The Directory Entry & Directory Iteration

In this lesson, we'll see different ways of iterating over a path.

WE'LL COVER THE FOLLOWING

While the `path` class represent files or paths that exist or not, we also have another object that is more concrete: it's `directory_entry` object. This object points to existing files and directories, and it's usually obtained by iterating using filesystem iterators.

What's more, implementations are encouraged to cache the additional file attributes. That way there can be fewer system calls.

Traversing a Path with Directory Iterators

You can traverse a path using two available iterators:

- `directory_iterator` - iterates in a single directory, input iterator
- `recursive_directory_iterator` - iterates recursively, input iterator

In both approaches the order of the visited filenames is unspecified, each directory entry is visited only once.

If a file or a directory is deleted or added to the directory tree after the directory iterator has been created, it is unspecified whether the change would be observed through the iterator.

> In both iterators the directories `.` and `..` are skipped.

You can iterate through a directory using the following pattern:

```
for (auto const & entry : fs::directory_iterator(pathToShow))
{
   ...
}
```

Or another way, with an algorithm, where you can also filter out paths:

```
std::filesystem::path inPath = /* GetInputPath() */;
std::vector<std::filesystem::directory_entry> outEntries;
std::filesystem::recursive_directory_iterator dirpos{ inPath };

std::copy_if(begin(dirpos), end(dirpos), std::back_inserter(outEntries),
             some_predicate);
```

> `some_predicate` is a predicate that takes `const directory_entry&` and returns `true` or `false` depending on if a given `directory_entry` object matches our filter or not. All matching paths are pushed back to the `outEntries` vector. See "Filtering Files Using Regex" in the Examples section of this chapter to see the use case of this technique. Also, instead of the output vector of directory entries, you can use a vector of paths since directory entries can convert into paths.

`directory_entry` Methods

Here's a list of `directory_entry` methods:

Operation	Description
`directory_entry::assign()`	replaces the path inside the entry and calls `refresh()` to update the cached attributes
`directory_entry::replace_filename()`	replaces the filename inside the entry and calls `refresh()` to update the cached attributes

`directory_entry::refresh()`	updates the cached attributes of a file
`directory_entry::path()`	returns the path stored in the entry
`directory_entry::exists()`	checks if a directory entry points to existing file system object
`directory_entry::is_block_file()`	returns true if the file entry is a block file
`directory_entry::is_character_file()`	returns true if the file entry is a character file
`directory_entry::is_directory()`	returns true if the file entry is a directory
`directory_entry::is_fifo()`	returns true if the file entry refers to a named pipe
`directory_entry::is_other()`	returns true if the file entry is refers to another file type
`directory_entry::is_regular_file()`	returns true if the file entry is a regular file
`directory_entry::is_socket()`	returns true if the file entry is a named IPC socket
`directory_entry::is_symlink()`	returns true if the file entry is a symbolic link
`directory_entry::file_size()`	returns the size of the file pointing by the directory entry
`directory_entry::hard_link_count()`	returns the number of hard links

`)`	referring to the file
`directory_entry::last_write_time()`	gets or sets the last time write for a file
`directory_entry::status()`	returns status of the file designated by this directory entry
`directory_entry::symlink_status()`	returns the symlink_status of the file designated by this directory entry

---

In the next lesson, we will look at some additional supporting non-member functions that the `filesytem` library provides!