Nama: Scholastica Celine Wahyudi

NIM: 2702214470

Kelas: LK01

Boogle dictionary documentation

This program has a restriction where the words entered has to be an alphabet, no symbols or space allowed. This restriction happens because the size is limited to 52 only.

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
#include <ctype.h> //Included to be able to use tolower

//Declaring colors for aesthetic purposes
#define RED "\x1b[31m"
#define RESET "\x1b[0m"
#define BLUE "\x1b[36m"
```

Struct and global variable initialization

```
const int size = 52; //The max size of the tree (52 is the amount of alphabets in
English, both uppercase and lowercase)

/* 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

```c
/*
    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(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

```c
/*
    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;
}
```

Inserting the Node

```
/*Inserting a new node for the trie.
    Parameters explanation:
    - key: The string to be inserted into the trie.
    - desc: The description of the word.

    If the trie is still null, we can create the root of the trie with a star.
Then we create a temporary string and fill it with a null terminator. Then we
create a new Node, curr, to loop it inside the while(*key) statements. The index
variable is used to identify which index in the child of the current node, hence
why it's subtracted with 'a' or 'A', depending if the current character is in
uppercase or lowercase;

    Now if the word already exists, we only update the description. Otherwise,
it'll mark the last node as the end of the word and create a new description for
the word.
*/
void insertNode(const char *key, const char *desc) {
    if (root == NULL) root = createNode('*');
    char temp[100];
    temp[0] = '\0';
    int index = 0;

    Node *curr = root;
    while (*key) {
        if (islower(*key)) index = *key - 'a';
        else if (isupper(*key)) index = *key - 'A' + 26;

        if (!curr->child[index]) //if the child node doesn't exists, create a new
one
            curr->child[index] = createNode(*key);
        curr = curr->child[index];
        temp[0] = '\0';
        strncat(temp, key, 1); // Append current character to the temp variable
        key++;
    }

    if (exists(temp)) { //Checks if the word alreadt exists or not
        strcpy(curr->desc, desc);
    } else {
        curr->data_end = 1;
        strcpy(curr->desc, desc);
    }
}
```

The validation for inputting the slang and the description

```
/*
    This function is the main function of the first case. It contains inputs from
the user that are validated by the program. The validation is as follows:
    - The slang word: it will loop itself while the flag is equals to 1. The
length of the word must be more than one, contains no spave, numbers, or symbols
    - The description: it will loop itself while the input is less than one word.
*/
void inputSlang_menu() {
    int wordFlag = 0;
    int len;
    char inputSlang[100];
    char inputDesc[500];

    /* While the word flag is equals to 1, it'll repeat the input process*/
    do {
        printf("Input a new slang word [Must be more than 1 characters and
contains no space]: ");
        scanf("%[^\n]", inputSlang); getchar();
        len = strlen(inputSlang);

        if(len < 2){ //This section checks the length
            printf(RED "\nInput must be more than 1 character and have no
spaces\n" RESET);
            printf("Press enter to continue..."); getchar();
            system("cls");
            wordFlag = 1;
            continue;
        }

        for(int i = 0; i < len; i++){ //this section checks if there's space or
not
            if(inputSlang[i] == ' '){
                printf(RED "\nInput must be more than 1 character and have no
spaces\n" RESET);
                printf("Press enter to continue..."); getchar();
                system("cls");
                wordFlag = 1;
                break;
            } else if(!isalpha(inputSlang[i])){ //this section checks if the
input is an alphabet or not
                wordFlag = 1;
                printf(RED "\nInput must be all alphabets without numbers or
symbols\n" RESET);
```

```
                printf("Press enter to continue..."); getchar();
                system("cls");
                break;
            } else wordFlag = 0;
        }
    } while(wordFlag);

