Tute 9: More Design Patterns

Programming Concepts:

What is the command pattern? Could this help you with your design of the controller in assigment 2?

The command pattern is a common approach to dealing with multiple function calls in an object-oriented way. Each different object stored in a “command collection” represents a different function call and a different command in the system. All command objects inherit from a single abstract command class and the command to be called is resolved at runtime based on menu selection or using input or something similar.

What is binary i/o? How does it differ from ascii?

In both cases (ascii and binary io) we are reading or writing bytes. It is just the meaning of those bytes that differ. Up to now, you have probably associated the char datatype with characters and probably been encouraged by your teachers to do so. But that’s abstraction that we don’t want anymore. A char (in C and C++) is simply an integer that takes up one byte. We have historically represented characters in one byte because character representation grew out of the ascii standard.

With binary I/O we will use chars to represent bytes in memory which includes non-printable characters. We don’t care what the thing we are representing is, we just write out the bytes to a file in the format that they can be read back into memory directly.

What does it mean to serialise an object? What issues do we need to consider?

When writing out data held in memory to a file, we need to replace any memory addresses and other contextual information with numbers that represent that data (such as unique identifiers). This is because pointers have no meaning outside a running program - the same data may be loaded into a different memory address on the next run of the program.

Another issue we need to consider when writing binary data is byte alignment. Because of the rules of memory packing, typically all data that is bigger than a byte must be aligned on an even byte. If you are not careful about this, your program may insert some garbage bytes and this may impact your program when you read these bytes back in.

Another issue is **endianness**. This is the byte order that data is stored in memory. The order within each byte is the same but the order of bytes in data such as ints is placed in a different order on different systems. On little endian systems, the byte with the lowest value is stored first and on big endian systems, the opposite order is used. You cannot move a binary file that has

been written with one endianness onto a system that uses the opposite endianness without some conversion.

What functions do we use for binary i/o?

We use the functions istream.read() and ostream.write. Please see https://github.com/muaddib1971/cpt323/blob/master/chat_examples/week4/read_nums_asc.cpp for an example of using ostream.write(). You should be able to write a program that reads in data using istream.read() based on this example.

Compilation: None this week.

Errors: None this week.

Debugging: Comments

Exercises:

Consider a datastructure such as the following. We have an array of students. For each student we want to store on disk their student number, name and results for each course they have studied where the results are a linked list of pairs of course code and result out of 100.

Write a program that inserts some students with a variable number of grades into the datastructure.

Next write the routine to serialize this data to a binary (not ascii) file.

See https://github.com/muaddib1971/cpt323/tree/master/chat_examples/week9 for an example solution.

Sample Data:

Fred Nerks, s6757389
- Grades:
  - COSC1076, 89
  - COSC1111, 67
  - COSC1234, 43

David, Wannabe, s7849003
- Grades:
  - COSC1108, 78
  - COSC1243, 89

Paul Malcanchio, s6788398
- Grades:
  - COSC1111, 45