Name: Ian Mulya Chiuandi
NIM: 2702218891
Class Code: LY01
AOL Data Structure Documentation

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h> //Using bool so that I don't have to use char and get confused

typedef struct node{
	char letter;
	char description[100]; //Store description for every possible ending point of a word.
	node *branch[128]; //ASCII is 0 - 127.
	bool end_of_slang;
} node;

node *root = NULL;

void menu_select(); //Calls menu function for other functions to be able to call it.
```

Here, I first include <stdio.h> to enable the use of input-output functions, as well as <stdlib.h> and <string.h> to be able to use and handle the required functions for the trie. Then, I include <stdbool.h> so that I don't have to use char or int and get confused later on just because of boolean variables.

The trie node is formed containing a char variable (letter), a char array (description) deliberately sized [100] elements as a hard limit, the edges of the trie (labelled branch, as an array of pointers) sized [128] elements to store all possible ASCII values, and the end of word boolean variable (end_of_slang), critical for the trie data structure to verify if a word is valid/has ended or not.

I then make a structure pointer (node*) labelled root and set to NULL, which is the root of the trie. Then, I declare a function titled "menu_select", in order to use later.

```
node *trie_node(char chr){
	node *trie = (node*) malloc(sizeof(node));
	trie->letter = chr;
	trie->end_of_slang = false;
	strcpy(trie->description, "x"); //Description is limited to more than 1 letter, so 1 letter is enough to set as a neutral state/placeholder.
	for(int i = 0; i < 128; i++){
		trie->branch[i] = NULL;
	}
	return trie;
}
```

For the above, I make a function labelled “trie_node” that accepts a char variable and makes a node for the trie data structure. The temporary node (labelled trie) uses malloc set to the size of the node structure, and then is filled with the inputted char variable, setting the end_of_slang bool to false, and copying “x” to the description of the node as a placeholder. This is because of the fact that description cannot be 1 letter only, so this can be used as a temporary sign that this node has not yet been filled, since the insert function later will require the user to input both the word and the description. After that, the node’s edges are set to NULL, using a for loop, for every single element in the array of pointer. Then, the node is returned from the function.

```
//Recursively pushes letter from string. At the end, attach description to the end_of_slang node.
void insert(node *curr, char *chr, char desc[]){
	while(*chr != NULL){ //Traverse per *chr while it exists.
		if(curr->branch[*chr] == NULL){ //Makes new trie nodes if they don't exist.
			curr->branch[*chr] = trie_node(*chr);
		}
		curr = curr->branch[*chr]; //Traverse to next node.
		chr++; //Move to next character on string.
	}
	curr->end_of_slang = true;
	if(strcmp(curr->description, "x") == 0){ //Check if curr has description.
		puts("");
		puts("Successfully released new slang word.");
	}
	else{
		puts("");
		puts("Successfully updated a slang word.");
	}
	strcpy(curr->description, desc);
}
```

This function is the insert function, which inserts a letter from the inputted string and traverses the array of char/string per letter. In essence, it first goes into a while loop

checking that the character on the inputted string isn't NULL. If the character isn't NULL, then it goes into an if statement checking if the current node's edge with the value of the character is NULL. If it is NULL, then this means that there doesn't exist any node yet for that character, and so makes a new node on that edge.

Regardless of the if statement, we then traverse to the next node and increment the char pointer in order to advance to the next letter on the inputted string. After the while loop ends (this means that we have iterated over all letters on the string), then we can set the end_of_slang boolean to true, which will indicate a valid word on the trie.

From this point on, the if statement relates to the description of the word, which happens to be stored on the node where the end_of_slang boolean is true. If the description is just "x", then we know this is a new slang word (since x is the default description), so we can indicate to the user that this is a new slang word. On the other hand, we notify that we have updated the slang word's description. At the end, we copy the inputted description to the the current node's description.

