Data Structures AOL

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <windows.h>
#include <conio.h>
```

- `<stdio.h>`: Provides standard input/output functions like `printf` and `scanf`
- `<stdlib.h>`: Provide `malloc` for memory allocation, potentially used in `createNode`.
- `<string.h>`: string manipulation functions like `strcpy` and `strcmp`
- `<windows.h>` used for `system("cls")` to clear the console screen
- `<conio.h>` used for `getch()` for getting a character without waiting for Enter, used for pausing the console output

```
struct Node{
    char slang[100];
    char desc[100];
    int end;
    Node *child[26];
};
```

struct Node defines the structure of a node in the trie

- `char slang[100]`: Stores the complete slang word
- `char desc[100]`: Holds the description of the slang word
- `int end`: A flag (0 or 1) indicating whether the current node represents the end of a complete slang word
- `Node *child[26]`: array of pointers to child nodes, where its' size is 26 represents a-z

```
Node* createNode(char slang[], char desc[]){
    Node *newNode = (Node*)malloc(sizeof(Node));
    strcpy(newNode->slang, slang);
    strcpy(newNode->desc, desc);
    for(int i = 0; i<26; i++){
        newNode->child[i]=0;
    }
    newNode->end = 0;
    return newNode;
}
```

- Allocates memory for a new `Node` using `malloc`.
- Copies the provided `slang` and `desc` using strcpy.
- Initializes all child pointers (`child`) to 0, indicating no children initially.
- Sets `end` to 0, signifying a not a complete word.
- Return the newNode.

```
Node *root = createNode("", "");
```

- `Node *root = createNode("", "");`: Creates a root node with empty `slang` and `desc` strings and no children, to start the trie

```
void insertNode(char slang[], char desc[]){
    Node *curNode = root;

    for(int i = 0; i<strlen(slang); i++){
        int index=slang[i]-'a';
        if(!curNode->child[index]){
            curNode->child[index]=createNode("", "");
```

```
        }
        curNode = curNode->child[index];
    }
    curNode->end = 1;

    if(strcmp(curNode->slang, "")==0) puts("Successfully Release
    else puts("Successfully Updated new slang word");

    strcpy(curNode->slang, slang);
    strcpy(curNode->desc, desc);
    puts("Press enter to continue...");
    getch();
}
```

- Takes `slang` and `desc` strings as function parameter/input.
- Starts at the `root` node which represented by `curNode`
- Iterates through each character in the `slang` string:
  - Calculates the index `index` for the current character using `slang[i] - 'a'`.
  - If a child node doesn't exist at the calculated index, a new node is created using `createNode` and linked as the child.
  - `curNode` is then updated to point to the child node for the next character.
- After iterating through the entire `slang` string, sets `end` to 1 in the current node
- Prints "Successfully Released new slang word" if a new word was inserted
- And print "Successfully Updated new slang word" existing word was updated.
- Pauses the console using `getch()`

```
void printAll(Node *start){
    if(start){
        if(start->end==1){
            num++;
```

```
            printf("%d. %s\n", num, start->slang);
        }
        for(int i = 0; i<26; i++){
            printAll(start->child[i]);
        }
    }
}
```

- Prints all slang words stored in the trie.
- Takes a pointer to a starting node `star`.
- If `start` is not `NULL`:
  - Checks if `end` is 1 (terminal node with a word).
    - If yes, increments a counter `num` and prints the word number and the word itself.
  - Recursively calls `printAll` on each child node `child[i]` to traverse the trie and print all words.

