MORSE

Program Developer's Guide

MUHAMMAD IBRAHIM SHOEB

BSc in Computer Science Engineering, BME
OZLVV3

OVERVIEW

This is a switch command line controlled program that handles Text to Morse and vice versa.

- It encodes Latin to Morse
- It decodes Morse to Latin
- Exit and saves a file named "history.txt" with decoding data.

— For ***Data Structure***, binary trees are made, and Structure “struct” is used to organize the data.

— ***File Handling*** is used to write the decoded morse statistics which saves the “history.txt” file.

— ***Functions*** are also used in every task to reuse the code multiple times just by calling it.

SOFTWARE AND HARDWARE REQUIREMENTS

- No special hardware requirements
- It can be compiled with any standard compiler
- This program is developed and tested in VS Code

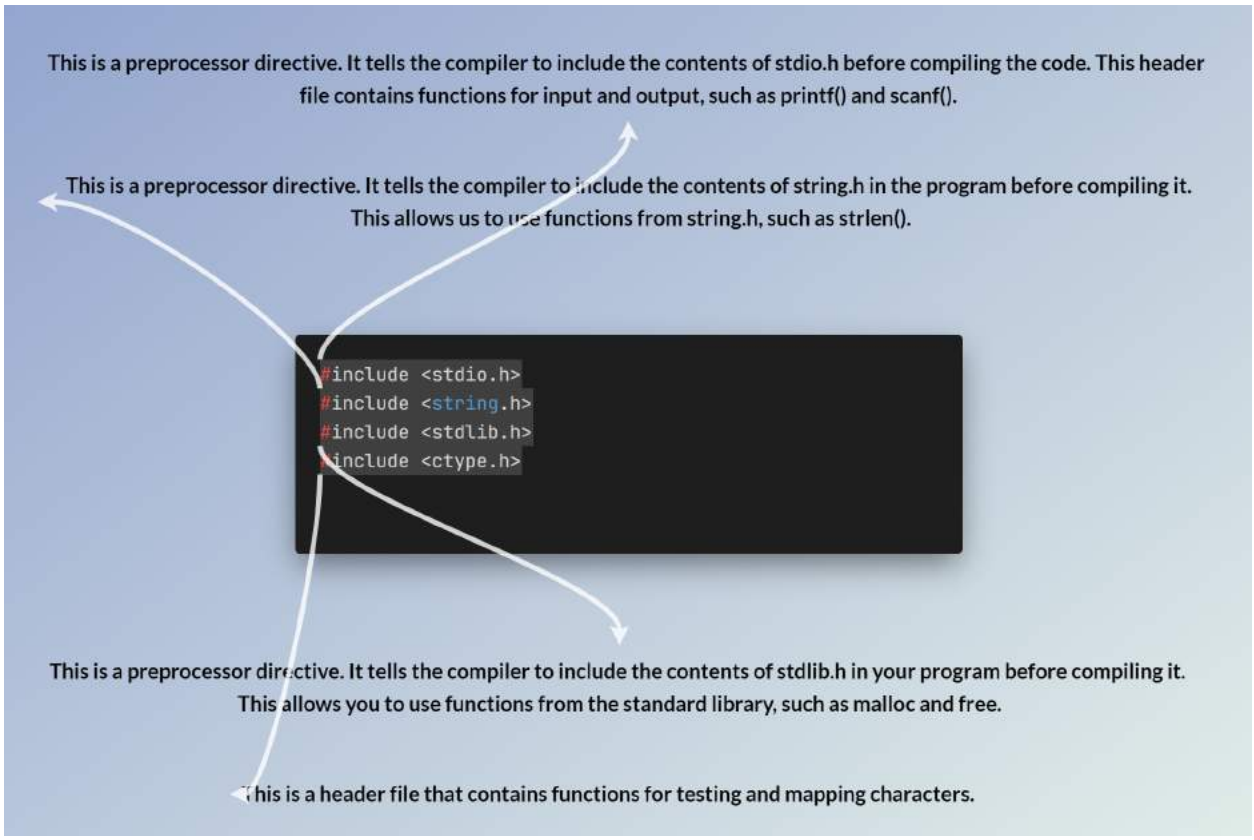
FILE FORMAT

This program uses a “.txt” file for writing the statistics of converted morse. It saves the “history.txt” file in the same folder where the program is written so a user can see the statistics whenever he wants to see that.

LIBRARIES

This program only contains C Standard libraries, no external libraries have been used.

Below are the libraries that are used in the program.