```
//Recursively matches letter from string.
node *search(node *curr, char *chr){
	while(*chr != NULL){
		if(curr->branch[*chr] == NULL) return NULL; //Character does not exist in trie, so return NULL.
		curr = curr->branch[*chr];
		chr++;
	}
	return curr; //Returns the node itself if word exists.
}
```

This is the search function, which searches whether or not a specific combination of letter exists in the trie. Because of its use in multiple other function, the function's actual use is just basic, and confirms whether or not any specific combination of letters exists already, even if "end_of_slang" is false. We enter a while loop, checking of the current letter is or is not NULL, and if it isn't, then we enter an if statement checking if the current node's edge is NULL. If so, then we return NULL, since this means that the letter does not exist in the trie (at least, with a specific combination of letters). Regardless of that, we traverse through the trie, and then increment the char pointer to move to the next letter of the inputted string. Once the while loop ends (at the end of the string), we know that every letter has already been checked, and so we can return the current node both as a confirmation that the search was successful and as a node which we can copy the description from, since the search function always return the node which "end_of_slang" is true.

```
//Prints all slang words.
void print_all(node *curr, char *hold_word, int depth, char *prefix_str, int *count){
	if(curr == NULL) return;

	char *buffer = (char*) malloc((depth+2)*sizeof(char)); //Create a buffer to store the previous resulting segment + the new segment.
	strcpy(buffer, hold_word);

	if(curr->end_of_slang == true){ //Check for end of slang.
		printf("%d. ", *count); //No. of words printed.
		(*count)++; //Increments the count outside of this function.

		if(prefix_str != NULL) printf("%s", prefix_str); //For use in printing prefix prints. (See below function.)
		buffer[depth] = '\0'; //Ends the string in buffer.
		printf("%s\n", buffer); //Outputs everything in buffer.
	}

	for(int i = 0; i < 128; i++){
		if(curr->branch[i]){ //Check if the next branch exists.
			buffer[depth] = i; //Sets the ASCII value (i) into the buffer[depth], which becomes a char.
			print_all(curr->branch[i], buffer, depth+1, prefix_str, count);
		}
	}

	free(buffer);
}
```

This specific function is the “print all” function, which also happens to be the function where all the words get printed from inside the trie, from a certain node (whcih will be important later). First, we check if the current node is NULL. If so, we return and exit out of the function, since this means that there doesn’t exist any node at all in the trie (this is just a contingency, since there is another check on the actual function forming the “view all words” menu option).

Moving on, we make a char pointer, denoted as a buffer, with the size being that of the depth of the trie we are in, plus 2 (just in case of overflow), multiplied by the size of the char variable. Then, we copy the part of the word we previously had (from recursion or the input) into the buffer. This makes it so that the word gets formed, since trie only contains a letter for each node, and we need a way to keep track of the word formed by the trie traversal.

Then, we check if end_of_slang is true. If so, then this means that the current node is where a word ends. We then print the number of words printed (a requirement on menu option 4), increment the count variable outside of the function, check again if the char pointer “*prefix_str” is or isn’t NULL. This if statement is only ever true if we call this function from another function, “print_prefix”. Basically, this check just allows

the prefix of a word to be printed first, since the actual “print all” function prints from a certain node. Regardless, since we know that the word is ending, we can set to the buffer according to the depth index the character ‘/0’, which ends the word. Then, we just output everything in the buffer.

After that, we go into a for loop running up to 128 times in order to find the next available node edge we can traverse to. If such an edge exists, then we add the current integer of i into the buffer, which becomes an ASCII value and transforms into a letter/character in the buffer. We then undergo recursion, this time going to the next node, keeping the buffer, incrementing the depth, keeping the prefix_str, and the integer pointer “count”.

After all the printing is done, we free the buffer.

