Scientific Programming with C++

READING AND WRITING DATA

Learning objectives

- After this lecture and related assignments, you will…
  - know how to read data from files
  - know how to write data to files
  - be introduced to serialization

File streams: data types for reading and writing data

- ofstream
  - Write files
- ifstream
  - Read files
- fstream
  - Both of the above combined in one
- Remember headers:
  - iostream
  - fstream

Working with files on the computer

- Before accessing, files must be opened
  - Using the function std::fstream::open(filename, mode)
    - Filename is a string containing the name of the file
    - Mode describes what can be done with the file

member constant	stands for	access
in	**in**put	File open for reading: the ***internal stream buffer*** supports input operations.
out	**out**put	File open for writing: the ***internal stream buffer*** supports output operations.
binary	**binary**	Operations are performed in binary mode rather than text.
ate	**at e**nd	The ***output position*** starts at the end of the file.
app	**app**end	All output operations happen at the end of the file, appending to its existing contents.
trunc	**trunc**ate	Any contents that existed in the file before it is open are discarded.

https://cplusplus.com/reference/fstream/fstream/open/

- It is good practice to explicitly close the file after use

Example: reading files (1/2)

```
std::fstream fs;

fs.open("dummyfile.txt", std::fstream::in);
std::vector<std::string> fileContents;

if (fs.is_open()) {
    while (!fs.eof()) {
        std::string newString;
        fs >> newString;

        fileContents.push_back(newString);
    }
}

fs.close();

for (auto it : fileContents) {
    std::cout << it << std::endl;
}
```

```
Hello world
I am the second line of this text file
Goodbye!
```

```
Hello
world
I
am
the
second
line
of
this
text
file
Goodbye!
```

Example: reading files (2/2)

```
std::fstream fs;

fs.open("dummyfile.txt", std::fstream::in);
std::vector<std::string> fileContents;

if (fs.is_open()) {
    while (!fs.eof()) {
        char newString[256];
        fs.getline(newString,256, '\n');

        fileContents.push_back(newString);
    }
}

fs.close();

for (auto it : fileContents) {
    std::cout << it << std::endl;
}
```

Hello world
I am the second line of this text file
Goodbye!

Hello world
I am the second line of this text file
Goodbye!

Example: writing files

```
std::fstream fs;

// open file for writing a new file, truncate if one exists
fs.open("coefficients.dat", std::fstream::out |
std::fstream::trunc);
std::vector<double> data = { 2, 4.2, 0.5, -3.5 };

for (double dataPoint : data)
    fs << dataPoint << '\n';

fs.close();
```

```
2
4.2
0.5
-3.5
```

Serialization and deserialization

- Serialization: writing objects to files (or converting to bytes)
- Deserialization: reading objects from files (or byte stream)
- Useful when objects and their modified state must be stored
  - Saving to file
  - Communicating between programs over the network
- Tricky for many objects
  - Issues with ordering of bytes (big-endian vs little-endian), data types with dynamic size (std::string, many containers)
  - External libraries are helpful

Example: serializing a book object (1/3): introducing the objects

```
struct Book {
    Book(std::string title, std::string author, int pageCount) {
        this->title = title;
        this->author = author;
        this->pageCount = pageCount;
    }
    Book() {};
    std::string title = "TITLE MISSING";
    std::string author = "AUTHOR MISSING";
    int pageCount = 0;
};
```

```
Book musketeers("The Three Musketeers", "Alexandre Dumas", 560);
Book heights("Wuthering Heights", "Emily Brontë", 416);
Book ward("The Case of Charles Dexter Ward", "H. P. Lovecraft", 176);
```

Example: serializing a book object (2/3): converting to byte array

```
std::ofstream out;
out.open("books_bytes.dat", std::fstream::out | std::fstream::trunc);

out.write((char*)&musketeers, sizeof(musketeers));
out.write((char*)&heights, sizeof(heights));
out.write((char*)&ward, sizeof(ward));
out.close();

Book savedBook;
std::fstream in;
in.open("books_bytes.dat", std::fstream::in);
in.seekg(0);
in.read((char*)&savedBook, sizeof(savedBook));
Book readMusketeers = savedBook;

in.seekg(sizeof(savedBook));
in.read((char*)&savedBook, sizeof(savedBook));
Book readHeights = savedBook;

in.seekg(sizeof(savedBook)+sizeof(savedBook));
in.read((char*)&savedBook, sizeof(savedBook));
Book readWard = savedBook;
in.close();
```

- Conversion of Book to byte array
- Works on my computer, for now
- If sizes of the data types in the struct change, we lose backwards compatibility
  - Saved data cannot be read
  - For example, if we add attribute “publicationYear”

Example: serializing a book object (3/3): converting to text format

```cpp
// overload << operator of ostream to write Book data in string format
std::ostream& operator<<(std::ostream& os, const Book& book) {
    return os << book.title << "\t" << book.author << "\t" << book.pageCount << "\t";
}

// overload >> operator of ifstream to read Book data
std::fstream& operator>>(std::fstream& ifs, Book& book) {
    char newString[256];
    ifs.getline(newString, 256, '\t');
    book.title = newString;
    ifs.getline(newString, 256, '\t');
    book.author = newString;
    ifs.getline(newString, 256, '\t');
    book.pageCount = stoi((std::string)newString);
    return ifs;
}
```

```cpp
std::fstream out;
out.open("books.dat", std::fstream::out | std::fstream::trunc);
out << musketeers << heights << ward;
out.close();

Book savedBook;

std::fstream in;
in.open("books.dat", std::fstream::in);
in >> savedBook;
Book readMusketeers = savedBook;

in >> savedBook;
Book readHeights = savedBook;

in >> savedBook;
Book readWard = savedBook;
in.close();
```

- Code is cleaner
- Produces a text file that is readable by humans
- Still terrible with objects that have many attributes
  - Prefer 3rd party libraries!

Summary

- Data can be written to file and read from file
  - Simple data types are easier to handle
  - Working with text files can help
- Serialization
  - Concept is good to know
  - For implementation, prefer external libraries