#include <stdio.h> // included for standard input output function

#include <stdlib.h> // included for malloc function, a function necessary when making tries/linked list

#include <string.h> // included for strcpy (to copy the content of a string to another string) and strlen (to count the length of the string) function

struct tnode{ // declare struct

char ch; // to contain the letter of a node in a trie

bool word; // to know whether a node is an end of a word or not (1 = end of a word, 0 = not end of a word)

char definition[1001]; // to contain the definition of a word (only used if bool word = 1)

struct tnode \*next[26]; // pointers for the next nodes (an array of 26 because there are 26 letters in the alphabet)

};

void release(struct tnode \*\*root){ // declare release function, parameter is a linked list address (not just linked list, just as a good practice)

char newWord[101]; // to contain the new word

char newDef[1001]; // to contain the new word’s definition

bool valid = 0; // bool used to check if the input is valid or not (valid = 1, not valid = 0), declared 0 so the while loop underneath runs

while(!valid){ // while loop to check the new word input, runs as long as the input is not valid

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

scanf("%[^\n]", &newWord); // to get the input for the new word, %[^\n] is used to get input until enter is pressed

getchar(); // to get rid of buffer

if(strlen(newWord) <= 1) continue; // check if the length of the input is not at least 1 character, if it is, loop is repeated

bool space = 0; // bool used to check if the input have any space or not (have space = 1, doesn’t have space = 0)

for(int i = 0; i < strlen(newWord); i++){ // for loop to check the character of the input one by one

if(newWord[i] == ' '){ // if function that runs if the character checked is a space

space = 1; // because the input has space

break; // break for function to save time and because 1 space violates the rule already

}

}

if(space) continue; // if space = 1, loop is repeated

valid = 1; // valid will become true when the code reaches this line, to stop the loop from repeating

}

valid = 0; // redeclare valid to false because it’s getting used again in the next while loop

while(!valid){ // while loop to check the new definition input, runs as long as the input is not valid

printf("Input a new slang word description [Must be more than 2 words]: "); // to show instructions

scanf("%[^\n]", &newDef); // to get the input for the new word, %[^\n] is used to get input until enter is pressed

getchar(); // to get rid of buffer

int nspace = 0; // used to count the number of spaces in the input

for(int i = 0; i < strlen(newDef); i++){ // for loop to check the character of the input one by one

// if function that runs if the character checked is a space, nspace++ because it is used to count the number of spaces

if(newDef[i] == ' ') nspace++;

}

if(nspace < 2) continue; // check if there are at least 2 spaces (meaning at least two words) in the input, if not, loop is repeated

valid = 1; // valid will become true when the code reaches this line, to stop the loop from repeating

}

struct tnode \*curr = \*root; // declare curr as a temporary pointer for root so that the real root pointer won’t get affected

bool exist = 1; // bool to check whether the word inputted is in the tries already or not (yes = 1, no = 1)

for(int i = 0; i < strlen(newWord); i++){ // for loop to insert a new word into the trie

// if function to check whether the next node is empty or not, - 97 because ‘a’ in ASCII is 97

// index 0 saves letter ‘a’, index 1 saves letter ‘b’, and so on

if(curr->next[newWord[i] - 97] != NULL){

curr = curr->next[newWord[i] - 97]; // because the next node is filled, no need to fill it again, just go to the next node

}

else{

// reserve spaces for the new node using malloc and pointer that points to the spaces

// sizeof is used so that we reserve spaces with the size of the struct

struct tnode \*temp = (struct tnode\*) malloc(sizeof(struct tnode));

for(int i = 0; i < 26; i++) temp->next[i] = NULL; // make all next nodes of the new node NULL because they are empty

temp->ch = newWord[i]; // keep the character of the input in the node’s ch variable

temp->word = 0; // make bool word = 0 because it is the default

// make curr’s next[newWord[i] – 97] the new node, - 97 because ‘a’ in ASCII is 97

// index 0 saves letter ‘a’, index 1 saves letter ‘b’, and so on

curr->next[newWord[i] - 97] = temp;

// if function that runs when the for loop have reached the last character of the word inputted

if(i == strlen(newWord) - 1){ // i == strlen(newWord) – 1 because i starts at 0

temp->word = 1; // make bool word = 0 because the new node is for the last character of the word input

strcpy(temp->definition, newDef); // keep the new definition in the node using strcpy

