COP3503

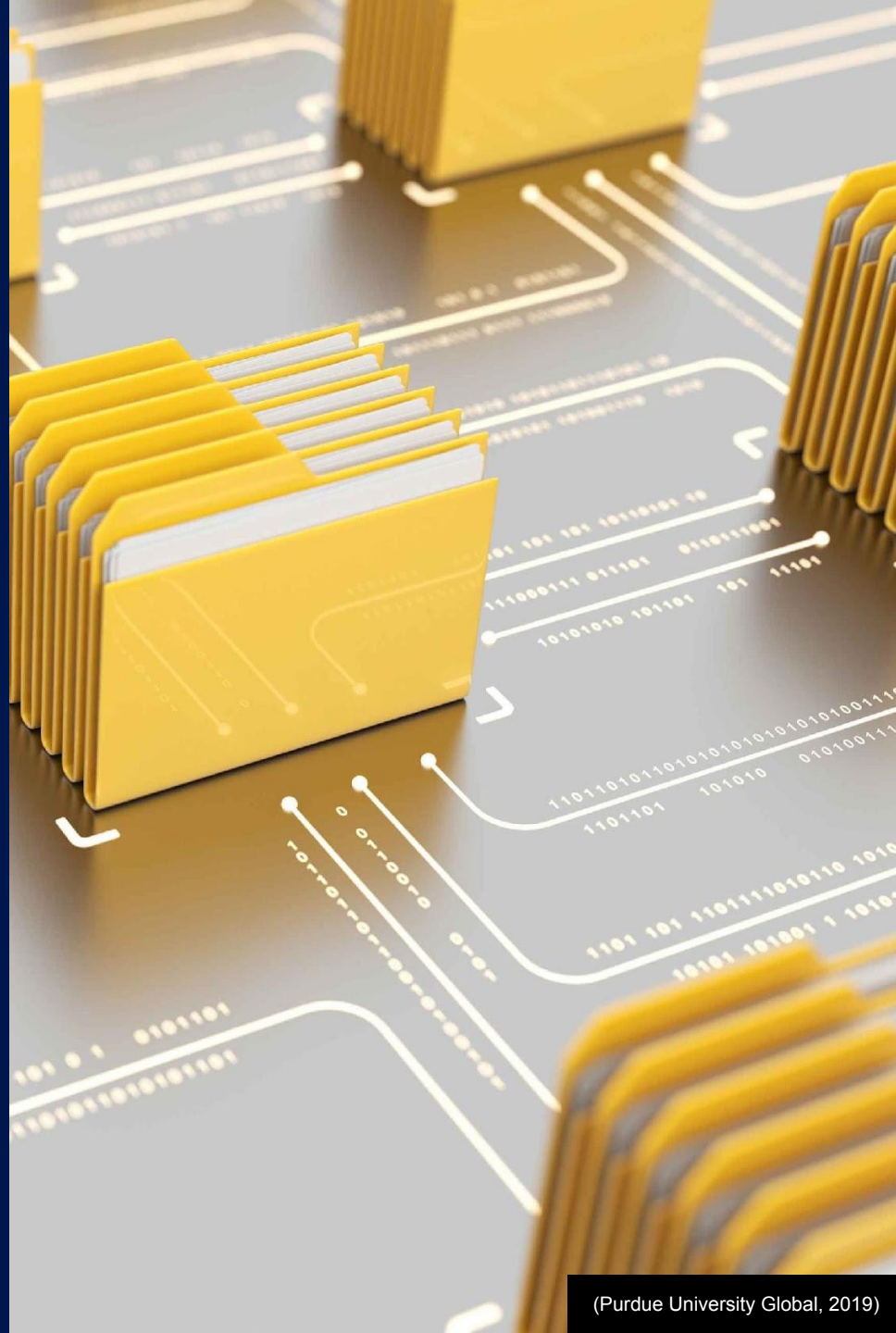
File I/O

File Input and Output (I/O)

- Virtually all data in computer applications is read from/stored in files.
  - Even “cloud” data is just files stored on someone else’s computer (Google, Microsoft, Apple, etc.).
- All files are made up of **collections of basic data types** – ints, floats, characters, etc.
  - Specific types of files (Word doc, Excel spreadsheet, etc.) are just collections of data.
- Data can be represented in **text** or **binary**.
  - Each with their own advantages/drawbacks

File Structure

- All files have **some** structure—some order to the data.
  - Even if you don't know what it is
- Some files are very simple and straightforward, others… not so much.
- Understanding that structure allows you do something with that file (read it, edit it, create new files of that type).
  - Think of **File->Save** or **File->Open** in a program.

