Nama : Alfandi Rifa'ul Nurhuda

NIM : 2702393673

Kelas : LM01

```c
// Include the standard input/output header file 'stdio.h
#include <stdio.h>
// Include the string header file 'string.h'
#include <string.h>
// Include the standard library header file 'stdlib.h'
#include <stdlib.h>
// Include the boolean header file 'stdbool.h'
#include <stdbool.h>
```

The code above have function to include the library:

1. #include <stdio.h>: This line includes the standard input/output library in C. This library contains functions for performing input and output operations like printf and scanf.
2. #include <string.h>: This line includes the string library in C. This library contains functions for manipulating arrays of characters (strings), such as strcpy, strlen, strcat, etc.
3. #include <stdlib.h>: This line includes the standard library in C. This library contains general purpose functions, including dynamic memory management, random number generation, communication with the environment, integer arithmetics, searching, sorting and converting.
4. #include <stdbool.h>: This line includes the boolean library in C. This library provides a bool type for C (which is not built-in in the language), and defines the macros true and false.

```c
// Constan variable to define the maximum length of the word
const int MAX_LENGTH = 100;
```

The code above is defining a constant interger variable and being used to set a maximum length for word. Which const variable cannot be change after initialized.

```c
// Constan variable to define the amount of alphabet
const int ALPHABET_SIZE = 26;
```

The code above is defining a constan interger variable and being used to represent the number of alphabet. Which const variable cannot be change after initialized.

```c
// Function to clear the screen with OS specific command
void clearScreen()
{
    // Checks if _WIN32 is defined, _WIN32 is defined in all window OS
    #ifdef _WIN32
        system("cls");
    // assume we are on a unix based system in which system(`clear`) is the command
    #else
        system("clear");
    #endif
```

```
}
```

The function above is used to clear a console screen. It uses preprocessor to check the macro of Operating System first to an then set a command of clear console screen. So we can use this function in another OS.

```
// Struct of trie node
    struct trieNode{
    bool isEndOfWord;
    char *desc;
    trieNode *child[ALPHABET_SIZE];
};
```

The struct above is define a strutcure trieNode to store a data. Each node in the trie represents a string (a prefix of the words in the trie). Each node has a number of child nodes equal to the number of letters in the alphabet (defined by ALPHABET_SIZE), where each child corresponds to a different letter.

- bool isEndOfWord: This boolean flag is used to mark the end of a word. If isEndOfWord is true, it means that the string represented by the path from the root of the trie to this node is a word in the trie.
- char *desc: This is a pointer to a character array (a string). It could be used to store additional information about the word. For example, if the trie is used to implement a dictionary, desc could be used to store the definition of the word.
- trieNode *child[ALPHABET_SIZE];: This is an array of pointers to the child nodes of this node. The size of the array is ALPHABET_SIZE, so there is one child for each letter in the alphabet.

```
// Function to convert character to index in range 0-25.
int charToIndex(char c)
{
   return (int)c - 'a';
}
```

The function above is used to convert a lowercase letter to an interger index in range 0-25. It takes a **char c** as parameter and subtracts the ASCII value of 'a' from it. So when you subtract the ASCII value of 'a' from a lowercase letter, you get a number in the range 0-25, where 'a' maps to 0, 'b' maps to 1, and so on up to 'z' which maps to 25.

```
// Function to create a new node
trieNode *createNode(char *slangWord, char *desc)
{
  trieNode *newNode = (trieNode*)malloc(sizeof(trieNode));

      // Initialize the node and set the isEndOfWord to false
      // Then allocate memory for the description and copy the description to the node
      if(newNode)
      {
        newNode->isEndOfWord = false;
        newNode->desc = (char*)malloc(strlen(desc) + 1);
        strcpy(newNode->desc, desc);

        // Initialize all child nodes to NULL
        // So that we can check if the child is present or not
```

```
        for(int i=0; i<ALPHABET_SIZE; i++)
        {
          newNode->child[i] = NULL;
        }
        return newNode;
  }
}
```

The code above is used to create a new node in a trie. This node can then be used to store a word (or a prefix of a word) in the trie. The desc field can be used to store additional information about the word, such as its definition or a description. The isEndOfWord field is used to mark the end of a word in the trie. The child array is used to store pointers to the child nodes of this node, where each child corresponds to a different letter in the alphabet.

```
// Function to search a node in the trie
trieNode *searchNode(trieNode *node, const char *slangWord)
{
        trieNode *current = node;

        // Traverse the trie and check if the node is present or not
        // If the node is present, then return the node
        for(int i=0; i<strlen(slangWord); i++)
        {
                int index = charToIndex(slangWord[i]);

                if(!current->child[index])
                {
                return NULL;
                }

                current = current->child[index];
        }

        // If the current node is the end of the word, then return the node
        return (current != NULL && current->isEndOfWord) ? current : NULL;
}
```

