# Student Information :

Name & Roll no. : 23BCE148 Krish Bhoraniya

23BCE149 Krish Patel

Subject : Data Structures (2CS501)

Topic : Morse Code Decoder/Encoder

**Project Overview**

This project involves the creation of a Morse code encoder and decoder using C++. The program allows the user to input a plain text message to encode into Morse code or input Morse code to decode it back into plain text. This Morse code system is implemented using a binary tree structure, with dots (.) and dashes (-) determining the paths to characters.

**Functional Details**

1. **Encode Function**:
   * Converts a user-provided plain text message into Morse code.
   * Uses a hash table (unordered map) to store Morse code representations of each character.
   * Each character is translated into Morse code, forming a single encoded string with characters separated by spaces, and words separated by a slash (/).
2. **Decode Function**:
   * Converts Morse code input back into plain text.
   * Traverses a binary tree structure using dots (.) and dashes (-) to locate the corresponding character for each Morse code symbol.

**Program Structure**

1. **Classes and Data Structures**:
   * **Node Class**: Defines a node in the Morse code tree.
     + Each node contains:
       - data: Character represented by the node.
       - dot: Pointer to the left child node (corresponding to a . in Morse code).
       - dash: Pointer to the right child node (corresponding to a - in Morse code).
   * **Binary Tree Structure**: Used to store and traverse Morse code.
     + A root node is created, with each left (dot) and right (dash) child representing subsequent dots and dashes for Morse characters.
   * **Unordered Map (Hash Table)**: Used for encoding plain text into Morse code.
     + Stores mappings of each character (A-Z, 0-9, and space) to its Morse code equivalent for fast look-up during encoding.

**Data Structures Used**

**1. Binary Tree:**

* A binary tree is used to represent the Morse code chart, where each character's path is represented by a sequence of dots and dashes.
* **Binary Tree Structure**:
  + The root node represents the starting point (no character).
  + Each left child represents a dot (.), while each right child represents a dash (-).
  + For example, .- (A in Morse) is represented by moving left for dot (.) to reach node E, then right for dash (-) to reach node A.
* **Traversal**: During decoding, each Morse code sequence guides a traversal of the tree from the root node, interpreting dots as left moves and dashes as right moves. Once a space is encountered, the character at the current node is added to the decoded message, and the traversal restarts at the root for the next character.
* **Advantages**:
  + Provides efficient, hierarchical representation of Morse code.
  + Allows direct traversal to characters with simple dot-dash rules, without searching through a list.

**2. Hash Table (Unordered Map):**

* An unordered map is used to store character-to-Morse code mappings for fast lookup during encoding.
* **Structure**:
  + Each entry in the hash table has a character as the key and its Morse code string as the value.
* **Advantages**:
  + Provides constant-time complexity for lookups (O(1)) in most cases, allowing efficient encoding of each character.
  + Reduces the need to traverse a tree for encoding, simplifying and speeding up the encoding process.

**Functions and Implementation**

**1. buildTree()**

* Creates the Morse code binary tree.
* Each node is initialized with a specific character (letter or number) and linked according to Morse code rules (dots go left, dashes go right).
* Returns the root node of the constructed binary tree.

**2. Encode()**

* Takes a plain text string as input and converts it to Morse code.
* For each character in the input:
  + Converts the character to uppercase.
  + Looks up the character in the hash table (morseCode).
  + Appends the Morse code string to the final encoded string.
* Displays the encoded Morse code string.

**3. Decode()**

* Takes a Morse code string as input and converts it back to plain text.
* Traverses the Morse code binary tree based on dots and dashes:
  + Moves to the left child for a dot (.) and the right child for a dash (-).
  + When encountering a space or slash (/), records the character at the current node, appends it to the decoded message, and resets the traversal to the root node.
* Displays the decoded plain text string.

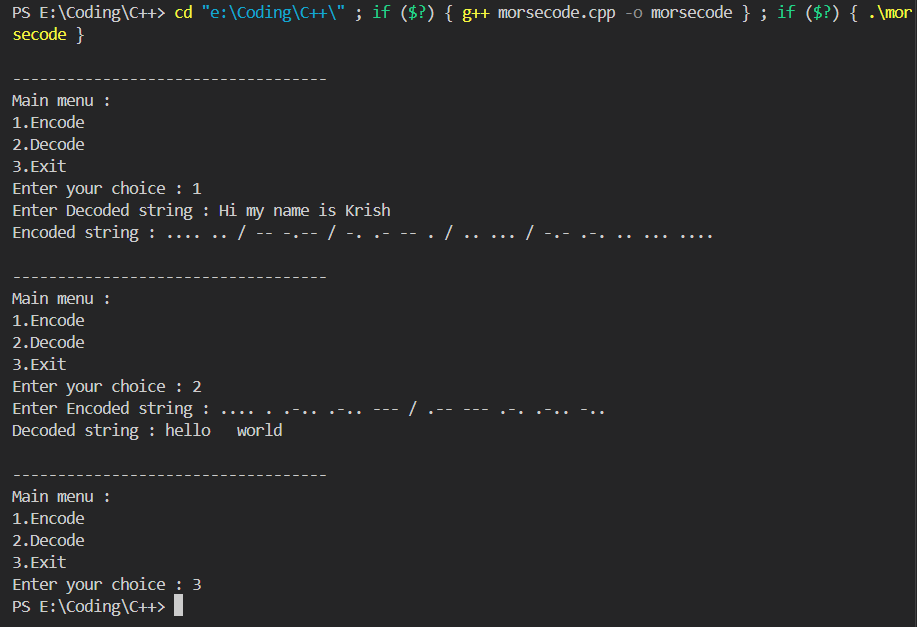
**Complexity Analysis**

1. **Encoding**:
   * Time Complexity: O(n), where nnn is the number of characters in the input string. Each character is mapped in constant time using the hash table.
   * Space Complexity: O(m), where mmm is the number of Morse mappings (characters) in the hash table.
2. **Decoding**:
   * Time Complexity: O(n) where n is the number of Morse code segments (characters) in the input string. Each character requires a traversal in the binary tree, which has a maximum height of around 5 (since Morse code paths are short), so each traversal is O(1) in practice.
   * Space Complexity: O(h) for the binary tree, where h is the number of unique Morse code characters and digits.

**Conclusion**

This Morse code encoder and decoder program demonstrates the efficient use of a binary tree for decoding and a hash table for encoding. The binary tree represents Morse code in a structured, traversable format, while the hash table enables quick character-to-code mappings. Together, these data structures optimize the Morse code translation process for both encoding and decoding. This approach could further be expanded to handle real-time Morse decoding or translation for larger text-based applications.

**Example program output :**

****