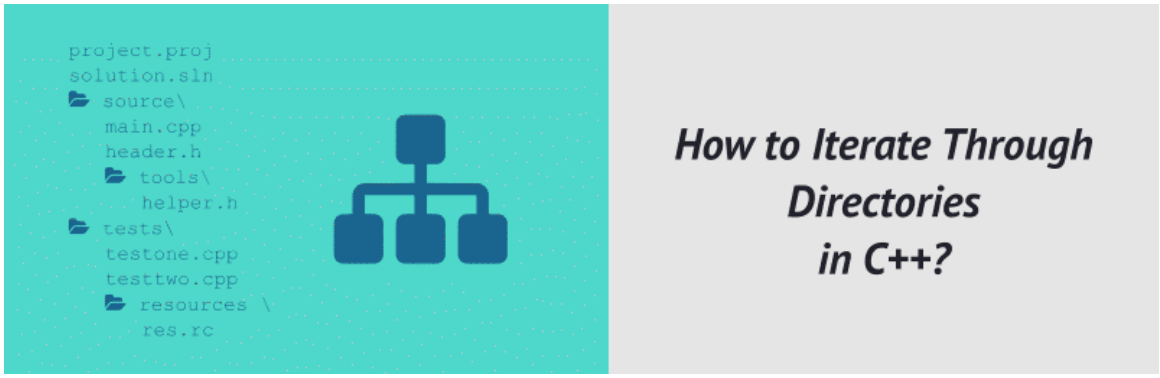
How to Iterate Through Directories in C++

Bartlomiej Filipek



How would you implement a function that searches for files with a given extension? For example, finding all text files? or *.cpp files? To code that solution you need a way to iterate through directories. Is that possible in C++ out of the box using the standard library? Let's see some techniques and new elements that C++17 added.

Intro

Let's assume you have the following directory structure:

```
books\
    cppbookA.txt
    cppbookA.pdf
    pythonbookX.txt
    pythonbookX.epub
    stlboob.txt
    stlbook.mobi
sources\
    licence.txt
    example1.cpp
    example2.cpp
```

How to filter all *.txt files from* `books\` *or* .cpp files from `sources\`?

The basic idea is to iterate through a directory and then check each entry if it's a regular file and if it has given extension.

Before C++17 there was no standard way of implementing the above solution. In the next few sections I'll show you a few possible APIs that are currently available, for example:

- in a POSIX system
- on Windows
- QT
- POCO
- BOOST

And then we'll move to C++17.

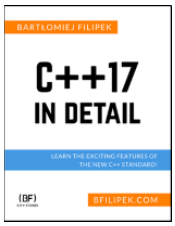
Let's go.

The Series

This article is part of my series about C++17 Library Utilities. Here's the list of the articles:

- Refactoring with `std::optional`
- Using `std::optional`
- Error handling and `std::optional`
- Everything You Need to Know About `std::variant` from C++17
- Everything You Need to Know About `std::any` from C++17
- `std::string_view` Performance and followup
- C++17 string searchers and followup

- Conversion utilities - about from_chars.
- How to get File Size in C++? and std:filesystem::file_size Advantages and Differences
- How To Iterate Through Directories this article!
- Working with the Path object from `std::filesystem` - soon!



Resources about C++17 STL:

- **C++17 In Detail** by Bartek!
- C++17 - The Complete Guide by Nicolai Josuttis
- C++ Fundamentals Including C++ 17 by Kate Gregory
- Practical C++14 and C++17 Features - by Giovanni Dicanio
- C++17 STL Cookbook by Jacek Galowicz

From C/POSIX

On Linux, using only the POSIX functionalities you can try with `dirent`:

```
#include <stdio.h>
#include <dirent.h>

int main(int argc, const char**argv) {
    struct dirent *entry = nullptr;
    DIR *dp = nullptr;

    dp = opendir(argc > 1 ? argv[1] : "/");
    if (dp != nullptr) {
        while ((entry = readdir(dp)))
            printf ("%s\n", entry->d_name);
    }

    closedir(dp);
    return 0;
}
```

