Bowling Assignment

Class 3

Arquitetura de Sistemas de Software

M.EIC 2023/2024

HW 04

Ricardo Almeida Cavalheiro up202005103@fe.up.pt

Index

1. Introduction

Bowling is a classic game enjoyed by people of all ages, and understanding its scoring system is essential for any enthusiast or developer tasked with implementing scoring algorithms.

In this individual project, the challenge is to create three different programs to calculate bowling scores, each using a distinct programming style or pattern. The programs will need to accurately interpret input from various files, representing rolls in a game of bowling, and apply the scoring rules correctly. The ultimate goal is to demonstrate proficiency in different software architectural approaches while effectively solving a relatively small and simple problem.

The key requirements for this project include a thorough understanding of the rules of bowling scoring. Additionally, the programs should be able to handle different types of inputs.

Overall, this project offers an opportunity to explore and compare different programming paradigms and architectural patterns. From procedural programming to object-oriented design with inheritance, and the use of pipes and filters, each approach brings its own advantages and challenges to the table that I will be talking about and explaining later.

Through this exercise, I gained a deeper understanding of software architecture principles. I learned to select the most appropriate approach for a given problem based on factors such as complexity, maintainability, and scalability.

2.1 Procedural

Procedural programming is a paradigm where the program is structured around functions or procedures that operate on data. In this approach, the focus is on breaking down the problem into smaller, manageable tasks and implementing each task as a separate function.

For the bowling scoring problem, I created three functions that: handle input parsing, separate the frames and calculate the score.

1. **read_rolls_from_file**: This function reads the input from a file and stores the rolls in a list. It's responsible for handling the input format and ensuring it is correctly processed and stored.

2. **frame_separator**: The second function separates the rolls appropriately into frames, taking into account the special rules for the final frame. It divides the rolls into nine normal rounds and one final frame, which may consist of up to three rolls if there are bonus rolls.

3. **calculate_score**: The third function implements the algorithm to calculate the total score based on the frames. It iterates through the frames, applying the scoring rules of bowling, such as adding up the pins knocked down in each frame and accounting for strikes and spares.

This procedural approach fits the problem well for several reasons:

- Simplicity: Procedural programming is straightforward and easy to understand, making it suitable for solving small to medium-sized problems like bowling scoring.

- Modularity: By breaking down the problem into smaller functions, each responsible for a specific task, the code becomes modular and easier to maintain and debug.

However, there are some limitations to consider:

- Scalability: While procedural programming is suitable for small-scale projects, it may become unwieldy for larger, more complex systems.

- Code Reusability: Procedural code tends to be less reusable than object-oriented code, as functions are tightly coupled with specific data structures and procedures.

Overall, the procedural approach is a good fit for the bowling scoring problem due to its simplicity, modularity and efficiency, making it a practical choice for implementing the required functionality.

2.2 Object-Oriented Programming

Object-oriented programming (OOP) is a paradigm that models software components as objects, each encapsulating data and behavior. In this approach, the problem is structured around classes and objects, with inheritance allowing for the creation of hierarchical relationships between classes. For the bowling scoring problem, an object-oriented approach involves creating classes to represent frames, with inheritance used to handle the differences between normal frames and the final frame.

Frame Class Hierarchy: The object-oriented solution begins by defining a base class Frame representing a generic frame in bowling. This class encapsulates data such as the rolls and provides methods for calculating the score. Then, subclasses NormalFrame and FinalFrame inherit from the Frame class, each implementing specific behavior related to scoring and handling strikes, spares, and bonus rolls. There is also a class BowlingGame that deals with creating the frames and calculating the total score of the game.

Inheritance: Inheritance is used to handle the special case of the final frame, which may have up to three rolls and additional scoring rules. By defining a separate subclass FinalFrame that inherits from Frame, the program can encapsulate the unique behavior of the final frame without duplicating code.

This object-oriented approach fits the problem well for several reasons:

- **Modularity and Reusability**: OOP promotes modularity by encapsulating data and behavior within objects, making it easier to reuse and extend code. By defining classes for frames and leveraging inheritance, the code becomes more modular and reusable.

- **Abstraction and Encapsulation**: OOP allows for abstraction and encapsulation of concepts, making the code easier to understand and maintain. The Frame class hierarchy abstracts away the details of individual frames, providing a clear interface for calculating scores.

- **Hierarchy for Special Cases**: Inheritance is well-suited for handling the differences between normal frames and the final frame. By defining subclasses and leveraging inheritance, the code can accommodate the unique scoring rules and behavior of the final frame without cluttering the base Frame class.