    /* This part creates two different versions of entering the description. One
for updating and one for adding. If the slang already exists, it'll print out the
update description prompt.*/
    if(exists(inputSlang)){
        system("cls");
        int isMoreThanTwo = 0;
        do { //checks if there's more than two words or not
            printf("Update the slang word description [Must be more than 2
words]: ");
            scanf("%[^\n]", inputDesc); getchar();
            len = strlen(inputDesc);

            for(int i = 0; i < len; i++){
                if(inputDesc[i] == ' ' && i != len - 1){ //If the current
iteration is a space and it's not empty next, it'll flag isMoreThanTwo and break
out of the loop
                    isMoreThanTwo = 1; break;
                }
            }
            if(!isMoreThanTwo){
                printf(RED "\nThe description must be more than 2 words!\n"
RESET);
                printf("Press enter to continue..."); getchar();
                system("cls");
            }
        } while(!isMoreThanTwo);
    }
    else{
        system("cls");
        int isMoreThanTwo = 0;
        do {
            printf("Input a new slang word description [Must be more than 2
words]: ");
            scanf("%[^\n]", inputDesc); getchar();
            len = strlen(inputDesc);

            for(int i = 0; i < len; i++){
                if(inputDesc[i] == ' ' && i != len-1){
                    isMoreThanTwo = 1;
                    break;
                }
            }
            if(!isMoreThanTwo){
                printf(RED "\nThe description must be more than 2 words!\n"
RESET);
                printf("Press enter to continue..."); getchar();
                system("cls");
            }
        } while(!isMoreThanTwo);
    }

    //Inserts the inputted word and description into the trie
    insertNode(inputSlang, inputDesc);
    puts("");
    printf("Press enter to continue..."); getchar();
}
```

Searching from the trie

```c
/*
    Now the searchFromTrie and exists function almost have the same structure,
but it has different functionality. The searchFromTrie function is used to return
the words being searched, so the searchWord_menu function could print the word
and the description being used. Whereas the exists function returns either one or
zero depending if the word exists or not. This function is being used in
searching prefix and for inserting a node.

    Onto the real explanation of the function, it first creates a curr pointer to
traverse through the Node. Then it will iterate through the length of the word to
detect if the word is in the node. If the child doesn't have the character, it
will return null. When the loop stops at the end of the key word, it will detect
if the current node is the end of the word or not. If yes, it'll return the node,
else it will return as null.
*/
char *searchFromTrie(const char *key) {
    Node *curr = root;
    int len = strlen(key);
    int index = 0;
    for(int i = 0; i < len; i++) {
        if (islower(key[i]))  index = key[i] - 'a';
        else index = key[i] - 'A' + 26;

        if(!curr->child[index])  return NULL;
        curr = curr->child[index];
    }
    if (curr && curr->data_end)  return curr->desc;

    return NULL;
}

int exists(const char* key) {
    Node* curr = root;
    int index = 0;
    while(*key) {
        if (islower(*key)) {
            index = *key - 'a';
        } else if (isupper(*key)) {
            index = *key - 'A' + 26;
        }
        if (!curr || !curr->child[index]) return 0;
        curr = curr->child[index];
        key++;
    }
    if((curr != NULL) && (curr->data_end)) return 1;
}


/*
    This menu is for printing the prompt and validating the input of the word
that the user wants to search for. First there's a do while loop to validate if
the input is more than one character and doesn't have spaces, just like the add
inputSlang function. Then it checks if the word exists in the trie or not using
the searchFromTrie function. If the word exists, it will print out the word and
the description.
*/
void searchWord_menu() {
    int len;
    char searchSlang[100];
    int spaceCount = 0;

    do {
        printf("Input a slang word to be searched [Must be more than 1 characters
and contains no space]: ");
        scanf("%[^\n]", searchSlang); getchar();
        len = strlen(searchSlang);

        for(int i = 0; i < len; i++){
            spaceCount = 0;
            if(searchSlang[i] == ' '){
                printf(RED "Input must be more than 1 character and have no
spaces" RESET);
                spaceCount++;
                break;
            }
        }
    } while(len < 2 || spaceCount > 0);

    char *description = searchFromTrie(searchSlang);
    if(description) {
        printf("Slang word  : %s\n", searchSlang);
        printf("Description : %s\n", description);
    } else {
        printf("The word doesn't exists\n");
    }

    printf("Press enter to continue..."); getchar();
}
```

Searching the dictionary based on the inputted prefix

```
/* Searching if the words have the prefix or not
    Parameters explanation:
    - prefix: Used to pass the prefix that's need to be searched from the
previous function.
    - temp: Used to hold the words being searched.
    - height: The current level of the trie, used to keep track of the current
height as it iterates through the trie.

