1. Included libraries and constants

```c
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define ALPHABET_SIZE 26
```

At the beginning of the program, three standard C libraries are included: stdio.h, stdlib.h, and string.h. The stdio.h library provides functions for input and output operations such as printf() and scanf(), which are essential for displaying messages to the user and receiving user input. The stdlib.h library is required for dynamic memory management using functions like malloc() to allocate memory and free() to deallocate memory. This is crucial in this program because each node in the Trie is dynamically created and later freed when no longer needed. Meanwhile, string.h is used for manipulating strings through functions like strcpy() and strlen(), which are used for handling the slang words and their descriptions.

Additionally, the line #define ALPHABET_SIZE 26 defines a constant representing the number of letters in the English alphabet. This constant is used throughout the program to declare the size of the children array in each Trie node, allowing each node to potentially point to 26 different children—one for each lowercase letter from 'a' to 'z'. Using a named constant like this improves readability and maintainability, as it clearly shows the intention behind the array size and allows for easy modification if necessary.

2. Trie Node Structure Definition

```c
typedef struct node

{

    struct node *children[ALPHABET_SIZE];

    int is_end_word;
```

```
    char slang_word[100];

    char description[1000];

} node;
```

This section defines the structure of a single node in the Trie data structure. The typedef struct node { ... } node; syntax creates a self-referential structure, which allows each node to point to other nodes (its children). The field children[ALPHABET_SIZE] is an array of 26 pointers to other nodes—one for each lowercase letter in the English alphabet. This setup allows the Trie to branch out for each letter, enabling fast prefix-based lookups and insertions.

The is_end_word field is an integer flag that indicates whether the current node marks the end of a valid slang word. If this value is set to 1, it means the path from the root to this node forms a complete slang word stored in the dictionary. The slang_word[100] and description[1000] arrays are used to store the actual slang word and its corresponding definition when a word is inserted. These are only populated in nodes where is_end_word is true. Using fixed-size character arrays simplifies memory management while ensuring each entry has sufficient space for meaningful content.

3. Function : create_node()

```
node *create_node()

{

    node *new_node = (node *)malloc(sizeof(node));

    for (int i = 0; i < ALPHABET_SIZE; i++)

    {

        new_node->children[i] = NULL;

    }

    new_node->is_end_word = 0;

    strcpy(new_node->slang_word, "");
```

```
    strcpy(new_node->description, "");

    return new_node;

}
```

This function is responsible for creating and initializing a new node in the Trie data structure. It starts by dynamically allocating memory using malloc() to store a node structure. The pointer new_node will point to this newly allocated memory block. Next, it initializes all elements of the children array to NULL, ensuring that the new node has no children yet—this means no letters have branched from it.

After setting the children, the function initializes the flag is_end_word to 0, indicating that the node does not yet mark the end of a valid slang word. Additionally, it clears the slang_word and description fields by copying empty strings into them using strcpy(). This step ensures that the node starts in a clean state with no accidental leftover data. Finally, the function returns a pointer to this newly created node so it can be linked into the Trie. This function is called whenever a new character branch needs to be added during insertion.

4. Function : insert(node *root, const char *word, const char *desc)

```
void insert(node *root, const char *word, const char *desc)

{

    node *current = root;

    for (int i = 0; word[i] != '\0'; i++)

    {

        int index = word[i] - 'a';



        if (!current->children[index])

        {
```

```
            current->children[index] = create_node();

        }



        current = current->children[index];

    }

    current->is_end_word = 1;

    strcpy(current->slang_word, word);

    if (strcmp(current->description, "") == 0)

    {

        printf("\nSuccessfully released new slang word.\n");

    }

    else

    {

        printf("\nSuccessfully updated a slang word.\n");

    }

    strcpy(current->description, desc);

}
```

This function inserts a new slang word into the Trie, or updates the description if the word already exists. It begins at the root node and iterates through each character of the input word. For each character, it calculates the corresponding index in the children array by subtracting 'a' from the character's ASCII value (so 'a' becomes 0, 'b' becomes 1, etc.). If the child node at that index does not exist, it creates a new one using create_node().

The current pointer is then updated to move deeper into the Trie along the path of the word. After the loop finishes, the function has reached the node that should represent the end of the word. It sets the is_end_word flag to 1 to mark this node as a complete slang word entry. The actual word is copied into the slang_word field of the node using strcpy(). To differentiate between inserting a new word and updating an existing one, it checks if the description field is still empty—if it is, a message about a successful new entry is printed; otherwise, a message about a successful update is shown. Lastly, it updates the description with the new input using strcpy().

This function ensures that each unique slang word is stored exactly once in the Trie, and that its description can be updated if needed. One limitation is that it assumes all letters in the input word are lowercase alphabetic characters (a–z); if input validation is weak, this could cause issues.

5. Function : search(node *root, const char *word)

```
void search(node *root, const char *word)

{

    node *current = root;

    for (int i = 0; word[i] != '\0'; i++)

    {

        int index = word[i] - 'a';



        if (!current->children[index])

        {

                printf("\nThere is no word \"%s\" in the dictionary.\n",
word);

            return;

        }
```

```
        current = current->children[index];
    }
    if (current->is_end_word)
    {
        printf("\nSlang word : %s\n", current->slang_word);
        printf("Description : %s\n\n", current->description);
    }
    else
    {
        printf("\nThere is no word \"%s\" in the dictionary.\n", word);
        return;
    }
}
```

This function is used to search for a slang word in the Trie and display its description if found. Starting from the root, it iterates over each character of the input word, converting each character into a corresponding index by subtracting 'a'. For each character, it checks if a child node exists at that index. If any character in the path is missing—meaning the child node is NULL—the function immediately concludes that the word does not exist in the dictionary and prints an appropriate message.

If the traversal completes successfully, the function then checks whether the current node is marked with is_end_word == 1, indicating the end of a valid slang word. If so, it prints the word and its stored description. Otherwise, even though all characters are found, the word is only a prefix and not a complete entry, so the function again prints a message saying the word does not exist. This behavior ensures the Trie accurately distinguishes between complete slang words and mere prefixes.

6. Function : print_data(node* current, int *num)

```
void print_data(node *current, int *num)

{

    if (current->is_end_word)

    {

        printf("%d. %s\n", *num, current->slang_word);

        (*num)++;

    }



    for (int i = 0; i < ALPHABET_SIZE; i++)

    {

        if (current->children[i])

        {

            print_data(current->children[i], num);

        }

    }

}
```

This recursive function is used to print all slang words stored in the Trie starting from a given node, in lexicographical (alphabetical) order. It first checks whether the current node marks the end of a valid slang word using the is_end_word flag. If so, it prints the word using the slang_word field, prefixed with a numbering counter pointed to by the num pointer. The counter is then incremented to ensure each word is printed with a unique sequence number.

After printing, the function iterates through all 26 possible children (from 'a' to 'z') using a for loop. If a child exists at a given index, it recursively calls print_data() on that

child. This traversal strategy ensures that all valid slang words beneath the current node are visited and printed in order. The function is useful for implementing features like displaying all slang words in the dictionary or listing words with a given prefix. A limitation is that it assumes all words use only lowercase alphabet letters; input outside this range could lead to incorrect behavior or be ignored.

7. Function : show_based_prefix(node *root, const char *word)

```
void show_based_prefix(node *root, const char *word)

{

    node *current = root;

    for (int i = 0; word[i] != '\0'; i++)

    {

        int index = word[i] - 'a';



        if (!current->children[index])

        {

                printf("\nThere is no prefix \"%s\" in the dictionary.\n",
word);

            return;

        }



        current = current->children[index];

    }

    int num = 1;

    printf("\nWords starts with \"%s\":\n", word);

    print_data(current, &num);
```

```
    printf("\n");

}
```

This function is used to display all slang words stored in the Trie that begin with a specific prefix. It begins at the root node and traverses through the Trie by following the path determined by the characters of the input prefix (word). For each character, the function calculates the index in the children array and checks whether the corresponding child node exists. If at any point a required child node is missing, the function concludes that the prefix is not present in the dictionary and prints an appropriate message.

If the prefix is successfully found, the function proceeds to print all slang words that extend from this prefix using the print_data() function. It initializes a counter variable num to 1 to number the results and displays a message indicating the prefix being used. The recursive call to print_data() then lists all words in lexicographical order starting from the matched prefix node. This feature is helpful for browsing all entries that start with a common root, such as seeing all slang words that begin with “sh” or “co”.

8. Function : show_all(node *root)

```
void show_all(node *root)

{

    node *current = root;

    int is_empty = 1;

    for (int i = 0; i < ALPHABET_SIZE; i++)

    {

        if (current->children[i])

        {

            is_empty = 0;

            break;
```

```
        }
    }
    if (!is_empty)
    {
        printf("List of all slang words in the dictionary:\n");
        int num = 1;
        print_data(current, &num);
        printf("\n");
    }
    else
    {
        printf("There is no slang word yet in the dictionary.\n");
    }
}
```

This function is responsible for displaying all slang words currently stored in the Trie. It first checks whether the Trie is empty by examining all 26 elements of the root node's children array. If every child pointer is NULL, it concludes that no slang words have been inserted and prints a message stating that the dictionary is empty.

If the Trie is not empty, the function prints a header message and initializes a counter variable num to 1. It then calls the print_data() function starting from the root node, which recursively prints every slang word in lexicographical order with sequential numbering. This feature gives users a complete overview of the data they have entered, and is particularly useful for verifying that insertions have succeeded or for browsing the entire collection of slang words. It assumes that all words use only lowercase letters and that valid words are only those marked by is_end_word.

9. Function : exit_trie(node *root)

```
void exit_trie(node *root)

{

    node *current = root;

    for (int i = 0; i < ALPHABET_SIZE; i++)

    {

        if (current->children[i])

        {

            exit_trie(current->children[i]);

        }

    }

    free(current);

}
```

This function is responsible for properly deallocating all memory used by the Trie to prevent memory leaks. It performs a post-order traversal of the Trie, meaning it recursively visits and frees all child nodes before freeing the current node. This ensures that all dynamically allocated memory for each node is released from the bottom up, maintaining proper cleanup.