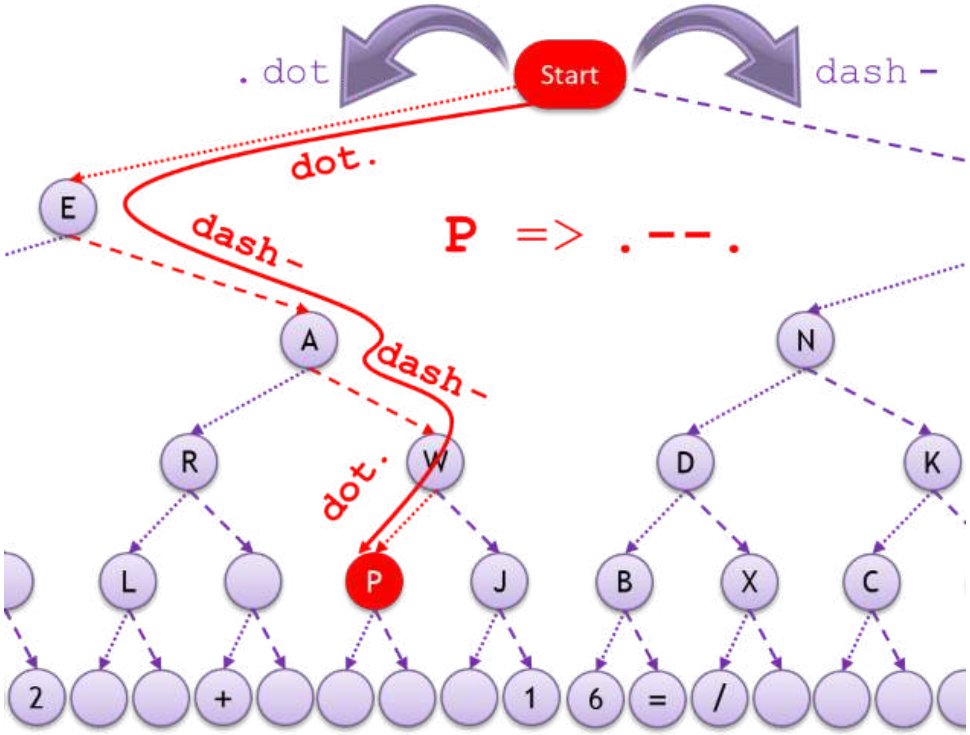
MAIN FUNCTION

The main is to to find the morse code is to use the “binary tree”, starting from the root of the tree (named “start” in the program), the flow of the tree connecting the root node to the desired character gives us the morse code.

Taking a few things into consideration

- Left branch corresponds to a dot (.)
- RIght branch corresponds to a dash (-)

Suppose we want the morse for the letter “p” in the binary tree which is “.--.”, below is the given diagram that shows how to find the character in the tree. (*image source: https://www.101computing.net/morse-code-using-a-binary-tree/*)

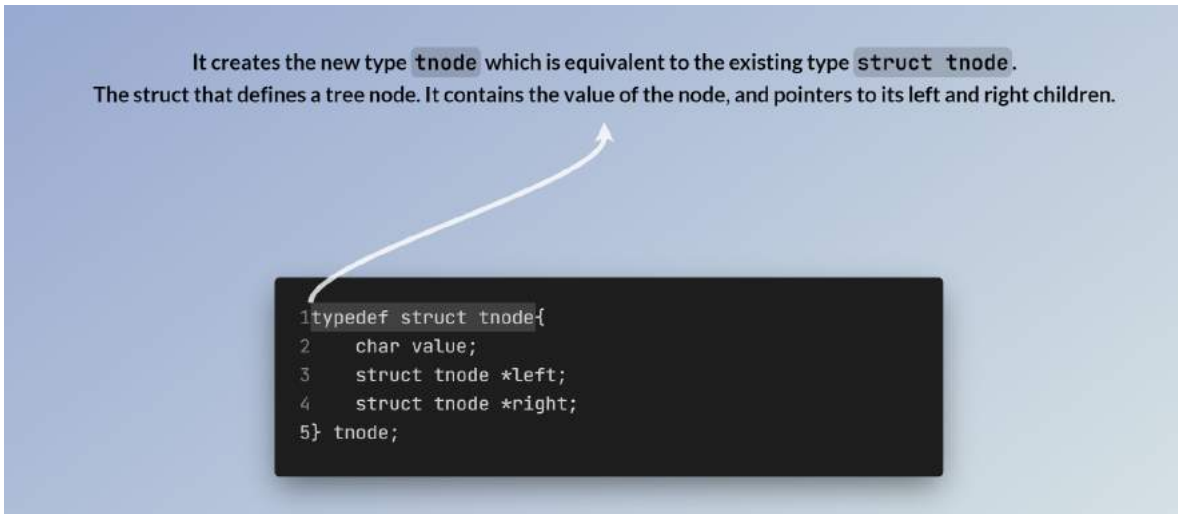


Steps:

