DEPARTMENT OF

COMPUTER SCIENCE AND ENGINEERING

Mini-Project Report

Data Structures
(CSE332P)

B. Tech Degree – 3 BTCS Section C

School of Engineering and Technology,

CHRIST (Deemed to be University),

Kumbalagodu, Bengaluru-560 074

September 2025

CHRIST
(DEEMED TO BE UNIVERSITY)
BANGALORE · INDIA

Certificate

This is to certify that HEMISH GOGINENI (2460372), MATHEW THOMAS (2460405), JOEL BIJU KAINIKAT (2460379) has successfully completed the Mini Project work for Data Structures –CSE332P in partial fulfillment for the award of Bachelor of Technology during the academic year 2025-2026.

FACULTY- INCHARGE

Name : - 1. HEMISH GOGINENI (2460372)
2. MATHEW THOMAS (2460405)
3. JOEL BIJU KAINIKAT (2460379)

TITLE

MORSE CODE TRANSLATOR USING BINARY TREE

Acknowledgement

We would like to express our sincere gratitude to Dr. Deepa Yogish for her valuable guidance, encouragement, and support throughout the development of our project, “Morse Code Translator (Encode & Decode) using C Language”. Her insights into programming concepts greatly helped us in successfully completing this work.

We also thank our institution for providing us with the necessary resources and environment to carry out this project. We are grateful to our families and friends for their constant motivation and support during this journey. Finally, we appreciate the cooperation and dedication of all our team members, whose collective effort made this project possible.

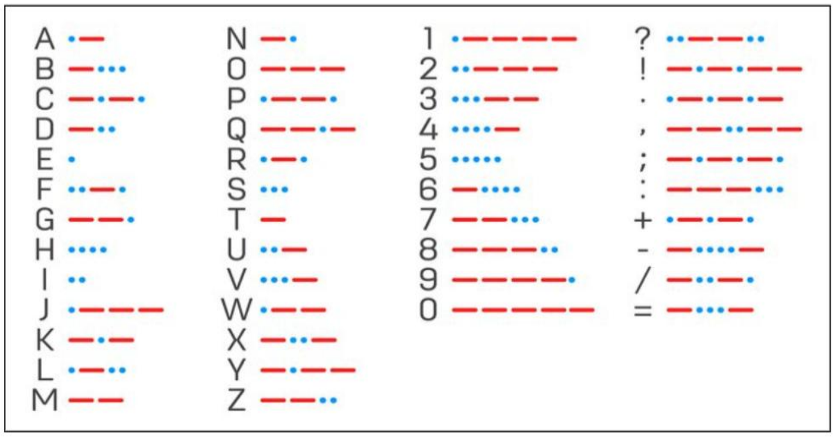
INDEX

INTRODUCTION

Communication is a fundamental aspect of human life, enabling the exchange of information, ideas, and emotions. Among the earliest methods of long-distance communication is **Morse Code**, developed in the 19th century by Samuel Morse and Alfred Vail. Morse Code translates letters, numbers, and some punctuation into sequences of **dots (.) and dashes (-)**, allowing messages to be transmitted through sound, light, or electrical signals. Despite being over 150 years old, Morse Code remains relevant in aviation, maritime communication, military operations, and amateur radio, especially in situations where verbal or digital communication is not feasible.

In computer science, Morse Code provides a practical example of applying **binary tree data structures**. A binary tree is a hierarchical structure where each node can have at most two children. In this project, dots correspond to the **left child** and dashes correspond to the **right child**, creating a clear and systematic representation of Morse sequences. Each path from the root to a leaf uniquely identifies a character, making decoding accurate and efficient.

The **Morse Code Translator project** is implemented in the C programming language and supports both encoding (text to Morse) and decoding (Morse to text). It serves as a hands-on exercise in understanding tree traversal, dynamic memory management, and modular programming. By combining historical communication techniques with modern programming concepts, the project not only reinforces theoretical knowledge but also demonstrates a real-world application of data structures, highlighting their relevance in practical problem-solving.



A	.-	N	-.	1	.----	?	..--..
B	-...	O	---	2	..---	!	-.-.--
C	-.-.	P	.--.	3	...--	.	.-.-.-
D	-..	Q	--.-	4	....-	,	--..--
E	.	R	.-.	5	.....	;	-.-.-.
F	..-.	S	...	6	-....	:	---...
G	--.	T	-	7	--...	+	.-.-.
H	....	U	..-	8	---..	-	-....-
I	..	V	...-	9	----.	/	-..-.
J	.---	W	.--	0	-----	=	-...-
K	-.-	X	-..-				
L	.-..	Y	-.--				
M	--	Z	--..				