The function starts by iterating through the children array of the current node. If a child node exists at any index, the function calls itself recursively on that child. After all children are processed and their memory is freed, the function then calls free(current) to deallocate the current node's memory. This function is typically called at the end of the program—usually from the main() function—when the user chooses to exit. It is critical in C programs that use dynamic memory to ensure clean program termination.

10. Function : Print_menu()

```
void print_menu()

{

    printf("====BOOGLE APPLICATION MENU====\n");

    printf("1. RELEASE A NEW SLANG WORD.\n");

    printf("2. SEARCH A SLANG WORD.\n");

       printf("3. VIEW ALL SLANG WORDS STARTING WITH A CERTAIN PREFIX
WORD.\n");

    printf("4. VIEW ALL SLANG WORDS.\n");

    printf("5. EXIT.\n");

    printf("YOUR INPUT HERE >> ");

}
```

This function displays the main menu of the application to the user. It prints a clear and formatted list of the available features in the Boogle application: inserting a new slang word, searching for an existing slang word, viewing all slang words with a specific prefix, displaying all slang words in the dictionary, and exiting the program. The menu provides a user-friendly interface for navigating the program's functionality.

At the end of the printed menu, the function prompts the user to enter their input by showing YOUR INPUT HERE >>, making it clear where the user should respond. This function is called each time the program loops back to await a new user command, ensuring consistent and guided interaction throughout the application's runtime. While simple, it plays a crucial role in the usability of the command-line interface.

11. Function : first_verif(const char *word)

```
int first_verif(const char *word)
```

```
{
    if (strlen(word) <= 1)
    {
        return 0;
    }

    for (int i = 0; word[i] != '\0'; i++)
    {
        if (word[i] == ' ')
        {
            return 0;
        }
    }

    return 1;
}
```

The first_verif() function is responsible for validating whether a slang word is acceptable before being inserted into the Trie. It enforces two basic input rules: the word must contain more than one character, and it must not contain any spaces. The first condition is checked using strlen(word) <= 1, which ensures that single-character or empty inputs are rejected. The second condition is checked in a loop that scans each character; if a space character (' ') is found, the function immediately returns 0, signaling that the input is invalid.

If the word passes both checks—being longer than one character and containing no spaces—the function returns 1, meaning the word is considered valid for insertion. This validation helps ensure that the slang words stored in the dictionary are meaningful,

consistent, and suitable for prefix-based searches in a Trie. It also prevents formatting or logical errors from user input that could otherwise disrupt Trie behavior.

12. Function : second_verif(const char *word)

```
int second_verif(const char *word)

{

    int space = 0;

    for (int i = 0; word[i] != '\0'; i++)

    {

        if (word[i] == ' ')

        {

            space++;

        }

    }



    return space >= 2;

}
```

The second_verif() function validates whether a slang word's **description** is sufficiently descriptive by ensuring it contains **at least three words**. This is determined by counting the number of spaces (' ') in the input string. The function initializes a space counter to zero, then iterates through the input word character by character. Each time a space is found, the counter is incremented.

At the end of the loop, the function returns the result of the expression space >= 2, which evaluates to 1 (true) if there are at least two spaces—implying the description contains at least three words. Otherwise, it returns 0 (false). This ensures users provide

meaningful and informative definitions for each slang word, improving the quality and usefulness of the data stored in the dictionary.

13. Function : main()

```
int main()

{

    node *root = create_node();

    char word[100];

    char word_desc[1000];

    int menu_option;

    do

    {

        system("cls");

        print_menu();

        scanf("%d", &menu_option);

        getchar();



        switch (menu_option)

        {

        case 1:

            do

            {

                    printf("Input a new slang word [Must be more than 1
characters and contains no space]: ");

                scanf("%[^\n]", word);
```

```
                getchar();
            } while (!first_verif(word));
            do
            {
                 printf("Input a new slang word description [Must be more
than 2 words]: ");
                scanf("%[^\n]", word_desc);
                getchar();
            } while (!second_verif(word_desc));
            insert(root, word, word_desc);
            system("pause");
            break;
        case 2:
            do
            {
                 printf("Input a slang word to be searched [Must be more
than 1 characters and contains no space]: ");
                scanf("%[^\n]", word);
                getchar();
            } while (!first_verif(word));
            search(root, word);
            system("pause");
            break;
        case 3:
            printf("Input a prefix to be searched: ");
```

```
            scanf("%s", word);
            getchar();
            show_based_prefix(root, word);
            system("pause");
            break;
        case 4:
            show_all(root);
            system("pause");
            break;
        case 5:
            exit_trie(root);
            printf("Thank you... Have a nice day :)");
            break;
        }
    } while (menu_option != 5);


    return 0;
}
```

The main() function serves as the **central control loop** of the application, orchestrating all interactions between the user and the underlying Trie data structure. It begins by initializing the Trie with a call to create_node(), creating an empty root node. Two character arrays, word and word_desc, are used to temporarily store user input for slang words and their descriptions. The menu_option variable is used to track the user's chosen action.

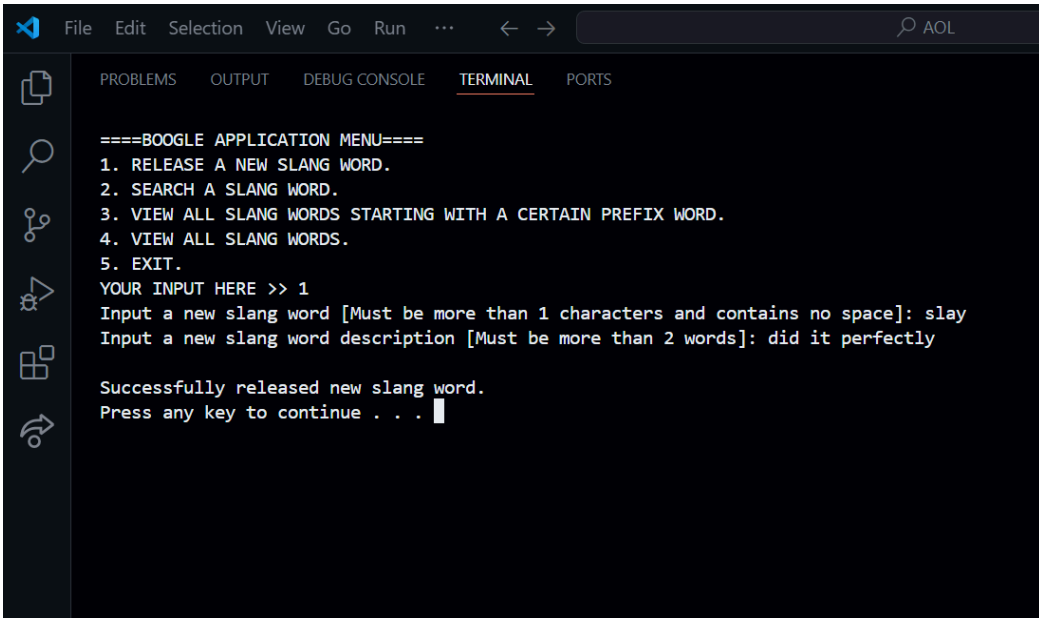
The program then enters a do-while loop that keeps running until the user selects option 5 (Exit). Each iteration clears the screen with system("cls") and displays the application menu using print_menu(). It reads the user's choice and branches using a switch statement:

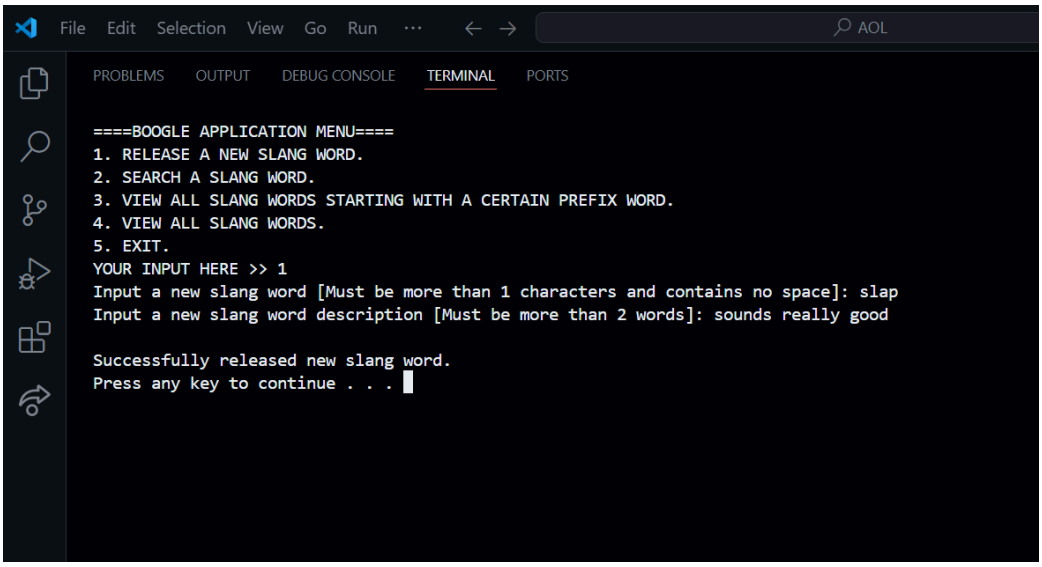
- **Case 1**: The user is prompted to input a new slang word and its description. Both inputs are validated using first_verif() and second_verif() to ensure correct formatting. Once valid, insert() adds or updates the word in the Trie.
- **Case 2**: The user enters a slang word to search. After validation, search() looks it up and displays the result.
- **Case 3**: The user enters a prefix, and show_based_prefix() lists all slang words that start with that prefix.
- **Case 4**: show_all() is called to display every slang word in the Trie.
- **Case 5**: The program calls exit_trie() to free all dynamically allocated memory and prints a farewell message.

