DATA STRUCTURE

ASSURANCE OF LEARNING – CASE STUDY

Fahimsyach Lokanta | 2702215946
Computer Science – Global Class | Alam Sutera

BINUS UNIVERSITY

Assignment

BOOGLE

Boogle is a company that creates and documents new slang words based on the internet. You as a programmer working at the Boogle company are asked to create an application that is useful for seeing what slang words have been released by Boogle.

The requirements are:

- The application consists of 5 menus:
    1. Release a new slang word.
    2. Search for a slang word.
    3. View all slang words starting with a certain prefix word.
    4. View all slang words.
    5. Exit

- If the user chooses menu 1 ("Release a new slang word"), then the program will:
    - Ask the user to input the new slang word. Validate that the slang word must be more than 1 character and contain no space.
    - Ask the user to input the description (meaning) of the new word. Validate that the description must be more than 1 word.
    - Store the newly released slang word in a Trie data structure along with its description.
    - If the slang word already exists in the Trie, then update the description with the new description.

- If the user chooses menu 2 ("Search a slang word"), then the program will:
    - Ask the user to input the slang word that wants to be searched. Validate that the slang word must be > 1 character and contain no space.
    - Search the input word in the Trie data structure.
    - If there is no such word, please show an empty message for the user and go back to the main menu.
    - If there is such a word, please show the word along with its description.

- If the user chooses menu 3 ("View all slang words starting with a certain prefix word"), then the program will:
    - Ask the user to input the prefix word that wants to be searched.
    - Search the input word in the Trie data structure.
    - If there is no such word, please show an empty message for the user.
    - If there is such a word, please show the list of words in the dictionary that starts with the prefix word in lexicographical order.

- If the user chooses menu 4 ("View all slang words"), then the program will:

    - If there is no word yet in the dictionary, please show an empty message for the user.
    - Else, please show the list of all words in the dictionary in lexicographical order.

- If the user chooses menu 5 (“Exit”), then the program will be closed.