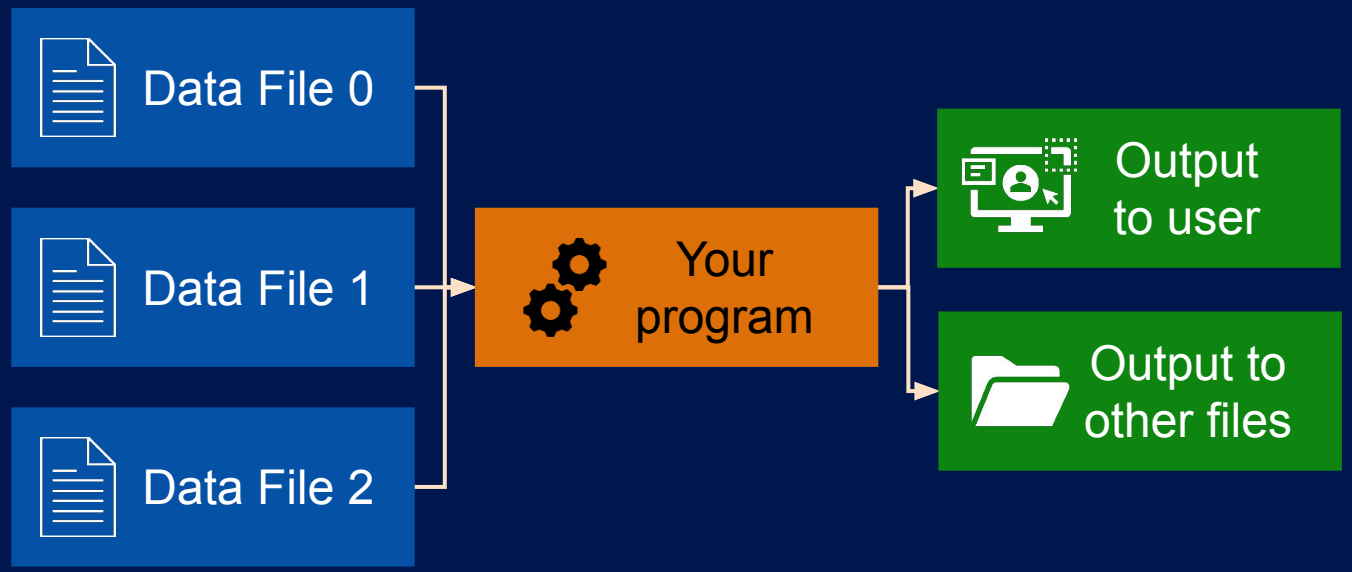


(Purdue University Global, 2019)

Why Use Files for Data?

- Simple applications can have all of their data "**hard-coded**".
  - Names of variables, numbers of loop iterations, etc. are **stored in source code**
  - Written by a programmer

- Many applications are **data-driven**.
  - Uses **external** sources of information to define behavior



Scenario: Address Book / Contact List

- Do you store all of the information for a contact list in source files?

No, that would be absurd!

- How many entries should there be?
- How would you possibly know in advance what those are?
- A program like that should allow for adding/deleting entries.

```
ContactList contacts;
contacts.AddEntry("Batman", "42 Batcave Lane", "555-BATS");
contacts.AddEntry("Superman", "86 Fortress o' Solitude Way", "555-2263");
contacts.Display();
// etc...
```

```
// A better alternative!
// Read some data from a file…
contacts.AddEntry(nameFromFile, addressFromFile, phoneNumberFromFile);
```

You, the programmer, don't have to know what these values are.

