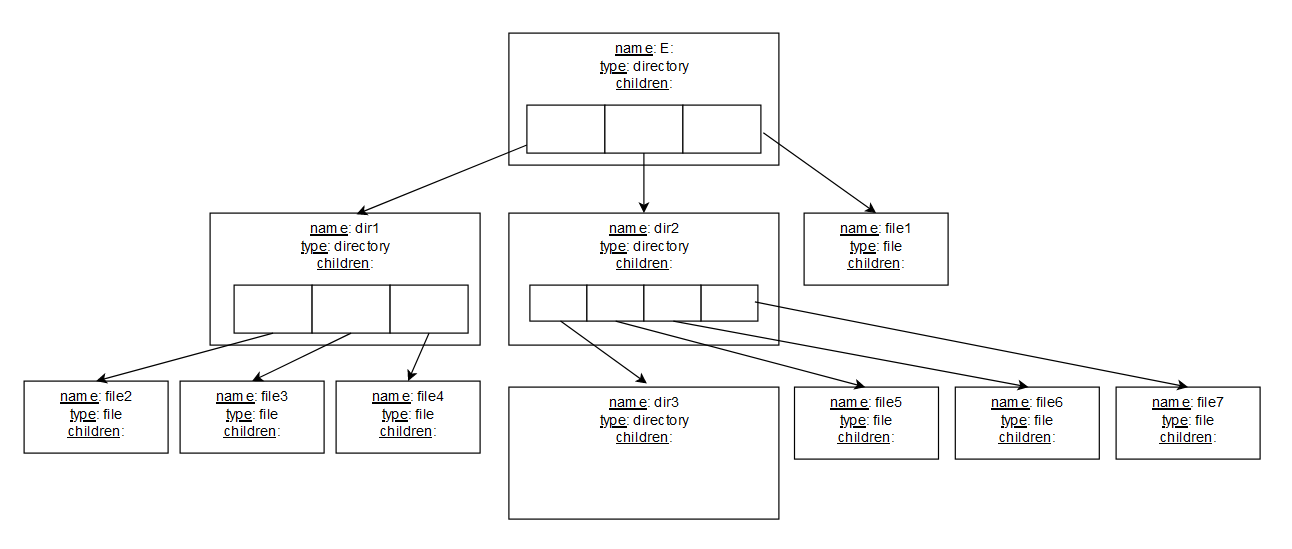
Project: Filesystem Simulator

For this project you will complete a class which simulates a filesystem structure based on a tree where each file/directory is a node. For example, a filesystem structure that looks like this:



Would be stored in the tree structure visualized as:

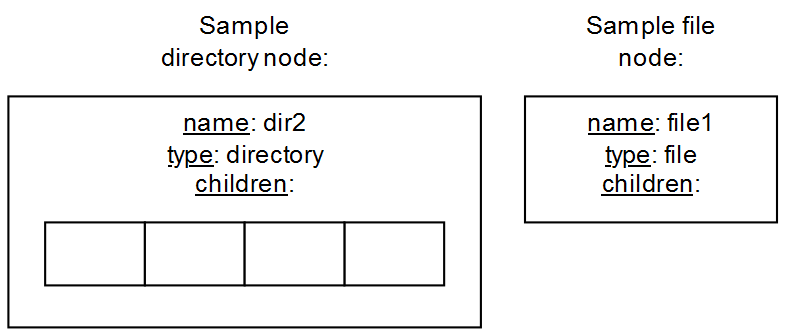


# Part 1: FSNode Class

Each file/directory in the filesystem tree is represented by a FSNode object. Each FSNode contains the private members:

|  |
| --- |
| name: string containing the file/directory name. For simplicity, names may only contain only alpha and digit characters (not even a period). |
| type: int containing 1 for directory, 0 for file. Note constants are defined for these types in FSNode.h. |
| children: vector of FSNode pointers to the children of this node. For a node of file type, the size of this vector must always be zero (no children).  For a node of directory type there is no limit on the number of children.  Children of a directory must have unique names, and the pointers in chidren are maintained in the vector in sorted order by name of the child.  It may be helpful to review/refresh the [vector](http://www.cplusplus.com/reference/vector/vector/) documentation for working with children. |

