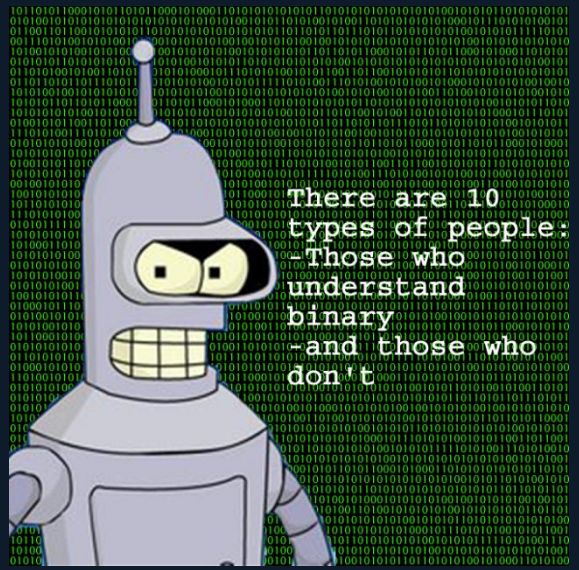
Reading and Writing Binary Files



Canberra Institute of Technology

aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

Contents

- Opening a file for binary read/write
- Reading a binary file
- Writing to a binary file
- Positioning the read and write markers

Binary Files

- Unlike text files, binary files are not human readable
- They store data using the internal format of the computer (binary numbers/bits)
- Basically we store data as it appears in memory
- Often referred to as Random Access Files

Binary Files

- Binary files are most often used with structures and classes.
- This is because the file is organised into fixed length records

Canberra Institute of Technology

aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

Pros and Cons

- Pros
  - More compact
  - Easier to modify
- Cons
  - Not human readable (possibly a good thing)
  - Less portable

Opening a binary file

- Very similar to opening text files
  - We need to use the parameter ios::binary
- Open a file for reading in binary mode:
  - fstream fin(“data.dat”, ios::in | ios::binary);
- Open a file for writing in binary mode:
  - fstream fout(“data.dat”, ios::out | ios::binary);

Reading a binary file

- Rather than using the >> operator, we want to read an entire record, or an entire chunk of data at once.
- We will use the following function
  - read(char* buffer, streamsize size);
- We read the data (bits) in and store it in *buffer*.
- *buffer* is usually a class or structure instance (passed as a char pointer).
- *size* determines how many bytes are to be read.

Example reading from a file

```
#include <iostream>
#include <fstream>

struct Vector3{
     float x, y, z;
};

void main()
{
    Vector3 myVec;
    std::ifstream fin("data.dat", std::ios::in | std::ios::binary);
    if(fin.good()) {
        // read until we get to the end of file
        while(!fin.eof() && fin.peek() != EOF) {
             fin.read((char*)&myVec, sizeof(Vector3));
             std::cout << myVec.x << " " << myVec.y << " " << myVec.z << std::endl;
        }
        fin.close();
    }
}
```

Writing to a file

- Writing to a file is just the opposite of reading, and the function looks very similar
  - write(const char* buffer, streamsize size);

Example writing to a file

```
#include <iostream>
#include <fstream>

struct Vector3{
    float x, y, z;
};

void main() {
    Vector3 myVecArray[] =  { {0, 1, 2}, {1, 2, 3}, {4, 5, 6}, {32, 1, 98} };
    int arraySize = sizeof(myVecArray)/sizeof(Vector3);

    //fill in the array here
    std::ofstream fout("data.dat", std::ios::out | std::ios::binary);
    if(fout.good()) {
        for(int i = 0; i < arraySize; i++) {
            fout.write((char*)&myVecArray[i], sizeof(Vector3));
        }
        fout.close();
    }
}
```

Writing to a file

- In that last example, we could have written out the whole array in one go:
  - fout.write((char*)myVec, sizeof(Vector3)*arraySize)
- This method is more efficient

Error checking

- ifstream::read and ofstream::write don’t return error codes
- Errors are signaled by the internal state flags

iostate **value (member constant)**	**indicates**	**functions to check state flags**				
		good()	eof()	fail()	bad()	rdstate()
goodbit	No errors (zero value iostate)	true	false	false	false	goodbit
eofbit	End-of-File reached on input operation	false	true	False	false	eofbit
failbit	Logical error on i/o operation	false	false	true	false	failbit
badbit	Read/writing error on i/o operation	false	False	true	true	badbit

Error checking

```
#include <iostream>
#include <fstream>

void main()
{
    std::ifstream fin("data.dat", std::ios::in | std::ios::binary);
    if ( (fin.rdstate() & std::ifstream::failbit ) != 0 )
    {
        std::cout << "Error opening 'data.dat'" << std::endl;
    }
}
```

Random Access - positioning

- Binary files allow us to read and write from/to anywhere in the file – not just from the beginning or end.
- We can move the read and write markers (a pointer to the position of the file we're currently writing to or reading from) around within our file.
- Files are accessed like arrays – the 0^{th} position is the start of the first byte.

Tellp, tellg

- These functions tell us the current positions of the read and write markers.
- They return an integer representing the current byte where the marker is positioned.
- Read:
  - tellg()
- Write:
  - tellp()

Seekg, seekp

- These functions will move the read and write markers to a specific byte in a file.
- This is useful for replacing one record that's in the middle of the file.
- Read:
  - seekg(int offset, seekdir fromWhere);
- Write:
  - seekp(int offset, seekdir fromWhere);

Seekdir?

- An enumeration with three values:
  - seek from the beginning
    - ios::beg
  - seek from the current location
    - ios::cur
  - seek from the end of file
    - ios::end

Examples

- fin.seekg(10, ios::beg)

  Set the reading position of fin to the 11th byte

- fin.seekp(5, ios::cur)

  Moves the writing position of fin five bytes to the right of its current position.

- fin.seekp(-8, ios::end)

  Moves the write position of fin 8 places before the end of the file

Summary

- Reading/Writing in binary is very similar to text
  - More compact
  - Easy to use with structures
  - Can write/read many records at the same time
- Use the std ifstream and ofstream classes
- Error states are written as internal class flags
- I/O streams can be randomly accessed via the tell/seek functions

References

- iostream - C++ Reference. 2015. *iostream - C++ Reference*. [ONLINE] Available at: http://www.cplusplus.com/reference/istream/iostream/. [Accessed 14 April 2015].
- C++ Binary File I/O. 2015. *C++ Binary File I/O*. [ONLINE] Available at:http://courses.cs.vt.edu/cs2604/fall02/binio.html. [Accessed 14 April 2015].

Canberra Institute of Technology

aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX