File I/O operations

C++ for Developers

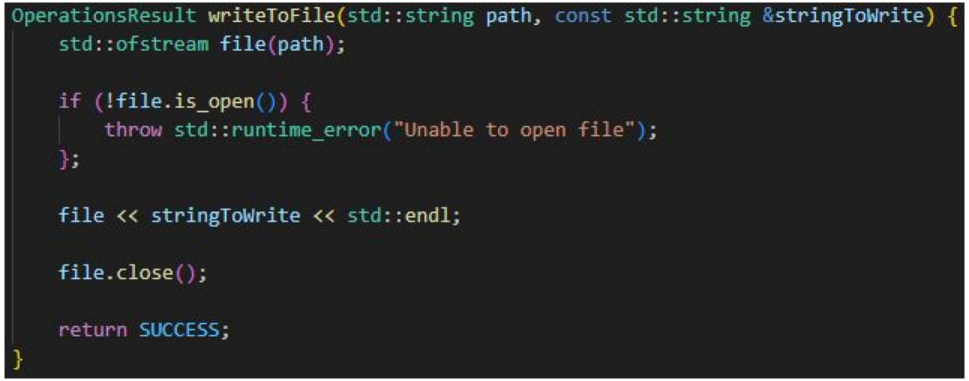
Operations

File reading operations are included in <fstream>, and may need to be combined with <string>.

- Writing to a file
- Appending to a file
- Reading from a file

Writing to a file

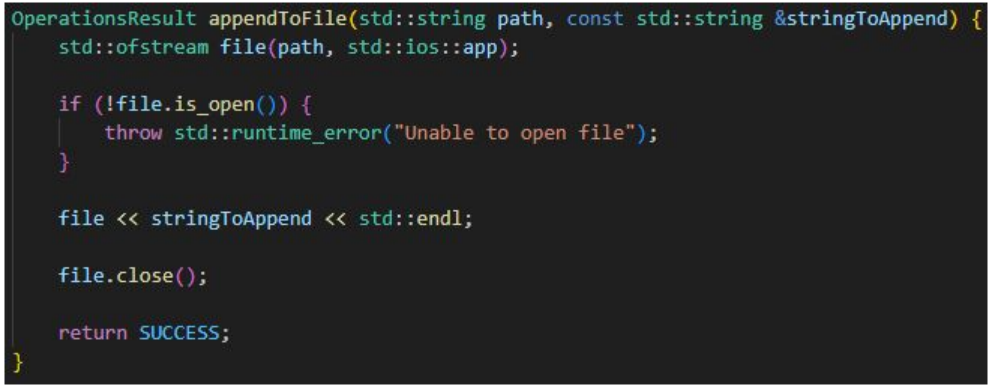
- Open file
- Check if file is open
- Write to file
- Close file
- return



```cpp
OperationsResult writeToFile(std::string path, const std::string &stringToWrite) {
    std::ofstream file(path);

    if (!file.is_open()) {
        throw std::runtime_error("Unable to open file");
    };

    file << stringToWrite << std::endl;

    file.close();

    return SUCCESS;
}
```

Appending to a file

- Open file with std::ios::app mode
- Check if file is open
- Write to file
- Close file
- return



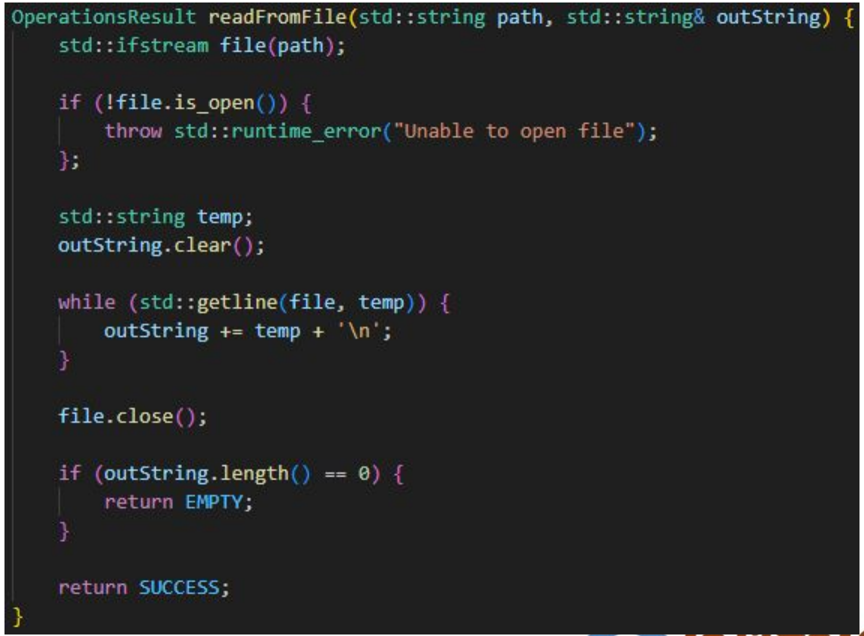
```cpp
OperationsResult appendToFile(std::string path, const std::string &stringToAppend) {
    std::ofstream file(path, std::ios::app);

    if (!file.is_open()) {
        throw std::runtime_error("Unable to open file");
    }

    file << stringToAppend << std::endl;

    file.close();

    return SUCCESS;
}
```

Difference between write and append?

Reading from a file

- Open file
- Check if file is open
- Read from file while there is a new line
- Close file
- return

Reminder: std::getline() is defined in <string>



```cpp
OperationsResult readFromFile(std::string path, std::string& outString) {
    std::ifstream file(path);

    if (!file.is_open()) {
        throw std::runtime_error("Unable to open file");
    };

    std::string temp;
    outString.clear();

    while (std::getline(file, temp)) {
        outString += temp + '\n';
    }

    file.close();

    if (outString.length() == 0) {
        return EMPTY;
    }

    return SUCCESS;
}
```

Link to I/O operations with files.

Find my example file here:

https://github.com/lafftale1999/cpp_for_developers/blob/main/week_2/2_file_input_output.cpp

Typical file formats

C++ for Developers

CSV format

```
Name,Age,Grade
John Doe,20,A
John Doe,20,A
John Doe,20,A
John Doe,20,A
```

Strengths:

- Super simple, lightweight.
- Easy to open in spreadsheets.
- Great for tabular data (rows & columns).

When to use:

- Import/export between spreadsheets and databases.
- Datasets for machine learning or statistical analysis.

JSON

```
[
	{
		"name": "John Doe",
		"age": 20,
		"grade": "A"
	},
	{
		"name": "John Doe",
		"age": 20,
		"grade": "A"
	}
]
```

Strengths:
- Human- and machine-readable
- Nested, hierarchical data
- Widely used

When to use:
- API / Web services
- Storing configuration files
- Mobile apps, web apps, server communication

lafftale for developers

XML

```
<Students>
    <Student>
        <Name>John Doe</Name>
        <Age>20</Age>
        <Grade>A</Grade>
    </Student>
    <Student>
        <Name>John Doe</Name>
        <Age>20</Age>
        <Grade>A</Grade>
   </Student>
</Students
```

Strengths:

- Structured
- Verbose

When to use:

- Legacy enterprise software

Exercise

C++ for Developers

Contacts

Create a program that prompts the user to enter a first and last name, phone number and address. Then save this to a .txt-file.

- The contact data type should be a struct.
- The user should be able to:
  - Add a contact, Show all contacts or Exit the program. The program runs until the user exits it manually.
- The program should save it in CSV form.

It is enough to read it from one file. I recommend trying to parse the JSON format for learning.

lafftale
for developers