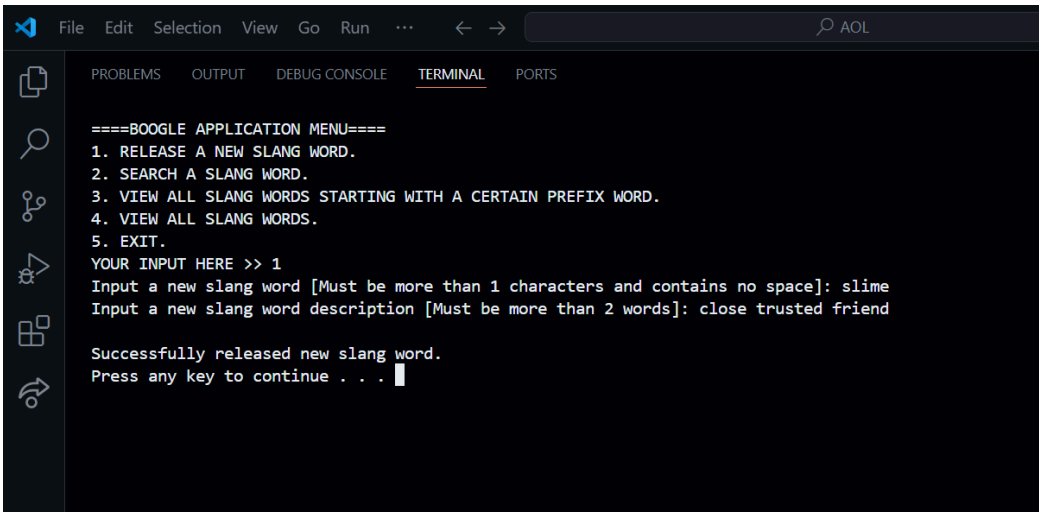
The use of the system("pause") ensures that users have time to read the results before the screen is cleared again. Overall, this main loop provides a structured, menu-driven user interface that coordinates validation, Trie operations, and memory management efficiently.

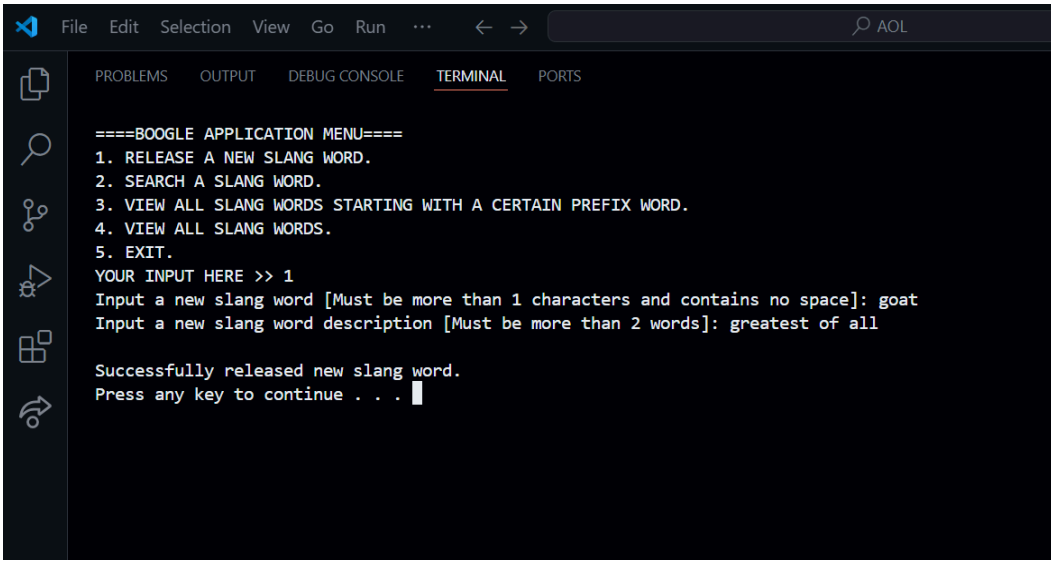
14. Custom Cases

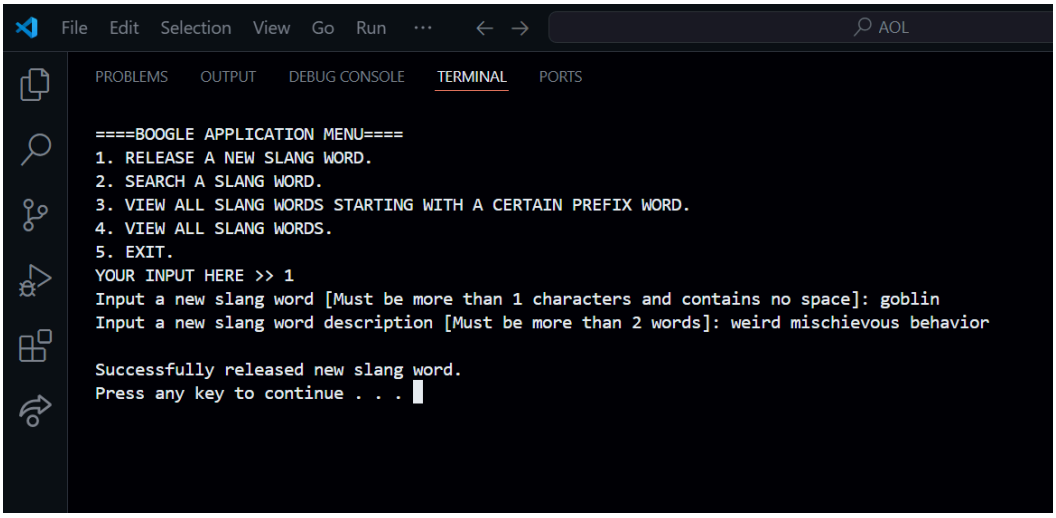
- Input of 15 slang words

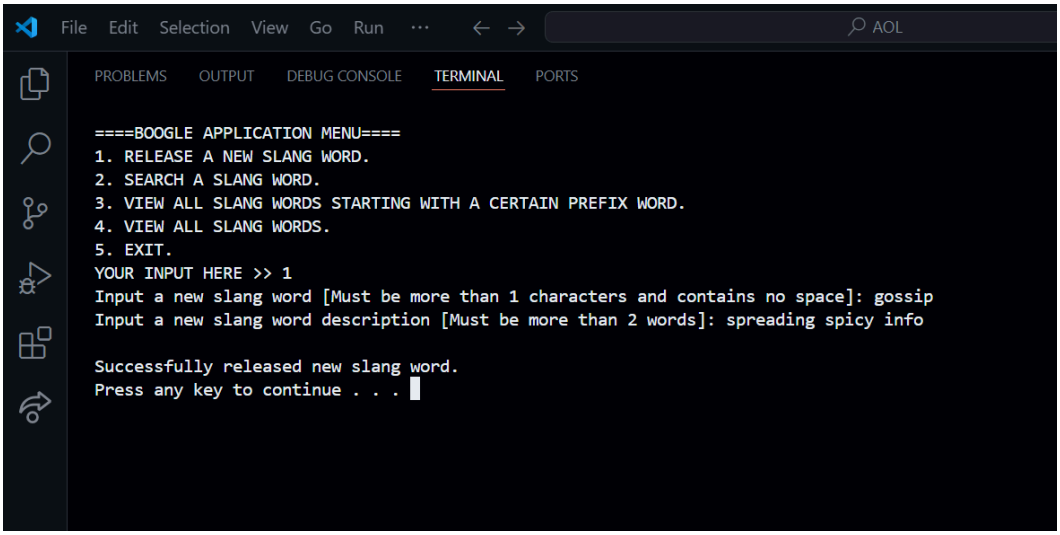

File Edit Selection View Go Run ··· AOL
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
====BOOGLE APPLICATION MENU====
1. RELEASE A NEW SLANG WORD.
2. SEARCH A SLANG WORD.
3. VIEW ALL SLANG WORDS STARTING WITH A CERTAIN PREFIX WORD.
4. VIEW ALL SLANG WORDS.
5. EXIT.
YOUR INPUT HERE >> 1
Input a new slang word [Must be more than 1 characters and contains no space]: slay
Input a new slang word description [Must be more than 2 words]: did it perfectly
Successfully released new slang word.
Press any key to continue . . .

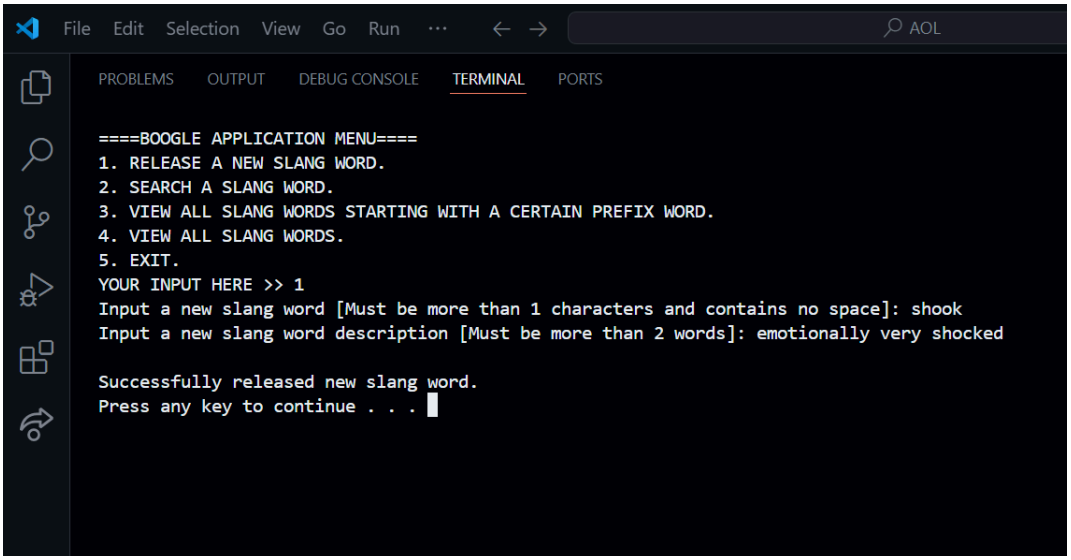

File Edit Selection View Go Run ··· AOL
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
====BOOGLE APPLICATION MENU====
1. RELEASE A NEW SLANG WORD.
2. SEARCH A SLANG WORD.
3. VIEW ALL SLANG WORDS STARTING WITH A CERTAIN PREFIX WORD.
4. VIEW ALL SLANG WORDS.
5. EXIT.
YOUR INPUT HERE >> 1
Input a new slang word [Must be more than 1 characters and contains no space]: slap
Input a new slang word description [Must be more than 2 words]: sounds really good
Successfully released new slang word.
Press any key to continue . . .