Play with code @Coliru

As you see you have basic API and three functions to iterate over a directory:

- `opendir()` to initialise the search and find the first entry
- `readdir()` to find the next entry
- `closedir()` to finish the search

While iterating, you get `dirent entry` which is declared as:

```
struct dirent {
    ino_t          d_ino;       /* inode number */
    off_t          d_off;       /* offset to the next dirent */
    unsigned short d_reclen;    /* length of this record */
    unsigned char  d_type;      /* type of file; not supported
                                   by all file system types */
    char           d_name[256]; /* filename */
};
```

See more description here.

As you see, it's a low-level API, and probably not what you want in modern C++ :)

To filter all files with a specific file extension, you'd have to extract the extension from the filename which is just a character array.

For a recursive version you can try `ftw()` - "File Tree Walk" - see documentation here.

On Windows, WinApi

Windows is not a POSIX system, but `dirent` is available in MinGW or Cygwin implementations. I've found even a stand-alone

helper: https://github.com/tronkko/dirent

Still, if you can, you might want to use the native Windows API.

The canonical example on this platform uses `FindFirstFile`, `FindNextFile` and `FindClose` (in fact it's very similar to `dirent` approach).

```
WIN32_FIND_DATA FindFileData;
HANDLE hFind = FindFirstFile(/*path*/, &FindFileData);
if (hFind == INVALID_HANDLE_VALUE) {
    printf ("FindFirstFile failed (%d)\n", GetLastError());
    return;
}

do {
    if (ffd.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY)
        _tprintf(TEXT("  %s   <DIR>\n"), ffd.cFileName);
    else
        _tprintf(TEXT("  %s \n"), ffd.cFileName);
} while (FindNextFile(hFind, &ffd) != 0);

FindClose(hFind);
```

While iterating you have access to members of `WIN32_FIND_DATA` which exposes members like path type, filename (`CHAR cFileName[MAX_PATH];`), access time, file size and more.

To get file extension, you can try with PathFindExtension

read more here: Listing the Files in a Directory - Windows applications | Microsoft Docs

I hope you have now the basic understanding of how you can iterate through a directory. Yet, it's not modern C++ and those are mostly low-level system calls. How about something high level?

3rd Party Libraries

In this section, you'll see three 3rd party libs that encapsulate old native APIs and expose a much cleaner interface. They are also multiplatform, so the same code will work on Windows and Linux.

The first: QT

QT

QT is an extensive framework, so you have separate components to work with the filesystem. For directory iteration, there's `QDirIterator`.

Here's a basic example (from the documentation):

```
QDirIterator it("/etc", QDirIterator::Subdirectories);
while (it.hasNext()) {
    qDebug() << it.next();
}
```

Only three lines of code! What's more, you have RAII (there's no need to close the directory search as it's encapsulated in `QDirIterator`).

You can get the filename or full file info from the iterator.

Poco

In Poco, which is a cross-platform C++ framework for building network and internet-based applications, there's a component `DirectoryIterator`:

```
#include <Poco/DirectoryIterator.h>
...

for (DirectoryIterator it(path); it != DirectoryIterator{}; ++it) {

}
```

As you see we have a similar pattern as in QT, an iterator (forward iterator), that allows going through entries.

BOOST Filesystem

The last lib that I'll mention is Boost Filesystem, which is a powerful library and well recognised by the community.

Here's the main tutorial if you want to have a quick overview: boost.org: Boost Filesystem tutorial.

And the canonical example of directory iteration:

```
#include <boost/filesystem.hpp>
using namespace boost::filesystem;

for (directory_entry& entry : directory_iterator(inputPath))
    std::cout << entry.path() << '\n';
```

This time, you can also see an iterator that wraps all the low-level system calls. Each entry has a path that you can access.