File Basics

- Every file has three things in common:
  1. **Directory**
     In which folder/directory is this file located?
  2. **Filename**
     The name of the file
  3. **Extension (optional)**
     What **type** of file is this?
     - Typically hints at: how is this file's data structured?
     - Also used by your operating system to decide what to do with the file
- Combine all three together for the full **path** of the file.

Complete File Path

```
D:\Documents\PG2\Lecture\File IO.pptx
```

Directory | Filename | Extension

```
C:\ThisIs_My_OddAnd.Long.File_nAmE.xyzqmwe
```

Directory | Filename | Extension

```
usr/local/bin/someFile.bbq
```

Directory | Filename | Extension

- Multiple folders within a directory are separated by **slashes**.
  - Forward or backward? Depends… more on this later
- If you're using **backslashes** in code, you may need to use the **escape sequence "\\"**.

```
string path = "D:\\Documents\\PG2\\Lecture\\FileIO.ppt";
```

Escape Sequences

Characters preceded by a backslash

There are others, but these are some of the most common.

- `\n` **Newline character**: This moves the output cursor to a new line
- `\t` **Tab character**: Prints a tab character, or moves the cursor to the next tab stop
- `\0` **NUL-terminator**: Terminates a string
- `\'` **Single-quote**: Indicates the apostrophe should be used as a character by itself
- `\"` **Double-quote**: Indicates the double-quote character should be used as a character by itself
- `\\` **Backslash**: Indicates backslash should be used as a character, not the start of an escape sequence

Why Use Escape Sequences?

- They might be helpful, or they might be required.
- Take the single quote character (the apostrophe): '

It can't be both an indicator of a character, and a character itself. (How do you differentiate?)

```
char letter = 'q'; //indicate a character, q
char punctuation = ''';
```

Error #1: quoted string should contain at least one character

Error #2: missing closing quote (for the 3rd character)

- So… how to store a single-quote character?

```
char correct = '\'';
```

`\'` treats that as the character itself, not a part of C++ syntax.

```
// Same for double-quotes
string quote = "He said \"Good morning\" to his neighbor.";
cout << quote << '\n'; // newline
```

```
He said "Good morning" to his neighbor.
Press any key to continue . . .
```

Double-Backslash \\ Escape Sequences

```
string wrongPath = "D:\Documents\PG2\Lecture\FileIO.ppt";
```

- What you **might** get:
  - (Some compilers may treat these as warnings, some as errors)

```
C4129   'D': unrecognized character escape sequence
C4129   'P': unrecognized character escape sequence
C4129   'L': unrecognized character escape sequence
C4129   'F': unrecognized character escape sequence
```

- If you tried printing this:

```
Microsoft Visual Studio Debug Console
D:DocumentsPG2LectureFileIO.ppt
```

```
string rightPath = "D:\\Documents\\PG2\\Lecture\\FileIO.ppt";
```

```
Microsoft Visual Studio Debug Console
D:\Documents\PG2\Lecture\FileIO.ppt
```

Backslash? Forward Slash?

- Depends on the OS (and maybe a program)
- Linux/MacOS use **forward slashes**.
- Windows uses **backslashes**, but forward tends to work everywhere as well.
- Some programs/OS may recognize “wrong” slashes and convert.
- Others will just yell at you until you do it right.

In this class, **use forward slashes** for every file path and you’ll be fine. That will work on MacOS, Linux, Windows, no problem!

\

/

Absolute Paths vs Relative Paths

Paths can be written in two ways.

- An **absolute path** is a fixed, unchanging path.
  - Should be avoided unless you **know** the location will be there (maybe a program running only on your computer?)

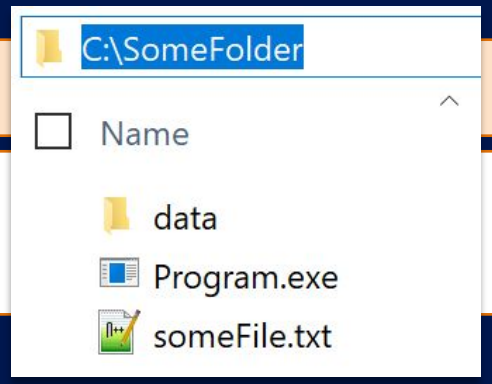
```
"C:/ThisFolder/SubFolder/someFile.txt"
"F:/User/Documents/data/results.xyz"
"K:/Unicorns/Lab1.cpp"
```

What if a program uses one of these, and you don't have that folder?