    The function starts with a while loop, iterating through the prefix word the
user inputted. Then it'll check which index the current character is in,
according if they're an uppercase or lowercase character. Then it'll pass down to
the next child with that index. In the end of the loop, it will detect if curr is
already the end of data or not, if yes it'll add a null terminator to the
temporary string. Then it'll loop in a for loop to determine if the next
character is null or not. If it's not null, it'll print using the print node
function with the information from that node.
*/
void searchPrefix(const char* prefix, char *temp, int height) {
    temp[height++] = *prefix;
    Node* curr = root;
    int found = 0;
    int index = 0;

    for(int i = 0; i < strlen(prefix); i++) {
        if (curr == NULL) return;
        if (islower(prefix[i]))  index = prefix[i] - 'a';
        else  index = prefix[i] - 'A' + 26;

        curr = curr->child[index];
        temp[height++] = *prefix++;
    }

    if (curr && curr->data_end)
        temp[height] = '\0';

    for (int i = 0; i < size; i++) {
        if (curr->child[i] != NULL) {
            found = 1;
            printNode(curr->child[i], temp, height);
        }
    }
}
```

```c
/*
    This function is used to find words with a certain prefix. It asks the user
to input the prefix they want to search then it'll call the function to find the
words with that prefix.
*/
void searchPrefix_menu(){
    char prefix[20];
    printf("Input a prefix to be searched: ");
    scanf("%[^\n]", prefix); getchar();
    char temp[100];
    temp[0] = '\0';

    //This if is to fix a bug where we enter a prefix with the same word as an
existing word, that existing word won't show up. If the prefix word already
exists in the node, it'll print the word immediately.
    if(exists(prefix)){
        printf("%s\n", prefix);
    }
    searchPrefix(prefix, temp, 0);

    puts("");
    printf("Press enter to continue..."); getchar();
}
```

Printing the words existing in the dictionary

```
/* Printing the word in the dictionary
    Parameters explanation:
    - root: The root of the trie. It's the starting point for traversing the
trie.
    - temp: A string being used to store the characters that are going to be
printed.
    - height: The current level of the trie, used to keep track of the current
height as it recurses through.

    This function works in recursion as it allows the function to work endlessly
as long as the trie has a node. It will first check if the trie is empty or not.
If the root is empty, displays an error message stating that the dictionary
doesn't have any words yet. The temp will now be copied into a local temp to
store the characters that are being printed. It will increment the height.

    If the current node is marked as the end of data (end of word), it will give
the local temp a null terminator, meaning it's the end of the string. Then it'll
print the local temp, excluding the first character. The first character is the
root of the trie, so that's why it's not printed.

    The for loop is to traverse depper into the trie. It'll traverse within the
size of the trie and if the root of the child of the current size exists, then
it'll call back the function. But in that recursion, it'll start from the child
of the root.
*/

void printNode(Node *root, char *temp, int height){
    char localTemp[100];
    if(!root) {
        puts("There are no words yet"); return;
    }

    strcpy(localTemp, temp);
    localTemp[height] = root->alphabet;
    height++;

    if(root->data_end){
        localTemp[height] = '\0';
        printf("%s\n", localTemp+1);
    }