OBJECTIVE AND RELEVANCE

Objective of the Project :

1. To develop a C program that can encode text into Morse code and decode Morse code into readable text.
2. To apply data structures concepts, specifically binary trees, in a practical scenario.
3. To enhance understanding of tree traversal techniques (pre-order, in-order, post-order) through a real-world application.
4. To provide a simple tool for communication using Morse code, which has historical and educational significance.

Relevance of the Project :

1. Educational Value: Demonstrates how binary trees can be applied for encoding and decoding binary-based information.
2. Practical Understanding: Helps students visualize traversal in binary trees and how it maps to real-world binary data (dots and dashes).
3. Programming Skills: Strengthens C programming skills through implementation of structures, pointers, and recursion.
4. Historical Significance: Morse code is a foundational communication method, so this project gives context to how data encoding works.
5. Foundation for Advanced Projects: Concepts learned here can be extended to Huffman coding, data compression, and other binary tree applications.

CONCEPTS APPLIED

The primary concept applied in this project is the **Binary Tree**, a fundamental data structure in computer science. A binary tree is a hierarchical structure where each node has at most two children, commonly referred to as the left and right child. This property makes it ideal for representing systems with **binary decisions**, such as Morse Code, where each symbol is either a dot (.) or a dash (-).

In this project, the Morse Code Translator uses a binary tree to store the entire alphabet. The **root node** of the tree serves as the starting point. Each dot in a Morse sequence directs the traversal to the **left child**, while each dash directs it to the **right child**. By following this path, the tree encodes the relationship between each Morse sequence and its corresponding character. For example, the letter E (represented by a single dot) is stored as the left child of the root, while T (represented by a dash) is the right child. More complex letters, like Q (--.-), are stored deeper in the tree.

This structure makes **decoding efficient** because traversing the tree according to the sequence of dots and dashes always leads to the correct character. For encoding, a **lookup table** is used to map characters directly to Morse sequences, providing fast access without tree traversal.

Additionally, the project demonstrates **modular programming** by separating the code into functions for node creation, insertion, encoding, and decoding. It also introduces concepts like **dynamic memory allocation**, string manipulation, and traversal algorithms, reinforcing practical application of **Data Structures and Algorithms (DSA)**.

In conclusion, the binary tree is the most suitable structure for this project because it naturally aligns with the **binary nature of Morse Code**, providing clarity, efficiency, and scalability. This concept bridges theoretical knowledge with a practical, real-world communication system.

Advantages of Using Binary Tree for Morse Code Translator

1. **Binary Nature Alignment**
   - Morse code is inherently binary: each symbol is either a dot (.) or a dash (-).
   - Binary trees have two children per node, making them a **natural fit** for representing dot-dash sequences.
2. **Efficient Decoding**
   - Traversing the binary tree along dot-dash paths guarantees reaching the correct character.
   - Each Morse sequence corresponds to a **unique path from root to leaf**, ensuring accurate and deterministic decoding.
3. **Scalability**
   - New symbols (numbers or punctuation) can be added easily by extending the tree without restructuring existing nodes.
4. **Memory Efficiency**
   - Only two pointers per node are needed, minimizing memory usage compared to more complex tree structures.
5. **Ease of Implementation**
   - Simple tree traversal logic (left for dot, right for dash) makes coding straightforward.
6. **Modularity**
   - Encoding uses a table lookup, while decoding uses tree traversal, separating concerns and enhancing maintainability.

WHY BINARY TREE OVER OTHER TREES ?

(BINARY TREE VS AVL,BST,SPLAY,etc)

1. **Morse Code Structure Matches Binary Tree**
    - Morse code is inherently **binary**: each symbol is either a **dot (·)** or a **dash (−)**.
    - A **binary tree** naturally represents this:
        - Left child → dot (·)
        - Right child → dash (−)
    - Other tree types (like n-ary trees) are unnecessary because Morse only has two options per step.
2. **Simple Traversal**
    - To decode a Morse code sequence, we **traverse the tree** following the dot/dash path.
    - **Binary tree traversal** (pre-order, in-order, post-order) is simple and efficient for this task.
    - Using a general tree would complicate traversal since it can have multiple children, requiring additional checks.
3. **Efficient Encoding and Decoding**
    - **Encoding:** Can use a **lookup table** for direct letter → Morse mapping.
    - **Decoding:** Binary tree allows **step-by-step traversal** for dot/dash to get the letter.
    - Other trees wouldn't provide a clear one-to-one mapping without extra logic.