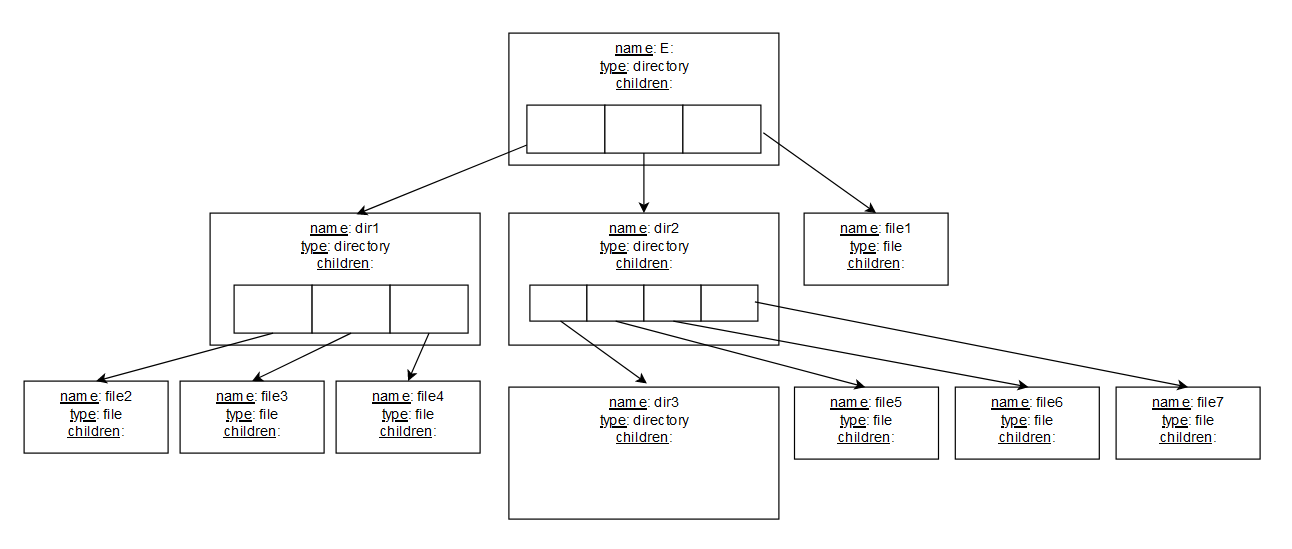
Here are visual representations of FSNode objects:



The FSNode class is already completed in FSNode.h and FSNode.cpp. At minimum, you’ll want to review the documentation in FSNode.h.

# Part 2: FilesystemTree Class

The FilesystemTree class contains a tree made up of FSNodes which represents a filesystem. It contains just a single private member variable, rootPtr, which is a pointer to the root of the filesystem. The root node is named E: in this visualization:



## The Root Node

Technically the root node is a directory type node, but it has some special features:

* The root node can never be deleted. So, a FilesystemTree object will always have *at least* a root node.
* The root node’s name is defined by the constant ROOT\_NAME in FilesystemTree.h. The root node is the *only* node which may contain non-alpha/digit characters in its name. This allows it to have a name such as “E:”. To allow this the FilesystemTree class is a friend of the FSNode class so that FilesystemTree can write the name of the root node directly into its name variable (rather than using FSNode’s setName() which doesn’t allow non-alpha/digit characters). You should not change ROOT\_NAME or the supplied testing code will not work.
* FilesystemTree should keep the root node’s name unique throughout the filesystem. In other words, no other file or directory can have the same name as the root node.

