C++ File I/O

Tom Latham

THE UNIVERSITY OF WARWICK

Input/Output streams

- We've so far seen that we can use the `std::cout` and `std::cin` objects to print output to the screen and to read input from the keyboard

- These are examples of I/O streams (hence the name of the `iostream` header!)

- We'd like to be able to use these to read from and write to files as well (helps to avoid a lot of typing!)

- Can use the types provided in the `fstream` header file

Output file streams

- To use an output file stream you must:
  - Include the appropriate header file:

```
#include <fstream>
```

  - Instantiate an instance of an `ofstream` type:

```
std::string name {"myoutputfile.txt"};
std::ofstream out_file {name};
```

  - Unlike with `std::cout`, you need to check that the file was correctly opened before you can write to it:

```
bool ok_to_write = out_file.good();
```

- Then you can use that instance exactly as you would use `std::cout`

```
out_file << "Some text\n";
```

Input file streams

- To use an input file stream you must:
  - Again include the appropriate header file:

```
#include <fstream>
```

  - Instantiate an instance of an `ifstream` type:

```
std::string name {"myinputfile.txt"};
std::ifstream in_file {name};
```

  - Again, you need to check that the file was correctly opened before you can read from it:

```
bool ok_to_read = in_file.good();
```

- Then you can use that instance exactly as you would use `std::cin`

```
char inputChar {'x'};
in_file >> inputChar;
```

Closing files (and opening new ones)

- With both input and output files you can do:

```
file.close();
```

- This closes the file and any further attempts to read or write will fail

- It is done automatically when a file stream object is destroyed (e.g. when it goes out of scope), so you don't always need to do this explicitly

- However, you do need to do this first if you want to then use the same file stream object to open another file (although the circumstances under which you would want to do this are rare – it is clearer to use a separate object for each file):

```
file.open("myfirstfile.txt");
...
file.close();
file.open("myotherfile.txt");
...
```

Appending rather than overwriting

- By default when you open an output file stream and the file already exists, it will overwrite the contents of the file

- However, it is possible to open an output file stream in a mode where whatever you write to it will instead be appended to the file:

```
std::ofstream out_file{ name, std::ios::app };
```

- For more details on this see, for example:

  http://en.cppreference.com/w/cpp/io/ios_base/openmode

Exercise on file I/O

- Now we have the knowledge necessary to implement reading the input text from file rather than the keyboard and to write the cipher text to a file rather than the screen

- You already have command line options that provide the names of these files

1. Implement the new code to perform these operations if the corresponding option is specified on the command line (if the option is not specified, the program should continue to read from the keyboard and/or write to screen, as appropriate)