File I/O – Text Files

Reading and writing text files

Contents

Canberra Institute of Technology

aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

Text Files

- Text files are human readable files that can be opened with almost any text editor.

- Therefore, when parsing our files, we need to be extra careful to catch any errors that may exist within our file.

Text Files

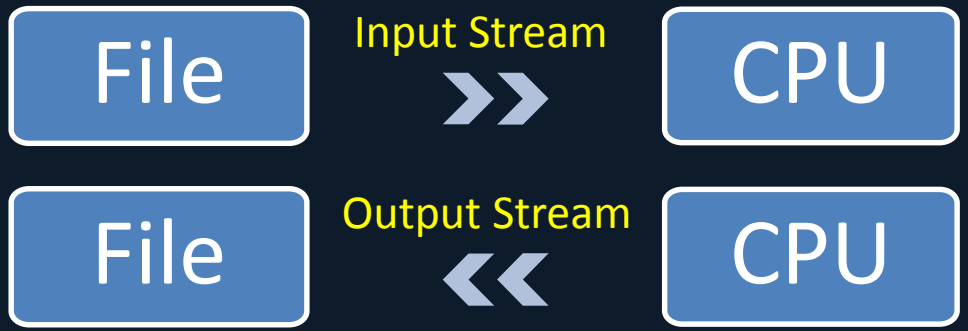
- Pros of Text Files:
  - Portable – Supported on all platforms.
  - Useful for saving pure text data (log files, conversation history).
  - Used to save settings that can be edited by humans.
  - Used for a few save game formats.
  - Easy to parse when using a standardised format.
    - JSON, CSV, XML, YAML

Text Files

- Cons of Text files
  - Need to convert binary to text and text to binary when used to store non-text data.
  - Need to write a custom parser when not using a standard format.
  - Doesn't work well for chunks of pure data (Images, sounds, videos, etc).

C++ Streams

- C++ uses streams for transferring a sequence of bytes using the stream operators << and >>.
- There are two types of streams:
  - Input streams receive a sequence of bytes.
  - Output streams send a sequence of bytes.



C++ Streams

- We have been using streams for sending and receiving information to the console window already.
  - Use the << operator to send information through a stream.
    - ```
      cout << "Outputting a string to the console" << endl;
      ```
  - Use the >> operator to receive information waiting to be retrieved.
    - ```
      cin >> myVariable;
      ```

File Streams

- To write to a file, we need to:
  - Include the fstream header.
    - Part of the std namespace.
  - Create our file stream handle.
  - Open a file:
    - ```
      file.open("File.txt", ios_base::out);
      ```
  - Write to file using the << operator.
  - Read from a file using the >> operator.
  - Always close the open file after use!
    - ```
      file.close();
      ```

Canberra Institute of Technology

aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

Open File – Parameter Flags

- The open() function's parameters tell it how to open the file:
  - std::ios_base::out
    - Open file for writing only, creates a new one if it does not exist and overrides any existing file.
  - std::ios_base::in
    - Open a file for reading only, file must exist.
  - std::ios_base::_Nocreate
    - Will not create a new file so the file must exist already.
  - std::ios_base::binary
    - Opens file for streaming binary data.
  - std::ios_base::app
    - Opens file, appends data to the end of the file; file must exist.

Combining Open Parameters

- We can use the | operator to specify multiple options:
  - Examples:
    - Open for writing, but don't create if file doesn't exist:

```
file.open("File.txt", ios_base::out | ios_base::_Nocreate);
```

    - Open for reading and read binary data:

```
file.open("File.txt", ios_base::in | ios_base::binary);
```

Safety First

- Always check that the file opened successfully before reading or writing.
  - Opening a file can fail for many reasons:
    - File does not exist.
    - File is already open.
    - User account does not have sufficient permissions.

```
file.open("File.txt", ios_base::out);

if(file.is_open())
{
    //Safe to write to file
    file << "Hello World" << endl;
}
else
{
    //Let the user know something went wrong.
}
```

Writing text to a file

- Very simple! Same thing as writing to the console.

```
file << "Hello World" << endl;
```

- Any object that has an overloaded << can send data to a stream, in our case, a file stream.

- If we can print an object out to the console window, we can write it to a file!

Reading in from a file

- Typically, to read a file, we need to know the exact format it was written in.
- The file must have been opened using either std::ios_base::in, or std::ios_base::app.
- We have many more options to choose from for reading than just the >> operator.

Reading in from a file

- Here are some other methods for reading from a file:
  - Reading up to the first space character:
    - `file >> myVariable;`
  - Reading a single character:
    - `get(char& c);`
  - Reading a specified number of characters:
    - `get(char* s, streamsize n);`
  - Reading a whole line:
    - `getline(char* s, streamsize n);`

Closing files

- Always close files after use.

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

  - When writing to a file, chunks of data are committed to a buffer in RAM. When the buffer is full, the data is written to the file and the buffer is cleared. Closing ensures this process completes before the program exits.
  - Closing frees the file to be opened by other software.

Pro Tip

- Research overloading the << operator. This will enable to you to send an object to the console window or a file.

```
class Point2D
{
public:
    float x, y;

    friend ostream &operator << (ostream &output, const Point2D &p)
    {
        output << p.x << ", " << p.y;
        return output;
    }
};
```

```
Point2D p;

p.x = 10;
p.y = 15;

cout << p << endl;
```

Canberra Institute of Technology

aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

Summary

- We can easily read and write text files using the << and >> operators, just remember to:
  - Include the fstream header
    - Part of the std namespace
  - Create a file stream handle.
  - Open the file.
  - Read or write the file.
  - Always close the open file after use!

References

- For in depth function definitions on all the C++ stream input and output operations:
  - cplusplus.com. *IO Library.* http://www.cplusplus.com/reference/iolibrary