```
//Calls the print_all function by passing the "prefix start node" as a starting point (prefix in the buffer).
void print_prefix(node *curr, char *chr, int *count){
	char *hold_prefix_pos = chr;
	while(*chr != NULL){
		//Contingency, just in case. In theory, this shouldn't happen, since we checked before using this function.
		if(curr->branch[*chr] == NULL) return;
		curr = curr->branch[*chr];
		chr++;
	}
	//Calls the print_all function to just print all words with prefix attached.
	print_all(curr, hold_prefix_pos, strlen(chr), hold_prefix_pos, count);
}
```

This specific function calls the print_all function, and holds a prefix to use within the print_all function. Hence, this is the “print prefix” function. First, we can make a new char pointer which holds the current character’s address. Then, we enter a while loop, which checks if the current character is NULL, where we iterate that if the next node in the edge is NULL, then we return.

Else, we traverse through the trie and increment the char pointer to go to the next letter in the prefix. Once we have arrived at the endpoint node of the prefix (the last letter of the prefix), then we can call the print_all function with our current node, the original char pointer of the prefix (from our char pointer holding the first character’s address), the length of the character, the original char pointer again, and the count variable.

```
//Function 1: Inputs new slang into the trie.
void new_slang(){
	puts("");

	//Inputs slang word.
	puts("Input a new slang word [Must be more than 1 character and contains no space(s)]: ");
	printf("> ");
	char str[30];
	scanf("%[^\n]", &str);
	getchar();

	int len = strlen(str);

	//Check for single character.
	if(len == 1){
		puts("ERROR: Slang only contains 1 character!");
		new_slang();
	}

	//Check for spaces.
	else if(strchr(str, ' ')){
		puts("ERROR: Slang contains spaces!");
		new_slang();
	}

	//Inputs slang description.
	puts("Input a new slang word description [Must be more than 1 word]: ");
	printf("> ");
	char desc[100];
	scanf("%[^\n]", &desc);
	getchar();

	//Check if two words.
	int index = 0;
	len = strlen(desc);
	while(desc[index] != '\0'){
		if(desc[index] == ' ') break;
		index++;
	}
	if(index == len || desc[index+1] == '\0' || desc[index+1] == ' '){ //Handles cases like "word " and "word  ".
		puts("ERROR: Description only contains 1 word!");
		new_slang();
	}

	//Insert string.
	insert(root, str, desc);

	puts("Press enter to continue.");
```

```
        getchar();
        menu_select();
}
```

This is the first actual menu option in the program, and inserts the inputted word and description into the “insert” function. First, it asks for a slang word, and waits for the input, storing it as a char array with a maximum of 30 characters. This is just a hard limit for the array, which is reasonable since most words do not go over 30 characters. The length of the string is determined. Then, the string is verified if it contains only 1 letter (from strlen, giving the length of a string) or that it contains spaces (from strchr, which returns the first instance of a character). If this verification fails, then we output an error message and let the user try again (this is based on the example test case on the question).

We then ask for the description input, which is then checked if it is more than 1 word. We take the strlen of the description, and we iterate through the char array using a while loop and incrementing the index. Then, we verify if the length of the description and the index count are the same, or if the description[index+1] is ‘\0’ or a space, which eliminates “words” that have two spaces and singular spaces. If this verification fails, then we output an error message and let the user try again.

If all this succeeds, then we insert the slang and description using the “insert” function. We then return to menu select.

```
//Function 2: Searches a slang exactly.
void search_slang(){
        puts("");

        //Inputs slang word.
        puts("Input a slang word to be searched [Must be more than 1 character and
contains no space(s)]: ");
        printf("> ");
        char target_str[30];
        scanf("%[^\n]", &target_str);
        getchar();

        int len = strlen(target_str);

        //Check for single character.
        if(len == 1){
                puts("ERROR: Slang only contains 1 character!");
                search_slang();
        }

        //Check for spaces.
        else if(strchr(target_str, ' ')){
                puts("ERROR: Slang contains spaces!");
                search_slang();
```

```
        }

        //Search
        puts("");
        node *search_node = search(root, target_str);

        //Check if the node is NULL or not.
        //Also, check if description is filled out for the word. If so, then output word and
description.
        //This description check is required, or search will also output true for prefixes.
        //This function of the search node is also being used by the view_all_prefix
function.
        if(search_node != NULL && strlen(search_node->description) > 1){
                printf("Slang word: %s\n", target_str);
                printf("Description: %s\n", search_node->description);
                puts("");
        }
        else printf("There is no such word as \"%s\" in the dictionary.\n", target_str);

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

This is the second function on the menu select, which asks for a slang to be searched and searches it, returning the description if found. As seen above, the function first asks for the slang word to be searched. The slang to be searched is inputted. We get the length of the word, and then we check if the word is more than a single character (length being 1) or if it contains spaces (using strchr). If this validation fails, then we allow the user to try again.

