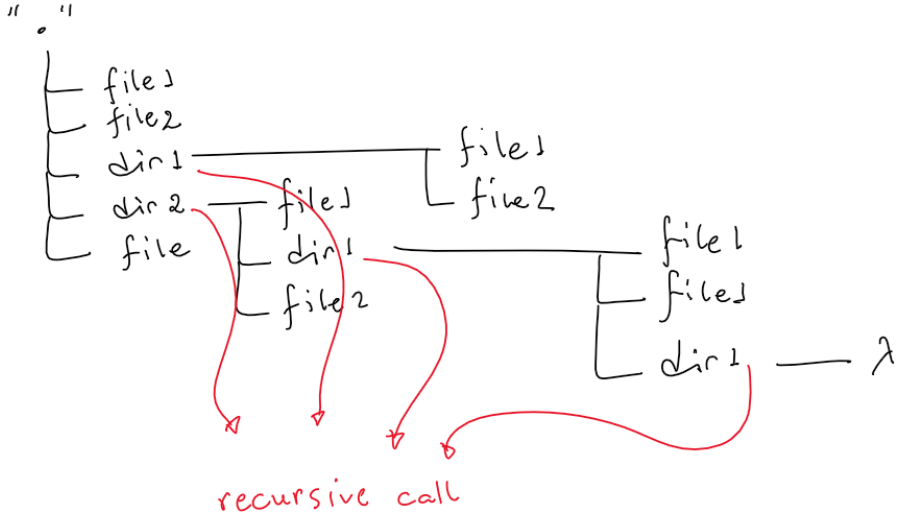
# Problem 2: File Recursion

The problem was addressed by creating a ***find_files()*** function that traverses a giving path looking for .c files. If an entry in the current path is a directory, the ***find_files()*** is called recursively using the next level path as parameter.

To keep track of the results found the algorithm uses a set object to accumulate the results. This object type was chosen since partial results from a higher-level directories are easily merged into lower level ones using set union operations.




The problem input ***n*** consists of a collection of entries, each one being a directory or file in all possible levels. Each entry is checked with ***os.path.isdict()*** or ***os.path.isfile()*** functions, which leads to an approximate ***O(n)*** time complexity.

Each recursive call returns a set accumulating a partial result which is united to the immediate lower-level result. The time complexity of ***union s | t*** operation in python sets is ***O(len(s)+len(t))***. Considering ***s*** and ***t*** equivalent to partial results from two consecutive directory levels and considering that the number of matched suffixes is limited by the input itself, the time complexity of the total onion operations is ***O(n)***. Thus, the time complexity of the overall algorithm results in ***O(n)***.

Regarding memory usage, the set collection used in this algorithm stores a string for each matched result, which is limited by the number of possible inputs, so it has a worst-case ***O(n)*** space complexity.