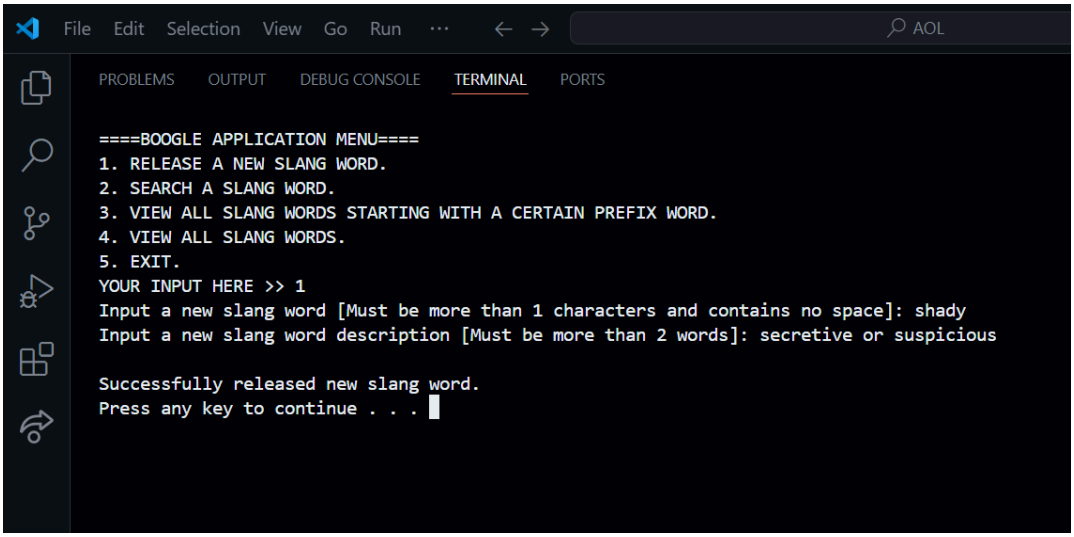

File Edit Selection View Go Run ··· AOL
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
====BOOGLE APPLICATION MENU====
1. RELEASE A NEW SLANG WORD.
2. SEARCH A SLANG WORD.
3. VIEW ALL SLANG WORDS STARTING WITH A CERTAIN PREFIX WORD.
4. VIEW ALL SLANG WORDS.
5. EXIT.
YOUR INPUT HERE >> 1
Input a new slang word [Must be more than 1 characters and contains no space]: slime
Input a new slang word description [Must be more than 2 words]: close trusted friend
Successfully released new slang word.
Press any key to continue . . .

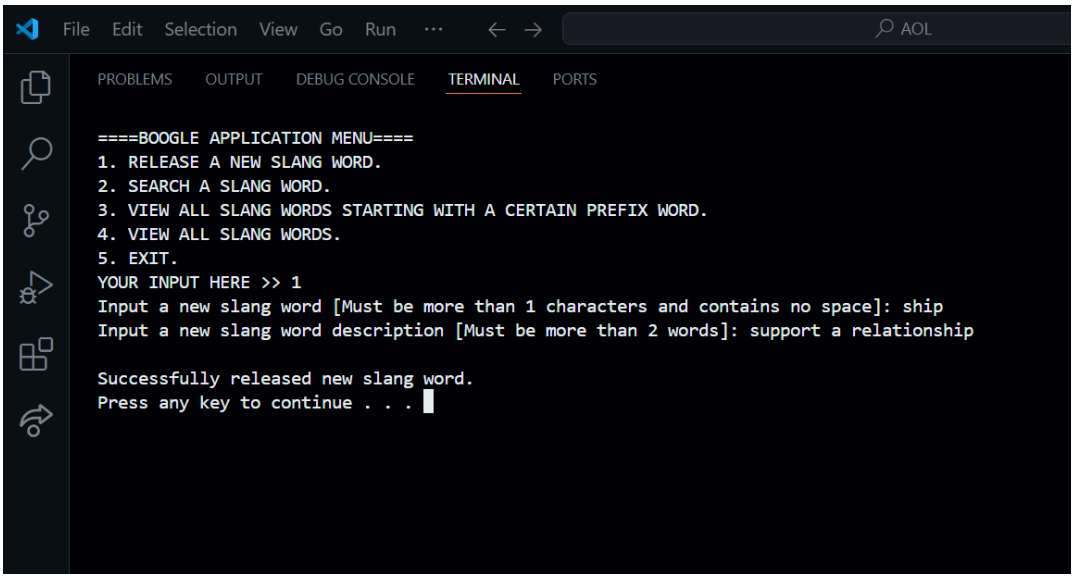

File Edit Selection View Go Run ···
AOL
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
====BOOGLE APPLICATION MENU====
1. RELEASE A NEW SLANG WORD.
2. SEARCH A SLANG WORD.
3. VIEW ALL SLANG WORDS STARTING WITH A CERTAIN PREFIX WORD.
4. VIEW ALL SLANG WORDS.
5. EXIT.
YOUR INPUT HERE >> 1
Input a new slang word [Must be more than 1 characters and contains no space]: goat
Input a new slang word description [Must be more than 2 words]: greatest of all
Successfully released new slang word.
Press any key to continue . . .

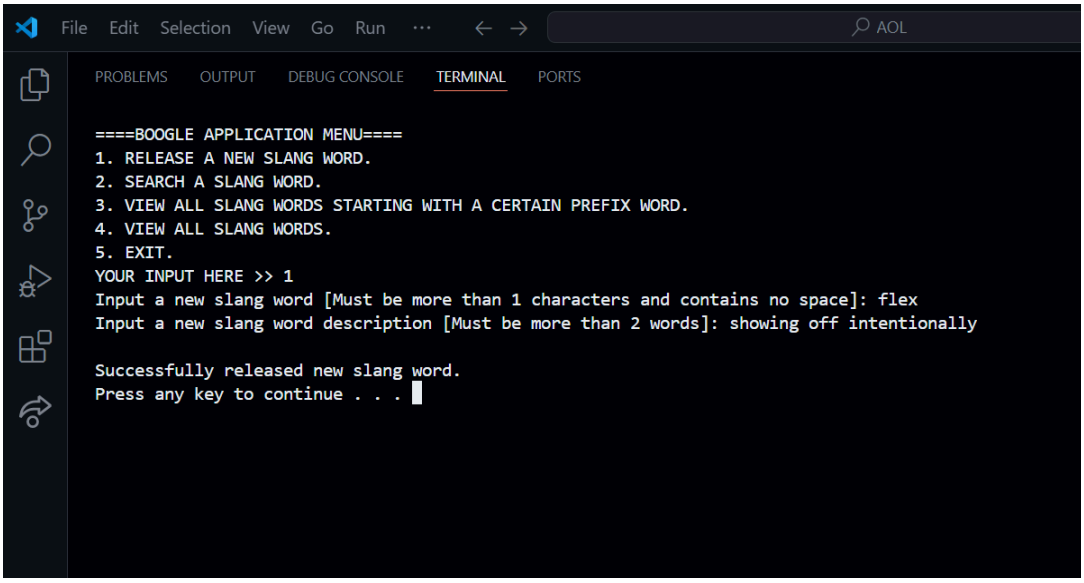

File Edit Selection View Go Run ···
AOL
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
====BOOGLE APPLICATION MENU====
1. RELEASE A NEW SLANG WORD.
2. SEARCH A SLANG WORD.
3. VIEW ALL SLANG WORDS STARTING WITH A CERTAIN PREFIX WORD.
4. VIEW ALL SLANG WORDS.
5. EXIT.
YOUR INPUT HERE >> 1
Input a new slang word [Must be more than 1 characters and contains no space]: goblin
Input a new slang word description [Must be more than 2 words]: weird mischievous behavior
Successfully released new slang word.
Press any key to continue . . .

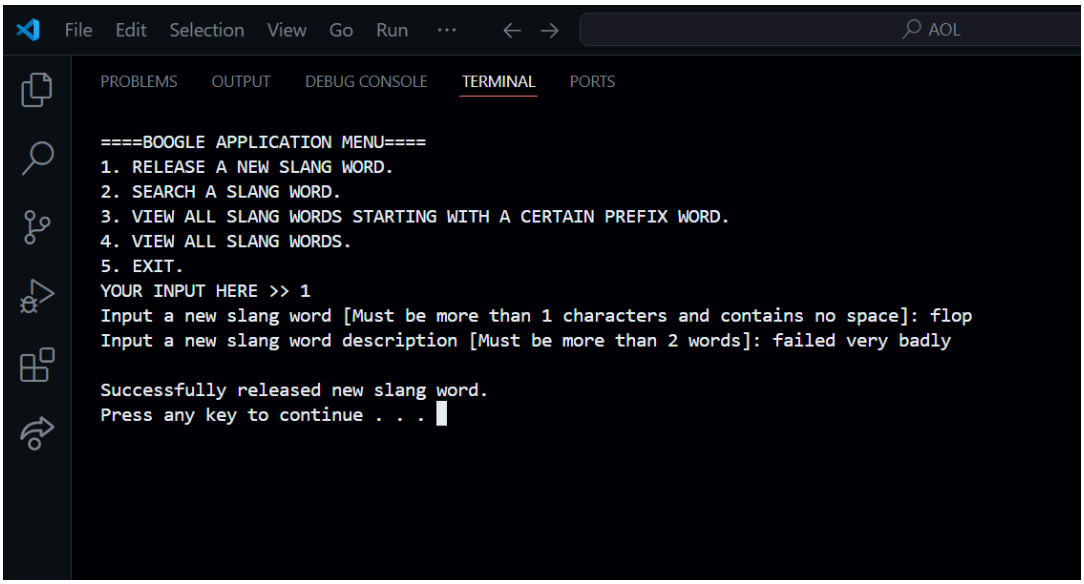

File Edit Selection View Go Run ···
AOL
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
====BOOGLE APPLICATION MENU====
1. RELEASE A NEW SLANG WORD.
2. SEARCH A SLANG WORD.
3. VIEW ALL SLANG WORDS STARTING WITH A CERTAIN PREFIX WORD.
4. VIEW ALL SLANG WORDS.
5. EXIT.
YOUR INPUT HERE >> 1
Input a new slang word [Must be more than 1 characters and contains no space]: gossip
Input a new slang word description [Must be more than 2 words]: spreading spicy info
Successfully released new slang word.
Press any key to continue . . .

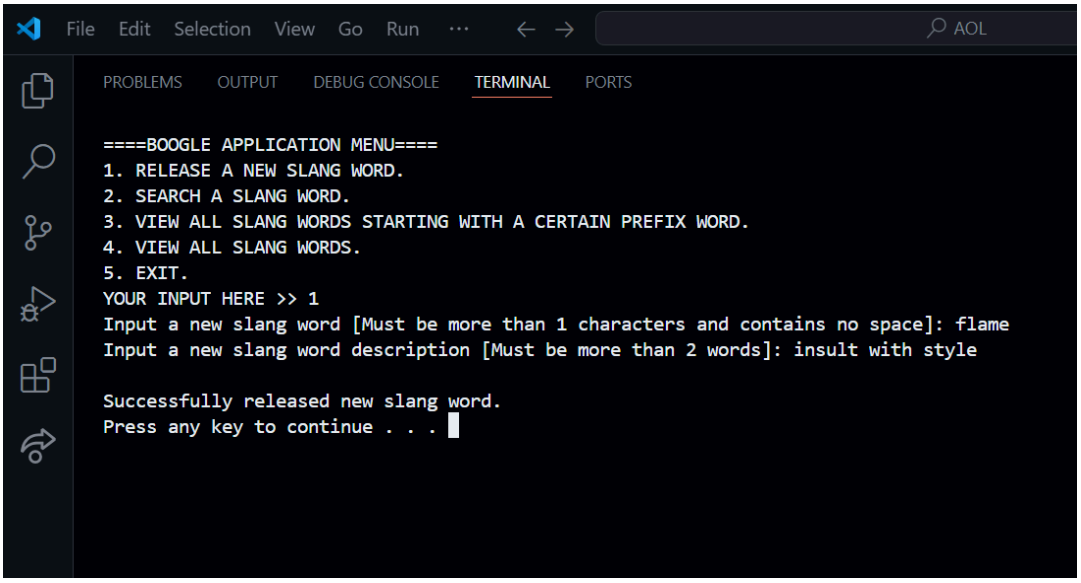

File Edit Selection View Go Run ···
AOL
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
====BOOGLE APPLICATION MENU====
1. RELEASE A NEW SLANG WORD.
2. SEARCH A SLANG WORD.
3. VIEW ALL SLANG WORDS STARTING WITH A CERTAIN PREFIX WORD.
4. VIEW ALL SLANG WORDS.
5. EXIT.
YOUR INPUT HERE >> 1
Input a new slang word [Must be more than 1 characters and contains no space]: shook
Input a new slang word description [Must be more than 2 words]: emotionally very shocked
Successfully released new slang word.
Press any key to continue . . .

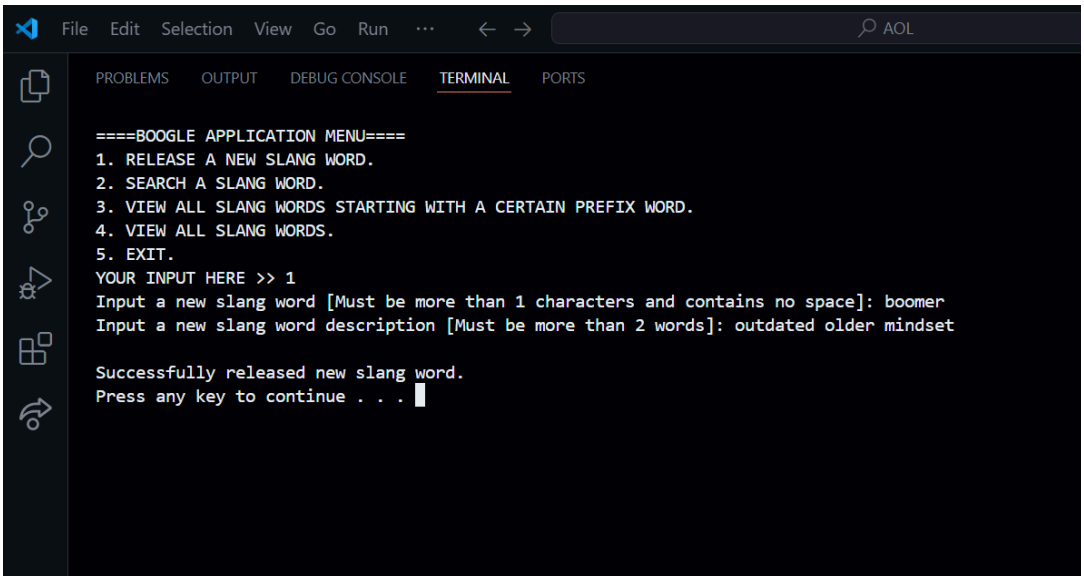

File Edit Selection View Go Run ···
AOL
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
====BOOGLE APPLICATION MENU====
1. RELEASE A NEW SLANG WORD.
2. SEARCH A SLANG WORD.
3. VIEW ALL SLANG WORDS STARTING WITH A CERTAIN PREFIX WORD.
4. VIEW ALL SLANG WORDS.
5. EXIT.
YOUR INPUT HERE >> 1
Input a new slang word [Must be more than 1 characters and contains no space]: shady
Input a new slang word description [Must be more than 2 words]: secretive or suspicious
Successfully released new slang word.
Press any key to continue . . .

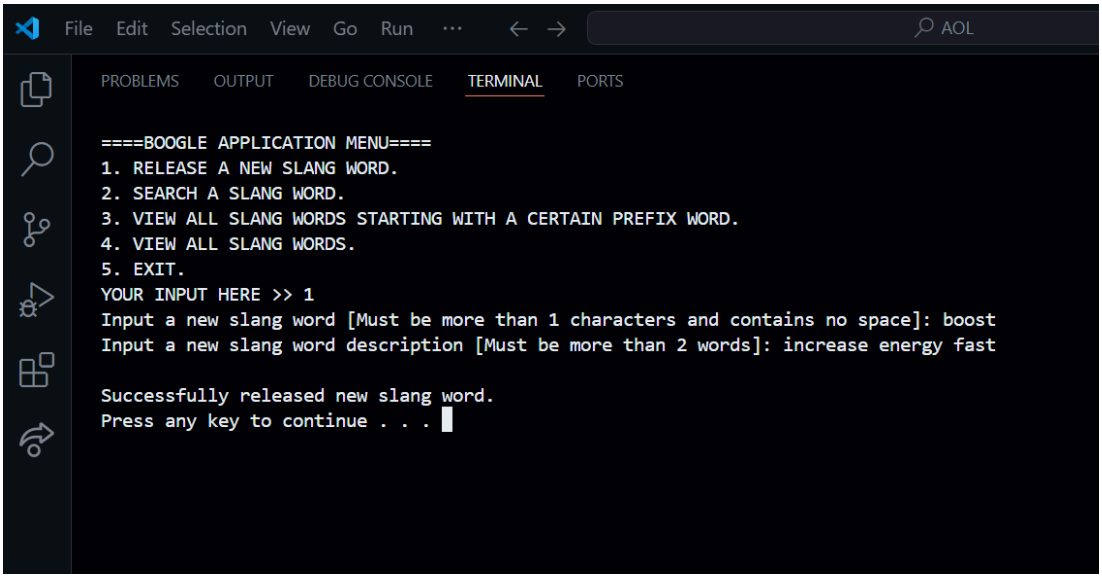

File Edit Selection View Go Run ···
AOL
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
====BOOGLE APPLICATION MENU====
1. RELEASE A NEW SLANG WORD.
2. SEARCH A SLANG WORD.
3. VIEW ALL SLANG WORDS STARTING WITH A CERTAIN PREFIX WORD.
4. VIEW ALL SLANG WORDS.
5. EXIT.
YOUR INPUT HERE >> 1
Input a new slang word [Must be more than 1 characters and contains no space]: ship
Input a new slang word description [Must be more than 2 words]: support a relationship
Successfully released new slang word.
Press any key to continue . . .

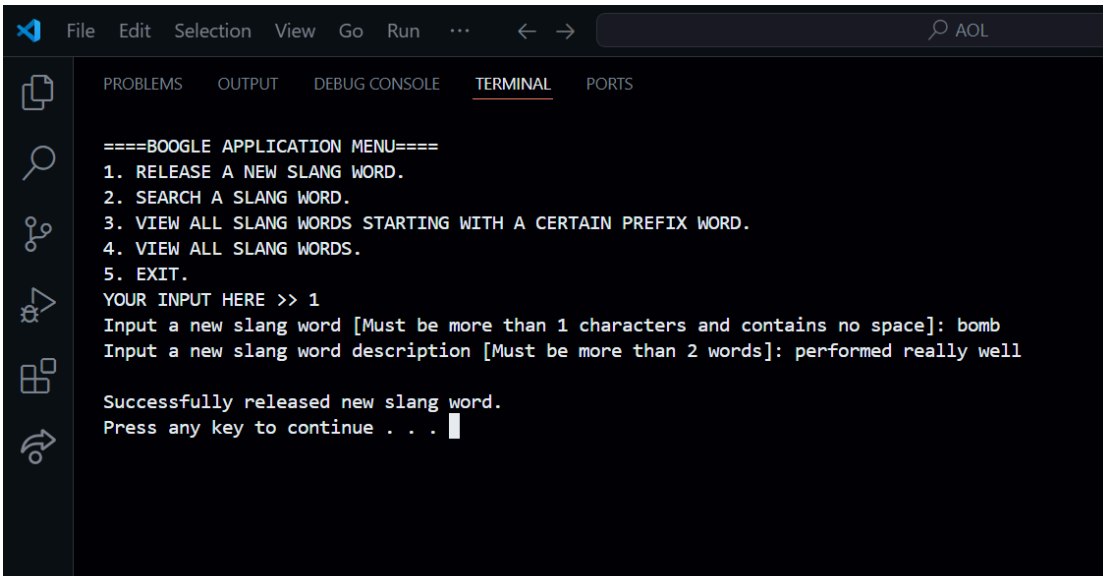

File Edit Selection View Go Run ···
AOL
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
====BOOGLE APPLICATION MENU====
1. RELEASE A NEW SLANG WORD.
2. SEARCH A SLANG WORD.
3. VIEW ALL SLANG WORDS STARTING WITH A CERTAIN PREFIX WORD.
4. VIEW ALL SLANG WORDS.
5. EXIT.
YOUR INPUT HERE >> 1
Input a new slang word [Must be more than 1 characters and contains no space]: flex
Input a new slang word description [Must be more than 2 words]: showing off intentionally
Successfully released new slang word.
Press any key to continue . . .


File Edit Selection View Go Run ···
AOL
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
====BOOGLE APPLICATION MENU====
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.
YOUR INPUT HERE >> 1
Input a new slang word [Must be more than 1 characters and contains no space]: flop
Input a new slang word description [Must be more than 2 words]: failed very badly
Successfully released new slang word.
Press any key to continue . . .

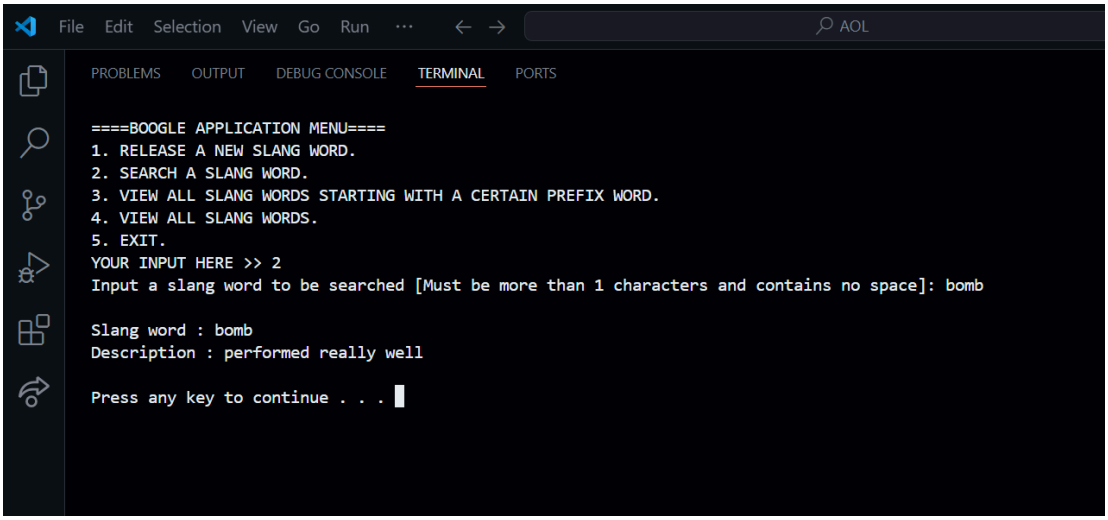

File Edit Selection View Go Run ···
AOL
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
====BOOGLE APPLICATION MENU====
1. RELEASE A NEW SLANG WORD.
2. SEARCH A SLANG WORD.
3. VIEW ALL SLANG WORDS STARTING WITH A CERTAIN PREFIX WORD.
4. VIEW ALL SLANG WORDS.
5. EXIT.
YOUR INPUT HERE >> 1
Input a new slang word [Must be more than 1 characters and contains no space]: flame
Input a new slang word description [Must be more than 2 words]: insult with style
Successfully released new slang word.
Press any key to continue . . .

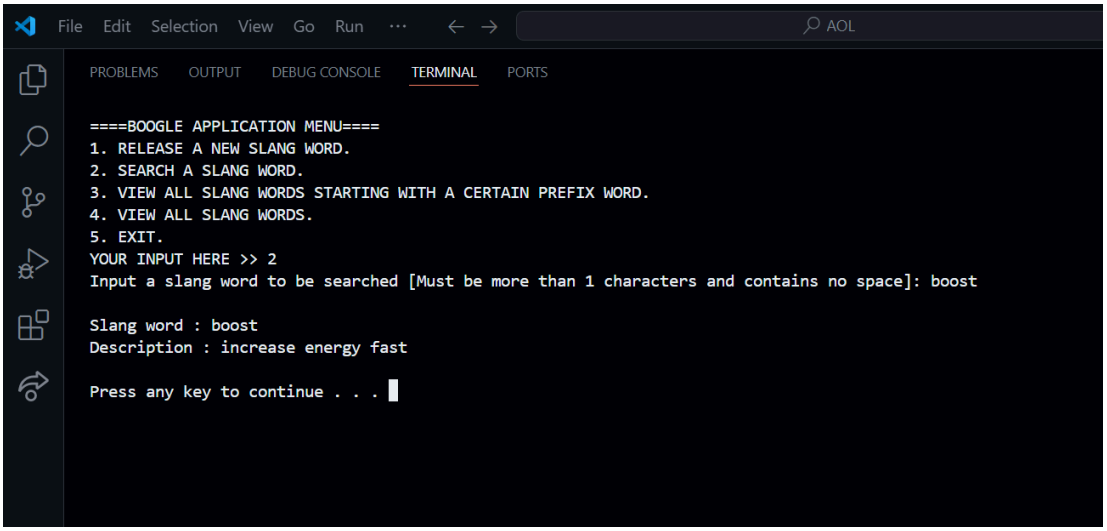

File Edit Selection View Go Run ···
AOL
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
====BOOGLE APPLICATION MENU====
1. RELEASE A NEW SLANG WORD.
2. SEARCH A SLANG WORD.
3. VIEW ALL SLANG WORDS STARTING WITH A CERTAIN PREFIX WORD.
4. VIEW ALL SLANG WORDS.
5. EXIT.
YOUR INPUT HERE >> 1
Input a new slang word [Must be more than 1 characters and contains no space]: boomer
Input a new slang word description [Must be more than 2 words]: outdated older mindset
Successfully released new slang word.
Press any key to continue . . .

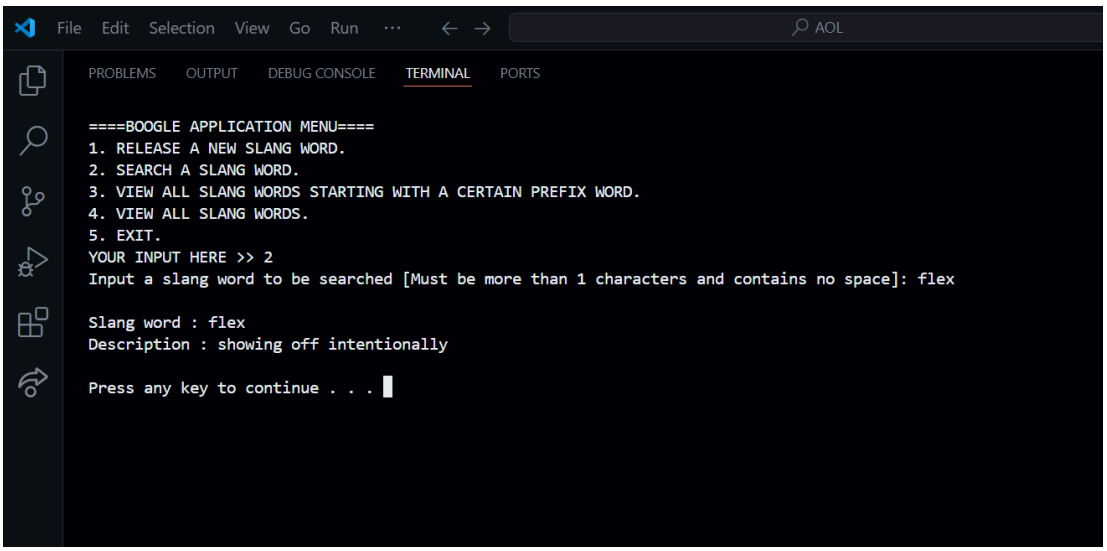


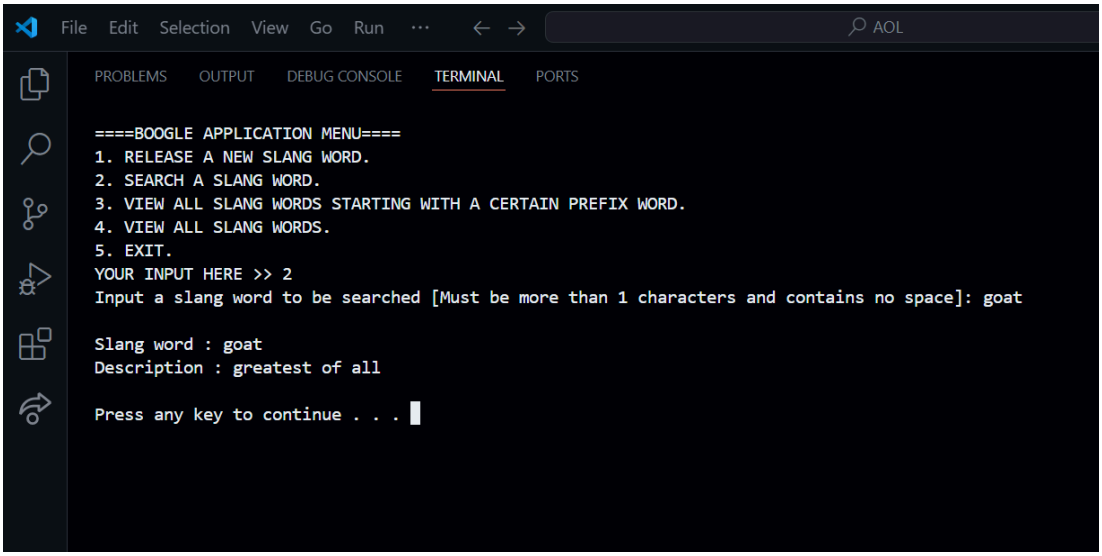


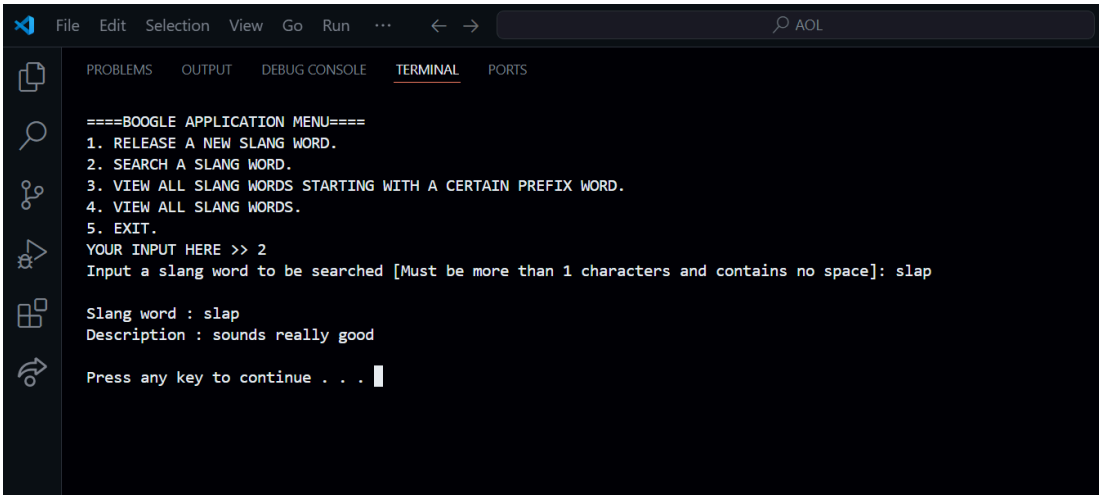
- Search 5 words


File Edit Selection View Go Run ···
AOL
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
====BOOGLE APPLICATION MENU====
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.
YOUR INPUT HERE >> 2
Input a slang word to be searched [Must be more than 1 characters and contains no space]: bomb
Slang word : bomb
Description : performed really well
Press any key to continue . . .

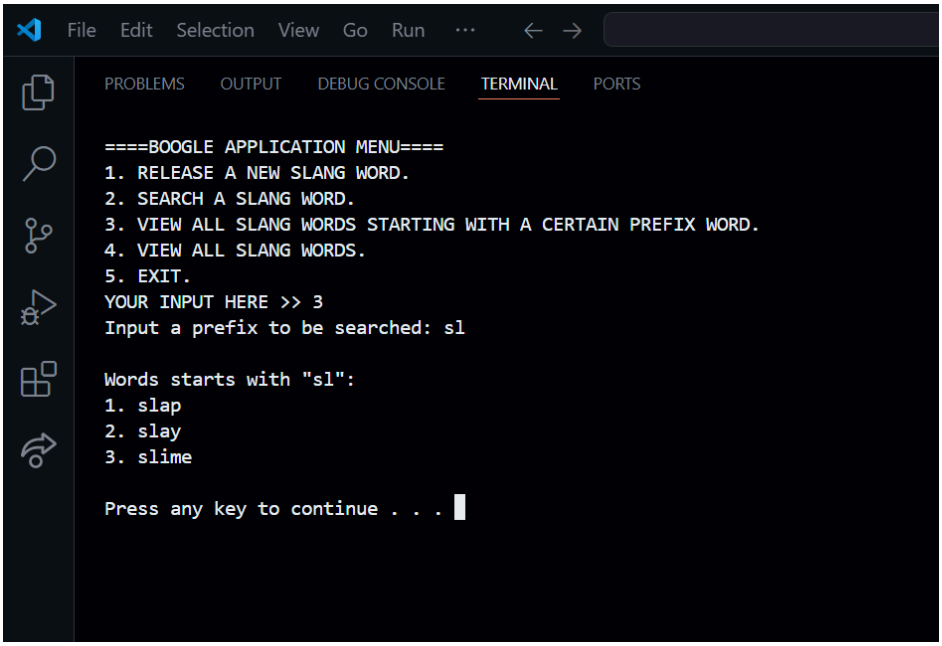

File Edit Selection View Go Run ···
AOL
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
====BOOGLE APPLICATION MENU====
1. RELEASE A NEW SLANG WORD.
2. SEARCH A SLANG WORD.
3. VIEW ALL SLANG WORDS STARTING WITH A CERTAIN PREFIX WORD.
4. VIEW ALL SLANG WORDS.
5. EXIT.
YOUR INPUT HERE >> 2
Input a slang word to be searched [Must be more than 1 characters and contains no space]: boost
Slang word : boost
Description : increase energy fast
Press any key to continue . . .

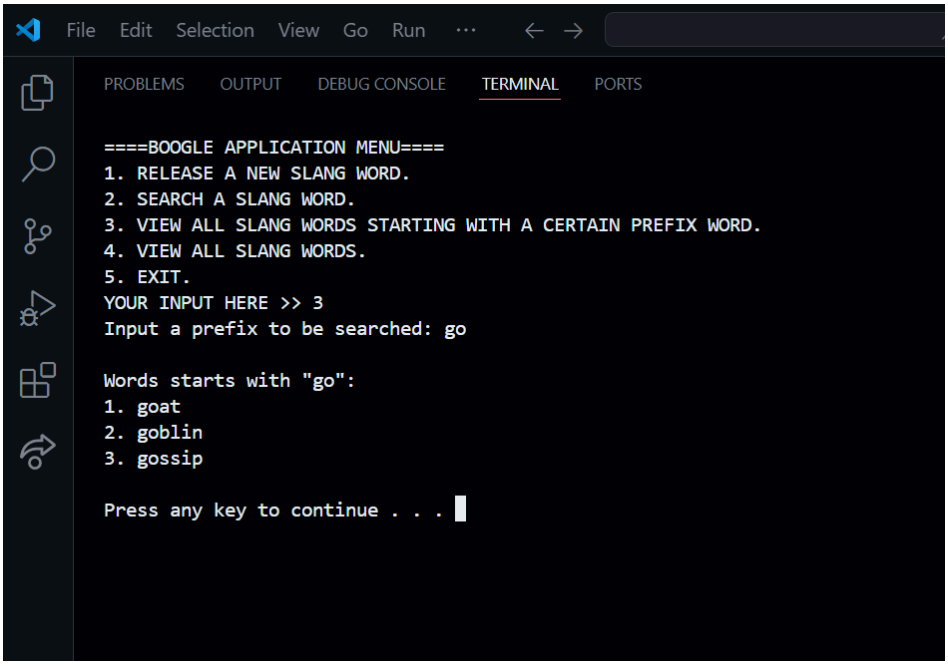

File Edit Selection View Go Run ···
AOL
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
====BOOGLE APPLICATION MENU====
1. RELEASE A NEW SLANG WORD.
2. SEARCH A SLANG WORD.
3. VIEW ALL SLANG WORDS STARTING WITH A CERTAIN PREFIX WORD.
4. VIEW ALL SLANG WORDS.
5. EXIT.
YOUR INPUT HERE >> 2
Input a slang word to be searched [Must be more than 1 characters and contains no space]: flex
Slang word : flex
Description : showing off intentionally
Press any key to continue . . .