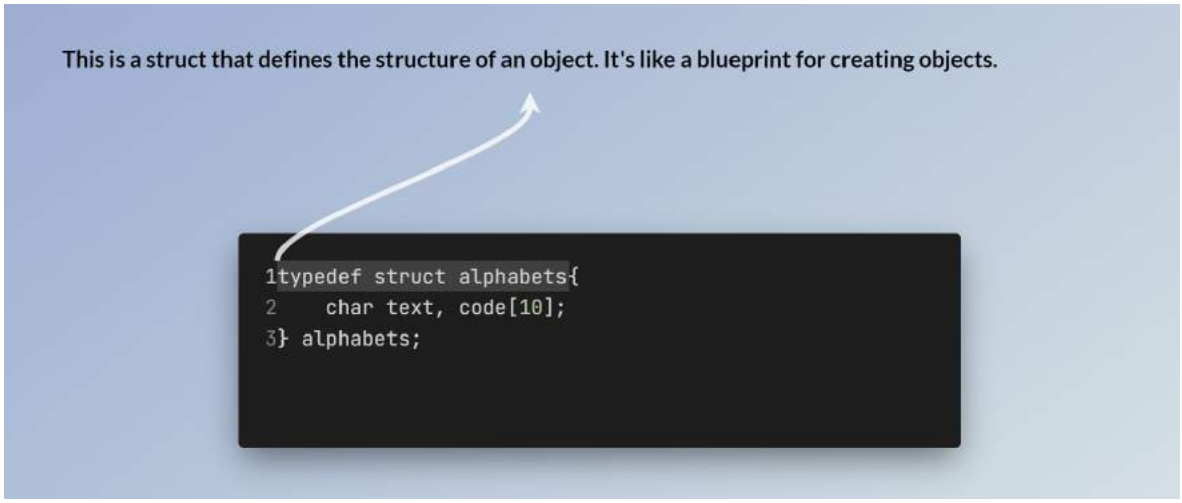
1. It receives a command parameter. It will create a struct letter array and initialize the counter array statistics and input character string.
2. It will build the binary tree as per the following explanations as above mentioned. We create the “start” node which will be the root and another temporary pointer named “hold” which will move through the tree.
3. It will make a loop for every character in a struct array. If the first morse is dot, it will create an empty node in the left. Of the node is dash, pointer will create a node on right and so on.
4. For the last morse element, it will create a node with the corresponding character value.
5. Switch case option gives user to select 3 options or cases to select according to which the user will have a menu and if a user selects all options and exits a program, it will create a “history.txt” file which will allow user to see the history of pervious decoded statistics of the morse.

STRUCTURES

1. **struct tnode:** which is basically a tree node. It contains a char variable which holds the value of each character node and two pointer (struct tnode *left, struct tnode*right) which points to the children of the parent node (root node). In this case there is left pointer and right pointer. Left points to dot and right points to dash.



2. **struct alphabets:** the struct we use in array containing the text and morse. It has char and strings.



FEW FUNCTIONS (other than the "*main*" function)

> NODE CREATION

```
tnode* new(char c);
```

This is the most important function which is used to build a binary tree. It is receiving the character as a parameter.

> TEXT

```
void text_to_morse(char c, char *text_convert[], char *num_convert[]);

void counts_text(int* counts_text_arr, char c);

void latin_statistics (int *counts_text_arr);
```

- **Function 1:** This is a function that converts text to morse code. It takes in three parameters: char c, char *text_convert[] and char *num_convert[]. The first parameter, c, is the character that will be converted to morse code. The second parameter, text_convert[], is an array of strings containing the alphabet in lowercase letters. The third parameter, num_convert[], is an array of strings containing numbers 0-9.
- **Function 2:** This is a function that takes in two parameters, an array of integers and a character. It counts the number of times each letter appears in the text file and stores it in an array.
- **Function 3:** This is a function that takes in an array of integers and returns the number of times each letter appears in the text.

> MORSE

```
int test_valid_morse(char *insert);

void morse_to_text (char *insert, tnode *start, int *counts_text_arr, int
*counts_morse_arr);

void counts_morse(char *arr, int *counts_morse_arr);

void morse_statistics (int *counts_morse_arr);
```

- **Function 1:** This is a function that takes in a string and returns an integer. The function checks if the string is valid morse code. If it is, then it returns 1,

otherwise 0.

- **Function 2:** This function is used to convert morse code into text. It takes in a string of morse code and converts it into text. The function also keeps track of the number of times each letter appears in the text and morse code, which is stored in two arrays that are passed by reference.
- **Function 3:** This function takes in a string and counts the number of times each letter appears in that string. It then stores the count for each letter in an array.
- **Function 4:** This is a function that takes in an array of integers and returns nothing. The function counts the number of times each letter appears in a text file, then stores the count for each letter in the array.

> **RELEASE MEMORY**

```
void freemem(tnode* start);
```

This is a function that takes in a pointer to the root of a tree and frees all memory associated with it.