Assignment Part 7 – C++

# Program Testing Procedure

Test sorting procedure with various arrays of books and observe output. Make sure the books are sorted by name, lexicographically, ascending. Please see the code for the test data, as there is too much to be shown here.

# Weekly Question

In terms of syntax and capabilities, C++ and Java objects are similar. Both enable objects via classes.  
However, the way object allocation, lifetime, and handling are done differs significantly. In Java, all object types (i.e. non-primitives) are allocated on the heap, and variables hold references to them – all handling of objects is done “by reference”. The lifetime of objects is managed by the garbage collector. In C++, objects are allocated on the stack unless one explicitly performs a heap allocation. Variables directly represent the storage and lifetime of their objects – handling of objects is done “by value”. The lifetime of an object is bound to the scope of the variable to which it is assigned. C++ also makes no distinction between primitive types and class/object types in terms of allocation and lifetime.  
In C++, objects more so represent allocated storage than pure OOP objects. This is highlighted by C++11’s introduction of “move semantics”, whereby an object’s state may be “moved” from its allocated storage to enable efficient memory management. In Java, an object conceptually encapsulates state and memory allocation.

# Reflection

C++’s const functionality adheres to the preservation of information, defence in depth, and security principles, increasing the reliability of the program. Variables may be marked as “const” to state that modification of them is not permitted/intended, which is enforced by the compiler. This enables compile-time detection of unintended mutation of program state.

C++’s multi-mode arithmetic and numerical conversions violates the defence in depth and security principles, decreasing the reliability of the program. When subtracting integer types, unsigned operands are not converted to signed types, resulting in the possibility of underflow. In the quicksort procedures, I have to explicitly use a signed type (std::ptrdiff\_t) for array indices, as some calculations result in negative numbers (this is not obvious from reading the code, however). Usage of unsigned types, which are more natural for array indices, would have resulted in silent bugs.

C++’s templates adheres to the abstraction principle, increasing the readability, writability, and expressivity of the program. Templates enable a construct to be parameterised on a type or value, allowing, for example, a single data structure or algorithm to be used for multiple data types. The program makes use of std::vector, a dynamically-sized array type, and std::swap(), a function for swapping object values, both of which are templated constructs. These constructs can simply be instantiated for the Book class, enabling reuse of existing, well-tested functionality.