    for(int i = 0; i < size; i++){
        if(root->child[i])  printNode(root->child[i], localTemp, height);
    }
}
```

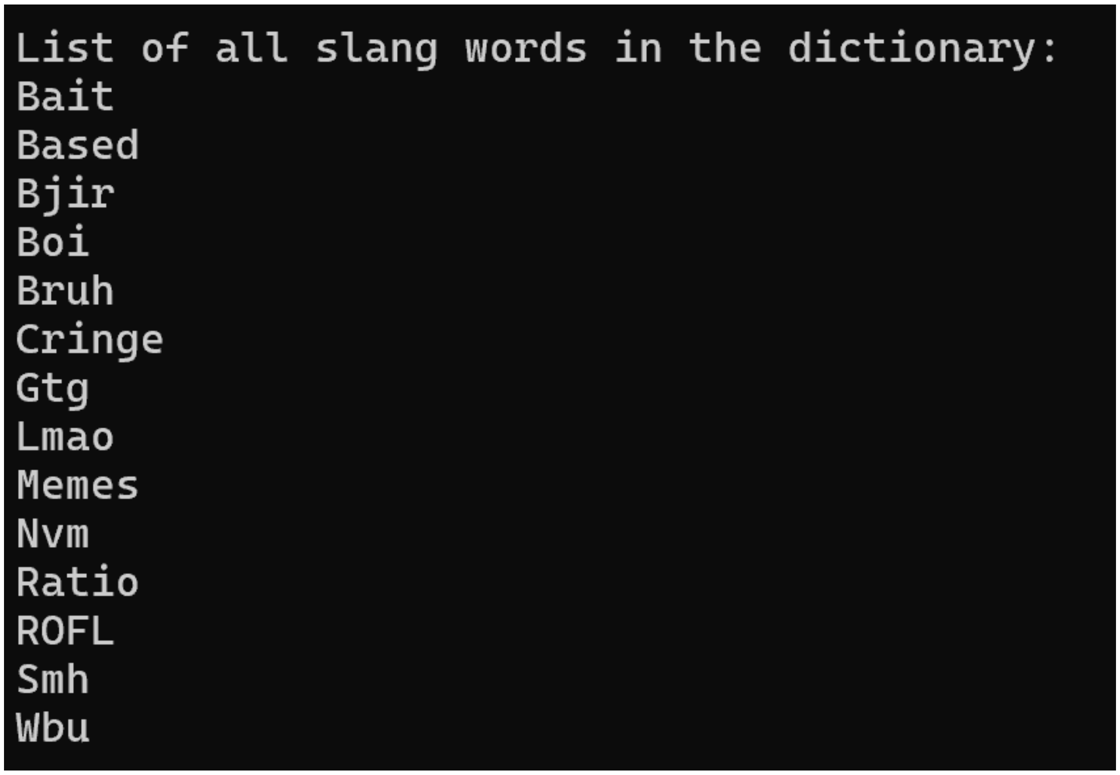
Printing the menu screen and the main function

```
/*
    A function to print the contents from the printNode function. temp is used in
the printNode function, hence why it's delared here and passed as a parameter.
Then it'll call the printNode function to actually print the words in the
dictionary.
*/
void printAll_menu(){
    puts("List of all slang words in the dictionary:");
    char temp[100];
    printNode(root, temp, 0);

    puts("");
    printf("Press enter to continue..."); getchar();
}

/*
    The main() function first declare the choice, then it'll show the menus to
the user. After that it asks for the user's input and validates if the input is
valid or not. The input is then put in a switch case to go to different menus.
All of that happens in a loop so that the user can go back to the main menu after
processing one part of the menu.
 */
int main() {
    int choice = 0;
    do {
        showMenu();
        printf("Your choice: ");
        scanf("%d", &choice); getchar();
        switch(choice) {
            case 1:
                system("cls");
                inputSlang_menu();
                break;
            case 2:
                system("cls");
                searchWord_menu();
                break;
            case 3:
                system("cls");
                searchPrefix_menu();
                break;
            case 4:
                system("cls");
                printAll_menu();
                break;
            case 5:
                break;
            default:
                system("cls");
                puts(RED "Invalid option\n" RESET);
                printf("Press enter to continue..."); getchar();
        }
    } while(choice != 5);

    puts(BLUE "Thank you.. Have a nice day :)" RESET);

