

COS30031 - Lab 10

102564760 – Ryan Chessum

Part A

- Created a struct with data.

```
struct Test
{
    int anInt;
    char aChar;
    float aFloat;

    void ShowValues()
    {
        cout << anInt << endl;
        cout << aChar << endl;
        cout << aFloat << endl;
    };
};
```

- Created an instance of the struct.

```
int main()
{
    int in = 6;
    char ch = 'a';
    float fl = 2.2;
    Test myData{ in, ch, fl };

    myData.ShowValues();
}
```

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

```
//open in write mode
fstream fs;
fs.open("test1.bin", fstream::out | fstream::binary);

//write data
fs << myData.anInt;
fs << myData.aChar;
fs << myData.aFloat;

//close file
fs.close();

return 0;
}
```

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

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

These flags can be combined with the bitwise OR operator (`|`).

(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

Name	Status	Date modified	Type	Size
Debug		12/11/2021 11:15 PM	File folder	
* Task 10.cpp		12/11/2021 11:21 PM	C++ Source	1 KB
Task 10.vcxproj		12/11/2021 6:29 PM	VC++ Project	8 KB
Task 10.vcxproj.filters		12/11/2021 6:29 PM	VC++ Project Filte...	1 KB
Task 10.vcxproj.user		12/11/2021 6:29 PM	Per-User Project O...	1 KB
test1		12/11/2021 11:15 PM	BIN File	1 KB

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

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	36	61	32	2E	32												6a2.2

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

Size: 5 bytes (5 bytes)

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

To specify the type write the code like this.

```
//open in write mode
fstream fs;
fs.open("test1.bin", fstream::out | fstream::binary | fstream::trunc);

//write data
fs.write(reinterpret_cast<const char*>(&myData.anInt), sizeof(myData.anInt));
fs.write(reinterpret_cast<const char*>(&myData.aChar), sizeof(myData.aChar));
fs.write(reinterpret_cast<const char*>(&myData.aFloat), sizeof(myData.aFloat));

//close file
fs.close();
```

Opening the file now gives us this.

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	06	00	00	00	61	CD	CC	0C	40							aïî.0

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.

HxD - [C:\Users\ryanc\OneDrive\Documents\GitHub\GamesProgramming\cos30031-102564760\10 - Lab - File Input Output\Task 10\Task 10\test1.bin]

File Edit Search View Analysis Tools Window Help

test1.bin

Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F Decoded text
 00000000 06 00 00 00 61 CD CC 0C 40 ...aïi.ë

Special editors

Data inspector

Binary (8 bit) 00000110

Int8	go to: 6
UInt8	go to: 6
Int16	go to: 6
UInt16	go to: 6
Int24	go to: 6
UInt24	go to: 6
Int32	go to: 6
UInt32	go to: 6
Int64	Invalid
UInt64	Invalid
LEB128	go to: 6
ULEB128	go to: 6
AnsiChar / char8_t	<input type="checkbox"/>
WideChar / char16_t	<input type="checkbox"/>
UTF-8 code point	<input type="checkbox"/> (U+0006)
Single (float32)	8.4077907859489E-45
Double (float64)	Invalid
OLETIME	Invalid
FILETIME	Invalid
DOS date	Invalid
DOS time	12:00:12 AM
DOS time & date	Invalid
time_t (32 bit)	1/01/1970 12:00:06 AM
time_t (64 bit)	Invalid
GUID	Invalid
Disassembly (x86-16)	push es
Disassembly (x86-32)	push es
Disassembly (x86-64)	Invalid

Byte order
☒ Little endian ☐ Big endian

☐ Hexadecimal basis (for integral numbers)

Offset(h): 0 Block(h): 0-3 Length(h): 4 Overwrite

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

HxD - [C:\Users\ryan\OneDrive\Documents\GitHub\GamesProgramming\cos30031-102564760\10 - Lab - File Input Output\Task 10\Task 10\test1.bin]

