments build one upon the other, to create an abstraction of a Linux Filesystem Hierarchy Standard. In simple words, the goal of the first three assignments will be to build something like what is depicted in the Figure below. The first part of the third assignment will build something like this, and from that point on, i.e. from the latter parts of the third assignment and the fourth assignment onwards, we will create different combinations of the tree to learn more complicated data structures.

When we use our computers (desktops, laptops, cell phones, tablets, etc), we regularly create and update files and directories, irrespective of the underlying Operating System (Windows, Linux, UNIX). The general hierarchy of files looks like in the Figure below. The **tree** command in Linux/UNIX based systems can get you a more clear picture of this. I implore you to explore the command, and understand how the hierarchy looks, just for curiosity.



Assignment 1

Pointers, Arrays, Linked Lists, Vector

Warm-up exercises **(60 pts)**

Create a pointer to a bunch of integers, an array and C++ std::vector of integers, and a linked list of integers. Each should follow these basic rules:

1. Pointer - Make a pointer pointing to 8 64 bit, or 8 byte, integers.
    a. Put the following 8 numbers: 65536, 78890, 12345, 98765, 87906, 45637, 863248, 4632768 consecutively, one after the other, into the memory pointed to by the pointer. (**2 pts**)
    b. Once the pointer points to these 8 numbers, print them out. (**2 pts**)
    c. Print the addresses of these numbers. (**2 pts**)
    d. Reallocate memory, to add 4 more 64 bit integers: 768, 765, 92034, 65789 (**3 pts**)
2. Array - Create an array of 8 64 bit, or 8 byte, integers.
    a. Put the following 8 numbers: 63556, 9887, 54321, 56789, 87906, 73654, 842368, 8672364 into the array. (**2 pts**)
    b. Print the array in reverse order. (**2 pts**)
    c. Print the addresses of these numbers in the array. (**2 pts**)
    d. What happens when you try to add a 9th element? (**2 pts**)
3. Vector - Create a vector of 64 bit integers. Perform the steps in order.
    a. Assign 100000 to 8 places in the vector. **Hint: Although there are multiple ways of doing this, you can do this in one line while declaring the vector itself. Other solutions are also acceptable.** (**2 pts**)
    b. Use push_back() 3 times to add 8000, 9000 and 7000, then print the vector (**Hint: Use iterator to print**). (**2 pts**)
    c. Use pop_back() 5 times, then print the vector. (**3 pts**)
    d. Use size() to print the size of the vector. (**2 pts**)
    e. Use insert() to insert 89748 at the beginning of the vector. (**2 pts**)
    f. Use emplace() to insert 50000 at the beginning of the vector. (**2 pts**)
    g. Use emplace_back to insert 78 at the end of the vector, then print the vector. (**2 pts**)
    h. Insert 20 **at** the 2nd, 4th and 7th position, then use pop_back() to erase the last element. Print the vector. (**3 pts**)
4. Linked List - Create a doubly linked list of 64 bit integers. Use **both struct** and **class** to create the linked list. Name the struct **cube_list** and the class **CubeList**. Be careful to point the **prev** and **next** pointers to **NULL**. Perform the steps in order.
    a. Create a linked list with cubes of the first 30 natural numbers. (**4 pts**)
    b. Add the cube of 50 to the start of the linked list. (**4 pts**)
    c. Add the cube of 60 to the end of the linked list. (**4 pts**)
    d. Add the cube of 40 to the middle of the linked list. **Hint:** Traverse through the two linked lists at different rates (think of traversing with **node->next** and **node->next->next**). (**5 pts**)
    e. Using the method in d, remove the middle element (before the cube of 40) from the linked list. (**4 pts**)

f. Remove the first element from the original linked list (i.e. 1). (**4 pts**)

Main Exercise **(40 pts)**

The definition of the class **NodeInfo** is given in the header file **NodeInfo.h**. You will be implementing the class and the functions in the file **NodeInfo.cpp**. The purpose of the variables and the functions in the class is given in the comments in the files.

The file **files_dirs** has data in the following order: **file/dir**, **filename**, **parent_directory**. The goal is to use the attributes in the class to create a linked list. Every node is connected to the next node in the file. Note that the files and directories are not sorted. We will simply create a linked list of files and directories.

Grading Rubrics:

1) Appending to the linked list using a correctly implemented **append** function. We will test this using our own printList function. (**30 pts**)
2) Printing the whole list in correct order using the given **printList** function. **Implement printList in a different file.** (**10 pts**)

Submission Details:

We will be running your submissions as scripts. Please follow the instructions to the tee, otherwise it can lead to evaluation issues with the scripts breaking and your grades being evaluated incorrectly.

Submission: Push to your github repository in the github organization for the class.

Name the files:
w1.cpp
w2.cpp
w3.cpp
w4.cpp
Main exercise source file - NodeInfo.cpp
Main exercise printList file - print_list.cpp

You should start the project with the given material. We will provide you with more test cases for the Main exercises after one week. You are encouraged to create some test cases of your own to test your code. You shouldn't need any starting code for the Warm-ups in this assignment, since it is supposed to be a revision of CSCE121. Please reach out for help to me, TA or your Peer Teachers if you face difficulties - even if the questions are along the lines of "Where do I begin?" or "What should be my first line of code as I begin coding?".

For anyone having difficulty beginning coding in C++, there is plenty of material and tutorials available online. This is an official guide: https://www.cplusplus.com/doc/tutorial/ that you can use. **You DO NOT need to cite any official cplusplus.com documentation that you use.**

Instructions:

1. Each warmup exercise should be coded in a separate file named w1.cpp, w2.cpp etc. respectively.
2. On running the warmup file, it should output all the subsections one after the another. The first part (a) of every warmup does not need any output. But the rest of the subsections should have an output.
3. The output should be formatted in the following way:

```
b:
100000
100000
c:
100000
100000
d:
100000
100000
e:
100000
100000
```

Each warmup will have a similar output structure.

4. Output files need not be pushed to github. Only source code is enough.
5. The answer for warmup 2, d subsection should be written in the comments in the source file (w3.cpp).
6. The output for NodeInfo.cpp should be formatted in the following way:

(file/directory name) (1/0 - 1 if directory)

```
alpha 1
bin 1
boot 1
dev 1
```

In comments at the top of the file in NodeInfo.cpp, mention the following:

1. List of students you discussed with.
2. Any external source you used (it is important to cite the sources you use, otherwise it is plagiarism).
3. If the code is not working, the concerns you faced (so that we can guide you with the correct resources to complete this assignment after the deadline, and for future assignments).