Else, we immediately call the “search” function using the word we inputted, and we put the results of this function into a node pointer. In order to check if the word actually exists, we first check if the node is NULL or not. If it’s NULL, then that means that the part of that word doesn’t exist in our trie. Otherwise, it may return the actual node, but the description is still “x”. So, we also check if the string length of the description is over 1 or not. If it is, then the word exists, and so the search is done. We have to check this, otherwise searching for a word even though that word doesn’t exist in our trie will output a false result. Once we pass this if statement, we output the slang word and the description. Else, we output to the user that the word they’re searching doesn’t exist. We then return to menu.

```
//Function 3: Prints all slang words with prefix attached.
void view_all_prefix(){
	puts("");

	//Inputs slang prefix.
	puts("Input a prefix to be searched: ");
	printf("> ");
	char prefix[30];
	scanf("%[^\n]", &prefix);
	getchar();

	//Search if prefix exists.
	node *search_node = search(root, prefix);

	//In this case, only check if node exists or not, since any such prefix of an existing words comes back true.
	if(search_node == NULL){
		printf("There is no such prefix as \"%s\" in the dictionary.\n", prefix);
	}
	else{
		int count = 1;
		puts("");
		printf("Words that start with \"%s\":\n", prefix);
		print_prefix(root, prefix, &count);
		puts("");
	}

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

The above function is the third function in the menu select, the function that prints all words with a specific prefix. First, the function asks the user to input a specific prefix. The prefix is inputted. We then search if the prefix exists, to which it returns to a node pointer. If the node is NULL, then the prefix doesn't exist, and we output that it doesn't exist to the user. Else, we declare an integer "count" set to 1, label the list title, and call the function print_prefix() starting from root with our prefix, which will print the words with the prefix attached. We then return to menu.

```
// Function 4: Prints all slang words.
void view_all_slang(){
	bool trie_exist = false;
	for(int i = 0; i < 128; i++){ //Loop all branches of root to find existing node. If exists, then set trie_exist true.
		if(root->branch[i] != NULL){
			trie_exist = true;
			break;
		}
	}

	if(trie_exist == false){ //Trie doesn't exist, return to menu.
		puts("");
		puts("There are no slang words yet in the dictionary.");
	}
	else{
		int count = 1;
		puts("");
		puts("List of all slang words in the dictionary:");
		print_all(root, "", 0, "", &count);
		puts("");
	}

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

This function is the fourth function in the menu select, and prints all words in the trie. First, we declare a boolean variable “trie_exist”, and set it to false. We then iterate through all possible edges of the root node, to see if any of them exist. If any of them exist, then the if statement allows the trie_exist variable to become true, and we break out of the loop. Else, the “trie_exist” variable remains false.

If it is false, then we output to the user that the trie doesn’t exist, and return to menu. If it is true, then we declare an integer “count” set to 1, label the list title, and call the print_all function starting from root and carrying no prefixes. After this is done, we return to menu.

```
void menu_select(){
	puts("1. Release a new slang word");
	puts("2. Search a slang word");
	puts("3. View all slang words starting with a certain prefix word");
	puts("4. View all slang words");
	puts("5. Exit");

	char choice;
	printf("Insert Choice: ");
	scanf("%c", &choice);
	getchar();

	switch(choice){
		case '1': //Make or update a slang word & description.
			new_slang();
			break;

		case '2': //Searches a slang word.
			search_slang();
			break;

		case '3': //Views all slang words starting with prefix.
			view_all_prefix();
			break;

		case '4': //Views all slang words.
			view_all_slang();
			break;

		case '5': //Exits program.
			puts("Thank you for using Boogle! Have a nice day!");
			exit(0);
			break;

		default: //Any choice not conforming to the selection above.
			puts("Choice unavailable, please choose again.");
			puts("");
			fflush(stdin); //In case of string inputs, which cause correct inputs to become false due to stdin carryover.
			menu_select();
	}
}
```

