1. Procedural

In the procedural approach, the program can be straightforwardly designed to read the input file, parse the rolls, and calculate the score sequentially. This fits the problem well because the scoring rules of bowling are inherently sequential, and the procedural style allows for a clear and linear implementation of these rules. Each step of the process, such as parsing rolls and calculating scores, can be represented by a function or a set of functions, making the code easy to understand and maintain.

2. Object-Oriented

Using an object-oriented approach allows for a more modular and organised design. The classes Game, Round, and Roll represent the entities involved in the game of bowling, and inheritance can be used to model relationships between them. Roll can be a base class with subclasses representing different types of rolls (Play, BonusPlay). Inheritance facilitates code reuse and enables polymorphism, which can be beneficial for extending the program in the future.

3. Pipes and Filters

Pipes and filters provide a flexible and modular way to design systems by breaking down a complex task into smaller, independent components. In the context of calculating bowling scores, pipes can be used to connect separate programs (filters) that handle different parts of the scoring process, such as parsing rolls and calculating scores. Each filter can perform a specific task and output its result, which is then passed as input to the next filter through a pipe. This approach fits well with the problem because it allows for easy composition and reuse of components, making the overall system more maintainable and scalable.

4. Summary

The best fit for the problem depends on various factors such as the size and complexity of the program, the programming environment, and the preferences of the developer. For a small and simple program like calculating bowling scores, the procedural approach may suffice and provide a straightforward solution. However, as the program grows in complexity or if there is a need for modularity and extensibility, the object-oriented or pipes and filters approach may be more suitable. Object-oriented design facilitates code organisation and reuse, while pipes and filters offer flexibility and scalability through modular components (similar to micro-services).

5. Summary on Software Architecture

Through this exercise, I learned that choosing the right software architecture pattern depends on the specific requirements and constraints of the problem at hand. Each architecture pattern has its strengths and weaknesses, and selecting the most appropriate one requires careful consideration of factors such as complexity, maintainability, and scalability. Additionally, I learned the importance of modularity and encapsulation in software design, as these principles help to create more flexible, understandable, and maintainable systems. Overall, this exercise highlighted the importance of understanding various architecture patterns and their applications to design efficient and robust software solutions.