The function above is used to search for a word in a trie data structure. It starts at a given node and traverses the trie according to the characters in slangWord. For each character, it calculates the corresponding index and moves to the child node at that index. If at any point the required child node does not exist, it returns NULL, indicating that the word is not in the trie. If it successfully traverses all the characters in slangWord, it checks if the current node marks the end of a word. If it does, it returns the current node; otherwise, it returns NULL. This function is used to check if a word exists in the trie and to retrieve the node that represents that word.

```
// Function to insert a node in the trie
void insertNode(trieNode *node, char *word, char *desc)
{
      // Search the node in the trie
      trieNode *existingNode = searchNode(node, word);

      // If the node is exist, then update the description
      if(existingNode != NULL)
      {
```

```
            free(existingNode->desc);
            existingNode->desc = (char*)malloc(strlen(desc) + 1);
            strcpy(existingNode->desc, desc);

            puts("\nSuccessfully updated a slang word.");

            return;
        }

        // If the node is not exist, then create a new node
        else
        {
            trieNode *current = node;

            for(int i=0; i<strlen(word); i++)
            {
                int index = charToIndex(word[i]);

                if(!current->child[index])
                {
                  current->child[index] = createNode(word, desc);
                }

                current = current->child[index];
            }

            current->isEndOfWord = true;

            strcpy(current->desc, desc);

            puts("\nSuccessfully released new slang word.");
    }
}
```

The function aboive is used to insert a word into a trie data structure or update an existing word. It first searches for the word in the trie. If the word is found, it updates the description of the word by freeing the existing description, allocating new memory, and copying the new description into the allocated memory. If the word is not found, it creates a new node for each character in the word that doesn't already exist in the trie. It then marks the last node as the end of a word and copies the description into this node. This function is used to add new words to the trie or update the descriptions of existing words.

```
// Function to validate the word is valid or not valid
bool validateWord(char *word)
{
        // Check the word length is more than 1 and contains space or not
        if(strlen(word) > 1)
        {
            for(int i=0; i<strlen(word); i++)
            {
                if(word[i] == ' ')
                {
                  return false;
                }
```

```
        }
        return true;
      }
return false;
}
```

The function above is used to check if a given word is valid based on certain criteria. The function checks if the length of the word is more than 1 and if it contains any spaces. If the word's length is more than 1 and it does not contain any spaces, the function returns true, indicating that the word is valid. If the word's length is 1 or less, or if it contains any spaces, the function returns false, indicating that the word is not valid. This function is used to ensure that only valid words, as per the defined criteria, are processed further.

```
// Function to validate the description is valid or not valid
bool validateDescription(char *desc)
{
        // Variable to count the words in the description
        int count = 0;

        // Check the description contains more than 2 words or not
        for(int i=0; i<strlen(desc); i++)
        {
          if(desc[i] == ' ')
          {
            count++;
          }
        }

        if(count >= 1)
        {
          return true;
        }

        return false;
}
```

The function above is used to verify if a given description is valid based on a specific criterion. The function counts the number of spaces in the description, which indirectly gives the number of words (assuming words are separated by spaces). If the count of spaces is greater than or equal to 1, implying the description contains at least two words, the function returns true, indicating that the description is valid. If the count is less than 1, meaning the description contains one word or less, the function returns false, indicating that the description is not valid. This function is used to ensure that only valid descriptions, as per the defined criteria, are processed further.

```
// Function to realese new slang word or updated existing slang word
void realeseNewSlangWord(trieNode *node)
{
        char slangWord[MAX_LENGTH];
        char desc[MAX_LENGTH];
        getchar();

        // Ask the user for input the slang word
        // The slang word must be more than 1 characters and contains no space
        // And code will keep asking the user until the input is valid
```

```
        do
        {
                printf("Enter the slang word [Must be more than 1 characters and contains no   space]: ");
                scanf("%[^\n]", slangWord);
                getchar();
        }
        while(!validateWord(slangWord));

        // Ask the user for input the description
        // The description must be more than 2 words
        // And code will keep asking the user until the input is valid
        do
        {
                printf("Enter the description [Must be more than 2 words]: ");
                scanf("%[^\n]", desc);
                getchar();
        }
        while(!validateDescription(desc));

        // Insert the node in the trie if all of the input requirement is valid
        insertNode(node, slangWord, desc);
}
```

The function above is used to interact with the user to gather a new slang word and its description, and then insert this information into a trie data structure. The function first prompts the user to enter a slang word, ensuring it is more than one character and contains no spaces by using a do-while loop and the **validateWord** function. It then asks the user to enter a description, ensuring it contains more than two words, again using a do-while loop and the **validateDescription** function. Once both inputs are validated, the function calls **insertNode** to insert the new slang word and its description into the trie. This function is used to facilitate user interaction and manage the addition of new slang words to the trie.

```
// Function to search a slang word in the trie
void searchSlangWord(trieNode *node)
{
        char slangWord[10];
        trieNode *searchedNode;

        // Ask the user for input the slang word
        // The slang word must be more than 1 characters and contains no space
        // And code will keep asking the user until the input is valid
        do
        {
                printf("Input a slang word to be searched [Must be more than 1 characters and contain no space]: ");
                scanf("%s", slangWord);
                getchar();
        }
        while(!validateWord(slangWord));

        // Search the node in the trie
        searchedNode = searchNode(node, slangWord);

        // If the node is exist, then print the slang word and the description
        if(searchedNode != NULL)
```

```
        {
            puts("");
            printf("Slang word : %s\n", slangWord);
            printf("Description : %s\n", searchedNode->desc);
            puts("");
        }

        // If the node is not exist, then print the message
        else
        {
            puts("");
            printf("There is no word \"%s\" in the dictionary.\n", slangWord);
        }
}
```

