Demo Example for Path Object

This lesson shows a demo code sample that we are going to discuss in the upcoming lessons. The code basically displays files in a given directory.

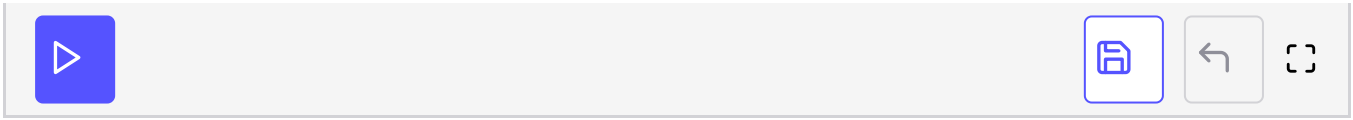
WE'LL COVER THE FOLLOWING

Demo

Instead of exploring the library piece by piece at the start, let's see a demo example: displaying basic information about all the files in a given directory (recursively). This should give you a high-level overview of how the library looks like.

```
#include <filesystem>
#include <iomanip>
#include <iostream>

namespace fs = std::filesystem;

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 " << fs::file_size(entry) << " bytes\n";
        }
        else
            std::cout << std::setw(level * 3) << "" << " [?]" << filenameStr << '\n';
    }
}
int main(int argc, char* argv[]) {
    try  {
        const fs::path pathToShow{ argc >= 2 ? argv[1] : fs::current_path() };
```

```
        std::cout << "listing files in the directory: "
                  << fs::absolute(pathToShow).string() << '\n';


        DisplayDirectoryTree(pathToShow);
    }
    catch (const fs::filesystem_error& err) {
        std::cerr << "filesystem error! " << err.what() << '\n';

    }
    catch (const std::exception& ex) {
        std::cerr << "general exception: " << ex.what() << '\n';
    }
}
```



Sample Outputs

We can run this program on a temp path `D:\testlist` and see the following output:

Running on Windows:

```
.\ListFiles.exe D:\testlist\
listing files in the directory: D:\testlist\
abc.txt, size 357 bytes
def.txt, size 430 bytes
ghi.txt, size 190 bytes
dirTemp
   jkl.txt, size 162 bytes
   mno.txt, size 1728 bytes
tempDir
   abc.txt, size 174 bytes
   def.txt, size 163 bytes
   tempInner
      abc.txt, size 144 bytes
      mno.txt, size 1728 bytes
      xyz.txt, size 3168 bytes
```

The application lists files recursively, and with each indentation, you can see that we enter a new directory.

Running on a Linux (Ubuntu 18.04 on WSL):

```
fenbf@FEN-NODE:/mnt/f/wsl$ ./list_files.out testList/
listing files in the directory: /mnt/f/wsl/testList/
```

```
a.txt, size 965 bytes
b.txt, size 1667 bytes

c.txt, size 1394 bytes
d.txt, size 1408 bytes
directoryTemp
   a.txt, size 1165 bytes
   b.txt, size 1601 bytes
tempDir
   a.txt, size 1502 bytes
   b.txt, size 1549 bytes
   x.txt, size 1487 bytes
```

Explanation

Let's now examine the core elements of this demo.

To work with the library, we have to include relevant headers. For the filesystem library it's:

```
#include <filesystem>
```

All the types, functions and names live in the `std::filesystem` namespace. For convenience it's useful to make a namespace alias:

```
namespace fs = std::filesystem;
```

And now we can refer to the names as `fs::path` rather than `std::filesystem::path`.

Let's start with the `main()` function where the logic of the application starts.

The program takes a single optional argument from the command line. If it's empty then we use the current system path:

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

`pathToShow` can be created from strings - from `argv[1]` if available. If not, then we take `current_path()` which is a helper free function that returns the current system path.

In the next two lines - line 26 and 27 - we display what the starting path for the iteration. The `absolute()` function "expands" the input path and it

converts it from a relative form into the absolute form if required.

The core part of the application is the `DisplayDirectoryTree()` function.

Inside, we use `directory_iterator` to examine the directory and find the files or more directories.

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

Each loop iteration yields another `directory_entry` that we need to check. We can decide if we should call the function recursively (when `entry.is_directory()` is `true`) or just show some basic information if it's a regular file.

As you see we have access to many methods of path and directory entry. For example, we use `filename` to return only the filename part of the path so we can display "tree" structure. We also invoke `fs::file_size` to query the size of the file.

After a little demo let's now have a closer look at `filesystem::path`, `filesystem::directory_entry`, supporting non-member functions and error handling.