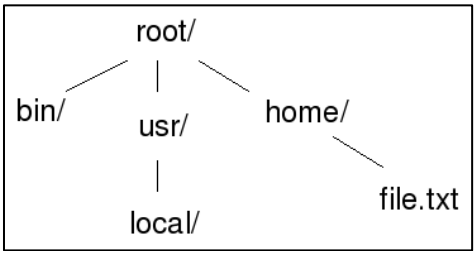
CENG 242

Homework #3

(Due: April 16th 2010, Friday 23:55)

In this homework, you will implement an abstract data type (called DirTree) representing the Unix Directory Tree, and provide interface functions for its manipulation.

A Unix Directory Tree is a hierarcy of directories, beginning with a single root directory at the top. Intermediate nodes and the root node have to be "directories". Leaf nodes can either be directories or files. A sample directory tree is below:



(Notice that the root directory is denoted by the string "root", instead of "/". We will use this modified representation in this homework for your convenience in parsing of paths.)

We will not give you the type signature of DirTree. You will implement it anyway you want and hide the type internals. The module you will implement should look like:

```
module DirTree (QuadTree, interface functions) where
   data DirTree = internal definition of type
    definitions of interface and auxiliary functions
```

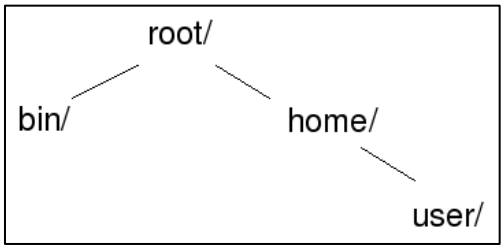
We kindly (!) ask you to implement and export the following interface functions on DirTree:

newDirTree :: DirTree
Creates an initial empty directory tree which is composed of a single "root" directory with no children.

mkdir :: DirTree -> String -> DirTree
Creates a directory in the path denoted by the second argument. The path is guaranteed to be an absolute path (beginning from the "root" directory, see the section I/O Specifications.)

`mkdir (newDirTree) "root/usr")` creates a "`usr`" directory in an empty directory tree:

`(mkdir (mkdir (mkdir (newDirTree) "root/home") "root/home/user") "root/bin")` results in:



You have to check wheher the given path really exists in the directory tree. If not, you should return the original directory tree with no modifications. For instance, `(mkdir (newDirTree) "root/null/newdir")` returns the empty tree with only the `root` directory.

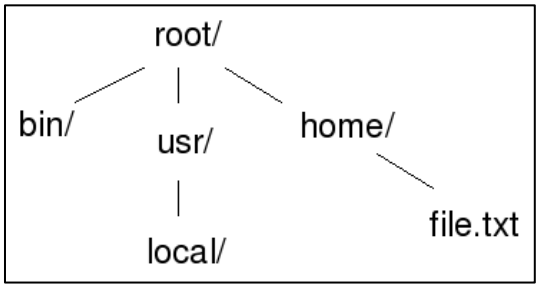
You have one guarantee, tough: It is guaranteed that we will not try to create an already existing directory. (i.e, a directory which has the name of an already existing file/directory in the given path.)

touch :: DirTree -> String -> DirTree

Creates a file in the path denoted by the second argument.

`(touch (newDirTree) "root/file.txt")` creates a file.txt in an empty directory tree:

`(touch (mkdir (mkdir (mkdir (mkdir (newDirTree) "root/bin") "root/usr") "root/home") "root/usr/local/") "root/home/user/file.txt")` results in:



The error checking conditions are similar to the mkdir function: You have to check whether the given path really exists in the directory tree. If not, you should return the original directory tree with no modifications. `(touch (newDirTree) "root/null/file.txt")` returns an empty directory tree composed of only root directory. You are guaranteed that we will not try to create an already existing file. (i.e, a file which has the name of an already existing file/directory in the given path.)

ls :: DirTree -> String

Lists the contents of a given directory. If the given directory path does not exists, or if it is a file instead of a directory, ls returns an empty string (“”). Take care that ls does not descend into a directory recursively.

```
ls (mkdir (mkdir (mkdir (mkdir (mkdir (newDirTree) “root/home”) “root/usr”)
"root/usr/bin") “root/usr/local”) “root/usr/local/include”) “root/usr”)
```

outputs

bin
local

(that is, “bin\nlocal\n”)

ls output should be chronological. Previously created items in a directory should be printed first. (In this example, bin is created before local directory.)

rm :: DirTree -> String → DirTree

Removes a given path (and everything under it) from the directory tree. (It works like rm -rf actually.) The given argument may be either a file or a directory, If it is a directory, all its descendants must also be removed. The given path may not exist, in this case the given directory tree should be returned with no modification. You can assume:
1) The root directory cannot be removed.
2) The current working directory (see below) cannot be removed.