File Edit Search View Analysis Tools Window Help

test1.bin

Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F Decoded text
00000000 06 00 00 00 0A CD CC 0C 40ii.0

Special editors

Data inspector

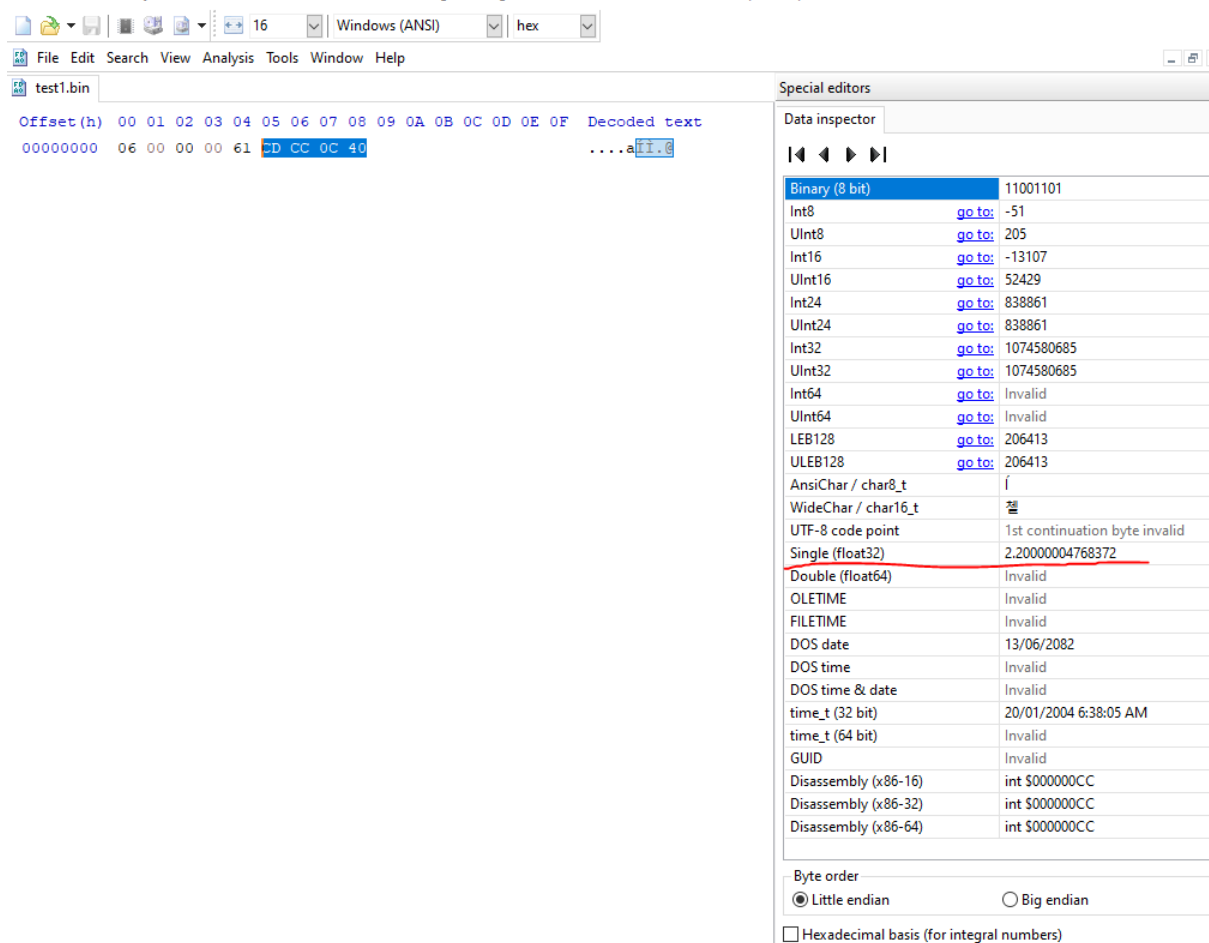
Binary (8 bit)	01100001
Int8	go to: 97
UInt8	go to: 97
Int16	go to: Invalid
UInt16	go to: Invalid
Int24	go to: Invalid
UInt24	go to: Invalid
Int32	go to: Invalid
UInt32	go to: Invalid
Int64	go to: Invalid
UInt64	go to: Invalid
LEB128	go to: -31
ULEB128	go to: 97
AnsiChar / char8_t	a
WideChar / char16_t	Invalid
UTF-8 code point	a (U+0061)
Single (float32)	Invalid
Double (float64)	Invalid
OLETIME	Invalid
FILETIME	Invalid
DOS date	Invalid
DOS time	Invalid
DOS time & date	Invalid
time_t (32 bit)	Invalid
time_t (64 bit)	Invalid
GUID	Invalid
Disassembly (x86-16)	popaw
Disassembly (x86-32)	popad
Disassembly (x86-64)	Invalid