C++ CODE with Explanation

```
// Header Includes
#include<stdio.h> // included for standard input output functions
#include<string.h> // included for string manipulation functions
#include<stdlib.h> // included for memory allocation functions
#include<ctype.h> // included for character type functions

#define SIZE 128 // maximum size for character arrays used

// Function to clear the input buffer
void ClearBuffer()
{
    while(getchar() != '\n');
}

// Function to wait for the Enter key press
void PressEnter()
{
    printf("Press ENTER to continue..."); getchar();
}

// Function to clear the screen
void ClearScreen()
{
    #if defined(_WIN32) || defined(_WIN64) // Windows
        system("CLS");
    #else // Unix-like systems
        system("CLEAR");
    #endif
}

// Function to check input validity
int checkInput(char* input, int min_length, int is_word)
{
    int len = strlen(input); // count length of the input string

    // Check if the input length is less than the minimum required
length
    if(len <= min_length)
    {
```

```
        return 0; // return 0 if the input length is invalid.
    }

    // Check if the input consists of alphabetic characters only
    if(is_word)
    {
        // Iterate through each character in the input string
        for(int i = 0; i < len; i++)
        {
            // Check if the character contains the alphabet
            if(!isalpha(input[i]))
            {
                return 0; // return 0 if the input characters are
invalid.
            }
        }
    }

    // Check if the input has more than the required number of words
    else
    {
        int word_count = 1; // initialize word count to 1 (suppose at
least one word)

        // Iterate through each character in the input string
        for(int i = 0; i < len; i++)
        {
            // If a space is encountered
            if(input[i] == ' ')
            {
                word_count++; // increment the word count.
            }
        }

        // If the word count is less than the required minimum
        if(word_count < min_length)
        {
            return 0; // return 0 if the input count is invalid.
        }
    }

    return 1; // return 1 if input is valid.
}

// Function to display the menu options
void Menu()
{
    puts("+-------------------------------------+");
```

```
    puts("|              B O O G L E            |");
    puts("+-------------------------------------+");
    puts("| 1. Release a slang word             |");
    puts("| 2. Search a slang word              |");
    puts("| 3. View all slang words with prefix |");
    puts("| 4. View all slang words             |");
    puts("| 5. Exit                             |");
    puts("+-------------------------------------+");
    printf(">> ");
}

// Trie node structure
typedef struct TrieNode
{
    char word; // variable to store words in the node.
    char* description; // variable to store description associated
with the word ending at this node.
    struct TrieNode* children[26]; // 26 letters in the alphabet
} TrieNode; // Define a shorthand for TrieNode structure.

// Function to create a new Trie node
TrieNode* createTrieNode(char word)
{
    TrieNode* node = (TrieNode*)malloc(sizeof(TrieNode)); // allocate
memory for the new node
    node->word = word; // set the character of the node

    // Initialize children's pointers to NULL
    for(int i = 0; i < 26; i++)
    {
        node->children[i] = NULL; // initialize description pointer to
NULL
    }

    node->description = NULL; // set description to NULL

    return node; // return the newly created node.
}

// Function to insert a new slang word into the Trie
void insertTrie(TrieNode* root, char* word, char* description)
{
    TrieNode* current = root; // set the current node as the root of
the Trie

    // Traverse the Trie based on the characters of the word
    for(int i = 0; i < strlen(word); i++)
    {
```

```
        int index = word[i] - 'a'; // calculate the index of the
current character

        // If the child node does not exist
        if(current->children[index] == NULL)
        {
            current->children[index] = createTrieNode(word[i]); //
create a new node for the character
        }
        current = current->children[index]; // move to the child node
    }

    // Free the memory allocated for the previous description if it
exists
    if(current->description!= NULL)
    {
        free(current->description);
    }

    current->description = strdup(description); // assign the new
description to the current node
}

// Function to search for a slang word in the Trie
char* searchTrie(TrieNode* root, char* word)
{
    TrieNode* current = root; // set the current node to the root of
the Trie

    // Iterate through each character of the word.
    for(int i = 0; i < strlen(word); i++)
    {
        int index = word[i] - 'a'; // calculate the index of the
current character based on its ASCII value

        // If the node corresponding to the current character doesn't
exist,
        if(current->children[index] == NULL)
        {
            return NULL; // return NULL if the word doesn't exist.
        }

        current = current->children[index]; // move to the next node
    }

    return current->description; // return the description of the last
node.
}
```

```c
// Recursive function to print words starting with a given prefix
void printPrefixWords(TrieNode* node, char* prefix, int* count)
{
     // If the current node is the end of a word, print the number
count and prefix word
    if(node->description != NULL)
    {
        printf("%d. %s\n", ++(*count), prefix);
    }

    // Recursively traverse all child nodes
    for(int i = 0; i < 26; i++)
    {
        // Check if the child node exists
        if(node->children[i] != NULL)
        {
            char newPrefix[SIZE]; // create a new prefix string
            sprintf(newPrefix, "%s%c", prefix, 'a' + i); // append the
current character to the prefix
            printPrefixWords(node->children[i], newPrefix, count); //
recursively call the function for the child node.
        }
    }
}

// Recursive function to print all words in the Trie in
lexicographical order
void printAllWords(TrieNode* root, char* prefix, int* count)
{
    // If the current node is the end of a word, print the number