Please notice that `directory_iterator` has also support for `begin` and `end` so that it can be used in range based for loop.

I mentioned Boost because it's a very well known and heavily used library, and also it was the foundation of the Filesystem TS that was published before C++17… and eventually, the committee merged it into the standard.

Summary of 3rd Party libs

As you see the interface is much cleaner and more helpful to use than native solutions. In a matter of a few lines of code, you can implement the task. Still, the main drawback is that you have to depend on the whole framework. For example link to all boost libs, or to the entire QT ecosystem.

Using C++17

So far you've seen several options that allow iterating through files. In all of the cases, you need to rely on the native API or third-party code. But Finally, since 2017 and C++17 you can depend only on the Standard Library!

This is possible through `std::filesystem` which was directly adopted from BOOST filesystem.

The code is similar to BOOST, have a look:

```
#include <filesystem>

namespace fs = std::filesystem;

const fs::path pathToShow{ argc >= 2 ? argv[1] : fs::current_path() };

for (const auto& entry : fs::directory_iterator(pathToShow)) {
    const auto filenameStr = entry.path().filename().string();
    if (entry.is_directory()) {
        std::cout << "dir:  " << filenameStr << '\n';
    }
    else if (entry.is_regular_file()) {
        std::cout << "file: " << filenameStr << '\n';
    }
    else
        std::cout << "??    " << filenameStr << '\n';
}
```

Play @Coliru

It's important to note (from cppreference):

> The iteration order is unspecified, except that each directory entry is visited only once. The special pathnames dot and dot-dot are skipped.

Recursive Version

The `directory_iterator` works only inside a single directory, but there's another class `recursive_directory_iterator` that allows iterating through the whole tree.

You can use `depth()` to check the current level of recursion. That might be helpful when you'd like to create a nicer output and add indentation:

```
for(auto itEntry = fs::recursive_directory_iterator(pathToShow);
         itEntry != fs::recursive_directory_iterator();
         ++itEntry ) {
    const auto filenameStr = iterEntry->path().filename().string();
    std::cout << std::setw(iterEntry.depth()*3) << "";
    std::cout << "dir:  " << filenameStr << '\n';
}
```

Play with the code @Coliru

You can also implement custom recursion and iterate with a regular iterator in a single directory.

For example:

```
void DisplayDirectoryTree(const fs::path& pathToScan, int level = 0) {
    for (const auto& entry : fs::directory_iterator(pathToScan)) {
        const auto filenameStr = entry.path().filename().string();
        if (entry.is_directory()) {
            std::cout << std::setw(level * 3) << "" << filenameStr << '\n';
            DisplayDirectoryTree(entry, level + 1);
        }
        else if (entry.is_regular_file()) {
            std::cout << std::setw(level * 3) << "" << filenameStr
                << ", size " << entry.file_size() << " bytes\n";
        }
        else
            std::cout << std::setw(level * 3) << "" << " [?]" << filenameStr << '\n';
    }
}
```

Play @Coliru

File Extensions

As for our main task - filtering files by extensions - it's straightforward!

C++17 exposes a path type and you can easily read its extension. just use: `path::extension()`.

For example:

```
std::filesystem::path("C:\\temp\\hello.txt").extension();
```

Compiler Support

on GCC and Clang Remember to add `-lstdc++fs` to link with the library.

Feature	GCC	Clang	MSVC
Filesystem	8.0	7.0	VS 2017 15.7

However, since GCC 5.3, Clang 3.9 and VS 2012 you could play with the experimental version - the TS implementation (include `<experimental/filesystem>`)

Summary

In this article, you've seen several ways ho to iterate through a directory in C++. Before C++17 you need to rely on some other libraries or system API, but now it's possible to use `std::filesystem::directory_iterator`.

I haven't shown the final code that iterates and then filters out the files by their extension. Can you implement it? Or maybe you'd like to share your experience with working with directories in C++. Let me know in comments!