```
Node* searchNode(char slang[]){
    Node *curNode = root;
    for(int i = 0; i<strlen(slang); i++){
        int index=slang[i]-'a';
        if(!curNode->child[index]){
            return NULL;
        }
        curNode = curNode->child[index];
    }
    if(strcmp(curNode->slang, "") == 0){
        return NULL;
    }
    return curNode;
}
```

- Takes a `slang` string as input to search for.
- Starts at the `root` node `curNode`.
- Iterates through each character in the `slang` string:
  - Calculates the index `index` as in `insertNode`.
  - If a child node doesn't exist at the index, the search fails, and `NULL` is returned.
  - `curNode` is updated to point to the child node for the next character.
- If the entire `slang` string is traversed and the current node's `end` is 1, the search is successful. The function returns the `curNode` pointer containing the complete slang word and its description.
- If the search reaches the end of the string but `end` is not 1, or if no child node is found at any point, the search fails, and `NULL` is returned.

```
void printPrefix(char slang[]){
    Node *curNode = root;
    for(int i = 0; i<strlen(slang); i++){
        int index=slang[i]-'a';
        if(curNode->child[index]==NULL){
            printf("\033[0;31mThere is no prefix \"%s\" in the d
            return;
        }
        curNode = curNode->child[index];
    }
    printAll(curNode);
}
```

- Finds all slang words starting with a given prefix.
- Takes a `slang` string (prefix) as input.
- Starts at the `root` node (`curNode`).
- Iterates through each character in `slang`:

    - Calculates the child index.
    - If a child node doesn't exist, prints an error message and exits the function.
    - Updates `curNode` to point to the child for the next character.
- Once the prefix is traversed, calls `printAll` on the final node (`curNode`) to print all words starting with that prefix (words in the subtree rooted at the final node).
- Waits for user input (`getch`).

```
int number;
char slang[100];
char desc[100];
char search[100];
```

- `int number`: Stores the user's menu choice.
- char `slang`, `desc`, and `search` for storing user input.

```
system("cls");
        puts("1. Release a new Slang word");
        puts("2. Search a slang word");
        puts("3. View all slang word starting with a certain pre
        puts("4. View all slang words");
        puts("5. Exit");
        do{
            printf(">>> ");
            scanf("%d", &number);
            if(number==5){
                puts("Thank you... Have a nice day :)");
                exit(0);
            }
```

```
        }while(number<0||number>5);
```

- `system("cls")`: Clears the console screen
- **Menu display**: Prints the menu options:
  1. Release a new Slang word
  2. Search a slang word
  3. View all slang word starting with a certain prefix word
  4. View all slang words
  5. Exit
- **User input**: Prompts the user to enter a choice `number` and reads their input using `scanf`.
- **Input validation**:
  - Uses a `do-while` loop to ensure the user enters a valid choice (1 to 5).
  - If the user enters an invalid number, they are prompted to re-enter a choice.

```
case 1:{
                while(1){
                    printf("Input a new slang word [Must be more
                    scanf(" %[^\n]", slang);
                    int count = 0;
                    for(int i = 0; i<strlen(slang); i++){
                        if(slang[i]==' '){
                            count += 0;
                        }else{
                            count += 1;
                        }
                    }
                    if(count==strlen(slang) && strlen(slang)>=2
```

```
                }
                while(1){
                    printf("Input a new slang word description
                    scanf(" %[^\n]", desc);
                    int count = 0;
                    for(int i = 0; i<strlen(desc); i++){
                        if(desc[i]==' '){
                            count += 1;
                        }
                    }
                    if(count>=2) break;
                }
                insertNode(slang, desc);
                break;
                }
```

- **Input validation loop**:
  - Prompts the user for a new slang word, ensuring it has more than one character and no spaces.
    - Uses a `while` loop to keep prompting until valid input is provided.
    - Checks the length of the `slang` string and verifies it doesn't contain spaces.
- **Input validation loop**:
  - Prompts the user for a description, ensuring it has at least two words.
    - Uses a `while` loop to keep prompting until valid input is provided.
    - Counts the number of spaces in the `desc` string to ensure it has at least two words (one or more spaces implies two or more words).
- `insertNode` **call**: Calls the `insertNode` function to add the new slang word and its description to the trie.
- **Success messages**: Prints messages indicating successful insertion or update (if the word already existed).

- `getch` **call**: Pauses the console output, waiting for the user to press a key (might need a portable alternative).