The function above is used to search for a slang word in a trie data structure based on user input. The function prompts the user to enter a slang word, ensuring it is more than one character and contains no spaces by using a do-while loop and the **validateWord** function. Once a valid word is entered, the function calls **searchNode** to search for the word in the trie. If the word is found, the function prints the slang word and its description. If the word is not found, the function informs the user that the word is not in the dictionary. This function is used to facilitate user interaction and manage the search of slang words in the trie.

```
// Function to print all words in the trie
void printAllWords(trieNode *node, char *prefix, int index, int *count)
{
        // If the node is the end of the word, then print the word
        if(node->isEndOfWord)
        {
            prefix[index] = '\0';
            printf("%d. %s\n",(*count)++, prefix);
        }

        // Traverse all child nodes and print the word
        // If the child is exist, then call the function recursively to print the word
        for(int i=0; i<ALPHABET_SIZE; i++)
        {
            if(node->child[i])
            {
                char newPrefix[index + 2];
                strncpy(newPrefix, prefix, index);
                newPrefix[index] = 'a' + i;
                newPrefix[index + 1] = '\0';
                printAllWords(node->child[i], newPrefix, (index + 1), count);
            }
        }

        // Set the last character of the prefix to null character
        // So that the prefix will be empty
        prefix[index] = '\0';
}
```

The function above is used to print all words stored in a trie data structure. The function traverses the trie recursively. If it reaches a node that marks the end of a word, it prints the word, which is constructed from the prefix and the current index. It then iterates over all

child nodes of the current node. If a child node exists, it constructs a new prefix by appending the corresponding character to the current prefix and calls itself recursively with the child node and the new prefix. After the recursive call, it sets the last character of the prefix to the null character, effectively removing the last character. This function is used to print all words in the trie in alphabetical order.

```
// Function to print all words with prefix in the trie
void printWordsWithPrefix(trieNode *node, char *prefix)
{
	trieNode *current = node;
	int len = strlen(prefix);
	char word[len + 1];
	int count = 1;

	// Traverse the trie and check if the prefix is exist or not
	for(int i=0; i<len; i++)
	{
		// Convert the character to index
		int index = charToIndex(prefix[i]);

		// If the child is not exist, then print the message
		if(!current->child[index])
		{
		  printf("There is no prefix \"%s\" in the dictionary\n", prefix);
		  return;
		}
		// Copy the prefix to the word so that the word will be the same as the prefix
		// Then set the current node to the child node
		word[i] = prefix[i];
		current = current->child[index];
	}

	// Set the last character of the word to null character
	// And then call the function to print all words with prefix
	word[len] = '\0';
	printf("Words start with \"%s\":\n", prefix);
	printAllWords(current, prefix, len, &count);
}
```

The function above is used to print all words in a trie data structure that start with a given prefix. The function first traverses the trie based on the characters in the prefix. If it encounters a character in the prefix that doesn't exist in the trie, it informs the user that the prefix doesn't exist in the dictionary and returns. If all characters in the prefix exist in the trie, it copies the prefix into a new word array and sets the current node to the node corresponding to the last character in the prefix. It then calls the function with the current node and the prefix to print all words in the trie that start with the prefix. This function is used to find and print all words in the trie that start with a specific prefix.

```
// Function to view all slang words with prefix
void viewAllSlangWordsWithPrefix(trieNode *node)
{
	char prefix[10];

	// Ask user to input prefix to search a slang word
```

```
        printf("Enter the prefix to be searched: ");
        scanf("%s", prefix);
        getchar();

        puts("");
        printWordsWithPrefix(node, prefix);
}
```

The function above is used to interact with the user to gather a prefix and then print all slang words in a trie data structure that start with this prefix. The function prompts the user to enter a prefix. Once the prefix is entered, the function calls **printWordsWithPrefix** to print all words in the trie that start with the entered prefix. This function is used to facilitate user interaction and manage the search of slang words in the trie that start with a specific prefix.

```
// Function to check if child is present or not
bool checkIfChildPresent(trieNode *node)
{
        for(int i=0; i<ALPHABET_SIZE; i++)
        {
          if(node->child[i] != NULL)
          {
            return true;
          }
        }

        return false;
}
```

The function above is used to check if a given node in a trie data structure has any child nodes. The function iterates over the array of child nodes of the given node. If it finds a child node that is not null, it returns true, indicating that the given node has at least one child node. If it doesn't find any non-null child nodes after checking all of them, it returns false, indicating that the given node does not have any child nodes. This function is used to determine if a given node in the trie is a leaf node or not.

```
// Function to view all slang words in the trie
void viewAllSlangWords(trieNode *node)
{
        bool childPresent = checkIfChildPresent(node);


        if(!childPresent)
        {
                getchar();
                puts("There is no slang word in the dictionary.");
                return;
        }


        int count = 1;
        char prefix[MAX_LENGTH];

        puts("List of all slang words in the dictionary:");
        printAllWords(node, prefix, 0, &count);
        puts("");
        getchar();
```

```
}
```

The function above is used to print all slang words stored in a trie data structure. The function first checks if the root node of the trie has any child nodes using the **checkIfChildPresent** function. If the root node has no child nodes, it informs the user that there are no slang words in the dictionary and returns. If the root node has at least one child node, it calls the **printAllWords** function with the root node and an empty prefix to print all words in the trie. This function is used to facilitate user interaction and manage the display of all slang words in the trie.

```
// Function to create the line "press enter to continue..."
void pressEnterToContinue()
{
	puts("Press enter to continue...");
	getchar();
	clearScreen();
}
```