4. **Memory Efficiency**
    - Binary trees only need **two pointers per node** (left and right), which is minimal.
    - N-ary trees or general trees would require **dynamic arrays or multiple pointers**, wasting memory unnecessarily.
5. **Ease of Implementation in C**
    - C handles **binary trees very efficiently** using struct with two pointers.
    - Implementing an n-ary tree for Morse would require extra complexity in memory management and traversal logic.
6. **Scalability for Morse Code**
    - Morse code has a **fixed alphabet** (A–Z, 0–9, some punctuations), so a **binary tree perfectly fits** without needing a more complex structure.
    - Other tree types are better for large, variable-branching data, which Morse code doesn't need.

A **binary tree** is the most natural and efficient choice for a Morse code translator in C because it directly mirrors the binary nature of Morse code, simplifies decoding, uses memory efficiently, and is easy to implement. Other tree types are either overkill or unnecessarily complex for this task.

ALGORITHM AND PESUDOCODE

The Morse Code Translator project operates in **two primary modes**: **Encoding (Text → Morse)** and **Decoding (Morse → Text)**. Each mode applies a different approach, but both rely on structured data handling and algorithmic traversal.

Encoding Algorithm (Text to Morse) :

1. Accept a text input from the user.
2. Convert all characters to uppercase for uniformity.
3. For each character in the input string:
    - If the character is a letter or digit, retrieve its Morse code using a lookup table.
    - Print the Morse code sequence followed by a space.
    - If the character is a space, print / to indicate a word separator.
    - For any unrecognized character, print ? as a placeholder.
4. Repeat until all characters in the input are processed.

Pseudocode for Encoding :

```
function encode(text):
   for each character c in text:
      if c is space:
         print "/"
      else:
         morse = lookup(c)
         if morse exists:
```

```
        print morse + " "
    else:
        print "? "
```

Decoding Algorithm (Morse to Text) :

1. Accept Morse code input from the user, with letters separated by spaces and words separated by /.
2. Split the input into individual Morse sequences.
3. For each Morse sequence:
    - Start at the root of the binary tree.
    - For each symbol in the sequence:
        - Move to the left child if the symbol is a dot (.).
        - Move to the right child if the symbol is a dash (-).
    - After traversing the sequence, print the character stored in the node.
    - If a / is encountered, print a space to separate words.
    - If traversal reaches a null node, print ? for an invalid sequence.

Pseudocode for Decoding :

```
function decode(morse):
    split morse by spaces into tokens
    for each token:
        if token == "/":
            print " "
        else:
```

```
node = root
for symbol in token:
    if symbol == ".":
        node = node.left
    else if symbol == "-":
        node = node.right
print node.character
```

The encoding algorithm uses a direct table lookup for simplicity and fast conversion of text to Morse code. In contrast, the decoding algorithm uses binary tree traversal, following the dot-dash paths to retrieve characters efficiently.

This approach ensures accuracy, scalability, and modularity, as each function handles a specific task, making the code easy to maintain and extend. The binary tree guarantees deterministic decoding performance, while the table provides quick encoding.

Overall, the design demonstrates how data structures and algorithms can be chosen to optimize efficiency, readability, and adaptability for real-world communication applications like Morse Code translation.

PROGRAM CODE

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

typedef struct Node {
    char ch;
    struct Node *dot, *dash;
} Node;

Node* newNode() {
    Node* n = (Node*)malloc(sizeof(Node));
    n->ch = 0;
    n->dot = n->dash = NULL;
    return n;
}

void insert(Node* root, const char* morse, char c) {
    Node* cur = root;
    int i;
    for (i = 0; morse[i]; i++) {
        if (morse[i] == '.') {
            if (!cur->dot) cur->dot = newNode();
            cur = cur->dot;
        } else {
            if (!cur->dash) cur->dash = newNode();
            cur = cur->dash;
        }
    }
    cur->ch = c;
}

typedef struct { char c; const char* m; } Map;
Map table[] = {
 {'A',".-"},{'B',"-..."},{'C',"-.-."},{'D',"-.."},
 {'E',"."},{'F',"..-."},{'G',"--."},{'H',"...."},
 {'I',".."},{'J',".---"},{'K',"-.-"},{'L',".-.."},
 {'M',"--"},{'N',"-."},{'O',"---"},{'P',".--."},
 {'Q',"--.-"},{'R',".-."},{'S',"..."},
 {'T',"-"},{'U',"..-"},{'V',"...-"},
 {'W',".--"},{'X',"-..-"},{'Y',"-.--"},{'Z',"--.."},
 {'0',"-----"},{'1',".----"},{'2',"..---"},
 {'3',"...--"},{'4',"....-"},{'5',"....."},
 {'6',"-...."},{'7',"--..."},{'8',"---.."},{'9',"----."}
};
int N = sizeof(table)/sizeof(table[0]);
Node* buildTree() {
   Node* root = newNode();
   int i;
   for (i = 0; i < N; i++) insert(root, table[i].m, table[i].c);
   return root;
}