}

curr = temp; // to move curr to the new node (the next node)

exist = 0; // exist = 0 because if this else function is run, the word inputted is definitely not in the tries

}

}

if(exist){ // if function that runs if the word inputted is in the tries, to set the new definition that was inputted

strcpy(curr->definition, newDef); // keep the new definition in the node using strcpy

// print a message to let the user know that the slang definition has been updated

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

}

else{ // else function that runs if the word inputted is not in the tries

printf("\nSuccessfully released new slang word.\n"); // print a message to let the user know that the new slang has been added

}

printf("\n");

}

void search(struct tnode \*\*root){ // declare search function, parameter is a linked list address (not just linked list, just as a good practice)

char wordSearch[101]; // to contain the word that want to be searched

bool valid = 0; ; // bool used to check if the input is valid or not (valid = 1, not valid = 0), declared 0 so the while loop underneath runs

while(!valid){ // while loop to check the new word input, runs as long as the input is not valid

printf("Input a slang word to be searched [Must be more than 1 characters and contains no space]: "); // to show instructions

scanf("%[^\n]", &wordSearch); // to get the input for the new word, %[^\n] is used to get input until enter is pressed

getchar(); // to get rid of buffer

if(strlen(wordSearch) <= 1) continue; // check if the length of the input is not at least 1 character, if it is, loop is repeated

bool space = 0; // bool used to check if the input have any space or not (have space = 1, doesn’t have space = 0)

for(int i = 0; i < strlen(wordSearch); i++){ // for loop to check the character of the input one by one

if(wordSearch[i] == ' '){ // if function that runs if the character checked is a space

space = 1; // because the input has space

break; // break for function to save time and because 1 space violates the rule already

}

}

if(space) continue; // if space = 1, loop is repeated

valid = 1; // valid will become true when the code reaches this line, to stop the loop from repeating

}

bool exist = 1; // bool to check whether the word inputted is in the tries already or not (yes = 1, no = 1)

struct tnode \*curr = \*root; // declare curr as a temporary pointer for root so that the real root pointer won’t get affected

for(int i = 0; i < strlen(wordSearch); i++){ // for loop to search the word you inputted in the trie

// go to the next node first on the first iteration because root is always empty

// go to the next node with index based on the letter of the word inputted

// - 97 because ‘a’ in ASCII is 97, index 0 means letter ‘a’, index 1 means letter ‘b’, and so on

curr = curr->next[wordSearch[i] - 97];

if(curr == NULL){ // if function to check whether the current node is empty or not

exist = 0; // if it’s empty, that means the word inputted is not in the trie, so exist = 0

break; // break for loop, no need to go further because the word is definitely not in the trie

}

// if function that runs when the for loop have reached the last character of the word inputted

if(i == strlen(wordSearch) - 1){ // i == strlen(newWord) – 1 because i starts at 0

if(curr->word == 0) exist = 0; // check if the word inputted is in the trie or not, by checking bool word, if not then exist = 0

break; // break for loop because no need to go further

}

}

// if function that runs if the word inputted is not in the tries

// print a message to let the user know that the word inputted is not in the trie

if(!exist) printf("There is no word \"%s\" in the dictionary.\n\n", wordSearch);

else{ // else function that runs if the word inputted is in the tries

printf("Slang word: %s\n", wordSearch); // print the slang word

printf("Description: %s\n\n", curr->definition); // print the slang defitinion

}

}

int num = 1; // int used for numbering in the list

// declare display function, parameter is a linked list address (not just linked list, just as a good practice), string words as a variable to contain the characters of the word that is going to be printed, int level to contain the level of the tree

void display(struct tnode \*\*root, char words[], int level){

struct tnode \*curr = \*root; // declare curr as a temporary pointer for root so that the real root pointer won’t get affected

if(curr->word){ // if function that runs if the bool word in curr is = 1, to print a word

words[level] = '\0'; // to indicate end of the string, because printf end when it meets \0

printf("%d. %s\n", num, words); // print number and also the word

num++; // so that the numbering of the list will go up

}

for(int i = 0; i < 26; i++){ // for loop to check every next node one by one

if(curr->next[i] != NULL){ // if function that runs if the next node is not NULL

words[level] = i + 'a'; // add the char into string words

display(&curr->next[i], words, level + 1); // call the display function recursively for each next node