- A **relative path** is relative to some other location (and a better alternative!)
  - Relative to the executable (by default), or from some other starting point

You write this:

```
"someFile.txt"
"data/results.xyz"
```



Your program **uses** it like this:

```
"C:/SomeFolder/someFile.txt"
"C:/SomeFolder/data/results.xyz"
```

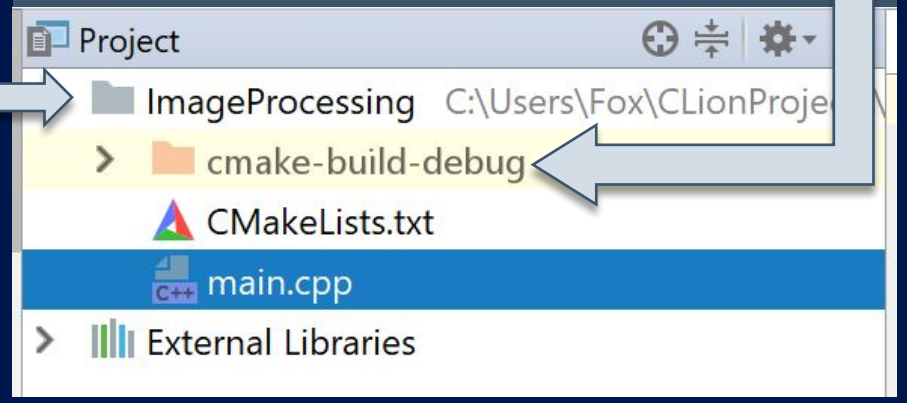
Where Does My IDE Read/Write Files?

- Depends on the IDE (these are default examples):
  - **Visual Studio**
    Looks in the same folder as your source code (right-click on your project in Solution Explorer, then select “Open Folder in File Explorer”)
  - **Clion**
    External files are read from the **`cmake-build-debug`** directory

External files (**and** the executable) are in **this** folder.

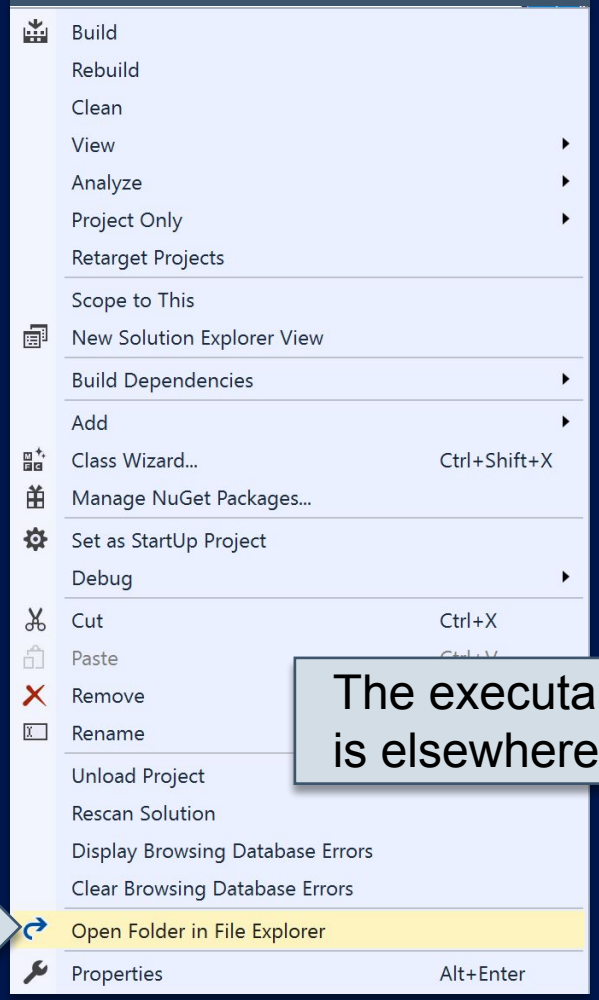
Your code is in this folder.

