PES UNIVERSITY
Electronic City Campus,1 KM before Electronic City, Hosur Road, Bangalore-100

PROJECT REPORT
on

“Dictionary and Thesaurus”

Submitted in partial fulfillment of the requirements for the IV Semester
Secure programming with C (UE19CS257C)

Bachelor of Engineering
IN
COMPUTER SCIENCE AND ENGINEERING

For the Academic year
2020-2021

BY

Lalitha Sravanti Dasu - PES2UG19CS201

Under the Guidance of
Vishwachetan D
Assistant Professor

Department of Computer Science and Engineering
PES UNIVERSITY EC CAMPUS
Hosur Road, Bengaluru -560100

PES UNIVERSITY EC CAMPUS

Hosur Road, Bangalore -560100

Department of Computer Science and Engineering

CERTIFICATE

Certified that the project work entitled **"Dictionary and Thesaurus"** is a bonafide work carried out by **Lalitha Sravanti Dasu** bearing USN: **PES2UG19CS201** of **PES University EC CAMPUS in** partial fulfillment for the special topic course **Secure Programming with C** of IV Semester in **Computer Science and Engineering** of the **PES University**, **Bangalore** during the year 2020-2021.

Signatures:

Project Guide: **Vishwachetan D** **Assistant Professor, Dept. of CSE,** **PES UNIVERSITY EC CAMPUS, Bengaluru**	**Dr. Sandesh B J** **Head, Dept of CSE** **PES UNIVERSITY EC CAMPUS,Bengaluru**

Declaration

I hereby declare that the project entitled **“Dictionary and Thesaraus”** submitted for the special topic course **Secure Programming with C** of IV Semester in **Computer Science and Engineering** of the **Pes University**, **Bangalore**, Bangalore is my original work.

Signature of the Student: Lalitha S.D.

Place: Bangalore, Karnataka

Date: 9th May, 2021

TABLE OF CONTENTS

PROBLEM DEFINITION AND NEED

Problem Definition

To develop a dictionary and thesaurus making use of the CERT recommendations.

Problem Need

Developing a dictionary and thesaurus using data structures like trie and linked list helps in performing search operations easily making it easy to look up the meaning or synonym of a word.

REQUIREMENT SPECIFICATION

Software requirements:

- C programming language:
  C is a popular programming language, it is also known for its many security flaws. Most of the vulnerabilities found in C programming language were buffer errors and input validations.

- GCC Compiler:
  The GCC implementation sets itself up as an early optimization pass. It builds a representation of the code from the GIMPLE internal representation. The gcc version has been difficult to support and maintain, due mainly to the fact that the GIMPLE intermediate language was never designed for static analysis.

- Splint Static Analysis Tool:

  Splint, short for **Secure Programming Lint**, is a programming tool for statically checking C programs for security vulnerabilities and coding mistakes. Formerly called LCLint, it is a modern version of the Unix lint tool.

  Splint has the ability to interpret special annotations to the source code, which gives it stronger checking than is possible just by looking at the source alone. Splint is used by gpsd as part of an effort to design for zero defects

SECURITY REQUIREMENT SPECIFICATION

1. Data Types used
   Various abstract data types have been used in the implementation of our project:
   - Integer
   - Character array used for strings
   - Stack used for trie delete operation
   - Structure used to define trie node in dictionary and word and meaning nodes in thesaurus

```
struct trienode {
        struct trienode *child[255];
        int endofword;
        char meaning[500];
};

struct stack{
        struct trienode *m;
        int index;
};

struct word
{
    char w[200];
    struct word* next;
};

struct meaning
{
    char sentence[500];
    struct word* next_word;
    struct meaning* next_meaning;
};
```

2. Functions used

void insert_meaning(struct meaning ,char *,char *);**
This function is used to insert a new word to the thesaurus whose meaning does not already exist.

void display_pairs(struct meaning*);
This function displays all the meanings and synonyms having the meaning.

void insert_synonym(struct meaning ,char *,char *);**
This function is used to add a new word to the thesaurus when the meaning already exists.

int check_meaning(struct meaning* ,char*);
This function returns 1 if the meaning exists in the thesaurus else 0.

struct trienode* getnode();
This function creates and returns a new trie node.

void del_thes(struct meaning*, char* , char*);
This function is used to delete a word from the thesaurus. If the word doesn’t have any synonyms, the meaning node is deleted as well.

void find_syn(struct meaning* ,char* , char*);
This function is used to find synonyms of a word. Displays synonyms if exist, “no synonyms found” if the word doesn’t have any synonyms and “Not Found” if the word is not present in the thesaurus.

void del_word(char *);
Deletes word from file on calling delete.

void add_to_file(char* ,char*);
Adds word-meaning pair to file on calling insert function.

void delete_trie(struct trienode *,char *);
Delete word from trie.

int check(struct trienode *);
Returns number of characters of a word (non-null nodes) in trie. Function used for trie delete operation.

void search_pre(struct trienode * , char *);
This function displays all words with a given prefix.

char* search(struct trienode * , char *);
This function returns the meaning of a word from trie.

struct stack pop();
Stack pop function used for trie delete.

void push(struct trienode *,int);
Stack push function used for trie delete.

void display(struct trienode *);
Display all words in the dictionary.

void file_to_dict(struct trienode* , struct meaning);**
Insert all words and meanings in file to the dictionary and thesaurus.

void insert(struct trienode* , char *, char *);
Insert new word to dictionary.

void destroy(struct meaning);**
Destroy thesaurus

void destroy_trie(struct trienode);**
Destroy Dictionary

CERT safety recommendations used:

EXP34-C. Do not dereference null pointers

Dereferencing a null pointer is undefined behavior. On many platforms, dereferencing a null pointer results in abnormal program termination, but this is not required by the standard.