}

}

}

// declare display (with prefix) function, parameter is a linked list address (not just linked list, just as a good practice), string prefixSearch to contain the prefix of the words that are going to be printed, string words as a variable to contain the characters of the word that is going to be printed, int level to contain the level of the tree

void displayPrefix(struct tnode \*\*root, char prefixSearch[], char words[], int level){

struct tnode \*curr = \*root; // declare curr as a temporary pointer for root so that the real root pointer won’t get affected

if(curr->word){ // if function that runs if the bool word in curr is = 1, to print a word

words[level] = '\0'; // to indicate end of the string, because printf end when it meets \0

printf("%d. %s%s\n", num, prefixSearch, words); // print number, the prefix, and the rest of the word

num++; // so that the numbering of the list will go up

}

for(int i = 0; i < 26; i++){ // for loop to check every next node one by one

if(curr->next[i] != NULL){ // if function that runs if the next node is not NULL

words[level] = i + 'a'; // add the char into string words

displayPrefix(&curr->next[i], prefixSearch, words, level + 1); // call the display function recursively for each next node

}

}

}

// declare view (with prefix) function, parameter is a linked list address (not just linked list, just as a good practice)

void viewprefix(struct tnode \*\*root){

char prefixSearch[101]; // to contain the prefix that want to be searched

printf("Input a prefix to be searched: "); // to show instructions

scanf("%[^\n]", &prefixSearch); // to get the input of the prefix that want to be searched

getchar(); // to get rid of buffer

bool exist = 1; // bool to check whether the prefix is in the trie or not (exist = 1, doesn’t exist = 0)

struct tnode \*curr = \*root; // declare curr as a temporary pointer for root so that the real root pointer won’t get affected

for(int i = 0; i < strlen(prefixSearch); i++){ // for loop to search the prefix you inputted in the trie

// go to the next node first on the first iteration because root is always empty

// go to the next node with index based on the letter of the word inputted

// - 97 because ‘a’ in ASCII is 97, index 0 means letter ‘a’, index 1 means letter ‘b’, and so on

curr = curr->next[prefixSearch[i] - 97];

if(curr == NULL){ // if function to check whether the current node is empty or not

exist = 0; // if it’s empty, that means the prefix inputted is not in the trie, so exist = 0

break; // break for loop, no need to go further because the prefix is definitely not in the trie

}

}

// if function that runs if the prefix is not in the trie

// print a message to let the user know that the prefix is not in the trie

if(!exist) printf("There is no prefix \"%s\" in the dictionary.\n\n", prefixSearch);

else{ // else function that runs if the prefix is in the trie

int level = 0; // to contain the level of the trie, = 0 because it starts at the root

char words[101] = {}; // to contain the word that is going to be printed

// call displayPrefix function, passing the root address, the prefix that want to be searched, string words, and int level

displayPrefix(&curr, prefixSearch, words, level);

}

num = 1; // reset num to 1

printf("\n");

}

void viewall(struct tnode \*\*root){ // declare view function, parameter is a linked list address (not just linked list, just as a good practice)

struct tnode \*curr = \*root; // declare curr as a temporary pointer for root so that the real root pointer won’t get affected

bool empty = 1; // bool to check if the trie is empty or not (empty = 1, not empty = 0)

for(int i = 0; i < 26; i++){ // for loop to check if the trie is empty or not, by checking the next node one by one

if(curr->next[i] != NULL){ // if function to check whether the next node is empty or not

empty = 0; // empty = 0 because the trie contains something

break; // break for loop, no need to go further because the trie is definitely not empty

}

}

if(empty){ // if function that runs if the trie is empty

// print a message to let the user know that the trie is empty

printf("There is no slang word yet in the dictionary.\n\n");

return; // stop viewall function because no need to go further

}

int level = 0; // to contain the level of the trie, = 0 because it starts at the root

char words[101] = {}; // to contain the word that is going to be printed

display(&curr, words, level); // call display function, passing the root address, string words, and int level

num = 1; // reset num to 1

printf("\n");