CLion



Your code **and** external files are in this folder (by default).

Visual Studio



The executable is elsewhere.

Should You Change Your Code to Adapt to Your IDE?

- No!
- Never!
- Really, don't!
- Change your IDE settings or move directories/files around if needed—work around proper code!
- Your code should work in as many situations as possible.

Image code that looks like this…

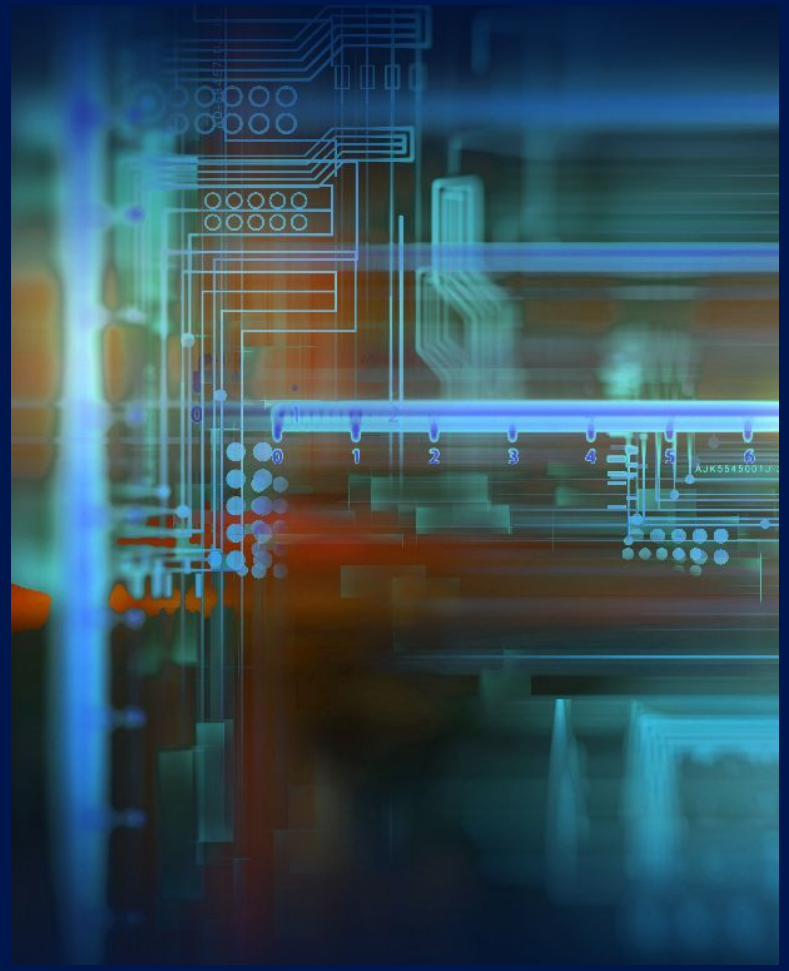
```
// uncomment if this is Sarah's machine
//file.open("C:/Sarah/Documents/file.txt"
// for Bob's machine
file.open("Q:/MagicalStuff/Unicorns/file.txt"
```

```
if (using Visual Studio)
    file.open("../data/file.txt");
else if (using CLion)
    file.open("someFile.txt");
```

Recap

- Programs typically revolve around **data** (lots of it!).
- Most of our **data is stored in files**.
- A **data-driven program** reads and writes those files.
  - We want to avoid “hard-coding” data in our source code files.
- Files have **paths** that contain information about what and where they are.
- Paths can be **absolute** (fixed) or **relative** to something else (typically an executable).
  - Relative paths are preferable, and work in more situations.
- IDEs may muck-up paths a little bit, but they can be adjusted to fit our code.

Conclusion



Placeholder for the instructor’s welcome message. Video team, please insert the instructor’s video here.

Thank you for watching.

References

Purdue University Global. (2019). Graphic of computer folders [Online Image]. Purdue University Global. https://www.purdueglobal.edu/blog/online-learning/manage-organize-computer-files/