Byte order
☒ Little endian ☐ Big endian

☐ Hexadecimal basis (for integral numbers)

Offset(h): 4 Block(h): 4-4 Length(h): 1 Overwrite

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



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...

Size: 9 bytes (9 bytes)

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.

- Wrote code to read the data stored in the file

```
int in;
char ch;
float fl;

fstream fs;
fs.open("test1.bin", fstream::in | fstream::binary);

fs.read(reinterpret_cast<char*>(&in), 4);
fs.read(reinterpret_cast<char*>(&ch), 1);
fs.read(reinterpret_cast<char*>(&fl), 4);

fs.close();

Test myData{ in, ch, fl };

myData.ShowValues();
```

```
Microsoft Visual Studio Debug Console
6
a
2.2
```

It reads the file successfully!

Part B

- Wrote code to read each line of the .txt file

```
fstream fs;
string str;

fs.open("test2.txt", fstream::in);

while (getline(fs, str))
{
    cout << str << endl;
}

fs.close();
```

```
Microsoft Visual Studio Debug Console
# This is a commentline. The next line is deliberately blank.
12:string value:13.57
done
```

- Edited code to ignore lines if they are blank or commented with a '#'

```
fstream fs;
string str;

fs.open("test2.txt", fstream::in);

while (getline(fs, str))
{
    if (str.size() != 0 && str.at(0) != '#')
    {
        cout << str << endl;
    }
}

fs.close();
```

```
Microsoft Visual Studio Debug Console
1
12:string value:13.57
6
7
done
```

- Split the string by ':'

```

vector<string> split(string str)
{
    stringstream ss(str);
    vector<string> result;

    while (ss.good())
    {
        string s;
        getline(ss, s, ':');
        result.push_back(s);
    }

    return result;
}

```

```


fstream fs;
string str;

fs.open("test2.txt", fstream::in);

while (getline(fs, str))
{
    if ((str.size() != 0) && (str.at(0) != '#'))
    {
        for (auto s : split(str))
        {
            cout << s << endl;
        }
    }
}

fs.close();

```

 Microsoft Visual Studio Debug Console

```

12
string value
13.57
done

```

Part C


- Created Json file
- Downloaded Json library
- Wrote code to read the file using library


```
fstream fs;

fs.open("test3.json", fstream::in);
json jf = json::parse(fs);

cout << "exp: " << jf["exp"] << endl;
cout << "health: " << jf["health"] << endl;
cout << "jsonType: " << jf["jsonType"] << endl;
cout << "level: " << jf["level"] << endl;
cout << "name: " << jf["name"] << endl;
cout << "uuid: " << jf["uuid"] << endl;

fs.close();
```

 Microsoft Visual Studio Debug Console

```
exp: 12345
health: 100
jsonType: "player"
level: 42
name: "Fred"
uuid: "123456"

done
```