Nama: Scholastica Celine Wahyudi

NIM: 2702214470

Kelas: LK01

This program has a restriction where the words entered has to be all lowercase.

Starting from the program’s initialization

#include <stdio.h> //Included for standard input and output

#include <string.h> //Included to be able to manipulate strings using strcpy, strcmp, and strcat

#include <stdlib.h> //Included to be able to do memory allocation

//Declaring colors for aesthetic purposes

#define RED "\x1b[31m"

#define RESET "\x1b[0m"

#define BLUE "\x1b[35m"

Struct and global variable initialization

const int size = 26; //The max size of the tree (26 is the amount of alphabets in English)

/\* Declaring a struct for the Trie

    It consists of the alphabets for each Node, description for the leaves of the tree, an integer that indicates if the current Node is a leaf or not, and a pointer to the next Node

\*/

struct Node {

    char alphabet;

    char desc[500];

    int data\_end;

    Node \*child[size];

};

Node \*root = NULL; //root as a global variable

Declaring functions in the start of the program

/\*

    Here I declare every function in this program so it can be accessed from everywhere in this file

\*/

Node \*createNode(char alphabet);

void insertNode(const char \*key, const char \*desc);

void inputSlang\_menu();

char \*searchFromTrie(const char \*key);

int exists(Node \*root, const char \*word);

void searchWord\_menu();

void searchPrefix(const char \*prefix, char \*temp, int height);

void searchPrefix\_menu();

void showMenu();

void printNode(Node \*root, char \*temp, int height);

void printAll\_menu();

Create Node function

/\*

    Creating a new Node. First we allocate the memory for newNode. Then we "construct" the alphabet and data\_end. Alphabet is used to contain every alphabet in the word and data\_end is to detect whether the current alphabet is a leaf or not. And like every tree data structure, we first make the child NULL to avoid any clashes

\*/

Node \*createNode(char alphabet){

    Node \*newNode = (Node\*) malloc(sizeof(Node));

    newNode->alphabet = alphabet;

    for(int i = 0; i < size; i++)

        newNode->child[i] = NULL;

    newNode->data\_end = 0;

    return newNode;

}