CENG 242
Homework #4
(Due: May 9th 2010, Sunday 23:55)

In this homework, you will implement `DirTree` class with C++ which was the subject of the third homework also.

As in the previous homework, the `DirTree` class will represent the Unix Directory Tree and provide the same interface functions and some more. The directory tree structure is same as the one which was explained in the previous homework. The functions that you have to implement is given below in the class declaration. You are free to add new functions or data types in your class.

```
class DirTree {
      public:
            DirTree();  // Creates an initial empty directory tree which is
                        // composed of a single "root" directory with no children.
            DirTree(const DirTree & rhs); // copy constructor
            ~DirTree(); // destructs the directory tree

            void mkdir(const string & path); // creates a directory in the given
                        // path. If the directory already exists
                        // in the given path, throw an Error typed value
                        // DIRECTORYEXISTS. If the directory is tried to be created
                        // in an invalid path, throw an Error typed value
                        // NOSUCHFILEORDIRECTORY. Ex;
                        // mkdir("root/home/user/Desktop/newDir");
                        // the directory named "newDir" is asked to be created
                        // under "Desktop".
            void touch(const string & path); //creates a file in the given path
                        // path. If the file already exists
                        // in the given path, throw an Error typed value
                        // FILEEXISTS (don't confuse with the actual unix touch
                        // command because you don't get an error like this, but in
                        // this case you should throw an Error).
                        // If the directory is tried to be created
                        // in an invalid path, throw an Error typed value
                        // NOSUCHFILEORDIRECTORY. Ex;
                        // touch("root/home/user/Desktop/file.txt");
                        // the file named "file.txt" is asked to be created under
                        // "Desktop".
            string ls();      //lists contents of the working directory. Output
                        // should be chronological. Previously created items in a
                        // directory should be printed first. Each item should be
                        // followed by a newline character. Ex; a possible output
                        // can be "newDir\nfile.txt\n".
            void rm(const string & path); // removes a directory or a file in the
                        // given path. If it is a directory, all its descendants
                        // must also be removed. If the file or directory doesn't
```

```
            // exist or if the given path is invalid, throw an Error
            // typed value NOSUCHFILEORDIRECTORY. If the working
            // directory or the ancestors of the working directory or
            // the root directory is tried to be removed
            // then throw an  Error typed value INVALIDACTION. Ex;
            // rm("root/home/user/Desktop/dir"); the directory named
            // "dir" is asked to be removed.
    string locate(const string & path); //Lists everything
            // in the directory tree that has an
            // occurrence of the given string in its absolute path
            // (including its name). the output of locate 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. A newline character '\n' should follow each path.
    string pwd();     // outputs the working directory (the directory which
            // you are standing in). When the tree is first created,
            // the current directory is root. The output should be the
            // absolute path like "root/home/user/Desktop" not
            // "Desktop".
    void cd(const string & path); //changes the working directory. If the
            // given path is invalid, throw an Error typed value
            // NOSUCHFILEORDIRECTORY.
    void cp(const string & copyFrom, const string & copyInto); //copies a
            // file into a file or a directory. If copyInto is a file
            // and it doesn't exist then it is created with the
            // name copyInto. If copyInto is a directory,
            // copyFrom is copied (into a file named copyFrom)
            // inside of the directory copyInto. If an
            // invalid path is given as an input, throw an  Error typed
            // value NOSUCHFILEORDIRECTORY. (Since we just hold file
            // names, this command is a simple version of the actual
            // one)
    void mv(const string & moveFrom, const string & moveTo);   // moves a
            // file or a directory into a directory or renames a file
            // or directory.
            // Case1: moveFrom and moveTo are files. If moveTo doesn't
            // exists, then renames moveFrom as moveTo. If moveTo
            // exists, just removes moveFrom.
            // Case2: moveFrom is a file and moveTo is a directory.
            // Moves the file moveFrom into the directory moveTo.
            // Case3: moveFrom and moveTo are directories. If moveTo
            // doesn't exist, then renames moveFrom as moveTo. If
            // moveTo exists, moves the directory moveFrom under
            // moveTo.
            // If an invalid path is given as an input, throw an  Error
            // typed value NOSUCHFILEORDIRECTORY.

    friend ostream & operator<<(ostream & ost, const DirTree & dt);
            // prints the directory tree. The format of the output must
            // be same as the show function in the previous homework.
```

```
};
```

Error is an enumerated type:

enum Error {NOSUCHFILEORDIRECTORY, DIRECTORYEXISTS, FILEEXISTS, INVALIDACTION};

I/O Specifications:

1. Pathnames can be absolute pathnames or relative pathnames. An absolute pathname begins with the root directory and follows the tree branch by branch until the path to the desired directory or file is completed (i.e, in the format "root/home/user", that is descending from the root directory). A relative pathname starts from the working directory. To do this, it uses a couple of special symbols to represent relative positions in the file system tree. These special symbols are "." (dot) and ".." (dot dot). However, "." (single dot) isn't going to be given in the pathnames. On the other, hand ".." can be used. The ".." symbol refers to the working directory's parent directory. For ex; you are in "root/home/user/Desktop/dir1/dir5", you can refer another directory named dir2 under Desktop with "root/home/user/Desktop/dir2" or "../../dir2". Or you are in "root/home/user/Desktop", you can refer directory named dir5 under dir1 with "root/home/user/Desktop/dir1/dir5" or "dir1/dir5". 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, you should throw Error with the given Error types.
3. Except the ".." (dot dot) directory, directory names won't include a dot (.) and file names will always include a dot (.) which means files will always have extensions. So that you can tell a given name is a directory or a file by looking at its name. Ex; "sample.txt" is a file but "sample" is a directory.

Specifications:

4. All the work should be done individually.
5. In evaluation, black box method will be used. So be careful about the name of functions, data structures etc.
3. You will submit your code through Cow system.
4. In error conditions mentioned above, only throw the error. Do **not** write any code to catch it in your submission.
5. Your codes should be written in C++ and you should submit a tar file **hw4.tar**. The tar file should include **hw4.h**, **hw4.cpp**. In hw4.h, you should include class declarations, structures, enumerated types (Error, etc). In hw4.cpp, you should include the definitions. You should not have **main** function or any other global function in these files.
6. You should test your codes in **inek** machines by compiling with **g++** before submitting.