#define MAX_MORSE_LEN 8   // longest Morse symbol length

// ----------------- DECODE (Morse -> Text) -----------------
void decode(Node* root, const char* line) {
   char buf[MAX_MORSE_LEN + 1];
   int bi = 0;
   int i, k;
   Node* cur;
   for (i = 0; line[i]; i++) {
      if (line[i] == ' ') {
         buf[bi] = 0;
         if (bi > 0) {
            cur = root;
            for (k = 0; buf[k]; k++)
               cur = (buf[k]=='.'?cur->dot:cur->dash);
            if (cur && cur->ch) printf("%c", cur->ch);
            else printf("?");
            bi = 0;
         }
      }
      else if (line[i] == '/') {
         printf(" ");  // word separator
      }
      else {
         if (bi < MAX_MORSE_LEN) buf[bi++] = line[i];
      }
   }
   // flush last buffer
   if (bi > 0) {
        buf[bi]=0;
        cur = root;
        for (k=0; buf[k]; k++)
            cur = (buf[k]=='.'?cur->dot:cur->dash);
        if (cur && cur->ch) printf("%c", cur->ch);
        else printf("?");
    }
    printf("\n");
}

// ----------------- ENCODE (Text -> Morse) -----------------
const char* encodeChar(char c) {
    int i;
    c = toupper((unsigned char)c);
    for (i=0; i<N; i++) {
        if (table[i].c == c) return table[i].m;
    }
    return NULL;
}

void encode(const char* text) {
    int i;
    const char* m;
    for (i=0; text[i]; i++) {
        if (text[i]==' ') {
            printf("/ ");   // word separator
        } else {
            m = encodeChar(text[i]);
            if (m) printf("%s ", m);
            else printf("? "); // unknown character
        }
    }
    printf("\n");
}

int main() {
    Node* root = buildTree();
    char input[256];
    printf("MORSE CODE TRANSLATOR (type EXIT to quit)\n");
    printf("Type 'D' for decode mode or 'E' for encode mode\n");
    printf("Letters separated by spaces, words by '/'\n\n");

    while (1) {
        printf("Enter command (D/E/EXIT): ");
        if (!fgets(input, sizeof(input), stdin)) break;

        input[strcspn(input, "\n")] = 0; // remove newline
        if (strcasecmp(input, "EXIT") == 0) {
            printf("Exiting program...\n");
            break;
        }
        if (strcasecmp(input, "D") == 0) {
            printf("Enter Morse code: ");
            if (!fgets(input, sizeof(input), stdin)) break;
            input[strcspn(input, "\n")] = 0;
            printf("Decoded text: ");
            decode(root, input);
        }
        else if (strcasecmp(input, "E") == 0) {
            printf("Enter text: ");
            if (!fgets(input, sizeof(input), stdin)) break;
            input[strcspn(input, "\n")] = 0;
            printf("Encoded morse: ");
            encode(input);
        }
        else {
            printf("Invalid command. Type 'D', 'E', or 'EXIT'.\n");
        }
    }