count and word
    if(root->description != NULL)
    {
        printf("%d. %s\n", ++(*count), prefix);
    }

    // Recursively traverse all child nodes
    for(int i = 0; i < 26; i++)
    {
        // Check if the child node exists
        if(root->children[i] != NULL)
        {
            char newPrefix[SIZE]; // create a new prefix string
            sprintf(newPrefix, "%s%c", prefix, 'a' + i); // append the
current character to the prefix.
            printAllWords(root->children[i], newPrefix, count); //
recursively call the function for the child node.
```

```
        }
    }
}

// Entry point of the program
int main()
{
    int choices; // variable to store user choices.
    int totalSlang = 0; // variable to store user choices.
    TrieNode* root = createTrieNode('\0'); // create root node of Trie

    do // main loop for menu selection.
    {
        ClearScreen(); Menu(); // clear screen and display menu

        // Check choices input validated as number or between
available choices
        if(scanf("%d", &choices) != 1 || choices < 1 || choices > 5)
        {
            printf("Invalid choice. Please input between 1 and 5.\n");
            ClearBuffer(); PressEnter();
        }

        else
        {
            ClearBuffer();
            printf("\n");
        }

        switch(choices) // Switch based on user choice.
        {
            case 1: // Release a new slang word.
            {
                char slang_word[SIZE], slang_desc[SIZE]; // variables
to store slang word and description

                do // loop for inputting slang word.
                {
                    printf("Input a new slang word [Must be more than
1 character and contains no space]: ");
                    scanf("%s", slang_word);
                    ClearBuffer();
                }
                while(!checkInput(slang_word, 1, 1)); // validate
slang word input

                do // loop for inputting slang description.
                {
```

```
                    printf("Input a new slang word description [Must
be more than 2 words]: ");
                    fgets(slang_desc, SIZE, stdin);
                }
                while(!checkInput(slang_desc, 3, 0)); // validate
slang description input

                insertTrie(root, slang_word, slang_desc); // insert
slang word and description into Trie
                totalSlang++; // increment total slang
                puts("\nSuccessfully released new slang word.");

                PressEnter();
                break;
            }

            case 2: // Search for a slang word.
            {
                char slang_search[SIZE]; // variable for search word

                do // loop for inputting slang word to be searched.
                {
                    printf("Input a slang word to be searched [Must be
more than 1 character and contains no space]: ");
                    scanf("%s", slang_search);
                    ClearBuffer();
                }
                while(!checkInput(slang_search, 1, 1)); // validate
slang word input to be searched.

                // Check if the search word exists in Trie
                if(searchTrie(root, slang_search) != NULL)
                {
                    // print the result if exists
                    printf("\nSlang word : %s\n", slang_search);
                    printf("Description: %s\n", searchTrie(root,
slang_search));
                }

                else
                {
                    printf("\nThere is no word \"%s\" in the
dictionary.\n", slang_search); // print the error message
                }

                PressEnter();
                break;
            }
```

```
            case 3: // View all slang words starting with a prefix.
            {
                TrieNode* current = root; // set current node to root
of Trie.
                int i, count = 0; // variable to store counting.
                char prefix[SIZE]; // variable to store prefix.

                printf("Input a prefix to search: ");
                scanf("%s", prefix);
                ClearBuffer();

                // Traverse Trie based on prefix
                for(int i = 0; prefix[i] != '\0'; i++)
                {
                    int index = prefix[i] - 'a'; // calculate index of
current character

                    // Check if the prefix doesn't exist
                    if(current->children[index] == NULL)
                    {
                        printf("\nThere is no prefix '%s' in the
dictionary.\n", prefix); // print the error message
                        PressEnter(); return 0;
                    }
                    current = current->children[index]; // move to
next node
                }

                 // If the loop without break, print words starting
with prefix
                if(prefix[i] == '\0')
                {
                    printf("\nWords starting with prefix '%s':\n\n",
prefix);
                    printPrefixWords(current, prefix, &count);
                    printf("\n"); PressEnter();
                }

                break;
            }

            case 4: // View all slang words in the dictionary.
            {
                int count = 0; // variable to store count of words

                // Check if Trie is empty
                if(totalSlang == 0)
```

```
                {
                    printf("There is no slang word yet in the
dictionary.\n"); // print the error message
                }

                else
                {
                    printf("List of all slang words in the
dictionary:\n");

                    // Check if Trie is not empty
                    if(root != NULL)
                    {
                        printAllWords(root, "", &count); // print all
words in Trie
                    }
                }

                printf("\n"); PressEnter();
                break;
             }

            case 5:
            {
                free(root); // free memory allocated for Trie nodes.
                printf("Thank you... Have a nice day :)"); // print exit
message
                break;
            }
        }
    }
    while(choices!=5); // continue loop until user chooses to exit

    return 0; // exit program with success status.
}
```

