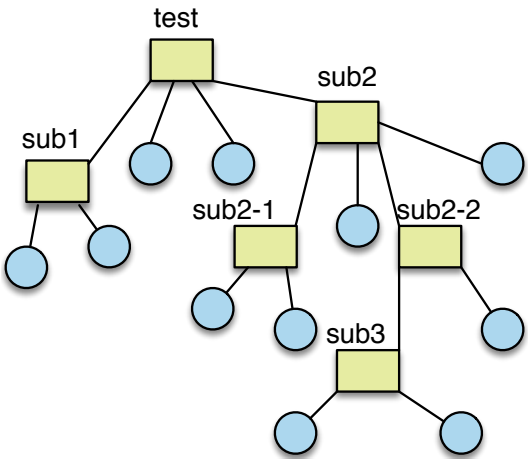
4: Sniff a Directory Tree

CSCI 4547 / 6647 Systems Programming
March 2022

1 Goals

1. To write and use a recursive function.
2. To find sniff-words in any and all files in a directory tree.



1.1 Preparation

Create directories (green) and files (blue) as shown on the right in your test directory. Note that the diagram has up to 3 levels of subdirectories. Every directory should have a couple of files. Then add soft links in at least two directories and add several hard links. Each one should go from one subdirectory to a file in a different directory.

2 main()

As in Program 3, call banner() and print a welcome message on the screen. Construct a Params object and pass it to the constructor of the Sniff class. Call `Sniff::run()` to process the directory tree. Call bye() from the tools library.

3 Sniff::run()

Add this function to the Sniff class. The parameter should be a string, the the simple name of the starting directory. Move

- As in Program 3, set `current` and `path` to the simple name of the starting directory.
- Change directory to `current`.
- Call `travel` with `current` as the parameter.
- When all the recursive calls return, you have constructed a vector of all the files in your directory tree that contain sniff-words.
- Loop through your vector of files. Print a blank line and a header line for each file, then print the list of sniff words in the FileID object

4 The travel() function

Define a `void` recursive function called `travel` in the Sniff class. The parameters will be the pathname (a string) and the simple name of the directory to be processed next. This function will handle directory entries for both files and subdirectories. File entries will be processed and, maybe, stored in the `vector<FileID>` object that is a member of the Sniff class. Directory names will be processed recursively.

When `travel` is first called, the parameters should be the pathname and simple name of the place you will start the sweep operation. In the body of `travel()`:

- Open the directory `current`. (The chdir() should already be done.
- Read each entry in `current`. If it is not a directory process it as in P3.
- If it is a directory, prepare for and make a recursive call on `travel()`. The first parameter should be the concatenation of the current path, '/', and the simple name of the new directory.
- By doing the concatenation operation in the call itself, you avoid needing to REMOVE the last link of the path when you return from the recursion.
- When the recursion returns, change directory to `..`