URL Shortener Using Data Structures (C++ Terminal-Based Project)

1. Introduction

The URL Shortener project is a mini Data Structures and Algorithms (DSA) project implemented in C++. The objective of this project is to design and implement a system that converts long URLs into shorter, unique codes and allows retrieval of the original URL using the shortened version. This project recreates the core logic behind popular URL shortening services such as Bitly and TinyURL using fundamental data structures and algorithms. Unlike web-based implementations, this project is fully terminal-based, focusing on algorithmic and structural design.

2. Objectives

• Implement a URL shortening system using C++. • Apply hashing for efficient storage and retrieval. • Implement Base62 encoding algorithm manually. • Demonstrate understanding of unordered maps. • Create a menu-driven terminal application. • Practice modular programming in C++.

3. System Overview

The system consists of three main components: 1. Base62 Encoding Module 2. URL Shortener Class 3. Main Terminal Interface Workflow: 1. User enters a long URL. 2. System generates a unique numeric ID. 3. ID is converted into a Base62 encoded string. 4. Short code is stored in a hash map. 5. User retrieves the original URL using the short code.

4. Data Structures Used

The core data structure used is unordered_map. Key → Short Code Value → Original URL Hash tables provide average O(1) insertion and search time, making them efficient for URL lookup systems.

5. Algorithms Used

Base62 Encoding Algorithm: 1. Take a numeric ID. 2. Divide the number by 62. 3. Store the remainder. 4. Map remainder to Base62 character. 5. Repeat until number becomes 0. 6. Reverse the result string. Time Complexity: O(log■■ N)

6. Features

• Terminal-based interface • Unique short code generation • Fast retrieval using hashing • Modular project structure • Error handling for invalid input

7. Time and Space Complexity

Shortening a URL: - Hash table insertion: O(1) - Base62 conversion: O(log N) Retrieving a URL: - Hash lookup: O(1) Space Complexity: - O(N), where N is number of stored URLs.

8. Limitations

• Data is not persistent (lost after program exits). • No database integration. • No authentication system. • No custom alias feature.

9. Future Improvements

• Add file-based persistent storage. • Add click tracking counter. • Add expiration time for URLs. • Develop GUI or web-based interface. • Deploy as backend service.

10. Conclusion

This project successfully demonstrates practical implementation of core data structures and algorithms using C++. Hashing ensures constant-time retrieval, while Base62 encoding efficiently generates compact short codes. The project strengthens understanding of hashing, encoding, algorithm efficiency, and modular programming, serving as a strong foundation for backend system development.