SPECIFIC EXPLANATION

USER INTERACTION ENHANCEMENTS

```c
// Function to clear the input buffer
void ClearBuffer()
{
    while(getchar() != '\n');
}

// Function to wait for the Enter key press
void PressEnter()
{
    printf("Press ENTER to continue..."); getchar();
}

// Function to clear the screen
void ClearScreen()
{
    #if defined(_WIN32) || defined(_WIN64) // Windows
        system("CLS");
    #else // Unix-like systems
        system("CLEAR");
    #endif
}
```

- ClearBuffer() flushes the buffer input. This makes the program ensure no additional unexpected activity will occur during the next inputs.
- PressEnter() pause for the user to press Enter when desired by the user.
- ClearScreen() keeps the screen clean to ensure readability and a glance for the user while reducing visual clutter.

VALIDATION FUNCTION

```
// Function to check input validity
int checkInput(char* input, int min_length, int is_word)
{
    int len = strlen(input); // count length of the input string

    // Check if the input length is less than the minimum required
length
    if(len <= min_length)
    {
        return 0; // return 0 if the input length is invalid.
    }

    // Check if the input consists of alphabetic characters only
    if(is_word)
    {
        // Iterate through each character in the input string
        for(int i = 0; i < len; i++)
        {
            // Check if the character contains the alphabet
            if(!isalpha(input[i]))
            {
                return 0; // return 0 if the input characters are
invalid.
            }
        }
    }

    // Check if the input has more than the required number of words
    else
    {
        int word_count = 1; // initialize word count to 1 (suppose at
least one word)

        // Iterate through each character in the input string
        for(int i = 0; i < len; i++)
        {
            // If a space is encountered
            if(input[i] == ' ')
            {
                word_count++; // increment the word count.
            }
        }

        // If the word count is less than the required minimum
        if(word_count < min_length)
        {
            return 0; // return 0 if the input count is invalid.
        }
    }

    return 1; // return 1 if input is valid.
}
```

- checkInput ensures users' inputs are valid. It requires three parameters: input string input, integer minimum length min_length and flag is_word to check whether the input is a single word or not. If the length of the input is less than or equal to the minimum length specified, it means the input is not valid then return 0. If is_word is true, then check if all characters in the input are alphabetic. Otherwise, count words in the input and make sure that they exceed the specified minimum number. If any of these conditions fail return 0 else return 1 meaning valid input.

TRIE DATA STRUCT

```c
// Trie node structure
typedef struct TrieNode
{
    char word; // variable to store words in the node.
    char* description; // variable to store description associated
with the word ending at this node.
    struct TrieNode* children[26]; // 26 letters in the alphabet
} TrieNode; // Define a shorthand for TrieNode structure.

// Function to create a new Trie node
TrieNode* createTrieNode(char word)
{
    TrieNode* node = (TrieNode*)malloc(sizeof(TrieNode)); // allocate
memory for the new node
    node->word = word; // set the character of the node

    // Initialize children's pointers to NULL
    for(int i = 0; i < 26; i++)
    {
        node->children[i] = NULL; // initialize description pointer to
NULL
    }

    node->description = NULL; // set description to NULL

    return node; // return the newly created node.
}

// Function to insert a new slang word into the Trie
void insertTrie(TrieNode* root, char* word, char* description)
{
    TrieNode* current = root; // set the current node as the root of
the Trie

    // Traverse the Trie based on the characters of the word
    for(int i = 0; i < strlen(word); i++)
    {
        int index = word[i] - 'a'; // calculate the index of the
current character

        // If the child node does not exist
        if(current->children[index] == NULL)
        {
            current->children[index] = createTrieNode(word[i]); //
create a new node for the character
        }
        current = current->children[index]; // move to the child node
    }

    // Free the memory allocated for the previous description if it
exists
    if(current->description!= NULL)
    {
        free(current->description);
    }

    current->description = strdup(description); // assign the new
description to the current node
}

// Function to search for a slang word in the Trie
char* searchTrie(TrieNode* root, char* word)
{
    TrieNode* current = root; // set the current node to the root of
the Trie

    // Iterate through each character of the word.
    for(int i = 0; i < strlen(word); i++)
    {
        int index = word[i] - 'a'; // calculate the index of the
current character based on its ASCII value

        // If the node corresponding to the current character doesn't
exist,
        if(current->children[index] == NULL)
        {
            return NULL; // return NULL if the word doesn't exist.
        }

        current = current->children[index]; // move to the next node
    }

    return current->description; // return the description of the last
node.
}
```

- A TrieNode is like a piece of a puzzle. It shows one letter of a word. Each Trie Node can connect to other Trie Nodes. These connections show the next letters in the word. This way of storing words is special. It makes it easy to find words fast. It is useful for things like auto-complete and dictionaries.

- The createTrieNode function is made to create a Trie node and gets a new space for a Trie node. It sets the character for the node and makes an array of children, with all values as NULL. It also sets the description pointer to NULL. After setting up the node, the function gives back the newly made node.

- The insertTrie function adds a new slang word to the Trie data structure. It uses the root node, word, and description as inputs. It starts at the root. For each letter in the word, it checks if a node exists. If not, it makes a new node using createTrieNode.

It links this new node to the current node. After going through the whole word, it sets the last node's description to the given one. If a description already exists from that word, free that first description memory. Then it assigns a new description.