This is the menu select function, and mostly consists of outputs regarding the selection of options. We first print the selection of 5 choices, and then ask the user for their input. Doing this, the input falls into a switch case detailing the 5 choices. For input “1”, we call the new_slang() function. For input “2”, we call the search_slang() function. For input “3”, we call the view_all_prefix() function. For input

“4”, we call the view_all_slang() function. For input “5”, we output a friendly message, and exit the program. Any other choices not listed will fall under “default”, which will output to the user that the choice is unavailable and that they should try again. We flush the standard input just in case the user inputted more than 1 character (for example, if the user accidentally inputted a string), and we call the menu_select() function to return to menu to let the user to try again.

```
int main(){
	//Sets root.
	root = trie_node(' ');
	//In this case, sets the root node to ' '.
	//Technically, the character here doesn't really matter that much, because we immediately traverse by branch.
	//It's just set to ' ' for convenience.

	//Calls menu.
	menu_select();

	return 0;
}
```

Finally, in the main function, we first set the root before all other functions run, which we insert an arbitrary character ‘ ‘. The character inserted is not important, since we never print root directly, and immediately go to the edges of the trie for all the functions. After that, we call the menu_select() function so that the user can interact with the program.

Custom Cases:

Inserting 15 Words/Slangs (Count the 1s)

```
Press enter to continue.

1. Release a new slang word
2. Search a slang word
3. View all slang words starting with a certain prefix word
4. View all slang words
5. Exit
Insert Choice: 1

Input a new slang word [Must be more than 1 character and contains no space(s)]:
> patapon
Input a new slang word description [Must be more than 1 word]:
> a weird creature

Successfully released new slang word.
Press enter to continue.

1. Release a new slang word
2. Search a slang word
3. View all slang words starting with a certain prefix word
4. View all slang words
5. Exit
Insert Choice: 1

Input a new slang word [Must be more than 1 character and contains no space(s)]:
> geli
Input a new slang word description [Must be more than 1 word]:
> can mean jelly, or shuddering

Successfully released new slang word.
Press enter to continue.

1. Release a new slang word
2. Search a slang word
3. View all slang words starting with a certain prefix word
4. View all slang words
5. Exit
Insert Choice: 1

Input a new slang word [Must be more than 1 character and contains no space(s)]:
> jack
Input a new slang word description [Must be more than 1 word]:
> this means "i dunno"

Successfully released new slang word.
Press enter to continue.

1. Release a new slang word
2. Search a slang word
3. View all slang words starting with a certain prefix word
4. View all slang words
5. Exit
Insert Choice: 1

Input a new slang word [Must be more than 1 character and contains no space(s)]:
> bee
Input a new slang word description [Must be more than 1 word]:
> an annoying person

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

```
1. Release a new slang word
2. Search a slang word
3. View all slang words starting with a certain prefix word
4. View all slang words
5. Exit
Insert Choice: 1

Input a new slang word [Must be more than 1 character and contains no space(s)]:
> flower
Input a new slang word description [Must be more than 1 word]:
> a shy person

Successfully released new slang word.
Press enter to continue.

1. Release a new slang word
2. Search a slang word
3. View all slang words starting with a certain prefix word
4. View all slang words
5. Exit
Insert Choice: 1

Input a new slang word [Must be more than 1 character and contains no space(s)]:
> fermi
Input a new slang word description [Must be more than 1 word]:
> a scientist, or a paradox

Successfully released new slang word.
Press enter to continue.

1. Release a new slang word
2. Search a slang word
3. View all slang words starting with a certain prefix word
4. View all slang words
5. Exit
Insert Choice: doge
Choice unavailable, please choose again.

