Design and Implementation Report

The project is designed to manage the arrival of new animals at a zoo. It reads data from a text file containing animal arrival records, assigns a random name to each animal based on its species, and updates a report file with the new entries. The project utilizes three primary text files:

- **animalNames.txt:** Contains lists of names for different species (e.g., Hyena, Lion).
- **arrivingAnimals.txt:** Contains the details of arriving animals, with each record formatted as six comma-separated fields.
- **newAnimals.txt:** The output report file where the updated records are appended.

Design Choices

The Animal struct is used to encapsulate all relevant information about an animal. This organizes the data and makes it easier to pass between functions.
The functionality is divided into several functions:

    - **loadAnimalNames:** Reads and parses animal names from the names file.
    - **loadArrivingAnimals**: Reads and parses arrival records.
    - **assignName:** Matches a species with a random name from the names file.

This modular design increases code readability and makes debugging or future modifications easier. Basic error handling is implemented for file operations to ensure that the program reports if a file cannot be opened or read. This is essential for diagnosing issues during runtime.
The **assignName** function utilizes rand() (seeded with the current time) to randomly choose a name from the available names for the given species. This simulates a more dynamic assignment of names.

Implementation Details

The arrivingAnimals.txt file is expected to have each record on a single line with exactly six comma-separated values. The first field must be in the format "age species" (e.g., "4 Hyena") to ensure proper parsing. The report file has been renamed to "newAnimals.txt" as required. The program appends new records to this file, which allows for cumulative updates over multiple runs.

Conclusion

This project demonstrates a modular approach to processing animal arrival records in C++. The design emphasizes readability, maintainability, and proper error handling. The included

comments and this report detail the key design choices and implementation strategies used, fulfilling the submission requirements for both source code documentation and a design report.