- The searchTrie function goes through the Trie word by word. It checks for letters that make up a slang word. It moves from one word to the next word part. If all letters match a stored slang word, it gives the meaning of that word. If no match, it says the word is not there.

```
// Recursive function to print words starting with a given prefix
void printPrefixWords(TrieNode* node, char* prefix, int* count)
{
     // If the current node is the end of a word, print the number
count and prefix word
    if(node->description != NULL)
    {
        printf("%d. %s\n", ++(*count), prefix);
    }

    // Recursively traverse all child nodes
    for(int i = 0; i < 26; i++)
    {
        // Check if the child node exists
        if(node->children[i] != NULL)
        {
            char newPrefix[SIZE]; // create a new prefix string
            sprintf(newPrefix, "%s%c", prefix, 'a' + i); // append the
current character to the prefix
            printPrefixWords(node->children[i], newPrefix, count); //
recursively call the function for the child node.
        }
    }
}

// Recursive function to print all words in the Trie in
lexicographical order
void printAllWords(TrieNode* root, char* prefix, int* count)
{
    // If the current node is the end of a word, print the number
count and word
    if(root->description != NULL)
    {
        printf("%d. %s\n", ++(*count), prefix);
    }

    // Recursively traverse all child nodes
    for(int i = 0; i < 26; i++)
    {
        // Check if the child node exists
        if(root->children[i] != NULL)
        {
            char newPrefix[SIZE]; // create a new prefix string
            sprintf(newPrefix, "%s%c", prefix, 'a' + i); // append the
current character to the prefix.
            printAllWords(root->children[i], newPrefix, count); //
recursively call the function for the child node.
        }
    }
}
```

- The printPrefixWords displays each word that begins with a specific prefix. It navigates the Trie utilizing a loop-like method, starting from the prefix's node. Once it reaches the completion of a word (a node featuring a filled description), it highlights the word along with its associated prefix.

- The printAllWords prints all words in the Trie in alphabetical order. It begins at the Trie's root and recursively moves through all child nodes. When it reaches the end of a word, it prints the word along with its prefix.

CUSTOM CASE

THE SLANG

Here are some slang words and their meanings that I used for this case:

1) Slang: "cool"
   Description: "Something that is impressive”

2) Slang: "cope"
   Description: "To deal effectively with something difficult”

3) Slang: "lit"
   Description: "Something that is amazing”

4) Slang: "lol"
   Description: "Laughing out loud"

5) Slang: "fam"
   Description: "Short term for family"

6) Slang: "flex"
   Description: "Brag out accomplishments"

7) Slang: "chill"
   Description: "Relaxed or calm"

8) Slang: "cheugy"
   Description: "Something as outdated"

9) Slang: "yeet"
   Description: "Expression of excitement"

10) Slang: "yearn"
    Description: "To have a strong desire "

11) Slang: "bae"
    Description: "Term calling for lovers"

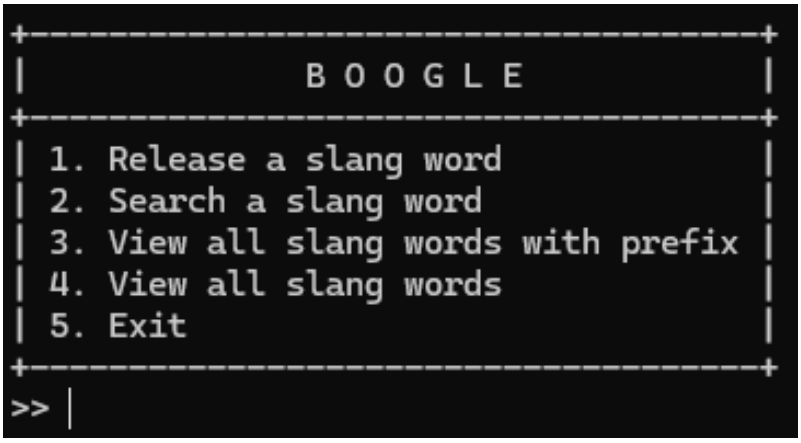
12) Slang: "sike"
    Description: "Misspelling of psych"

13) Slang: "savage"
    Description: "Fearless or fierce behavior"

14) Slang: "drab"
    Description: "Dull and boring"

15) Slang: "drip"
    Description: " Stylish or sophisticated appearance”

BOOGLE: MAIN MENU



This is the menu that appears when the program is executed. Users can input program choices.

