**COS30031 - Lab 10**

**102564760 – Ryan Chessum**

**Part A**

* Created a struct with data.

Text

Description automatically generated

* Created an instance of the struct.

Text

Description automatically generated

* Wrote code to open a binary file and write to it.

Text

Description automatically generated

Q: There are different file open modes: What are they?

Table

Description automatically generated

(table from: <https://www.cplusplus.com/reference/fstream/fstream/open/>)

in -> open file for reading

out -> open file for writing

binary -> performs operations in binary

ate -> output position starts at the end of the file

app -> operations happen at the end of the file

trunc -> discards the content within the file before performing operations

(The difference between ate and app are that ate sets the write position to the end upon opening the file while app sets it to the end before each operation.)

Q: What happens if you don’t “close” the file? Is it something we need to worry about?

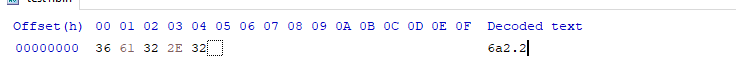
There is a very, very small chance it could corrupt the file but otherwise the file stream will go out of scope and the file will be closed automatically. So if we forget it we probably don’t need to worry too much. Despite it normally being fine, it is best to remember to close it. If you are writing to too many open files at once your program can crash or fail to open more files when you need to.

* Code runs and file was created

Graphical user interface

Description automatically generated

* Stuff is being written to the file. The hex viewing program I am using even confirms it’s what I put in.



Q: How many bytes are in the file? Is this expected based on the size of the variable types?



It’s 5 bytes. It seems that it saved each value as a char the way I saved to the file initially so I has 1 byte for each character saved in the file.

To specify the type write the code like this.

Text

Description automatically generated

Opening the file now gives us this.

Graphical user interface

Description automatically generated with low confidence

The hex viewer can’t really tell us what the text is right away. It got ‘a’ right though.

If we check each chunk of data we can see it would be as the appropriate data type.

Graphical user interface, application, table

Description automatically generated

The int we used must have been a 32 bit integer as it takes up 4 bytes.

Graphical user interface, application, table

Description automatically generated

A is still a char so it only takes up 1 byte.

Graphical user interface, table

Description automatically generated

Our float is also now stored as a float. Pretty interesting how you can see floats don’t always have 100% precision.

So if we check the size again…



It’s now 9 bytes. Still makes sense as floats and larger integers take a bit more space. Still tiny though and makes sense for our small data types.