```
case 2:{
                while(1){
                    printf("Input a slang word to be searched [N
                    scanf(" %[^\n]", search);
                    int count = 0;
                    for(int i = 0; i<strlen(search); i++){
                        if(search[i]==' '){
                            count += 0;
                        }else{
                            count += 1;
                        }
                    }
                    if(count==strlen(search) && strlen(search)>=
                }
                Node *temp = searchNode(search);
                if(temp){
                    puts("\033[0;32mFOUND!!!\033[0m");
                    printf("Slang word\t: %s\n", temp->slang);
                    printf("Description\t: %s\n", temp->desc);
                }else{
                    printf("\033[0;31mThere is no word \"%s\" in
                }
                puts("Press enter to continue...");
                getch();
                break;
            }
```

- **Input validation loop**:
  - Prompts the user for a slang word to search for, ensuring it has more than one character and no spaces.

    - Uses a `while` loop to keep prompting until valid input is provided.
- `searchNode` **call**: Calls the `searchNode` function to find the entered slang word in the trie.
- **Search result**:
    - If the word is found:
        - Prints success messages with the found word and its description.
    - If the word is not found:
        - Prints an error message indicating the word doesn't exist in the dictionary.
- `getch` **call**: Pauses the console output, waiting for the user to press a key.

```
case 3:{
                num = 0;
                while(1){
                    printf("Input a prefix to be searched: ");
                    scanf(" %[^\n]", search);
                    int count = 0;
                    for(int i = 0; i<strlen(search); i++){
                        if(search[i]==' '){
                            count += 0;
                        }else{
                            count += 1;
                        }
                    }
                    if(count==strlen(search) && strlen(search)>=
                }
                printPrefix(search);
                puts("Press enter to continue...");
                getch();
                break;
            }
```

- **Input validation loop**:
  - Prompts the user for a prefix to search for, ensuring it has more than one character and no spaces.
    - Uses a `while` loop to keep prompting until valid input is provided.
- `printPrefix` **call**: Calls the `printPrefix` function to find and display all slang words that start with the entered prefix.
- `getch` **call**: Pauses the console output, waiting for the user to press a key.