BOOGLE: INPUT (CASE 1)

```
>> 1

Input a new slang word [Must be more than 1 character and contains no space]: c
Input a new slang word [Must be more than 1 character and contains no space]: c o
Input a new slang word [Must be more than 1 character and contains no space]: c o o l
Input a new slang word [Must be more than 1 character and contains no space]: cool
Input a new slang word description [Must be more than 2 words]: something
Input a new slang word description [Must be more than 2 words]: something that
Input a new slang word description [Must be more than 2 words]: something that is trendy or impresive

Successfully released new slang word.
Press ENTER to continue...
```

When the user's choice input is 1, it will prompt the user to input slang word and slang description to be released. The slang word prompt will re-print when the user input is <1 character and spaces. The slang description will re-print when the user input is <3 words. When both slang word and slang description input are valid, they will be stored in the Trie data structure. Next, a message appears when the slang word and the description are successfully released and prompts the user to press ENTER, then the screen clears and re-prints the menu.

There's a special case when you try to input a slang word that already exists, when you input the same word with a different description, it will update the old description with the new input.

- Current description for slang word "cool":

```
>> 2

Input a slang word to be searched [Must be more than 1 character and contains no space]: cool

Slang word : cool
Description: something that is trendy or impressive

Press ENTER to continue...
```

- Inputting word that already exist with a different description:

```
>> 1

Input a new slang word [Must be more than 1 character and contains no space]: cool
Input a new slang word description [Must be more than 2 words]: terms for "trendy"

Successfully released new slang word.
Press ENTER to continue...
```

- New description for slang word “cool”:

```
>> 2

Input a slang word to be searched [Must be more than 1 character and contains no space]: cool

Slang word : cool
Description: terms for "trendy"

Press ENTER to continue...
```

BOOGLE: SEARCH (CASE 2)

When the user’s choice input is 2, it will prompt the user to input a slang word to be searched. The slang word search prompt will re-print when the user input is <1 character and spaces. When the slang word to be searched input is valid, the program searches in the Trie data structure.

- If the word does not exist in the Trie data structure, it will print not exist:

```
>> 2

Input a slang word to be searched [Must be more than 1 character and contains no space]: d
Input a slang word to be searched [Must be more than 1 character and contains no space]: d a
Input a slang word to be searched [Must be more than 1 character and contains no space]: dab

There is no word "dab" in the dictionary.
Press ENTER to continue...
```

- If the word does exist in the Trie data structure, it will print with the description (5 examples):

```
>> 2

Input a slang word to be searched [Must be more than 1 character and contains no space]: lit

Slang word : lit
Description: Something that is amazing

Press ENTER to continue...
```

```
>> 2

Input a slang word to be searched [Must be more than 1 character and contains no space]: fam

Slang word : fam
Description: Short term for family

Press ENTER to continue...
```

```
>> 2

Input a slang word to be searched [Must be more than 1 character and contains no space]: chill

Slang word : chill
Description: Relaxed or calm

Press ENTER to continue...
```

```
>> 2

Input a slang word to be searched [Must be more than 1 character and contains no space]: yeet

Slang word : yeet
Description: Expression of excitement

Press ENTER to continue...
```

```
>> 2

Input a slang word to be searched [Must be more than 1 character and contains no space]: bae

Slang word : bae
Description: Term of the loved one

Press ENTER to continue...
```

BOOGLE: VIEW BY PREFIX (CASE 3)

When the user's choice input is 3, it will prompt the user to input a prefix to print slang words based on the prefix. It will process the Trie data structures to find and print all slang words that come with the prefix that are inputted by the users.

- If the prefix word does exist in the Trie data structure (5 examples):

```
>> 3

Input a prefix to search: co

Words starting with prefix 'co':

1. cool
2. cope

Press ENTER to continue...
```

```
>> 3

Input a prefix to search: ye

Words starting with prefix 'ye':

1. yearn
2. yeet

Press ENTER to continue...
```

```
>> 3

Input a prefix to search: li

Words starting with prefix 'li':

1. lit

Press ENTER to continue...
```

```
>> 3

Input a prefix to search: ch

Words starting with prefix 'ch':

1. cheugy
2. chill

Press ENTER to continue...
```

```
>> 3

Input a prefix to search: dr

Words starting with prefix 'dr':

1. drab
2. drip

Press ENTER to continue...
```

- If the prefix word does not exist in the Trie data structure, it will print not exist:

```
>> 3

Input a prefix to search: ha

There is no prefix 'ha' in the dictionary.
Press ENTER to continue...
```

BOOGLE: VIEW ALL (CASE 4)

When the user's choice input is 4, it will print all slang words based on the prefix in lexicographical order. It will process the Trie data structures to find and print all slang words.