The function above is s used to pause the execution of a program and wait for the user to press the enter key before continuing. The function first prints the message **"Press enter to continue..."** to the console. It then calls the **getchar()** function, which waits for the user to enter a character. Once the user presses the enter key, the **getchar()** function returns and the function calls **clearScreen()** to clear the console. This function is typically used to give the user time to read output before it is cleared or more output is printed.

```
// Function to display the menu of the program
// And call the function based on the user input
void mainMenu(trieNode *node)
{
	clearScreen();
	int choice;

	// Display the menu to show to user
	puts("1. Realese 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");
	// Ask the user for input the choice
	printf("\nEnter your choice: ");
	scanf("%d", &choice);
	clearScreen();

	// Call the function based on the user input
	switch(choice)
	{
		// If the choice is 1, then call the function to realese new slang word
		case 1:
		realeseNewSlangWord(node);
		pressEnterToContinue();
		break;

		// If the choice is 2, then call the function to search a slang word
		case 2:
```

```
            searchSlangWord(node);
            pressEnterToContinue();
            break;

            // If the choice is 3, then call the function to view all slang words with prefix
            case 3:
            viewAllSlangWordsWithPrefix(node);
            pressEnterToContinue();
            break;

            // If the choice is 4, then call the function to view all slang words
            case 4:
            viewAllSlangWords(node);
            pressEnterToContinue();
            break;
            // If the choice is 5, then exit the program
            case 5:
            puts("Thank you... Have a nice day :)");
            exit(0);
            break;

            // If the choice is invalid, then print the message
            default:
            puts("Invalid choice\n");
            break;
            }
}
```

The function above is serves as the primary user interface for the program. It first clears the screen and then displays a menu of options to the user. The user can choose to release a new slang word, search for a slang word, view all slang words starting with a certain prefix, view all slang words, or exit the program. The user's choice is captured and the corresponding function is called. If the user chooses to release, search, or view slang words, the program will pause and wait for the user to press enter before continuing, providing the user with ample time to read the output. If the user chooses to exit, a farewell message is displayed and the program terminates. If the user's choice doesn't match any of the provided options, an error message is displayed. This function is responsible for controlling the main flow of the program and facilitating user interaction.

```
int main()
{
            // Create a root node
            trieNode *root = createNode(" ", " ");

            // Display the menu to show to user
            while(true)
            {
              mainMenu(root);
            }

  return 0;
}
```

The function above is the entry point of the program. It first creates a root node for a trie data structure by calling the **createNode** function with two empty strings as arguments.

This root node will serve as the starting point for all operations on the trie. Then, it enters an infinite loop where it continuously calls the **mainMenu** function with the root node as an argument. The **mainMenu** function displays a menu to the user and performs various operations based on the user's choice. The loop continues indefinitely, allowing the user to perform multiple operations without restarting the program. The function ends by returning 0, which is a standard way to indicate that the program has ended successfully. However, this line is never actually reached due to the infinite loop.

Documentation of my code when running:

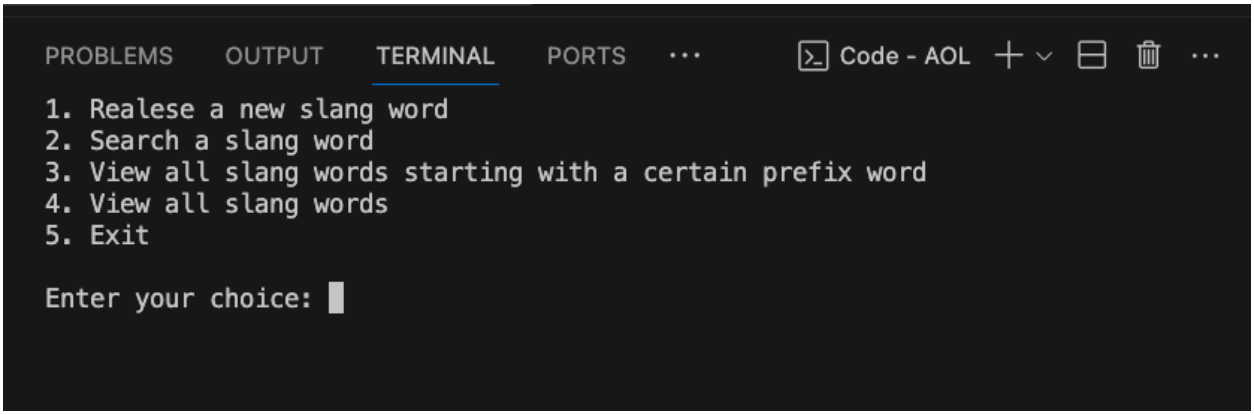
I want to test my code with requirement below:

```
/*

1. Slang word and descriprion i wanna to input :
    1. otw = on the way -> then updated to = same meaning as “on the way”
    2. otiwi = say that we wanna go with jokes
    3. dada = say good bye ekspression
    4. daaa = say good bye to the specially person
    5. daddy = call the father with another name
    6. yolo = you only live once
    7. yoi = say yes with cool
    8. yowes = say yes with hopeless ekspression
    9. yauda = forced to say yes
    10. yaya = say yes if feel understand
    11. haha = ekspreesion that we laugh out loud
    12. hah = ask something that make you feel confuse
    13. halo = say greeting warm to people
    14. haduh = ekspression that forget something
    15. halah = disagree for any agreement



2. Slang word i wanna to searh:
    1. hola -> there is no slang word same with this word

    1. otw
    2. hah
    3. yowes
    4. yaya
    5. daaa

3. Prefix word i wanna to search:
    1. ye -> there is no slang word same with this word

    1. yo
    2. ot
    3. da
    4. ya
    5. ha
*/
```