`(rm (mkdir (mkdir (mkdir (mkdir (newDirTree) “root/usr”) "root/usr/bin") “root/usr/local”) “root/usr/local/include”) “root/usr”)` results in the empty directory tree.

locate :: DirTree -> String → String

Lists everything in the directory tree that has an occurrence of the given string in its absolute path (including its name) . locate output should be chronological and depth-first. Previously created items in a directory should be printed first. Descendants of a directory should be printed before advancing to its siblings. The output of locate should contain absolute paths.

```
(locate (touch (touch (mkdir (mkdir (mkdir (mkdir (newDirTree) “root/home”)
“root/bin”) “root/usr”) "root/usr/bin") “root/home/file.bin”)
“root/usr/bin/matlab”) “bin”)
```

outputs:

root/home/file.bin
root/bin
root/usr/bin
root/usr/bin/matlab

(Note that the home directory is created first, so its descendant is printed first.)

pwd :: DirTree -> String

Outputs the current working directory. When the tree is first created, the current directory is root. mkdir et.c functions do not change the current working directory, the only function to change it is cd (see below).

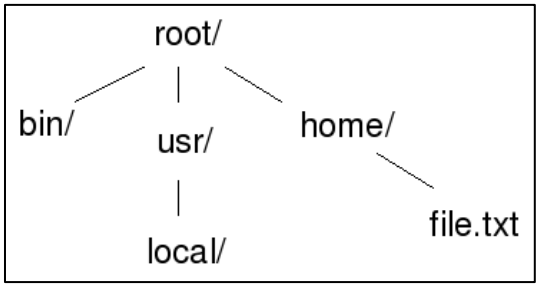
cd :: DirTree -> String → DirTree

Changes the current working directory. It does not output a string, only returns the modified directory tree. Its effect should be visible in a subsequent pwd call. Note that: (1) cd'ing to the current directory has no effect, (2) cd'ing to a non-existent directory has no effect, and (3) cd'ing to a file has no effect.

show :: DirTree -> String

DirTree should be an instance of Show class. The output will be Lisp-like with indentation (Yes, indentation is important.)

An example is:




```
("root/"
        ("bin/")
        ("usr/"
                ("local/")
        )
        ("home/"
                ("file.txt")
        )
)
```

In the show function, notice that you should display a trailing "/" at the end of names of directories. There are no other differences between the outputs of directories and files other than that.

You can give your show output to putStr function to display the trees with proper newlines. (i.e, putStr (show (newDirTree)))

I/O Specifications:

1. All paths given will be absolute paths for your convenience. (i.e, in the format "`root/usr/local`", that is descending from the root directory) You will not be given any relative paths (Illegal examples are: "`.`", "`../home`", "`e1111111/Desktop`") There will *not* be a trailing "/" at the end of arguments, even if the path ends with a directory.

2. The requests made may not be error-free. In case of an erroneous request, the function should return the argument `DirTree` with no modification. *Any* illogical request (examples include, but NOT limited to, trying to create a file inside another file, trying to create a directory inside a file, trying to remove a non-existent file or directory, trying to cd to a file, trying to cd to a non-existent directory, ls'ing a non-

existent directory) should return the original tree with no modification.

3. In the implementation you are free to use any internal representation and make auxiliary definitions. However you should hide the other definitions in local declaration blocks so that you will not define any other identifiers but the ones mentioned above in the global environment. Note that all modifying functions do not update the existing tree but return a new tree with modifications applied.

Specifications:

1. All the work should be done individually.
2. Your codes should be written in Haskell and have the name "hw3.hs"
3. In evaluation, black box method will be used. So be careful about the name of functions, data structures etc.
4. You will submit your code through Cow system.
5. You should test your codes in inek machines with hugs before submitting.