1. Release a new slang word
2. Search a slang word
3. View all slang words starting with a certain prefix word
4. View all slang words
5. Exit
Insert Choice: 1

Input a new slang word [Must be more than 1 character and contains no space(s)]:
> doge
Input a new slang word description [Must be more than 1 word]:
> a good boy

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

```
1. Release a new slang word
2. Search a slang word
3. View all slang words starting with a certain prefix word
4. View all slang words
5. Exit
Insert Choice: 1

Input a new slang word [Must be more than 1 character and contains no space(s)]:
> bnuuy
Input a new slang word description [Must be more than 1 word]:
> a misspelling of bunny

Successfully released new slang word.
Press enter to continue.

1. Release a new slang word
2. Search a slang word
3. View all slang words starting with a certain prefix word
4. View all slang words
5. Exit
Insert Choice: 1

Input a new slang word [Must be more than 1 character and contains no space(s)]:
> bum
Input a new slang word description [Must be more than 1 word]:
> homeless person

Successfully released new slang word.
Press enter to continue.

1. Release a new slang word
2. Search a slang word
3. View all slang words starting with a certain prefix word
4. View all slang words
5. Exit
Insert Choice: 1

Input a new slang word [Must be more than 1 character and contains no space(s)]:
> slumming
Input a new slang word description [Must be more than 1 word]:
> living illegally on the streets

Successfully released new slang word.
Press enter to continue.

1. Release a new slang word
2. Search a slang word
3. View all slang words starting with a certain prefix word
4. View all slang words
5. Exit
Insert Choice: 1

Input a new slang word [Must be more than 1 character and contains no space(s)]:
> kepler
Input a new slang word description [Must be more than 1 word]:
> a giant belt

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

(Updating a Slang)

```
1. Release a new slang word
2. Search a slang word
3. View all slang words starting with a certain prefix word
4. View all slang words
5. Exit
Insert Choice: 1

Input a new slang word [Must be more than 1 character and contains no space(s)]:
> king
Input a new slang word description [Must be more than 1 word]:
> a person on a metaphorical pedestal

Successfully updated a slang word.
Press enter to continue.
```

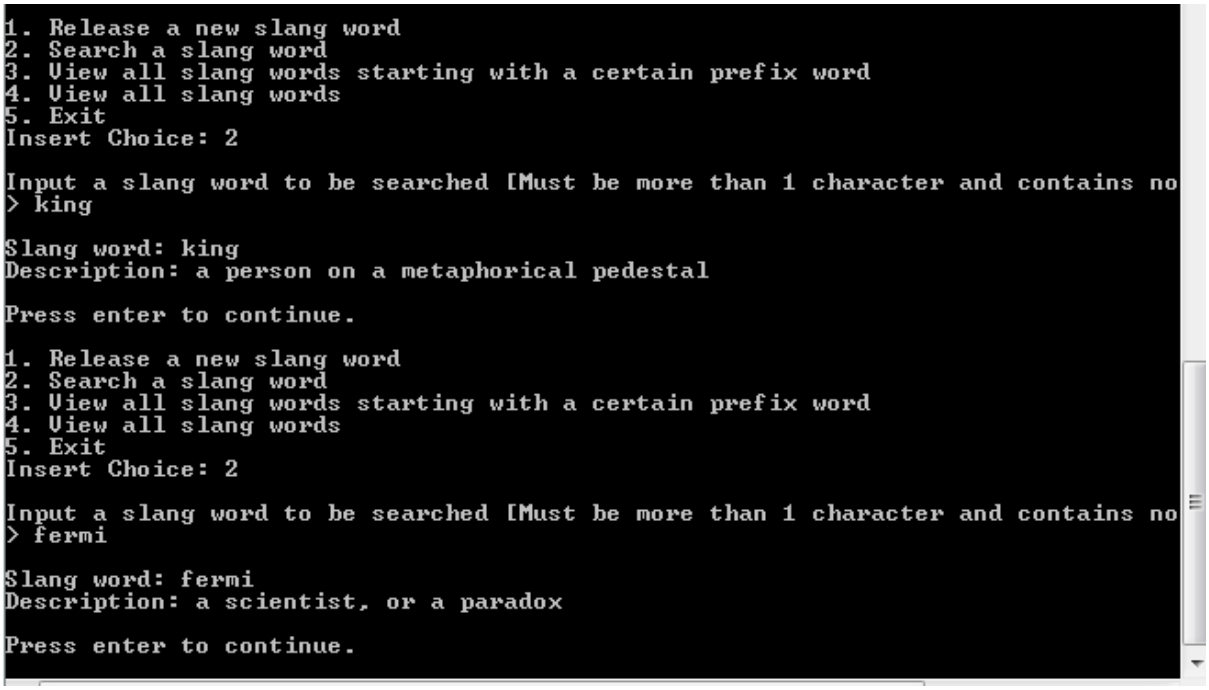
(Special Cases)