Note:

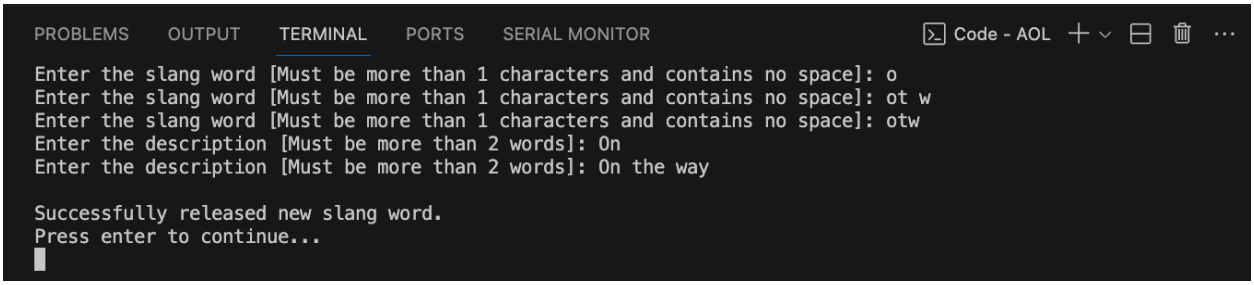
Actually, i will input all queris who i write above to dictionary, but i only include some of the documentation for brevity.

Main menu of the system:

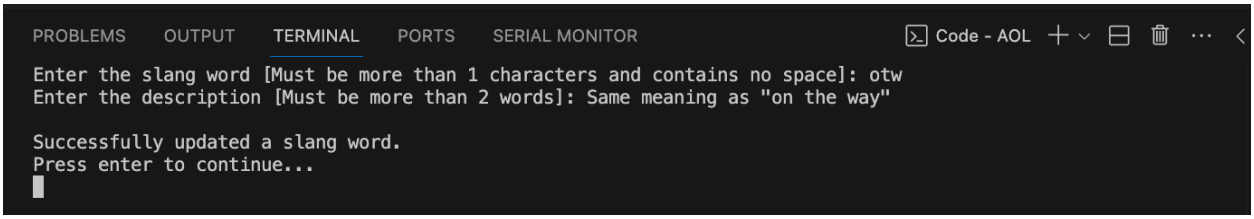


1. If i choose menu 1 (Realese a new slang word)

- Realese a new slang word



- Update a existing slang word

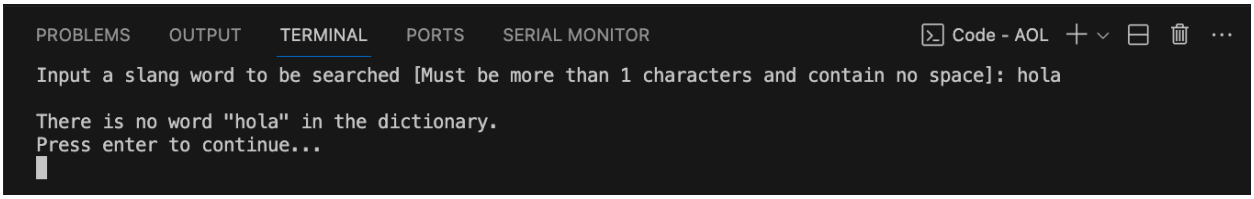


Note :

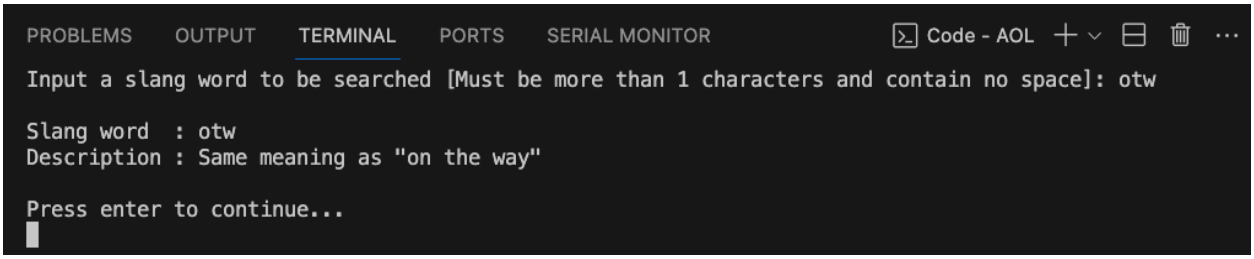
Actually, i have to input all the queries to the dictionary, but im not including the documentation.

