Project Report: Morse to English and English to Morse Translator

## Introduction

The Morse to English and English to Morse Translator project represents a practical implementation of a Morse code translation program using C++. This comprehensive project report delves into the extensive use of C++ features and object-oriented programming (OOP) concepts in this project. It provides an in-depth analysis of classes, polymorphism, inheritance, function overloading, constructors, and data structures, offering a comprehensive view of the development process.

Features and Concepts Implemented:

**1. Object-Oriented Programming**

Object-oriented programming (OOP) is a programming paradigm that revolves around the concept of objects. It enables the modeling of real-world entities in code by encapsulating data and behavior within classes. In this project:

Classes: The core of the project revolves around classes. Two main classes, MorseSource (the base class) and TextualMorseSource (a derived class), were defined to structure and organize the code efficiently. Classes facilitate the bundling of attributes and methods into coherent units, enhancing code modularity and maintainability.

Class Inheritance: Inheritance, a fundamental OOP concept, is employed in this project. The TextualMorseSource class inherits from the MorseSource base class. Inheritance allows the derived class to inherit attributes and behaviors from the base class. This hierarchical relationship promotes code reuse and supports the extension of functionality while maintaining a consistent interface.

**2. Classes and Inheritance**

The use of classes and inheritance is central to the project:

Base Class (MorseSource): The MorseSource class serves as the base class for both translation directions, Morse to English and English to Morse. It contains essential member variables: morseCode and englishText. These variables store the input and translation results, respectively. The base class encapsulates shared attributes and methods that apply to both translation directions.

Derived Class (TextualMorseSource): The TextualMorseSource class is derived from MorseSource. It extends the base class by adding specialized functionality for translating Morse code to English and vice versa. This relationship allows for code reusability and a clear separation of concerns.

**3. Polymorphism**

Polymorphism is a hallmark of OOP, enabling the use of objects of different classes through a common interface. In this project:

Virtual Function: Polymorphism is realized using virtual functions. The base class, MorseSource, declares a pure virtual function, virtual void translate() = 0;. The pure virtual function serves as a contract that derived classes must adhere to by providing concrete implementations. The virtual function translate provides a consistent interface for translation across different types of textual sources.

Function Overloading: The TextualMorseSource class exemplifies function overloading. Multiple versions of the translate function are implemented within the derived class. Each overloaded function accepts distinct parameters, allowing for seamless translation between Morse and English. This method simplifies the user experience by offering a single, intuitive entry point for both translation directions.

**4. Constructors**

Constructors are pivotal in object-oriented programming, responsible for initializing objects. In this project:

Base Class Constructor (MorseSource): The base class, MorseSource, features a versatile constructor that accepts an optional parameter, code. This parameter initializes the morseCode or englishText member variables based on the input. Constructors are indispensable for setting the initial state of objects, ensuring that objects are valid and ready for use upon creation.

Derived Class Constructors (TextualMorseSource): The TextualMorseSource class encompasses constructors that mirror those of the base class, providing similar optional code parameters. These constructors not only initialize the inherited members but also invoke the base class constructor to establish a consistent state. Constructor chaining ensures that the object is correctly initialized before any operation.

**5. Data Structures**

The efficient management and storage of data are crucial in any program. In this project:

Maps (std::map): Maps play a pivotal role in the project. Two maps are employed: morseToEnglish and englishToMorse. morseToEnglish maps Morse symbols to corresponding English characters, while englishToMorse maps English characters to Morse symbols. These mappings are pivotal for accurate translation in both directions. Maps are chosen due to their key-value pair structure, allowing for fast and efficient lookups.

Vectors (std::vector): Vectors are used as a flexible data structure for storing pairs of Morse symbols and their corresponding English characters. These pairs are employed during the initialization of the morseToEnglish map. Vectors enable dynamic resizing and efficient element access, ensuring that the mapping process is both adaptable and resource-efficient.