Learning Objectives: Writing

- **Demonstrate how to open a file and write to it using `ofstream`**
- **Explain what happens when you write to a file that does not exist**
- **Demonstrate how to write multiline strings to a file**
- **Differentiate between input, output, and append modes**

Writing to a File

Writing to a File

When writing to a file, you'll want to use `ofstream` instead of `ifstream`. Like before, create your string path, open the file, and check for whether it can be opened successfully.

```
string path = "student/text/practice1.txt";

try {
  ofstream file;
  file.open(path);
  if (!file) {
    throw runtime_error("File failed to open.");
  }
}

catch (exception& e) { //catch error
  cerr << e.what() << endl;
}
```

If the file is successfully opened, you can start writing to the file using the insertion operator `<<` followed by what you want to write in double quotes `""`. Remember to close the file, and if you want, you can print a message at the end telling the user that that the file was successfully written to.

```
string path = "student/text/practice1.txt";

try {
  ofstream file;
  file.open(path);
  if (!file) {
    throw runtime_error("File failed to open.");
  }
  file << "Hello there";
  file.close();
  cerr << "Finished writing to file.";
}

catch (exception& e) { //catch error
  cerr << e.what() << endl;
}
```

Click on the link to open the file and see what was wrttien: Open practice1.txt

challenge

Try these variations:

Open the `practice1.txt` file after each change to see what gets written.
* Change `"Hello there"` to `"Goodbye"`.
* Change `"Goodbye"` in your current code to `""`.
* Open `student` in the sidebar on the left. Open the `text` folder and right-click on `practice1.txt`. Select `Delete...` and `YES` to delete the file and then run the program again.

▼ **Why is there no error message?**
If you tell C++ to create an `ofstream` file, it will automatically create that file if one does not exist. Or it will overwrite the existing file with a new one. This is why you do not see an error message. You will only see an error message if for some reason the system is not able to create the file at all.

Open practice1.txt

Reading a Written File

If you want to read from the file after it was written to, you can create an `ifstream` to read from the file.

```
string path = "student/text/practice1.txt";

try {
  ofstream file;
  file.open(path);
  if (!file) {
    throw runtime_error("File failed to open.");
  }
  file << "Hello there";
  file.close();

  ifstream stream;
  string read;
  stream.open(path);
  while (getline(stream, read)) {
    cout << read << endl;
  }
  stream.close();
}

catch (exception& e) { //catch error
  cerr << e.what() << endl;
}
```

challenge

What happens if you:

- Change `"Hello there"` to `"Hi!"`.
- Add `file << " My name is AI." << endl;` to the line after `file << "Hi!";`?

Multiline Strings

Multiline Strings

In addition to being able to write and output string literals (e.g. `file << "Hi!";`), we can also write and output the content of variables (e.g. `file << var;`). Let's tweak the code from the previous page to write multiple messages to a text file called `practice2.txt`. We'll create three string variables, `text1`, `text2`, and `text3`. The first message will go into a string variable `text1`, the second will go into `text2`, and the third will go into `text3`.

```
string path = "student/text/practice2.txt";

try {
  ofstream file;
  file.open(path);
  if (!file) {
    throw runtime_error("File failed to open.");
  }
  string text1 = "Hello, ";
  string text2 = "your balance is: ";
  string text3 = "12.34";
  file << text1 + text2 + text3;
  file.close();

  ifstream stream;
  string read;
  stream.open(path);
  while (getline(stream, read)) {
    cout << read << endl;
  }
  stream.close();
}

catch (exception& e) { //catch error
  cerr << e.what() << endl;
}
```

Open practice2.txt

challenge

What happens if you:

- Change `string text3 = "12.34";` to `double text3 = 12.34;`?
- Change `file << text1 + text2 + text3;` to `file << text1 + text2 << text3;`?
- Split the code `file << text1 + text2 << text3;` into two lines:

```
file << text1 + text2 << endl;
file << text3;
```

- Change `file << text1 + text2 << endl;` to `file << text1 + text2 << '\n';`?
- Change `file << text1 + text2 << '\n';` to `file << "Hello, your balance is:\n12.34";` and remove `file << text3;`?

Open practice2.txt

Notice how you can also write the content of other types of data (double, int, etc.) to a file. You are not restricted to just strings. Also, there are multiple ways to write the same kind of content to a file.

Appending to a File

Appending to a File

You may have noticed that every time a file is opened using an `ofstream` object, a new file is always created, even if one already exists (the system just overwrites the existing file). If you want to add to an existing file, you have to tell C++ to open the file in **append** mode. Let's look at the code below and `TRY IT`.

```
string path = "student/text/practice3.txt";

try {
  ofstream file;
  file.open(path, ios::app); //open file in append mode
  if (!file) {
    throw runtime_error("File failed to open.");
  }
  string text = "Adding to the file.";
  file << text;
  file.close();

  ifstream stream;
  string read;
  stream.open(path);
  while (getline(stream, read)) {
    cout << read << endl;
  }
  stream.close();
}

catch (exception& e) { //catch error
  cerr << e.what() << endl;
}
```

Open practice3.txt

Since there is no `practice3.txt` file at the start of the program, C++ will create one. However, try running the code again and see what happens.

Open practice3.txt

You'll notice that the output `Adding to the file.` shows up twice in the file. This happens because we have included the tag `ios::app` as a second parameter when we opened up the file `practice3.txt`. By default, an `ofstream` object has the flag `ios::out` as a second parameter, which causes the file to always get overwritten. By changing the parameter to `ios::app`, we're telling the system to add to the file instead of overwriting it.

challenge

What happens if you:

- Change `ios::app` in the code to `ios::out`?
- Change `ios::out` to `ios::in`?
- Change `file << text;` to `file << text << text;`?
- Change `file << text << text;` to `file << "Hello";`?
- Change `ios::in` back to `ios::app`?

Open practice3.txt

If you follow through the challenges above, you'll notice that when the flag is set to input mode `ios::in`, the system will overwrite the content without creating a new file or overwriting the old one. On the other hand, `ios::app` will add to the end of the existing content. Lastly, `ios::out` creates a completely new file and writes to it.