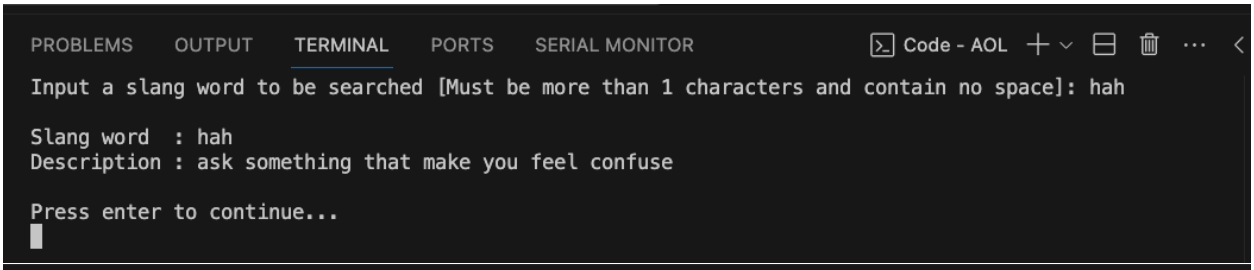
2. If i choose menu 2 (Search a slang word)

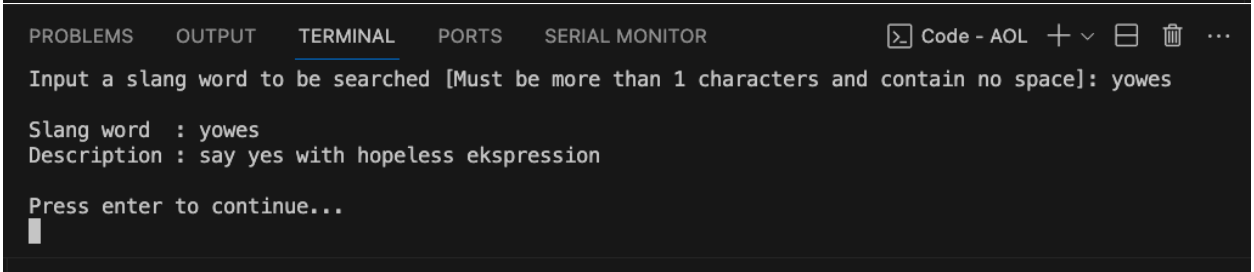
- There is no such word

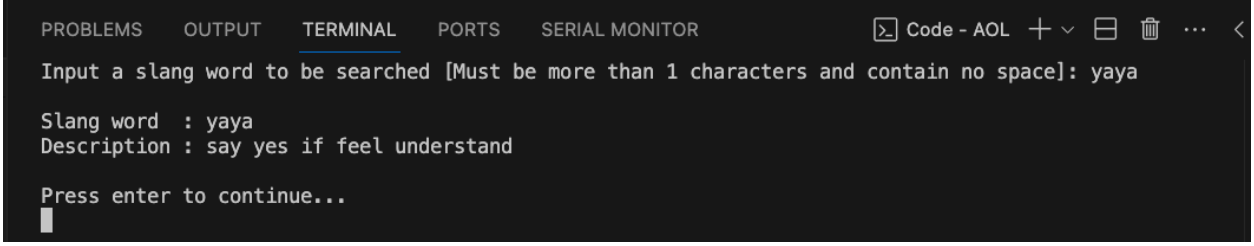


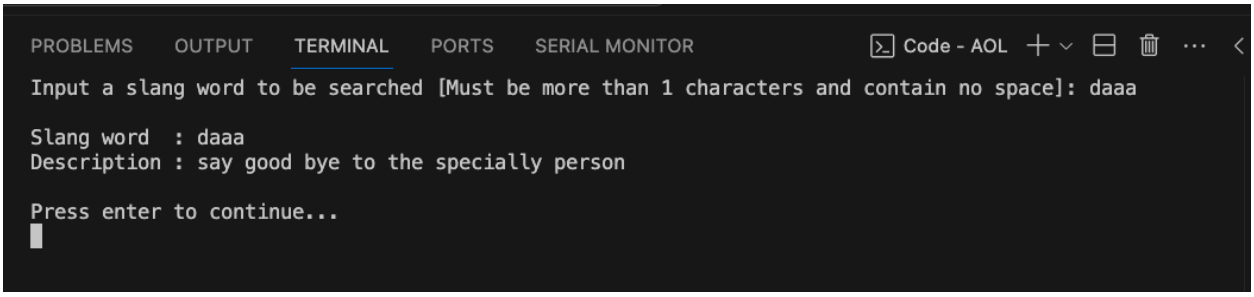
- There is such word (5 documentation)