}

int main(){ // declare main function

// reserve spaces for root using malloc and pointer that points to the spaces

// sizeof is used so that we reserve spaces with the size of the struct

struct tnode \*root = (struct tnode\*) malloc(sizeof(struct tnode));

root->word = 0; // set bool word in root = 0 because it’s empty

for(int i = 0; i < 26; i++) root->next[i] = NULL; // make all next nodes of root NULL because they are empty at first

int nchoice = 0; // to contain the int user input for menu choices, declared 0 so the while loop underneath runs

while(nchoice != 5){ // while loop for user to pick the options, will not stop until user input 5 (exit/end program)

// print choices for user to choose

printf("Choose one of the option\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("Input your choice: "); // to show instructions

scanf("%d", &nchoice); // to get the input for the choice

getchar(); // to get rid of buffer

switch(nchoice){ // switch case that will run depending on the number user input

case(1): // if user input 1, call release function

release(&root);

break; // to break from switch case

case(2): // if user input 2, call search function

search(&root);

break; // to break from switch case

case(3): // if user input 3, call viewprefix function

viewprefix(&root);

break; // to break from switch case

case(4): // if user input 4, call viewall function

viewall(&root);

break; // to break from switch case

}

}

printf("Thank you... Have a nice day :)\n"); // show a message to thank the user for using the program

return 0; // to know that the program run successfully

}

Custom case:

chill Same meaning as word "relax"

airhead A silly/foolish person

cringe Really embarrassing action/person

simp Person who does too much for who he/she likes

crash Same meaning as word "sleep"

dope Same meaning as word "cool"

crusty Same meaning as word "unclean"

sus Short for "suspicious"

crap Something has a bad quality

crispy Neat, good-looking, clean

cap Same meaning as "to lie"

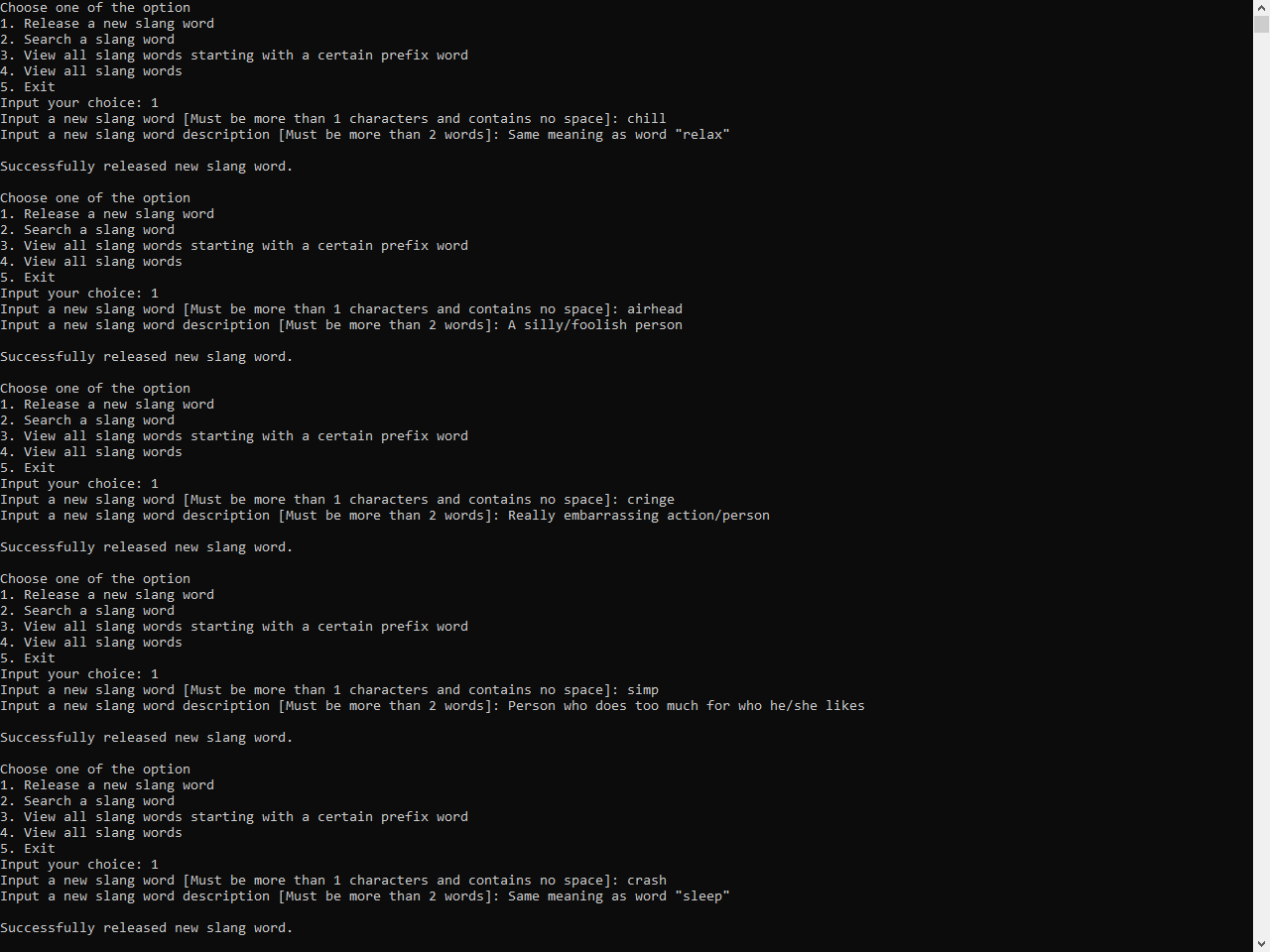
beef Same meaning as word "fight"