However, there are some potential drawbacks to consider:

- **Complexity**: Object-oriented solutions can introduce complexity, especially for smaller problems like bowling scoring. While inheritance provides flexibility, it may also add unnecessary overhead for simpler problems.

- **Overhead of Class Hierarchies**: Defining a class hierarchy for frames may introduce overhead compared to a procedural approach, especially if the problem does not require the use of inheritance.

Overall, the object-oriented approach is a good fit for the bowling scoring problem due to its modularity, reusability, and support for handling special cases like the final frame. While it may introduce some additional complexity, particularly in defining class hierarchies, these drawbacks are outweighed by the benefits of encapsulation and abstraction offered by OOP. Object-oriented programming enhances scalability, making it easier to manage and extend large and complex projects, allowing for more efficient development and maintenance as the project grows over time.

2.3 Pipes and Filters

The pipes and filters architectural pattern involves breaking down a task into smaller, independent components (filters) that can process data sequentially through a series of interconnected pipes. Each filter performs a specific operation on the input data and passes the result to the next filter via pipes. In the context of the bowling scoring problem, this approach entails creating separate programs (filters) to handle input parsing, frame separation, and score calculation, with pipes connecting these programs to facilitate data flow.

I created three programs where each one addresses a part of the problem:

1. **read_input.py**: The first program reads the input from a file and prepares it for further processing. This filter is responsible for parsing the input data and formatting it in a way that can be easily consumed by the subsequent filters. It separates the rolls into a list and passes them through the pipe to the next filter.
2. **frame_separator.py**: The second program receives the parsed input from the first filter via pipes and separates it into frames. It identifies the boundaries between frames and groups the rolls accordingly, considering the special rules for the final frame. This filter then passes the frames through the pipe to the next filter for scoring calculation.
3. c**alculate.py**: The third program calculates the total score based on the frames received from the previous filter via pipes. It applies the scoring rules of bowling, including handling strikes, spares, and bonus rolls in the final frame. Once the calculation is complete, this filter outputs the final score.

The advantages of the pipes and filters approach are:

- **Modularity**: The problem is decomposed into smaller, independent filters, each responsible for a specific task. This modular design makes it easier to understand, test, and maintain each component separately, promoting code reusability and scalability.

- **Flexibility**: Pipes facilitate the exchange of data between filters, allowing for flexibility in the design and composition of the solution. Filters can be combined and rearranged in different configurations to accommodate changes in requirements or optimize performance.

  However, there are some potential drawbacks to consider:

- **Complexity**: Managing the communication and data flow between filters via pipes can introduce complexity, especially for developers who are not familiar with this architectural pattern. Careful design and coordination are required to ensure the correct sequencing and interaction of filters.
- **Overhead**: Introducing pipes and filters may add overhead in terms of communication and data transfer between filters, especially if the data volume is large. Efficient data handling and optimization techniques are needed to mitigate this overhead and ensure optimal performance.

The pipes and filters approach may indeed overcomplicate the solution for the bowling scoring problem, especially given its relatively small size and straightforward nature. While the pipes and filters architectural pattern can offer benefits such as modularity and flexibility, it may not be the most practical or efficient choice for solving this particular problem.

In a problem like bowling scoring, where the task involves reading input from a file, parsing the data, and performing calculations based on a set of rules, a simpler approach such as procedural programming or object-oriented programming would likely suffice. These traditional programming paradigms are more straightforward to implement, making them better suited for smaller-scale problems like this.

3 Conclusion

In conclusion, for the bowling scoring problem, the best fit among the three architectural approaches (procedural programming, object-oriented programming (OOP), and pipes and filters) is subjective and depends on various factors such as the project's size, complexity, and requirements.

However, considering the relatively small scale and straightforward nature of the problem, a procedural approach emerges as the most practical choice. Procedural programming offers simplicity, directness, and ease of implementation, making it well-suited for tasks like input parsing, frame separation, and score calculation. Its linear and step-by-step execution aligns closely with the sequential nature of the problem, providing a clear and intuitive solution without unnecessary overhead.

Through this exercise, several key insights about software architecture have been gained.

Starting with the importance of selecting the appropriate architectural pattern based on the specific characteristics of the problem at hand. Different architectural patterns offer distinct advantages and trade-offs, and understanding their strengths and weaknesses is crucial for making informed design decisions.

The assignment demonstrates the value of considering factors such as complexity, scalability and maintainability when evaluating architectural choices. By carefully weighing these factors and selecting the most suitable architectural pattern, developers can effectively address the requirements of a given problem while ensuring the long-term success and sustainability of the software solution.