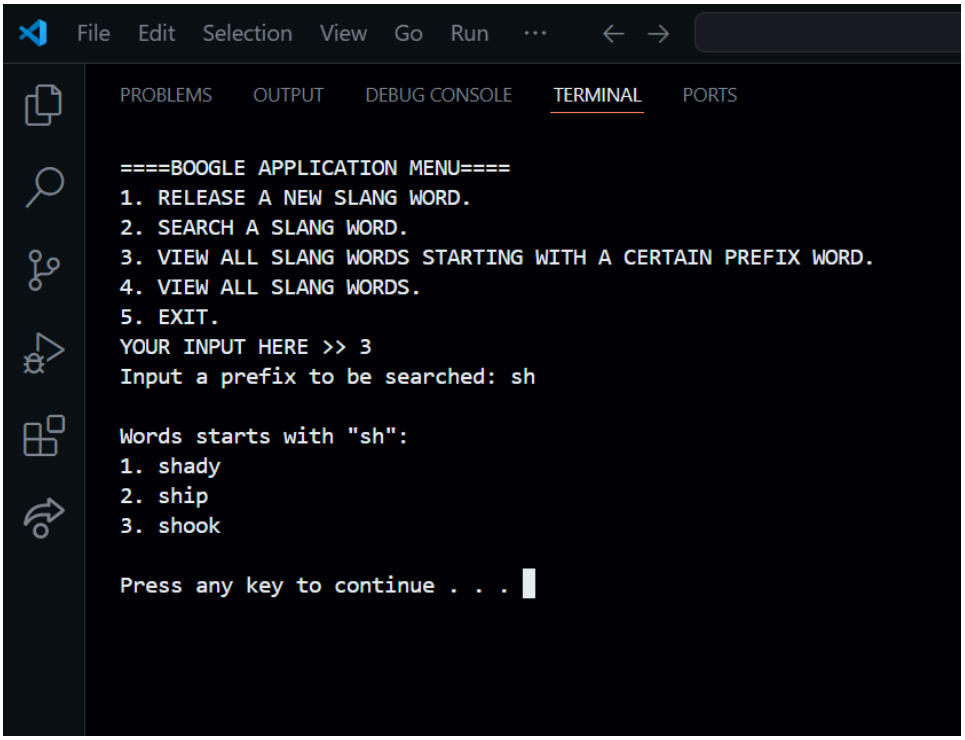


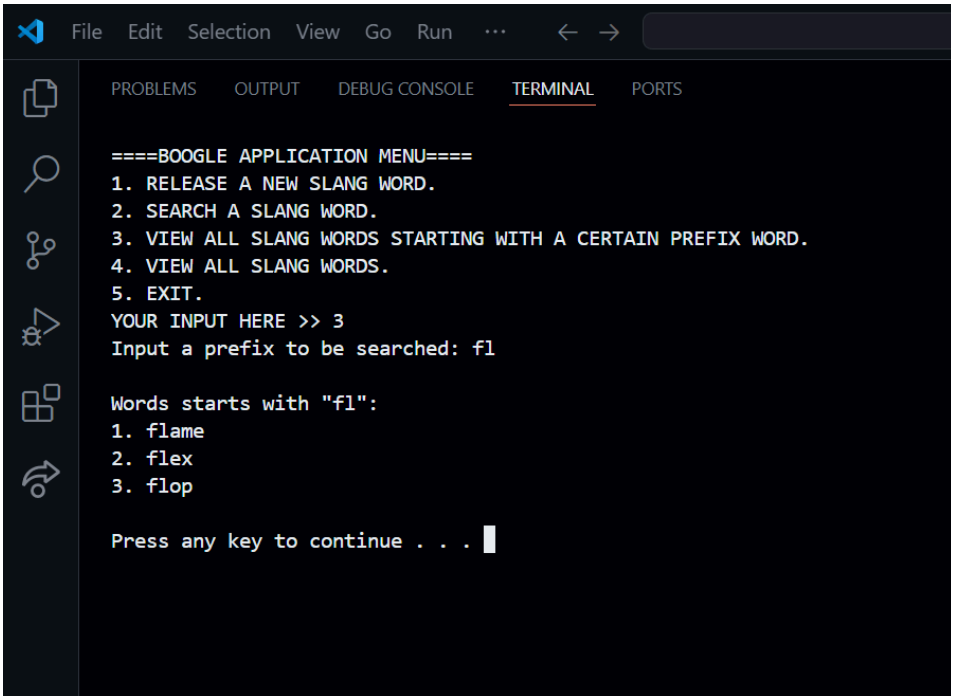


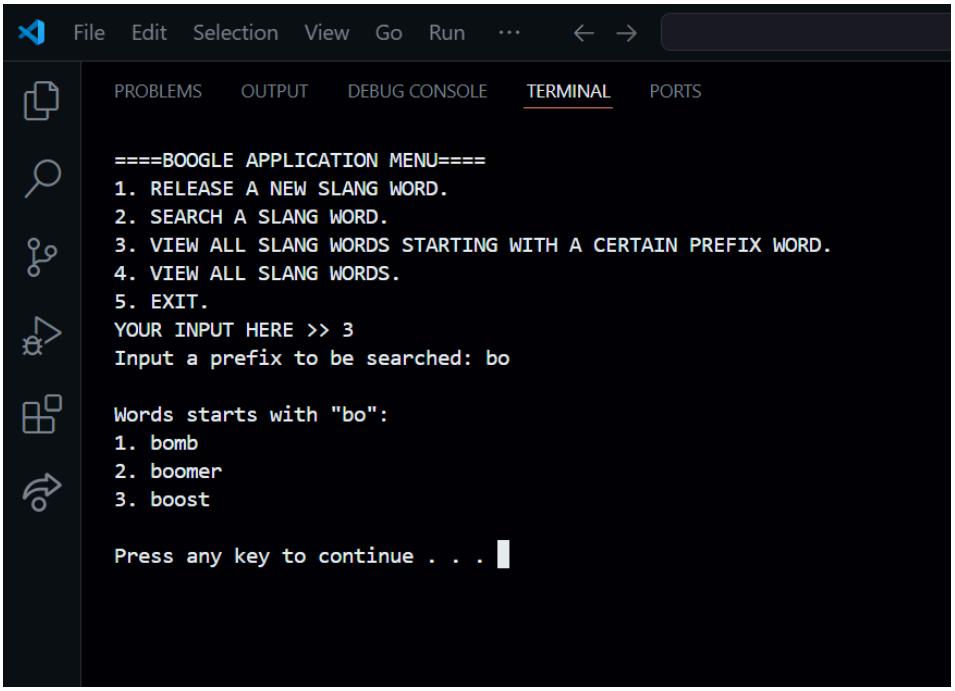
- View prefix 5 words


File Edit Selection View Go Run ···
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
====BOOGLE APPLICATION MENU====
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.
YOUR INPUT HERE >> 3
Input a prefix to be searched: sl
Words starts with "sl":
1. slap
2. slay
3. slime
Press any key to continue . . .

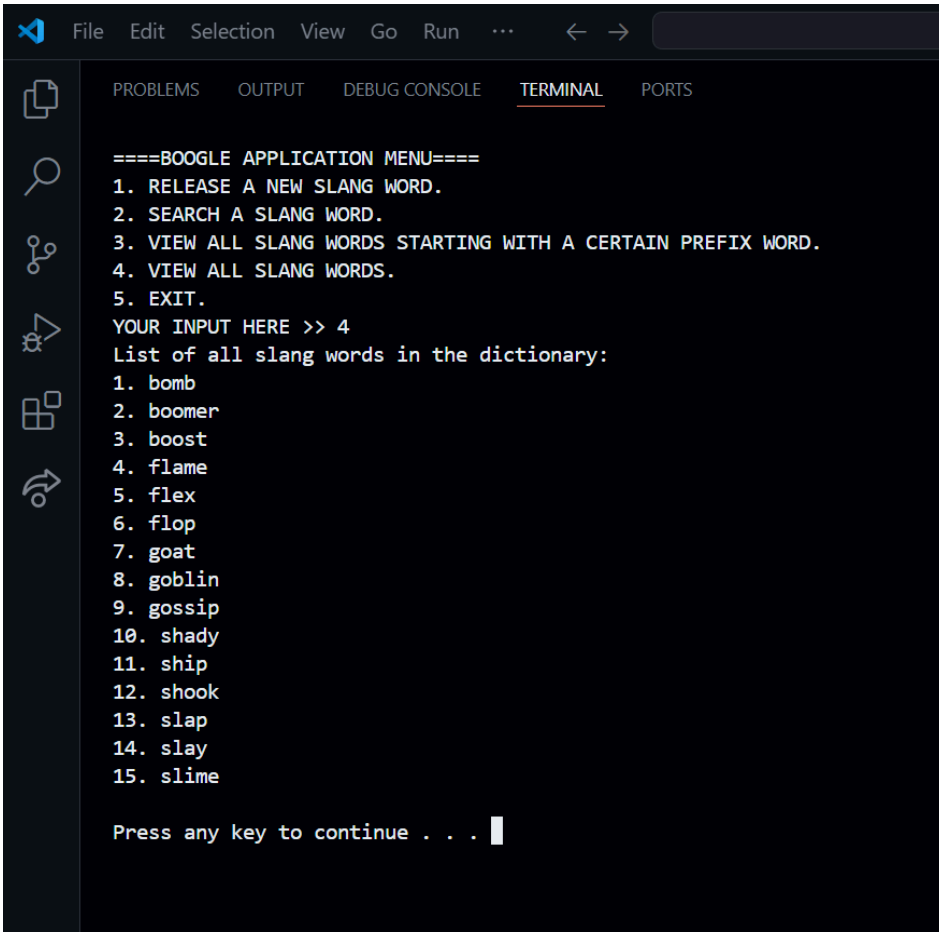

File Edit Selection View Go Run ···
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
====BOOGLE APPLICATION MENU====
1. RELEASE A NEW SLANG WORD.
2. SEARCH A SLANG WORD.
3. VIEW ALL SLANG WORDS STARTING WITH A CERTAIN PREFIX WORD.
4. VIEW ALL SLANG WORDS.
5. EXIT.
YOUR INPUT HERE >> 3
Input a prefix to be searched: go
Words starts with "go":
1. goat
2. goblin
3. gossip
Press any key to continue . . .


File Edit Selection View Go Run ···
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
====BOOGLE APPLICATION MENU====
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.
YOUR INPUT HERE >> 3
Input a prefix to be searched: sh
Words starts with "sh":
1. shady
2. ship
3. shook
Press any key to continue . . .


File Edit Selection View Go Run ···
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
====BOOGLE APPLICATION MENU====
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.
YOUR INPUT HERE >> 3
Input a prefix to be searched: fl
Words starts with "fl":
1. flame
2. flex
3. flop
Press any key to continue . . .


File Edit Selection View Go Run ···
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
====BOOGLE APPLICATION MENU====
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.
YOUR INPUT HERE >> 3
Input a prefix to be searched: bo
Words starts with "bo":
1. bomb
2. boomer
3. boost
Press any key to continue . . .

- View all


File Edit Selection View Go Run ···
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
====BOOGLE APPLICATION MENU====
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.
YOUR INPUT HERE >> 4
List of all slang words in the dictionary:
1. bomb
2. boomer
3. boost
4. flame
5. flex
6. flop
7. goat
8. goblin
9. gossip
10. shady
11. ship
12. shook
13. slap
14. slay
15. slime
Press any key to continue . . .