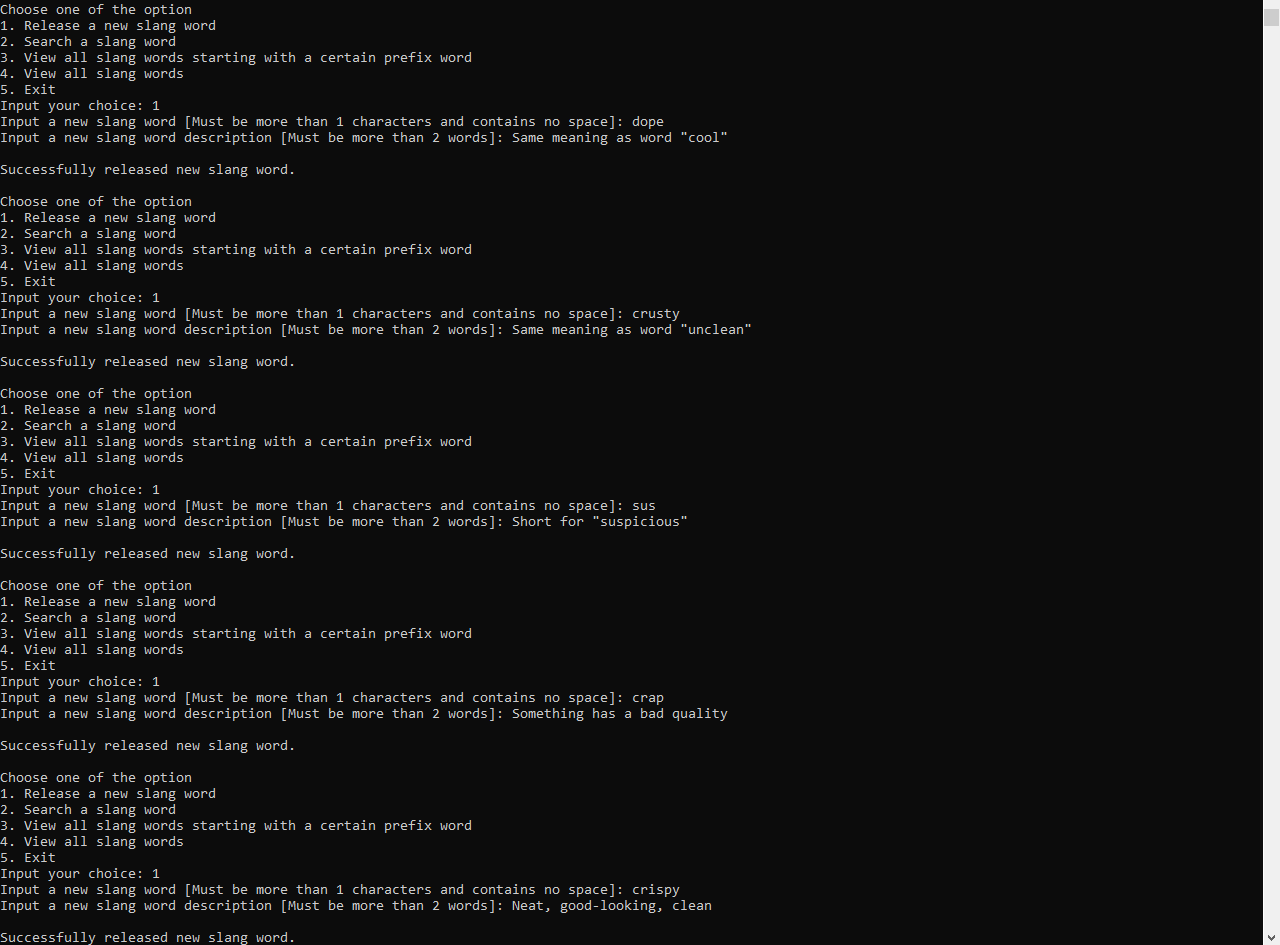