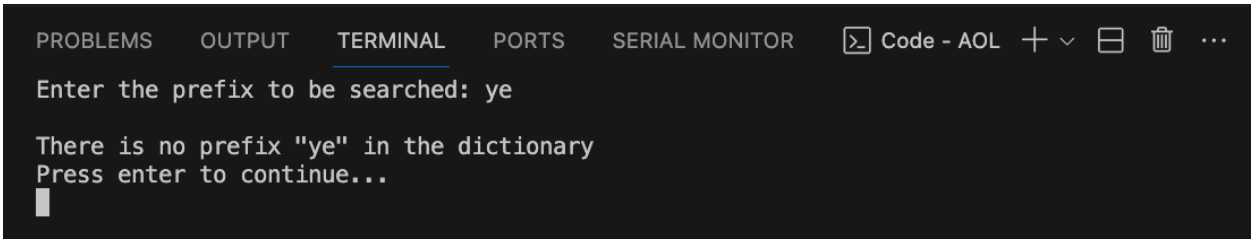




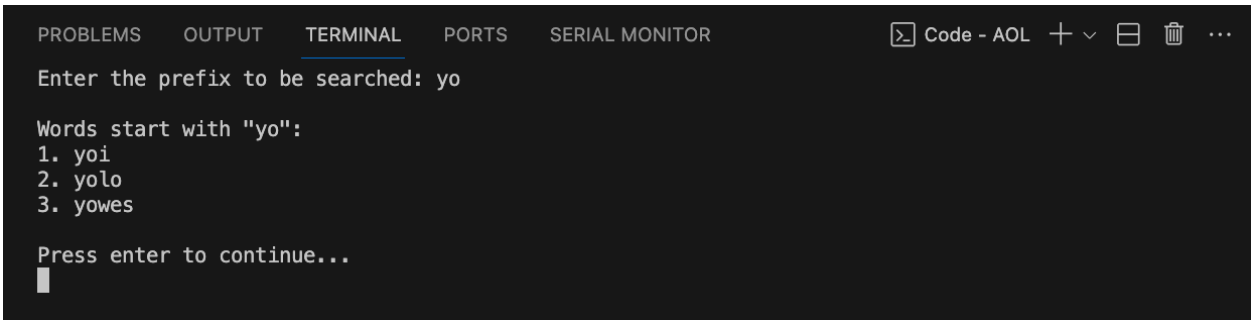


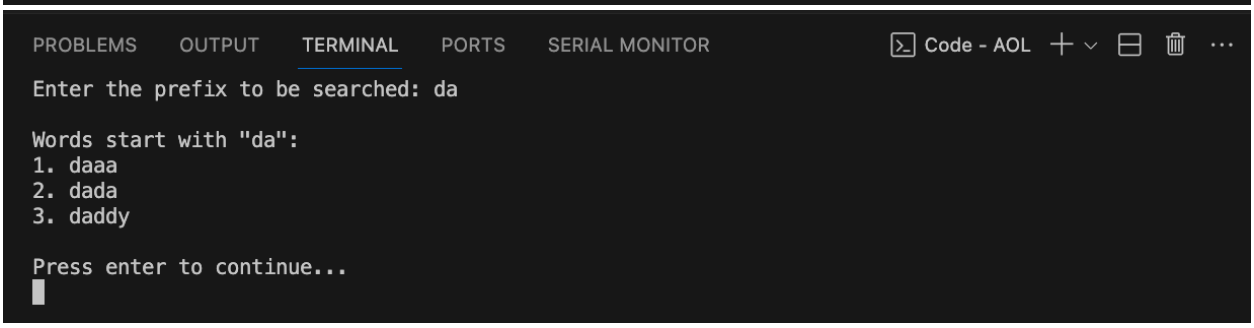
3. If choose 3 (View all word starting with a certain prefix word)

