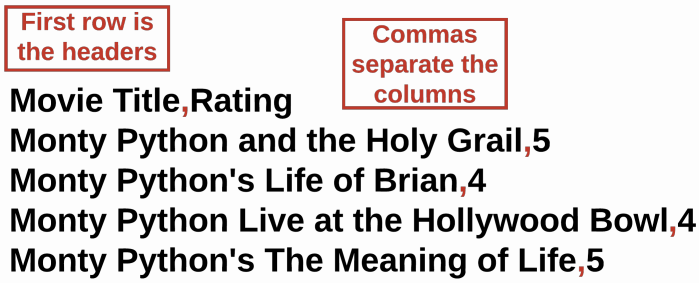
Learning Objectives: CSV

- **Define CSV**
- **Read data from a CSV file**
- **Iterate over a CSV file**
- **Print data from a CSV file with formatting**
- **Write data to a CSV file**

CSV Files

CSV Files

C++ can work with files besides just text files. Comma Separated Value (CSV) files are an example of a commonly used file format for storing data. CSV files are similar to a spreadsheet in that data is stored in rows and columns. Each row of data is on its own line in the file, and commas are used to indicate a new column. Here is an example of a CSV file.



Month Python CSV

You can read a CSV file in the same way you read a text file. First create an `ifstream` object and then open the CSV file using it.

```
string path = "student/csv/monty_python_movies.csv";

try {
  ifstream file;
  string read;
  file.open(path);
  if (!file) {
    throw runtime_error("File failed to open.");
  }
  while (getline(file, read, ',')) {
    cout << read + ' ';
  }
  file.close();
}

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

To iterate through the CSV file, we use `while (getline(file, read, ','))`. Since CSV files contain commas that separate information, we set the delimiter to `','`. Then we print the content by using `cout << read + ' '`. We add a space to separate the tokens from each other since it is not apparent that the information is tokenized from using just `cout << read;`.

challenge

Try this variation:

- Change `cout << read + ' ';` to `cout << read << endl;`?

By using `cout << read << endl;` you can clearly see each token line by line. Depending on your preference, you can choose to arrange the tokens in a variety of different formats.

Ignoring the Header

The first row of a CSV file is helpful because the header values provide context for the data. However, the first row is not useful if you want to know how many rows of data there are, or to calculate the avg value, etc. Here, you can also use the `ignore()` function to skip a specific number of characters.

```
string path = "student/csv/monty_python_movies.csv";

try {
  ifstream file;
  string read;
  file.open(path);
  if (!file) {
    throw runtime_error("File failed to open.");
  }
  file.ignore(19); //Ignore the first 19 characters from index
        0-18
  while (getline(file, read, ',')) {
    cout << read << endl;
  }
  file.close();
}

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

challenge

Try this variation:

- Change `file.ignore(19);` to `file.ignore(53);`?
- Change `file.ignore(53);` to `file.ignore(500);`?
- Change `file.ignore(500);` to `file.ignore(500, '\n');`?
- Add another `file.ignore(500, '\n');` below the first `file.ignore(500, '\n');`?

The `file.ignore(500, '\n');` command tells the system to skip the first 500 characters **or** up through the newline character `\n`. Since there are fewer than 500 characters, the system will skip everything up through the first occurrence of the newline. You can add additional ignore commands to ignore more lines of data if needed.

Printing CSV Data

Printing CSV Data

Iterating over the CSV file and printing each line does not produce visually pleasing output. The code below produces three columns of data, but there is no consistency in the spacing between columns. When printed, the data looks very disorganized and difficult to read.

```
string path = "student/csv/homeruns.csv";

try {
  ifstream file;
  string read;
  file.open(path);
  if (!file) {
    throw runtime_error("File failed to open.");
  }
  while (getline(file, read, ',')) {
    cout << read + " ";
  }
  file.close();
}

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

To better organize our CSV data, we can store the data into a vector and then format and print elements in a way that looks more organized.

```
string path = "student/csv/homeruns.csv";
vector<string> data;

try {
  ifstream file;
  string read;
  file.open(path);
  if (!file) {
    throw runtime_error("File failed to open.");
  }
  while (getline(file, read)) {
    stringstream ss(read);
    while (getline(ss, read, ',')) {
      data.push_back(read);
    }
  }
  file.close();
  for (int i = 0; i < data.size(); i++) {
    if (i % 3 == 0) {
      cout << setw(20) << left << data.at(i);
    }
    else if (i % 3 == 1) {
      cout << setw(15) << left << data.at(i);
    }
    else {
      cout << data.at(i) << endl;
    }
  }
}

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

To organize our data, we use conditionals to split our elements into three columns. `if (i % 3 == 0)` refers to the elements in the first column, `else if (i % 3 == 1)` refers to the second column, and `else` refers to the third. We use the `setw()` function to provide padding for our elements. For example, `setw(20)` means that the system will reserve 20 characters for the elements. If the element does not take up 20 characters, then white spaces will occupy those spaces. To use `setw()`, you'll need `#include <iomanip>` in the header of your file. The `left` tag forces the element to be aligned to the left side.

challenge

Try this variation:

- Change `setw(20)` to `setw(16)`?
- Change `setw(15)` to `setw(10)`?
- Delete the second `left` tag?
- Delete the first `left` tag?

  ▼ **Note**

  By default, standard streams are set to `right`. This is why deleting all of the `left` tags will effectively align the streams back to the right side. Additionally, you only have to set the `left` tag once for **all** streams that follow to align left as well.

Notice how the last column `Active Player` is not formatted and is therefore unaffected by the changes.

important

IMPORTANT

The order or placement of where you use `left` and `setw()` can affect all streams that follow. So it's important to keep track of the changes that take place as you print. For example outputting

Writing to CSV Files

Writing to a CSV File

Writing to a CSV files is similar to writing to a text file. First create an `ofstream` object to open the CSV file. Then you can write to the CSV file using the insertion operator <<. To read the CSV file, you can use the same syntax that was previously shown.

```
string path = "student/csv/writepractice.csv";

try {
  ofstream file;
  file.open(path);
  if (!file) {
    throw runtime_error("File failed to open.");
  }
  file << "Greeting,Language" << endl;
  file << "Hello,English" << endl;
  file << "Bonjour,French" << endl;
  file << "Hola,Spanish";
  file.close();

  ifstream file2;
  string read;
  file2.open(path);
  while (getline(file2, read, ',')) {
    cout << read + ' ';
  }
  file2.close();
}

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

Click to open CSV file: writepractice.csv

To organize the CSV data, you can add each token into a vector and then use conditionals to format and print the data like before.

```
string path = "student/csv/writepractice.csv";
vector<string> data;

try {
  ifstream file;
  string read;
  file.open(path);
  if (!file) {
    throw runtime_error("File failed to open.");
  }
  while (getline(file, read)) {
    stringstream ss(read);
    while (getline(ss, read, ',')) {
      data.push_back(read);
    }
  }
  file.close();
  for (int i = 0; i < data.size(); i++) {
    if (i % 2 == 0) {
      cout << setw(15) << left << data.at(i);
    }
    else {
      cout << data.at(i) << endl;
    }
  }
}

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