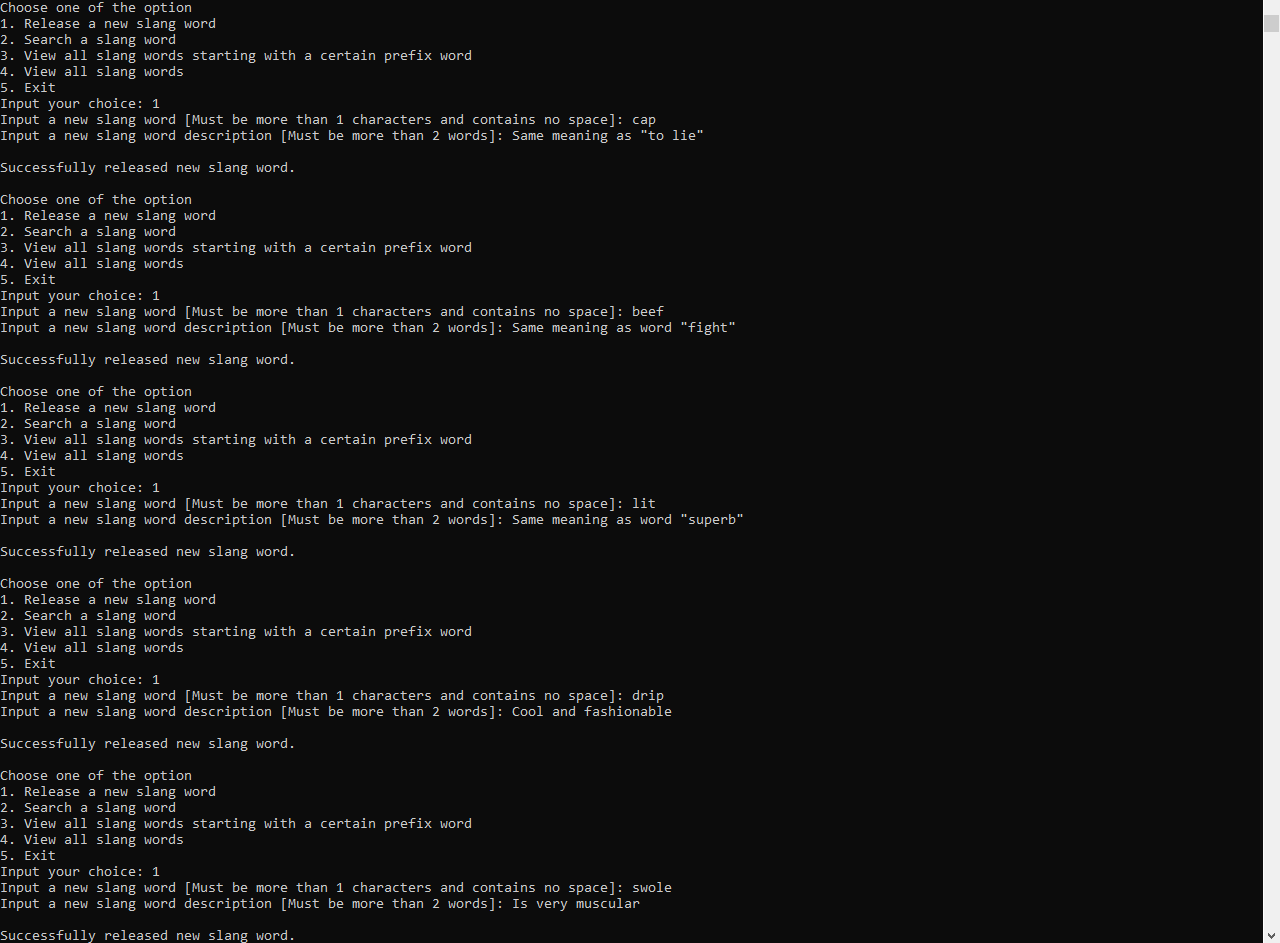
lit Same meaning as word "superb"