    return 0;
}
```

SAMPLE INPUT & OUTPUT SCREENSHOTS

Encoding Example (Text → Morse):

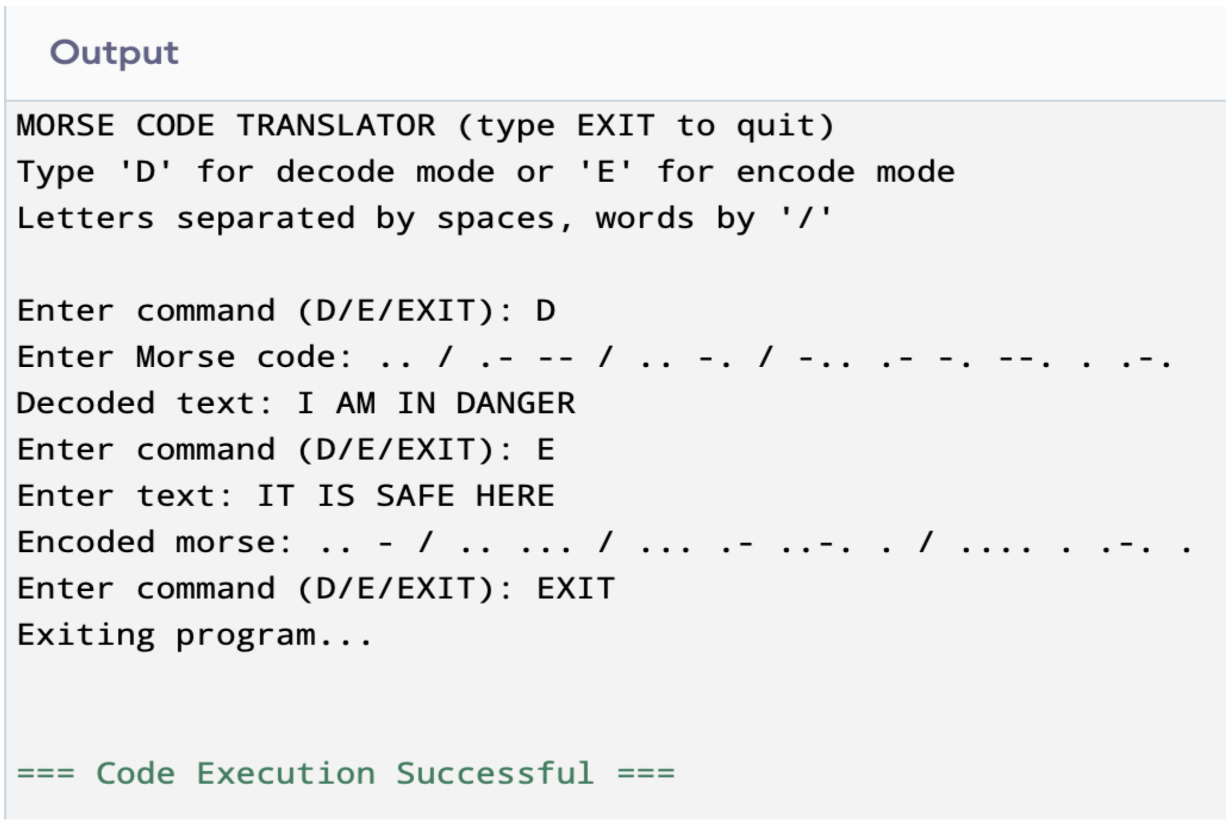
Input: HELLO WORLD

Output: .... . .-.. .-.. --- / .-- --- .-. .-.. -..

Decoding Example (Morse → Text):

Input: .... . .-.. .-.. --- / .-- --- .-. .-.. -..

Output: HELLO WORLD



Output

```
MORSE CODE TRANSLATOR (type EXIT to quit)
Type 'D' for decode mode or 'E' for encode mode
Letters separated by spaces, words by '/'

Enter command (D/E/EXIT): D
Enter Morse code: .. / .- -- / .. -. / -.. .- -. --. . .-.
Decoded text: I AM IN DANGER
Enter command (D/E/EXIT): E
Enter text: IT IS SAFE HERE
Encoded morse: .. - / .. ... / ... .- ..-. . / .... . .-. .
Enter command (D/E/EXIT): EXIT
Exiting program...


=== Code Execution Successful ===
```

RESULT AND CONCLUSION

The Morse Code Translator successfully demonstrates the **practical use of binary trees** for communication systems. It accurately converts **text to Morse code** and **Morse code to text**, handling multiple words and spaces efficiently.

Key Outcomes:

1. Reinforces **DSA concepts**: tree traversal, recursion, and dynamic memory.
2. Demonstrates **modular programming**, separating insertion, encoding, and decoding functions.
3. Validates **accuracy and reliability**: invalid Morse sequences are flagged with ?.
4. Combines **historical communication techniques** with modern programming.
5. Supports **interactive learning**, allowing users to encode/decode messages easily.

Conclusion:

The project bridges theory and practical application, highlighting how **data structures like binary trees** can efficiently model real-world problems. The design ensures **efficiency, scalability, and readability**, making it an ideal mini-project for learning programming and communication systems simultaneously.

REFERENCES

1. Samuel Morse and Alfred Vail, *Morse Code and Telegraphy*, 1844.
2. https://youtu.be/BqeX-4TshEA?si=tWVP8dvno3REOHvk
3. Kernighan, B. W., & Ritchie, D. M., *The C Programming Language*, 2nd Edition, Prentice Hall, 1988.
4. Online Morse Code Reference:https://morsecode.world/international/translator.html
5. GeeksforGeeks, “Morse Code Translator in C