```
case 4:{
                num = 0;
                printAll(root);
                puts("Press enter to continue...");
                getch();
                break;
            }
```

Case 4: View all slang words (4):

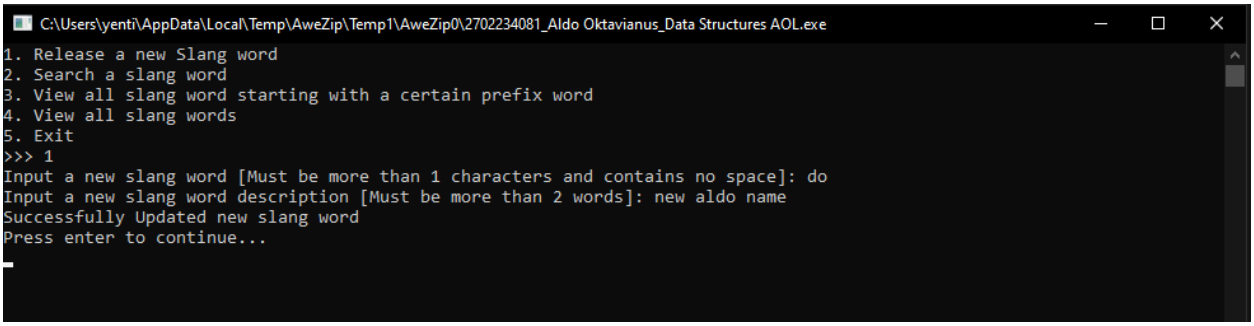
1. `num` **initialization**: Sets the `num` counter to 0 (used to track the number of words found).
2. `printAll` **call**: Calls the `printAll` function, starting from the `root` node, to traverse the trie and print all stored slang words.
