Non-member functions

Apart from the classes, C++ has pre-defined functions to help us fully use filesystem library.

WE'LL COVER THE FOLLOWING

- Read and set the last write time of a file
- Getting space information to the filesystem

Many non-member functions exist for manipulating the filesystem.

Non-member functions	Description
`absolute`	Composes an absolute path.
`canonical` and `weakly_canonical`	Composes a canonical path.
`relative` and `proximate`	Composes a relative path.
`copy`	Copies files or directories.
`copy_file`	Copies file contents.
`copy_symlink`	Copies a symbolic link.
`create_directory` and `create_directories`	Creates a new directory.
`create_hard_link`	Creates a hard link.
`create_symlink` and `create_directory_symlink`	Creates a symbolic link.

`create_directory_symlink`	
`current_path`	Returns the current working directory.
`exists`	Checks if path refers to an existing file.
`equivalent`	Checks if two paths refer to the same file.
`file_size`	Returns the size of the file.
`hard_link_count`	Returns the number of hard links to a file.
`last_write_time`	Gets and sets the time of the last file modification.
`permissions`	Modifies the file access permissions.
`read_symlink`	Gets the target of the symbolic link.
`remove`	Removes a file or an empty directory.
`remove_all`	Removes a file or a directory with all its content recursively.
`rename`	Moves or renames a file or directory.
`resize_file`	Changes the size of a file by truncation.
`space`	Returns the free space on the

	filesystem.
`status`	Determines the file attributes.
`symlink_status`	Determines the file attributes and checks the symlink target.
`temp_directory_path`	Returns a directory for temporary files.

The non-member functions for manipulating the filesystem

Read and set the last write time of a file

Thanks to the global function `std::filesystem::last_write_time`, you can read and set the last write time of a file. Here is an example, based on the `last_write_time` example from en.cppreference.com.

```
#include <iostream>
#include <chrono>
#include <fstream>
#include <filesystem>

namespace fs = std::filesystem;
using namespace std::chrono_literals;

int main(){

    fs::path path = fs::current_path() / "rainer.txt";
    std::ofstream(path.c_str());
    auto ftime = fs::last_write_time(path);

    std::time_t cftime = std::chrono::system_clock::to_time_t(ftime);
    std::cout << "Write time on server " << std::asctime(std::localtime(&cftime));
    std::cout << "Write time on server " << std::asctime(std::gmtime(&cftime)) << std::endl;

    fs::last_write_time(path, ftime + 2h);
    ftime = fs::last_write_time(path);

    cftime = std::chrono::system_clock::to_time_t(ftime);
    std::cout << "Local time on client " << std::asctime(std::localtime(&cftime)) << std::enc

    fs::remove(path);

}
```

Write time of a file

Line (1) gives the write time of the newly created file. You can use `ftime` in (2) to initialise `std::chrono::system_clock`. `ftime` is of type

`std::filesystem::file_time_type` which is in this case an alias for `std::chrono::system_clock`; therefore, you can initialise `std::localtime` in (3) and present the calendar time in a textual representation. If you use `std::gmtime` (4) instead of `std::localtime`, nothing will change. This puzzled me because the Coordinated Universal Time (UTC) differs 2 hours from the local time in Germany. That's due to the server for the online-compiler on en.cppreference.com. UTS and local time are set to the same time on the server.

Here is the output of the program. I moved the write time of the file 2 hours to the future (5) and read it back from the filesystem (6). This adjusts the time so it corresponds to the local time in Germany.

```
Write time on server Tue Oct 10 06:28:04 2017
Write time on server Tue Oct 10 06:28:04 2017

Local time on client Tue Oct 10 08:28:04 2017
```

Getting space information to the filesystem

The global function `std::filesystem::space` returns a `std::filesystem::space_info` object that has three members: capacity, free, and available.

- *capacity*: total size of the filesystem
- *free*: free space on the filesystem
- *available*: free space to a non-privileged process (equal or less than free)

All sizes are in bytes.

The output of the following program is from cppreference.com. All paths I tried were on the same filesystem; therefore, I always get the same answer.

```
#include <fstream>
```

```cpp
#include <iostream>
#include <string>
#include <filesystem>

namespace fs = std::filesystem;

int main(){

    std::cout << "Current path: " << fs::current_path() << std::endl;

    std::string dir= "sandbox/a/b";
    fs::create_directories(dir);

    std::ofstream("sandbox/file1.txt");
    fs::path symPath= fs::current_path() /=  "sandbox";
    symPath /= "syma";
    fs::create_symlink("a", "symPath");

    std::cout << "fs::is_directory(dir): " << fs::is_directory(dir) << std::endl;
    std::cout << "fs::exists(symPath): "  << fs::exists(symPath) << std::endl;
    std::cout << "fs::symlink(symPath): " << fs::is_symlink(symPath) << std::endl;


    for(auto& p: fs::recursive_directory_iterator("sandbox"))
        std::cout << p << std::endl;
    fs::remove_all("sandbox");

}
```

Space information

```
         Capacity       Free      Available
/    42140499968   18342744064   17054289920
usr  42140499968   18342744064   17054289920
```