**Reflective Report :**

**1) Sensible code separation (use of class and functions), encapsulation, inheritance, and data validation**

I believe on the Grade Descriptor my use of sensible code separation is around a low to mid First as I developed a rough system architecture of what my system is going to be before I started implementation to ensure I met object-oriented standards. So, in my project I have classes for each part of the application to to have a sensible code separation

I ensured to meet encapsulation so that each class’s methods operated on the data of that method only and ensured each class represents distinct entities in my application. I also restricted direct access to some of parts of the classes components by using access specifiers (private, protected, and public).

Also, to meet the inheritance rule of object-oriented programming Both File and Directory classes inherit from FilesystemEntity, demonstrating polymorphism and the power of abstraction.

Lastly to ensure the data the user inputs is valid I ensured to use the correct data types and validate each input by users.

So according to the grade descriptor I would award a grade point of **13 marks** as I could have used better inheritance and could have used the protect access specifier better and more often, but I went beyond what has been taught and designed my system before I started implementation and used techniques that may not have been covered in learning material.

**2) Use of OO in C++ including, polymorphism, runtime type operators, operator overloading**

I believe on the grade descriptor my use of Use of OO in C++ including, polymorphism, runtime type operators, operator overloading is around the mid to high 2.1.

I used polymorphism by having an abstract base class FilesystemEntity from which other classes like File and Directory inherit. This allows objects of these derived classes to be treated as objects of the base class, enabling polymorphic behaviour. For example, Display() method in my FilesystemEntity class is virtual and it is implemented distinctly in each subclass to present information relevant to files and directories, showcasing the practical use of virtual functions and method overriding.

I used runtime type operators many times in my project by using dynamic cast to check the runtime type of FilesystemEntity pointers and references, to determine whether they are actually pointing to a File or a Directory. Also used to find the correct file or directory when user wants to delete file or directory and to check for existing files or directories.

Lastly, I only used operator overloading slightly for “<<” however I didn’t use it that much therefore my overall grading may go down for this section, but I used the other topics we covered in the module.

So overall I believe I should get **11 marks** as I have showed a very good understanding of this area of the module however, I didn’t use operator overloading as much.

**3) Dynamic memory management and ownership without memory leaks**

I believe on the grade descriptor my use Dynamic memory management and ownership without memory leaks is around the high 2.1 – low 1st. In my project implementation I primarily used unique\_ptr for automated resource management. These smart pointers ensure that memory is automatically freed when no longer needed, effectively preventing memory leaks.

The Filesystem class, for instance, demonstrates ownership and scope-bound resource management by encapsulating a root Directory within a unique\_ptr. I used smart pointers in the File classes extending from FilesystemEntity, where each dynamically allocated instance is also managed by unique\_ptr.

So overall I believe I should get **12 marks** because of my use of smart pointers, such as unique\_ptr, in my project demonstrates my ability to effectively manage memory without leaks, ensuring robust and safe memory operations. This decision aligns with modern C++ best practices, favouring smart pointers over raw pointers to prevent common memory management errors. While I did not manually handle raw pointers, which might be seen as a limitation in showcasing manual memory management skills so that’s why I give 12 marks as I could have showed cased my manual memory management better.