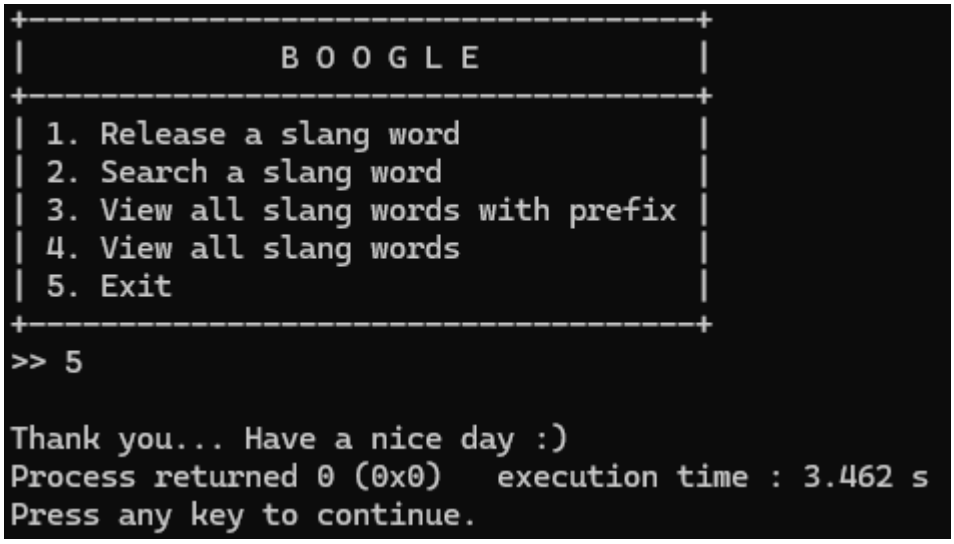
- If the dictionary does have any slang word exist:

```
>> 4

List of all slang words in the dictionary:
1. bae
2. cheugy
3. chill
4. cool
5. cope
6. drab
7. drip
8. fam
9. flex
10. lit
11. lol
12. savage
13. sike
14. yearn
15. yeet

Press ENTER to continue...
```

- If the dictionary does not have any slang word exist:

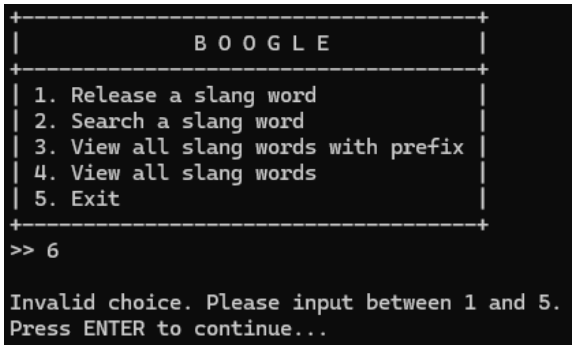
```
>> 4

There is no slang word yet in the dictionary.

Press ENTER to continue...
```

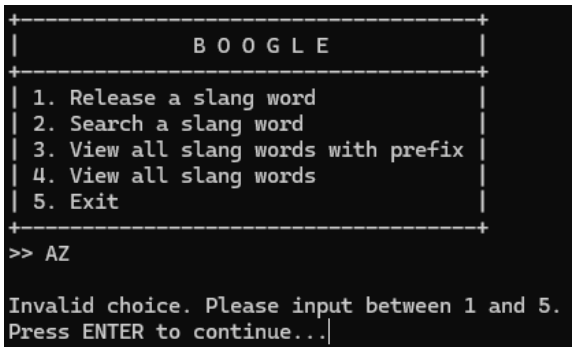
BOOGLE: EXIT (CASE 5)



When the user's choice input is 5, it will terminate the program process with printing exit message.

BOOGLE: ERROR CHOICES





When the user's choice input is not from the choices provided on the menu, for example, the user input numbers lower than 1 or higher than 5 or non-digit (alphabet). It will trigger the error message and prompt user to continue to input a correct choice.