    return 0;
}
```

Example of cases:

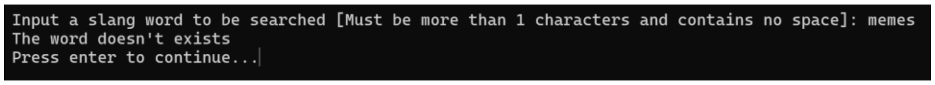
- Inputting 15 different words and descriptions
  Words that are inputted - the description:
  1. Based - A word used when you agree with something; or when you want to recognize someone for being themselves. Especially common in online political slang.
  2. Cringe - The feeling of physical discomfort without something physical happening to your body, that is usually caused by: second-hand embarrassment.
  3. Bjir - Bjir is the word "Njir" which was changed by shitposter in Indonesia.
  4. Ratio - When a reply on twitter gets more likes than the tweet it replied to.
  5. Bait - When someone states their opinion that is a obvious bait to get people mad and fight in the comment/reply section.
  6. Memes - Memes are a lifestyle and art used by teens and adults who are willing to actually live a life that doesn't include depression.
  7. Boi - This is what you say when someone says/does something stupid.
  8. Bruh - The reaction to every single situation possible. The most versatile word in existence.
  9. Smh - Meaning, "shaking my head", smh is typically used when something is obvious, plain old stupid, or disappointment.Wtf - The universal interrogative particle.
  10. Lmao - 'lmao' is an Internet slang, meaning laughing my ass off. It is generally known as a stronger version of 'laughing out loud'
  11. Rofl - ROFL is an abbreviation for 'rolling on (the) floor laughing', usually used in a similar context as 'LOL'.
  12. Gtg - Internet shorthand for "Got to go". Also, it can mean "good to go", depending on the context.
  13. Nvm - It is a abbreviation as "never mind" used mostly while texting
  14. Wbu - Chatspeak for 'what about you?'

If I print all of the words listed:

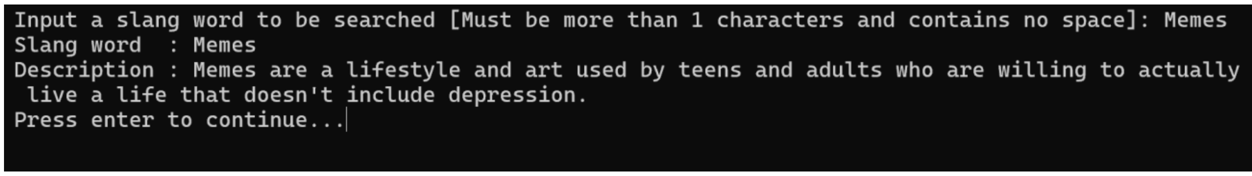


```
List of all slang words in the dictionary:
Bait
Based
Bjir
Boi
Bruh
Cringe
Gtg
Lmao
Memes
Nvm
Ratio
ROFL
Smh
Wbu
```

- Searching 5 words (case sensitive)
  1. Memes

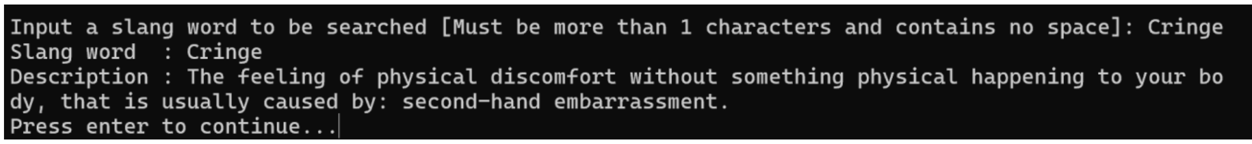