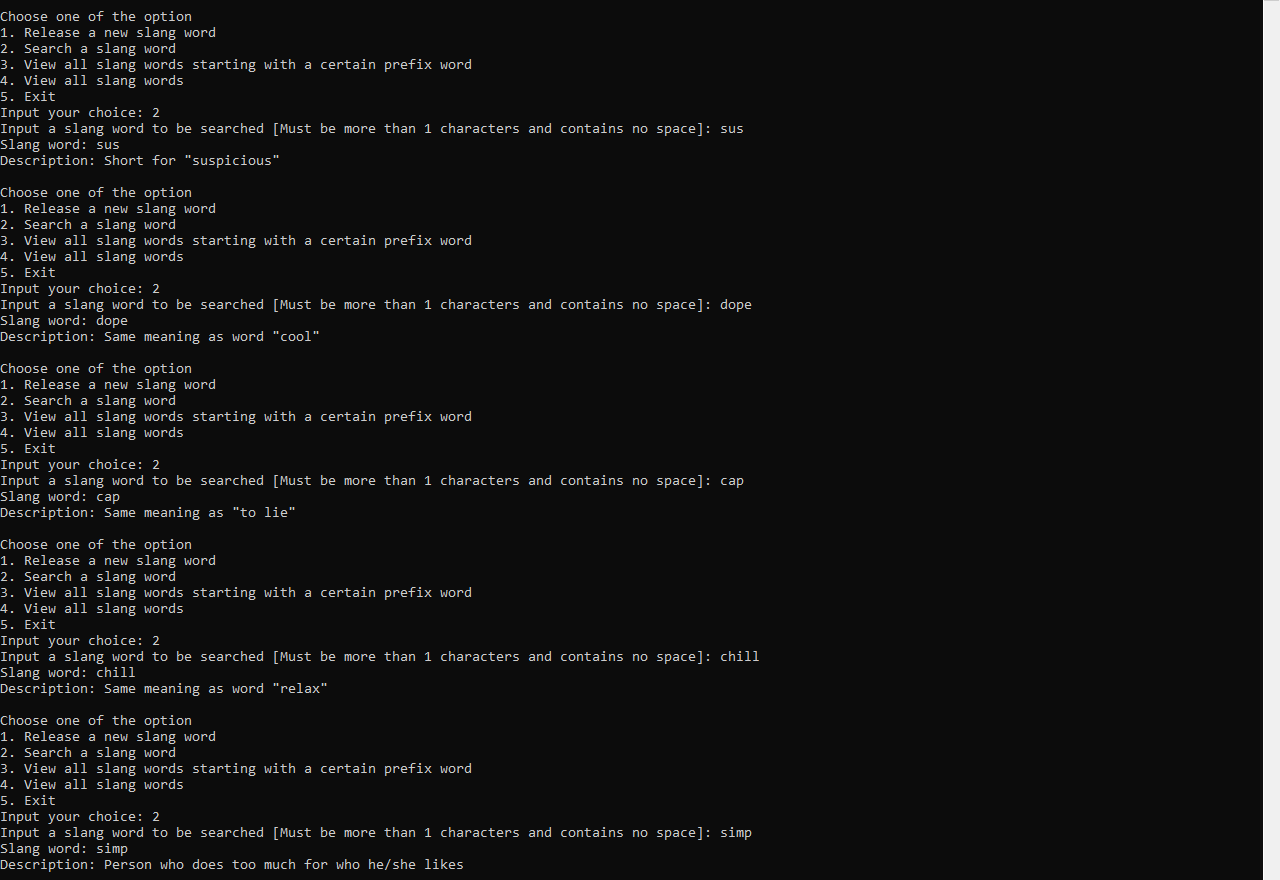
drip Cool and fashionable

swole Is very muscular

Input of 15 slang words:



Search 5 words: 

View prefix 5 words:



View all:

