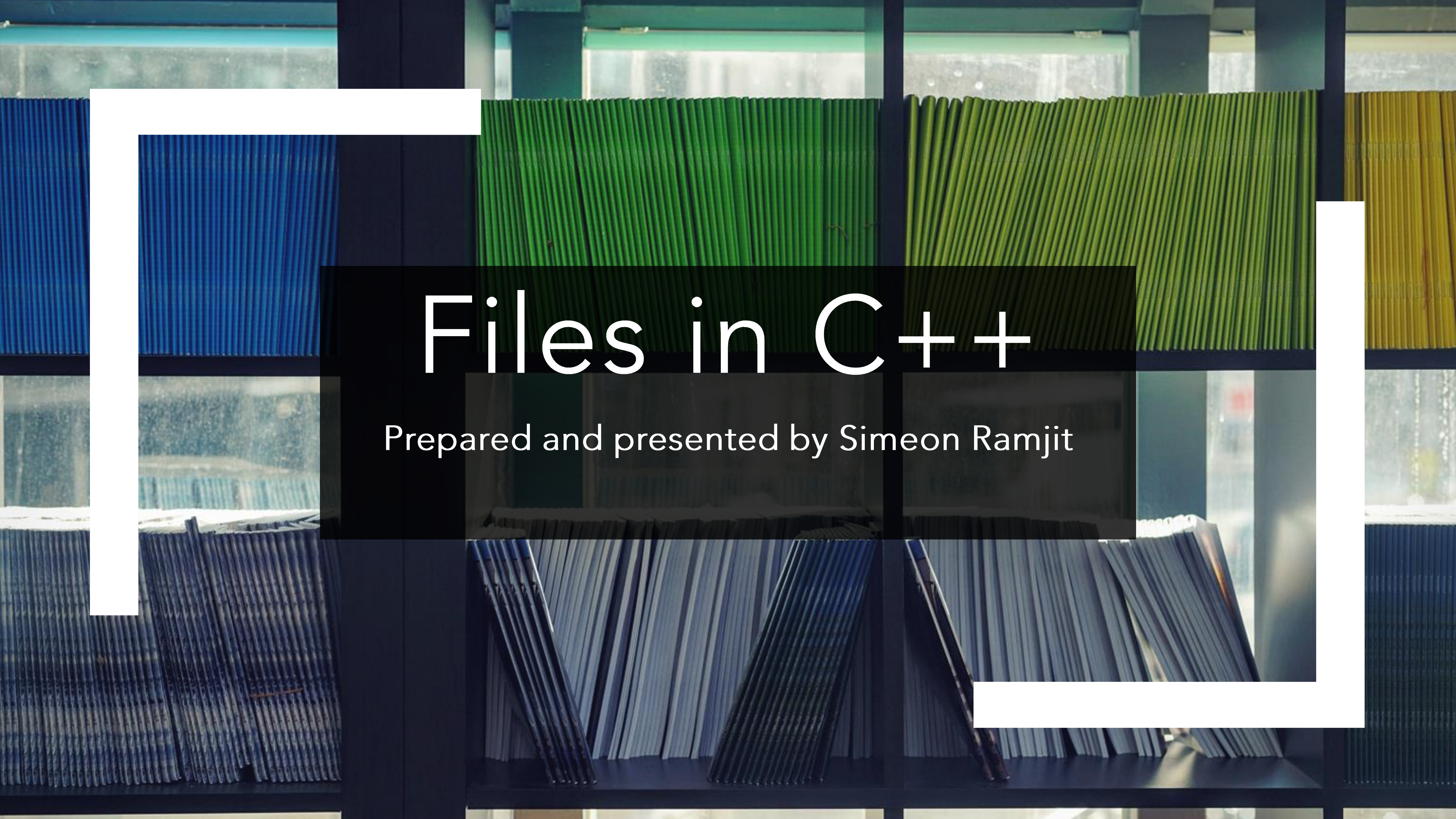

Files in C++

Prepared and presented by Simeon Ramjit

Files in C++

- A file is an object that stores data , information or settings used with a program.
- Files have varying encoding schemes that enable them to represent varying types of data.
- We would be looking at file Input-Output (I/O – reading and writing) operations using C++

File Classes

- C++ has three (3) classes that enable file I/O ops:
  - `ofstream` – enables a stream that writes to files only
  - `ifstream` - enables a stream that writes read to files only
  - `fstream` – enables a stream that allows reading & writing to files
- These classes were derived from the istream and ostream. You've already used objects from these classes.
- We use these streams the same way we use cin and cout except now we will stream to or from a physical file.

Opening Files

- To enable it's use in a program, the appropriate class has to be included at the start of the file, like all #include.

```
#include<fstream>
```

- Opening & closing files use a member function of the 'file' class. Whether it's to read or write, the method to open and close a file stays the same
- Files can be opened either to be read or to be written to.
  - To create an output stream `std::ofstream output_file`
  - To create an input stream `std::ifstream input_file`

Writing to files

- Let's put it all together!

```
#include<iostream>
#include<fstream>
int main() {
  std::ofstream outputFile; //declare output file stream
  outputFile.open("example file.txt");//We can use spaces normally,but dont
  outputFile << "I can write to a file! \n"; // /n-newline
  outputFile << "This is interesting :D";
  outputFile.close(); //Always close the file when you're done, why?
  return 0;
}
```

- If file that does not exist, is opened using an output stream it will be created. If it does exist it will be overwritten.

Reading Files

- Reading files is a bit more complex
- We need to check if the file exists and then read it
- We also to properly track where we are in the file to avoid working with uninitialized or garbage data

Reading Files

- Let's start off simple. Read in the file we just wrote to and display it in the console
- First we need to check if the file exists

```
#include<iostream>
#include<fstream>
int main() {
  std::ifstream input_file; //declare input file stream
  input_file.open("example file.txt");
  if (input_file) {
    std::cout << “File found!" << std::endl;
  }
  else {
    std::cout << "That file does not exist" << std::endl;
  }
  input_file.close();
  return 0;
}
```

Reading Files

- Re

```
#include<iostream>
#include<fstream>
#include<string>
int main() {
  std::ifstream input_file; //declare input file stream
  input_file.open("example file.txt");
  if (input_file) {
    std::string myText = ""; //declare string to hold text
    while (input_file >> myText) {//as long as data is valid
      std::cout << myText << std::endl;
    }
  }else {
    std::cout << "That file does not exist" << std::endl;
  }
  input_file.close();
  return 0;
}
```

Reading Files

- Remember we said we had to keep track of our position in the file?
- This is done via `while (input_file >> myText)`
- Using `while(!filestream.eof())` is bad practice. This is because the .eof() only sets the 'end of file' flag bit *after* data has been read in

```
while (!input_file.eof()) {
  int filestuff; // Seems like we're not at the end of the stream
  input_file >> filestuff;
  // now we read the file and discover it's the end
  // the eof  bit will be set (as well as the fail bit) and
  // we then go on to execute instructions on uninitialized data
}
```

Reading Files

- So how do we tell when we're at the end of the file?

```
while (input_file >> filestuff){
 // if we get to here then the read was successful. If it was not
 // the stream operator would return false and break the loop
 // We can now execute instructions on valid data.
}
```

- So we can safely read in a file, for the most part, but at this point you'd realize that we read in one word at a time and not the line itself.

Reading Files

- We need to use: `getline( std::ifstream filestream, std::string stringvar)`

```
#include<iostream>
#include<fstream>
#include<string>
int main() {
   std::ifstream input_file;
   std::string filename = "example file.txt"; //store filename in string
   input_file.open(filename);
   if (!input_file) {//file not open, throw error
     //Using \ to break up long cout statement
     std::cout << "Uh-oh, file not found. \
         Ensure file is in project directory" << std::endl;
   }else {//yay, do something with data
      std::string fileData = "";
      //use getline to read up to \n
      while (getline(input_file, fileData)) {
        std::cout << fileData << std::endl; }
      }   //what's missing? Check slide 5
   return 0;
}
```

Reading Files

- What if I have a .csv (comma separated file) ? Or just a file that indicates line endings by some other character that's not \n ?
- getline(…) has a delimiter that allows you to specify what character you'd like to read up to.
- Create a new file in yourproject directory called commaSeparatedValues.txt
- Enter the following text: comma,separated,text
- Now use getline with a comma as a delimiter

Reading Files

```
#include<iostream>
#include<fstream>
#include<string>
int main() {
  std::ifstream input_file;
  std::string filename = "commaSeparatedValues.txt"; //store filename in string
  input_file.open(filename);
  if (!input_file) {//file not open, throw error
   //Using \ to break up long cout statement
   std::cout << "Uh-oh, file not found. \
     Ensure file is in project directory" << std::endl;
  } else {//yay, do something with data
      std::string fileData = "";
      //use getline with custom delimiter ’,’
      while (getline(input_file, fileData, ',')) {
      std::cout << fileData << std::endl; }
  }   //what’s missing? Check slide 5
  return 0;
}
```

Files - Extras

- Sometimes you might need to reset the file position without closing and reopening the file (File I/O is expensive)
- **`tellg() and tellp()`** – Tells the position of streampos
- **`seekg() and seekp()`** - Sets the position of streampos
- http://www.cplusplus.com/doc/tutorial/files/

Questions?

simeon.ramjit@sta.uwi.edu

github.com/simeon9696/programmingworkshop