```
1. Release a new slang word
2. Search a slang word
3. View all slang words starting with a certain prefix word
4. View all slang words
5. Exit
Insert Choice: 1

Input a new slang word [Must be more than 1 character and contains no space(s)]:
> a
ERROR: Slang only contains 1 character!

Input a new slang word [Must be more than 1 character and contains no space(s)]:
> a a
ERROR: Slang contains spaces!

Input a new slang word [Must be more than 1 character and contains no space(s)]:
> bee
Input a new slang word description [Must be more than 1 word]:
> bee
ERROR: Description only contains 1 word!

Input a new slang word [Must be more than 1 character and contains no space(s)]:
```

Search 5 Words

```
1. Release a new slang word
2. Search a slang word
3. View all slang words starting with a certain prefix word
4. View all slang words
5. Exit
Insert Choice: 2

Input a slang word to be searched [Must be more than 1 character and contains no
> zamn

Slang word: zamn
Description: means like awesome, creepily

Press enter to continue.

1. Release a new slang word
2. Search a slang word
3. View all slang words starting with a certain prefix word
4. View all slang words
5. Exit
Insert Choice: 2

Input a slang word to be searched [Must be more than 1 character and contains no
> bnuuy

Slang word: bnuuy
Description: a misspelling of bunny

Press enter to continue.

1. Release a new slang word
2. Search a slang word
3. View all slang words starting with a certain prefix word
4. View all slang words
5. Exit
Insert Choice: 2

Input a slang word to be searched [Must be more than 1 character and contains no
> doge

Slang word: doge
Description: a good boy

Press enter to continue.

1. Release a new slang word
2. Search a slang word
3. View all slang words starting with a certain prefix word
4. View all slang words
5. Exit
Insert Choice: 2

Input a slang word to be searched [Must be more than 1 character and contains no
> rag

There is no such word as "rag" in the dictionary.
Press enter to continue.
```

(Above attempt is when the search is not found)



View Prefix 5 Words

```
1. Release a new slang word
2. Search a slang word
3. View all slang words starting with a certain prefix word
4. View all slang words
5. Exit
Insert Choice: 3

Input a prefix to be searched:
> s

Words that start with "s":
1. scum
2. slumming

Press enter to continue.

1. Release a new slang word
2. Search a slang word
3. View all slang words starting with a certain prefix word
4. View all slang words
5. Exit
Insert Choice: 3

Input a prefix to be searched:
> a
There is no such prefix as "a" in the dictionary.
Press enter to continue.
```

(Above is an attempt to search prefix “a”, which doesn’t exist)

View All

```
1. Release a new slang word
2. Search a slang word
3. View all slang words starting with a certain prefix word
4. View all slang words
5. Exit
Insert Choice: 4

List of all slang words in the dictionary:
1. bee
2. bnuuy
3. bum
4. doge
5. fermi
6. flower
7. geli
8. jack
9. kepler
10. king
11. patapon
12. scum
13. slumming
14. wowza
15. zamn

Press enter to continue.
```

(Fresh run of the program, before any slang was inserted)

```
1. Release a new slang word
2. Search a slang word
3. View all slang words starting with a certain prefix word
4. View all slang words
5. Exit
Insert Choice: 3

Input a prefix to be searched:
> a
There is no such prefix as "a" in the dictionary.
Press enter to continue.

1. Release a new slang word
2. Search a slang word
3. View all slang words starting with a certain prefix word
4. View all slang words
5. Exit
Insert Choice: 4

There are no slang words yet in the dictionary.
Press enter to continue.
```