3. `getch` **call**: Pauses the console output, waiting for the user to press a key.

Some test runs

Case 1

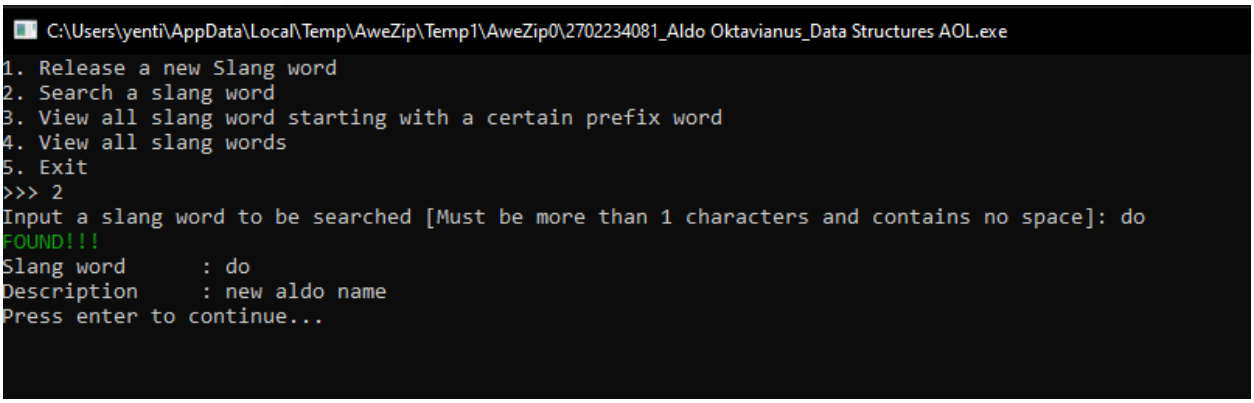
- New word

- same word

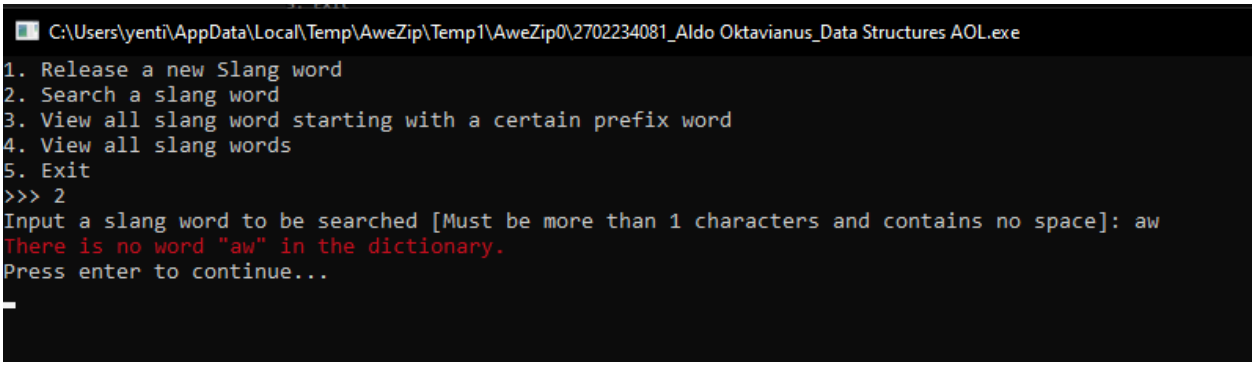


Case 2

- found



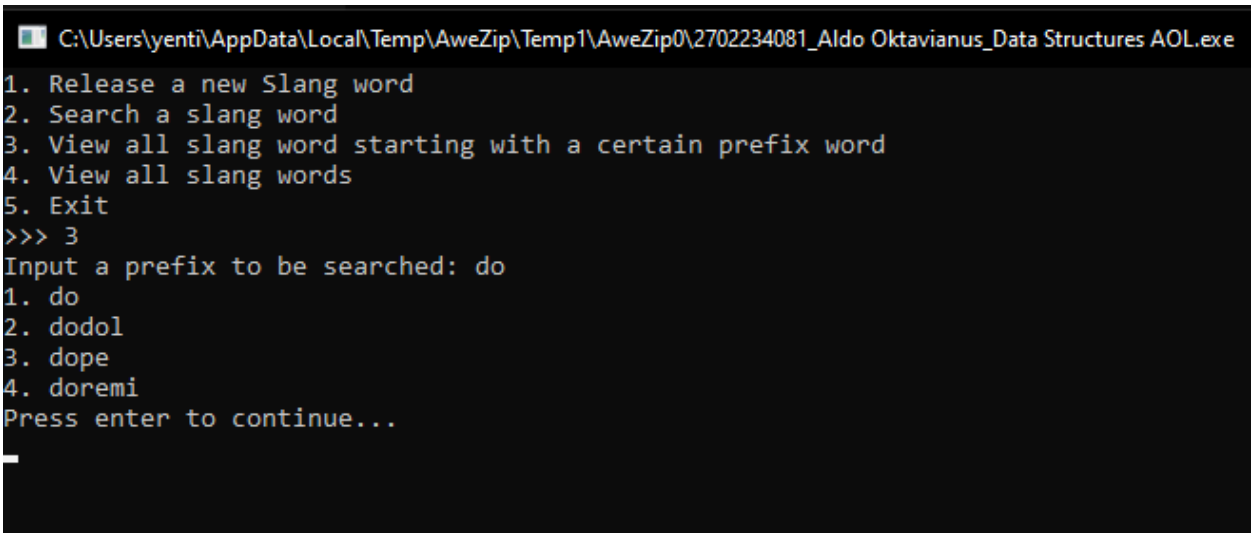
- not found



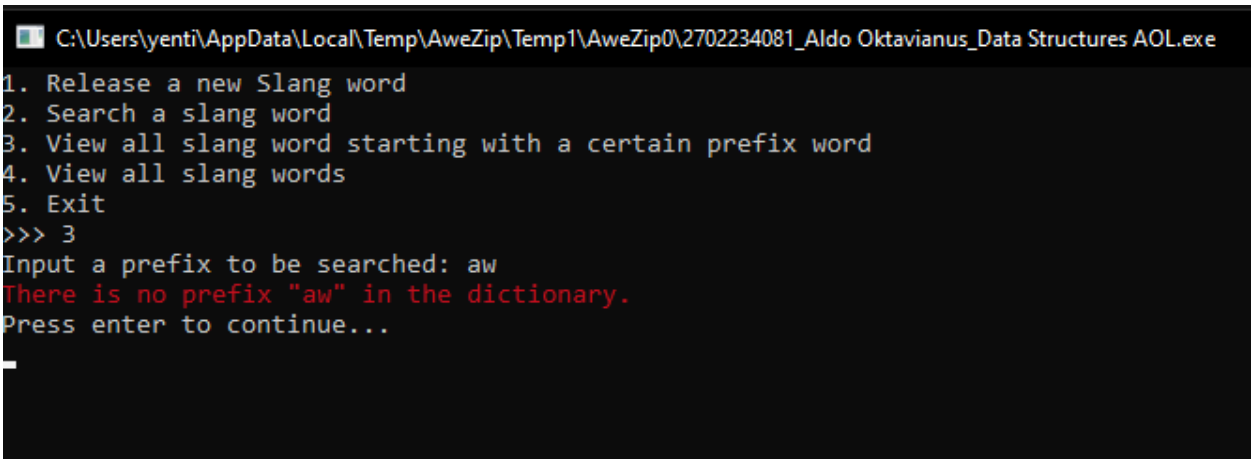
Case 3

After Inputing some more slang words

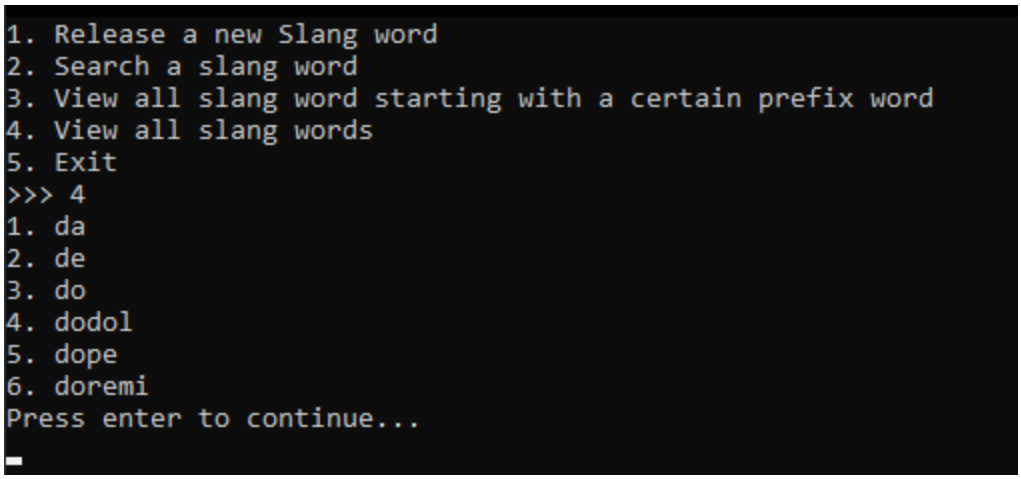
- found



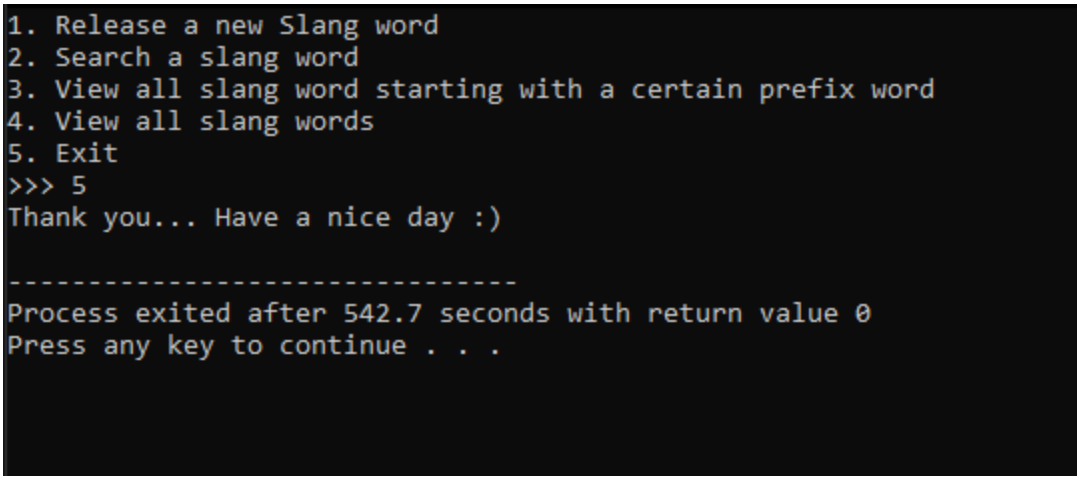
- not found



Case 4


1. Release a new Slang word
2. Search a slang word
3. View all slang word starting with a certain prefix word
4. View all slang words
5. Exit
>>> 4
1. da
2. de
3. do
4. dodol
5. dope
6. doremi
Press enter to continue...

Case 5


1. Release a new Slang word
2. Search a slang word
3. View all slang word starting with a certain prefix word
4. View all slang words
5. Exit
>>> 5
Thank you... Have a nice day :)

Process exited after 542.7 seconds with return value 0
Press any key to continue . . .