```
Input a slang word to be searched [Must be more than 1 characters and contains no space]: memes
The word doesn't exists
Press enter to continue...
```



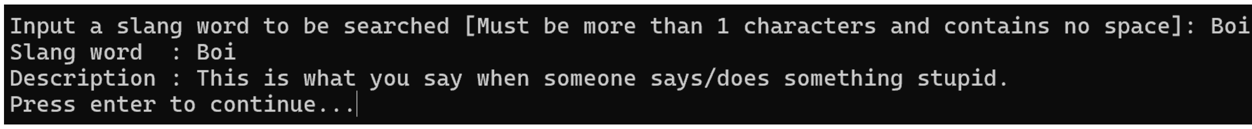
```
Input a slang word to be searched [Must be more than 1 characters and contains no space]: Memes
Slang word  : Memes
Description : Memes are a lifestyle and art used by teens and adults who are willing to actually
 live a life that doesn't include depression.
Press enter to continue...
```

  2. Cringe



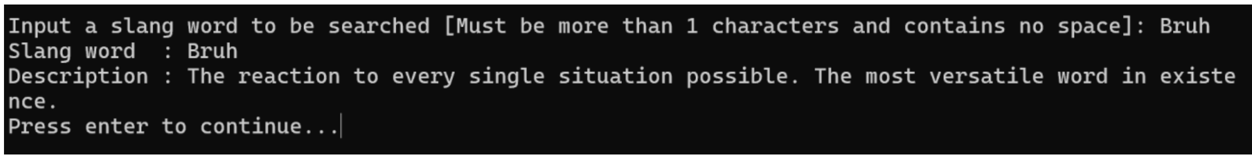
```
Input a slang word to be searched [Must be more than 1 characters and contains no space]: Cringe
Slang word  : Cringe
Description : The feeling of physical discomfort without something physical happening to your bo
dy, that is usually caused by: second-hand embarrassment.
Press enter to continue...
```

  3. Boi



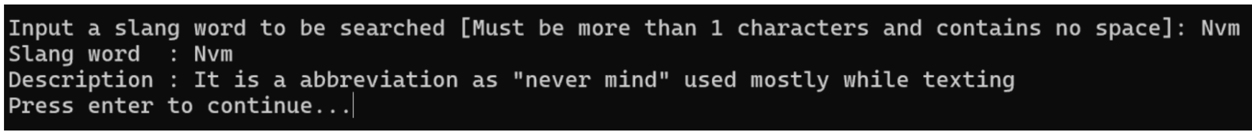
```
Input a slang word to be searched [Must be more than 1 characters and contains no space]: Boi
Slang word  : Boi
Description : This is what you say when someone says/does something stupid.
Press enter to continue...
```

  4. Bruh



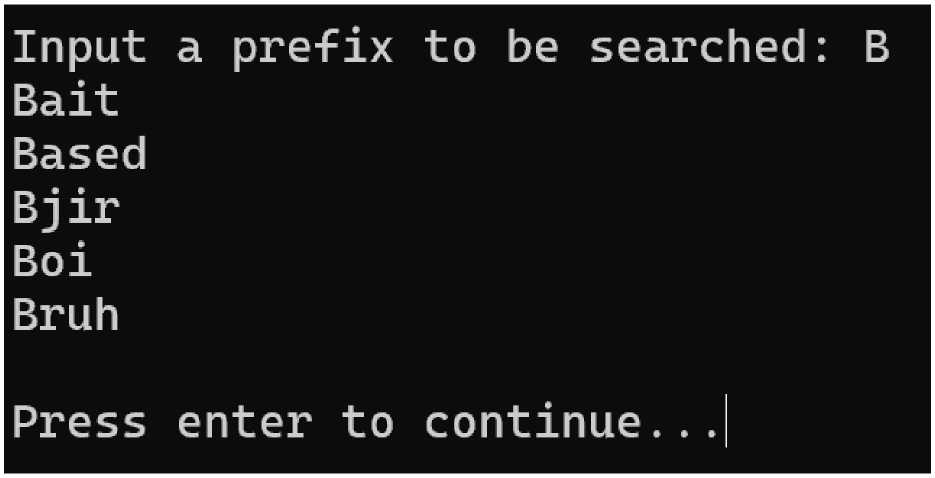
```
Input a slang word to be searched [Must be more than 1 characters and contains no space]: Bruh
Slang word  : Bruh
Description : The reaction to every single situation possible. The most versatile word in existe
nce.
Press enter to continue...
```

5. Nvm



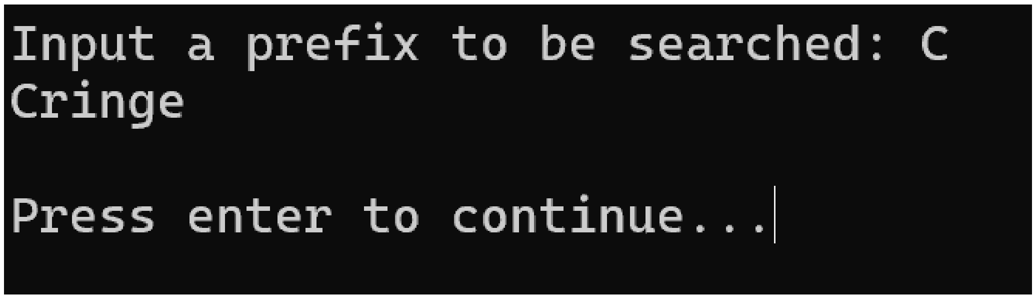
```
Input a slang word to be searched [Must be more than 1 characters and contains no space]: Nvm
Slang word  : Nvm
Description : It is a abbreviation as "never mind" used mostly while texting
Press enter to continue...
```

- Search based on prefix 5 times
  1. B



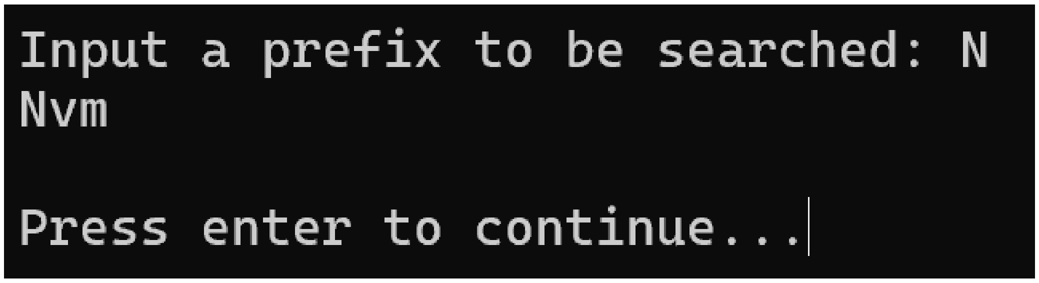
```
Input a prefix to be searched: B
Bait
Based
Bjir
Boi
Bruh