## Paths

You are most likely aware that the location of files and directories in a filesystem are often specified by *paths*. A full path begins with the name of root, then a series of directory names delimited with a separating character, finally ending with the name of the file or directory for which the path is being specified.

For example, the full path to file5 in the diagram above it might look like this:

E:/dir2/file5

The separating character in this example is ‘/’. So, we know from the path above that we can find file5 by beginning at the root (E:) and navigating into directory dir2.

The separating character for the FilesystemTree paths is specified by a constant in FilesystemTree.h named SEPARATING\_CHAR. You should not change this or the supplied testing code will not work.

FilesystemTree has several functions that work with paths, and they are always full paths. In other words, all paths begin at the root. FilesystemTree does not work with *relative paths* which may begin at locations other than root.

## Testing

The provided main.cpp contains a series of test functions for FilesystemTree which include expected output.

Included with the starter files is the file input.txt which, along with a provided parameterized constructor, creates a FilesystemTree for testing.

Also included in the starter files is a sample output file showing expected output when all the FilesystemTree testing functions are run.

## Functions Already Complete

Several functions in FilesystemTree are already completed for you. You’ll want to look at the documentation in FilesystemTree.h to make sure you understand what these completed functions do and how to use them:

* FilesystemTree()
* FilesystemTree(const std::string& fileName)
* FilesystemTree(const FilesystemTree& treeToCopy)
* pathToPointer(const std::string& path)
* operator=(const FilesystemTree& rTree)
* displayStats(std::ostream& outStream)
* recursiveStats(std::shared\_ptr<FSNode> startPtr)
* displayTree(std::ostream& outStream)
* recursiveDisplay(std::shared\_ptr<FSNode> nodePtr,

std::string preSpace, std::ostream& outStream)

## Functions to Complete

You’ll need to complete the FilesystemTree functions below. Several testing functions for FilesystemTree are provided in main.cpp. If you complete the functions in order from top to bottom in the table below it should correspond to the order of the FilesystemTree testing functions provided in main.cpp.

|  |
| --- |
| bool create(const std::string& pName, int pType,  const std::string& parentPath)  Attempts to add a node named pName of type pType to the tree at the path parentPath. Returns true if the node was successfully added.  This function should not allow a new node to be created with the same name as the root node, nor should it allow file/directories with the same name in the same parent directory. |
| bool remove(const std::string& pName,  const std::string& parentPath)  Attempts to remove the node named pName from the directory at the path parentPath. Returns true if the node was successfully removed.  This function should not allow the removal of the root node. |
| bool move(const std::string& pName,  const std::string& sourcePath,  const std::string& destPath)  Attempts to move the node named pName from the directory at the path sourcePath to the directory at destPath. If pName is a name of a directory it moves the directory and all its contents. Returns true if the move was successful.  This function should not allow the root node to be moved. |
| std::shared\_ptr<FSNode>  copySubTree(std::shared\_ptr<FSNode> rootPtr)  Returns a fully independent copy of the subtree which begins at rootPtr. This should work with any subtree or the entire tree beginning with the root node.  My implementation is recursive, though it could probably be done iteratively. |
| bool copy(const std::string& pName,  const std::string& sourcePath,  const std::string& destPath)  Attempts to copy the node named pName from the directory at the path sourcePath to the directory at destPath. If pName is a name of a directory it copies the directory and all its contents. Returns true if the copy was successful.  To work properly, this will need to be a deep copy, not just a change of pointers. You should probably make use of copySubTree().  This function should not allow the root node to be copied. |

The starter files should compile without issue. If you get an exception, then it probably isn’t finding the data file. In that case, change “input.txt” in main() to be the full path to the file on your system with backslashes doubled. For instance, if the file is on your system at C:\Users\bob\class\input.txt, then change “input.txt” to:

“C:\\Users\\bob\\class\\input.txt”

# Turn In

When finished upload FilesystemTree.cpp.

Jacob Kier

Johnson County Community College