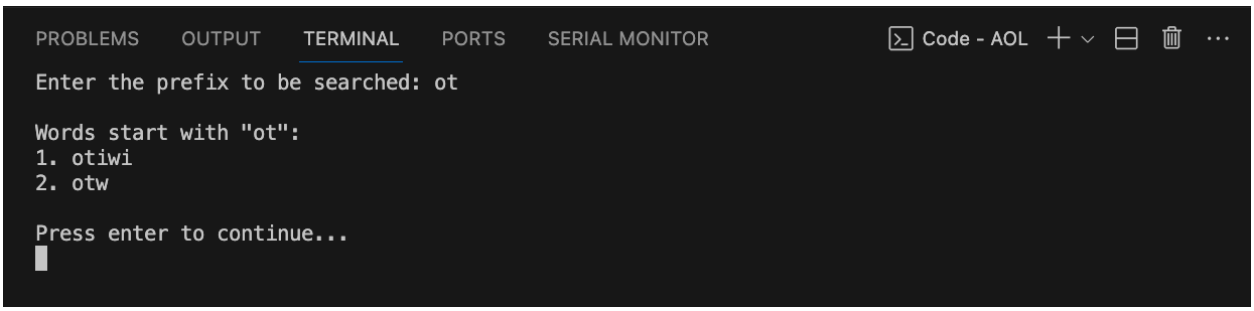
- There is no such word

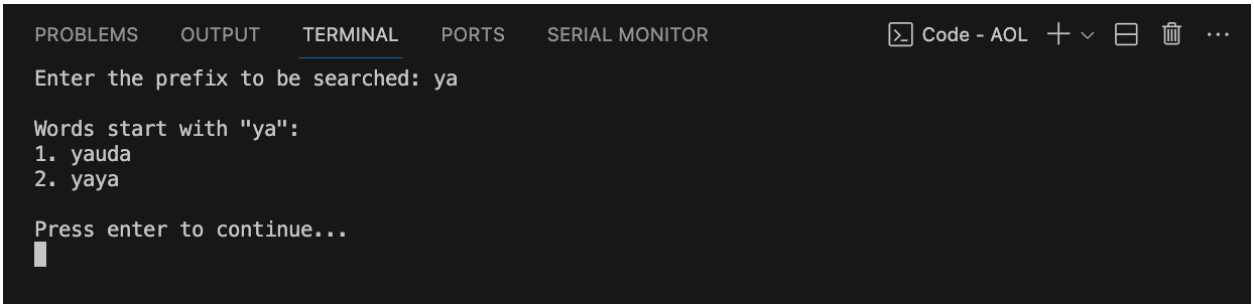


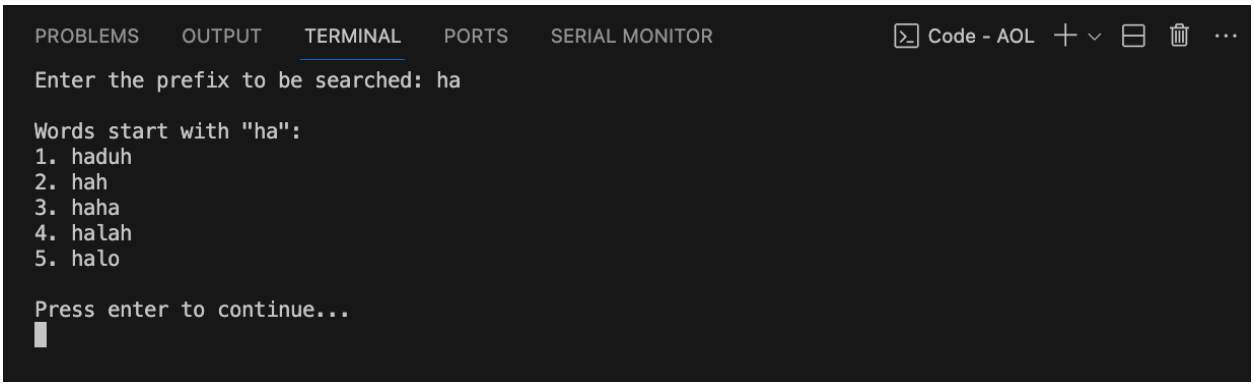
- There is such word (5 prefix word)





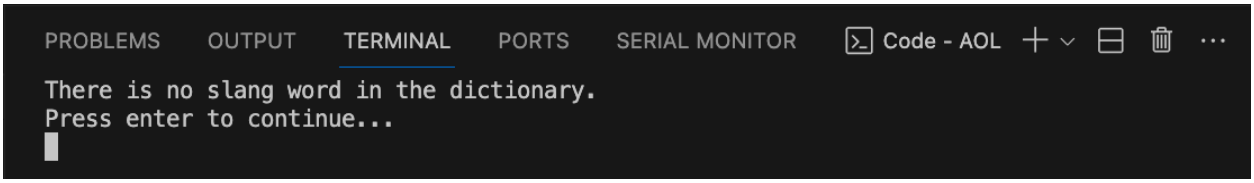




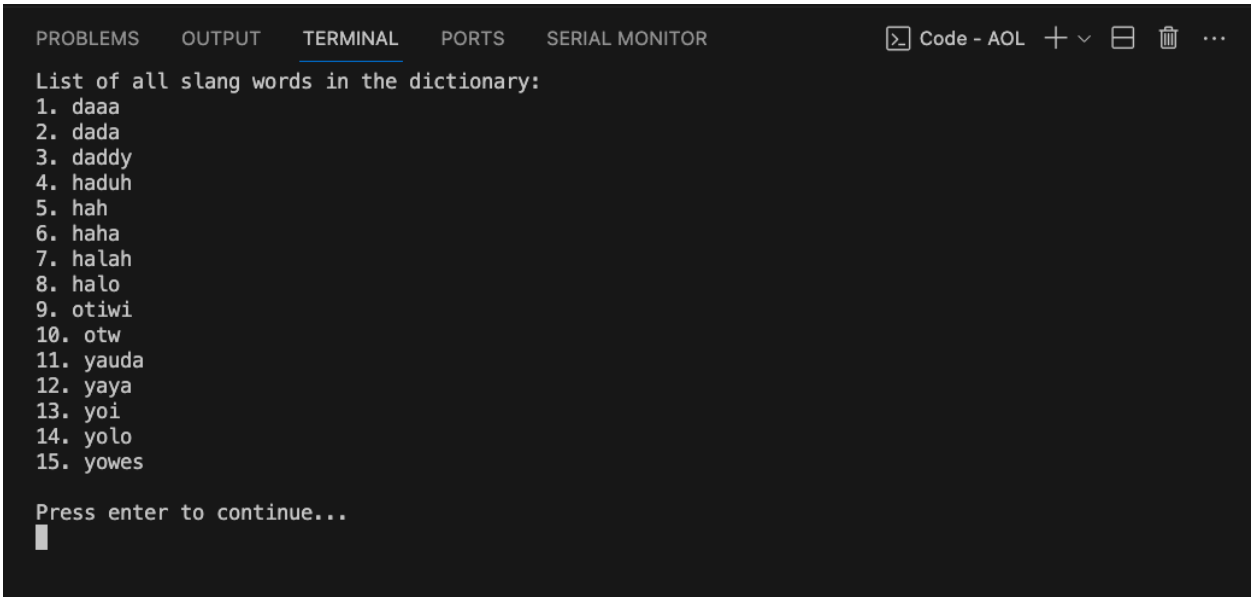


4. If i choose 4 (View all slang word)

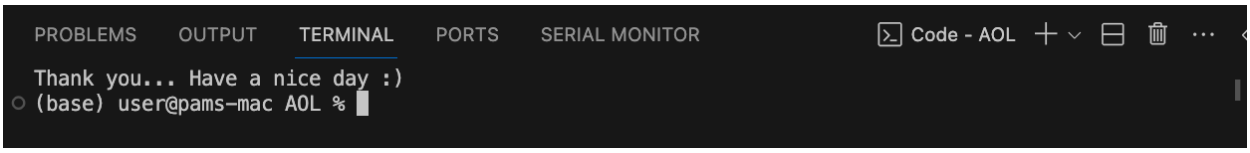
- There is no word yet in the dictionary



- There are a word in dictionary



5. If choose 5 (Exit)


PROBLEMS
OUTPUT
TERMINAL
PORTS
SERIAL MONITOR
Code - AOL
Thank you... Have a nice day :)
(base) user@pams-mac AOL %