Press enter to continue...
```

  2. C



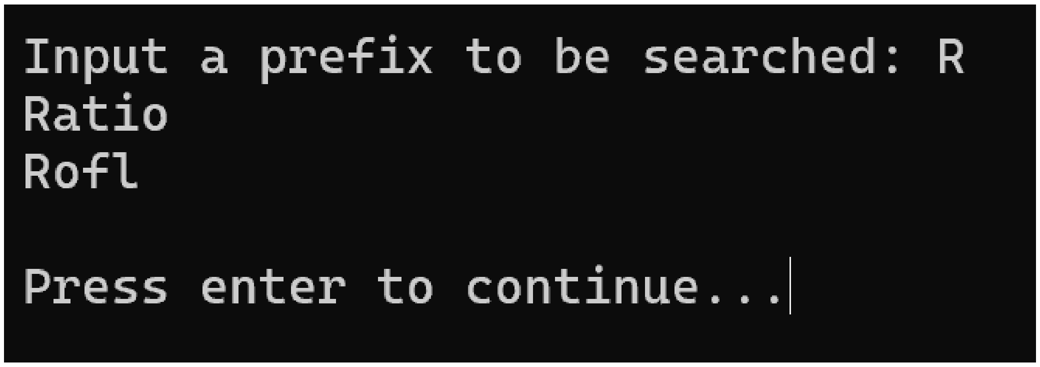
```
Input a prefix to be searched: C
Cringe

Press enter to continue...
```

  3. N



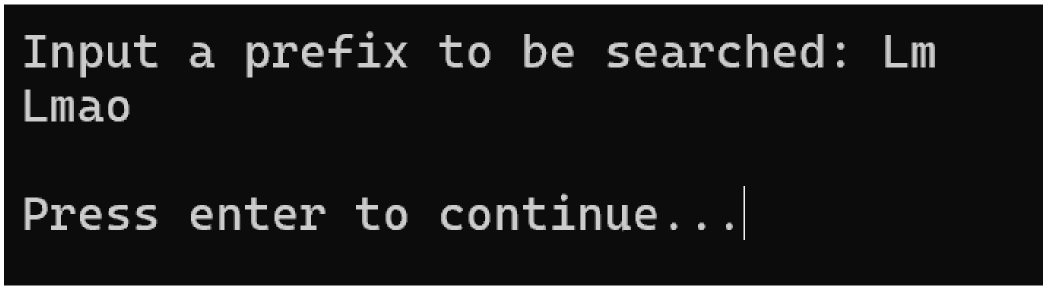
```
Input a prefix to be searched: N
Nvm

Press enter to continue...
```

  4. R



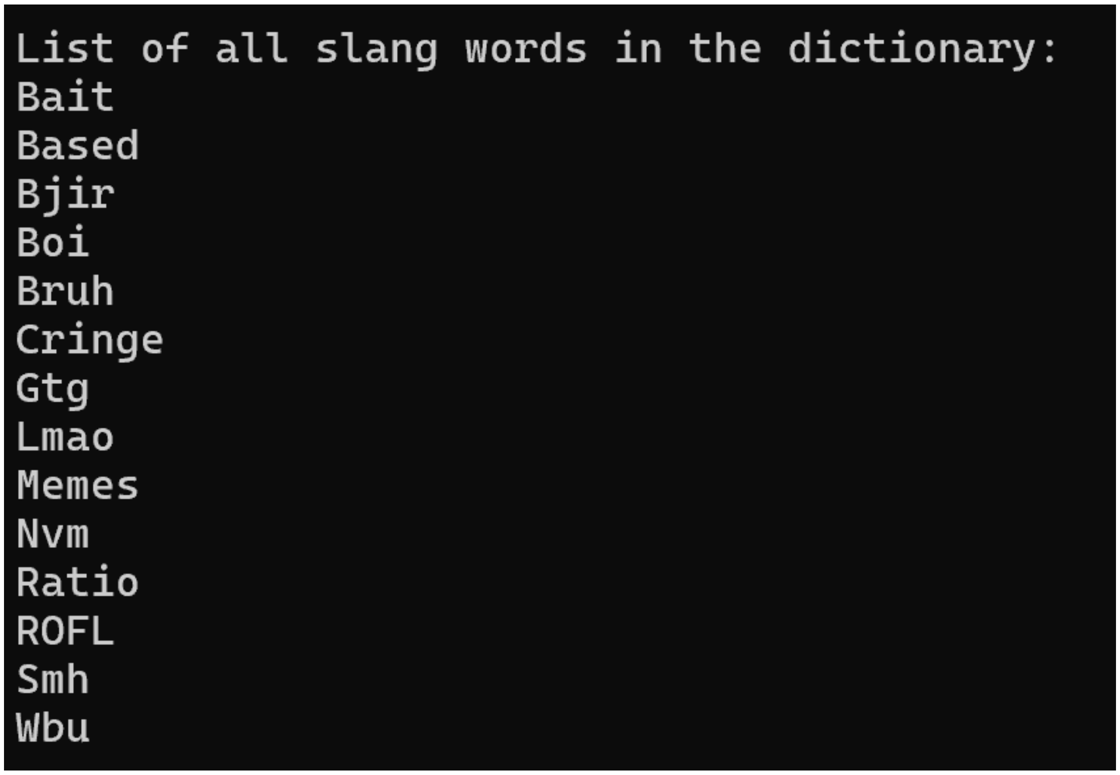
```
Input a prefix to be searched: R
Ratio
Rofl

Press enter to continue...
```

5. Lm



```
Input a prefix to be searched: Lm
Lmao

Press enter to continue...
```

- Print all



```
List of all slang words in the dictionary:
Bait
Based
Bjir
Boi
Bruh
Cringe
Gtg
Lmao
Memes
Nvm
Ratio
ROFL
Smh
Wbu
```

Reference:

- https://www.urbandictionary.com (to get words and descriptions)
- https://www.geeksforgeeks.org/trie-insert-and-search/
- https://github.com/TheAlgorithms/C/blob/master/data_structures/trie/trie.c