A common cause of program failures is when a null pointer is dereferenced. It occurs either because of missing a null check after a memory allocation function call, such as malloc or realloc, or it occurs because of invalid/wrong pointer assignments.

These types of vulnerabilities are dealt with by handling the cases where a pointer can be NULL and running the program accordingly.

EXP43-C. Avoid undefined behavior when using restrict-qualified pointers

The bounds-checking interfaces are alternative library functions that promote safer, more secure programming.
C11 Annex K defines the strcpy_s(), strcat_s(), strncpy_s(), and strncat_s() functions as replacements for strcpy(), strcat(), strncpy(), and strncat()
The alternative functions verify that output buffers are large enough for the intended result and return a failure indicator if they are not.

The strcpy() function does not specify the size of the destination array, so buffer overrun is often a risk. Using strcpy() function to copy a large character array into smaller one is dangerous, but if the string will fit, then it will not worth the risk. If destination string is not large enough to store the source string then the behavior of strcpy() is unspecified or undefined.
To tackle this we use strncpy(). The strncpy() function is

similar to strcpy() function, except that at most n bytes of src are copied. If there is no NULL character among the first n character of src, the string placed in dest will not be NULL-terminated. If the length of src is less than n, strncpy() writes additional NULL character to dest to ensure that a total of n character are written.
Stack smashed error encountered when appropriate size input was not given.

ERR33-C. Detect and handle standard library errors

The majority of the standard library functions, including I/O functions and memory allocation functions, return either a valid value or a value of the correct return type that indicates an error (for example, −1 or a null pointer). Assuming that all calls to such functions will succeed and failing to check the return value for an indication of an error is a dangerous practice that may lead to unexpected or undefined behavior when an error occurs. It is essential that programs detect and appropriately handle all errors in accordance with an error-handling policy.

DCL31-C. Declare identifiers before using them

The C11 Standard requires type specifiers and forbids implicit function declarations. The C90 Standard allows implicit typing of variables and functions. Consequently, some existing legacy code uses implicit typing. Some C compilers still support legacy code by allowing implicit typing, but it should not be used for new code. Such an implementation may choose to assume an implicit declaration and continue translation to support existing programs that used this feature.

DCL41-C. Do not declare variables inside a switch statement before the first case label

If a programmer declares variables, initializes them before the first case statement, and then tries to use them inside any of the case statements, those variables will have scope inside the switch block but will not be initialized and will consequently contain indeterminate values.

INT31-C. Ensure that integer conversions do not result in lost or misinterpreted data

CHECK IMPLICIT TYPE CONVERSION IN CERT. THIS MAY BE WRONG.

We avoid implicit type conversion as this might lead to erroneous values stored in the variables. We also provide bool values as the loop conditions instead of int values.

CERT, MSC12-C. - Detect and remove code that has no effect or is never executed (NOT ACCURATE BUT CLOSEST CERT RECOMMENDATION I FOUND FOR THIS WARNING. VERIFY)

If control reaches the closing curly brace (}) of a non-void function without evaluating a return statement, using the return value of the function call is undefined behavior.

We changed the structure of the main while loop such that the program doesn't quit abruptly and only quits as it reaches the final return 0 instruction.

DCL40-C. Do not create incompatible declarations of the same function or object

Two or more incompatible declarations of the same function or object must not appear in the same program because they result in undefined behavior.

EXP33-C. Do not read uninitialized memory

Local, automatic variables assume unexpected values if they are read before they are initialized.

EXP37-C. Call functions with the correct number and type of arguments

Do not call a function with the wrong number or type of arguments.

MEM30-C. Do not access freed memory

Evaluating a pointer—including dereferencing the pointer, using it as an operand of an arithmetic operation, type casting it, and using it as the right-hand side of an assignment—into memory that has been deallocated by a memory management function is undefined behavior. Pointers to memory that has been deallocated are called dangling pointers. Accessing a dangling pointer can result in exploitable vulnerabilities.

MEM35-C. Allocate sufficient memory for an object

The types of integer expressions used as size arguments to malloc(), calloc(), realloc(), or aligned_alloc() must have sufficient range to represent the size of the objects to be stored. If size arguments are incorrect or can be manipulated by an attacker, then a buffer overflow may occur. Incorrect size arguments, inadequate range checking, integer overflow, or truncation can result in the allocation of an inadequately sized buffer.

FIO42-C. Close files when they are no longer needed

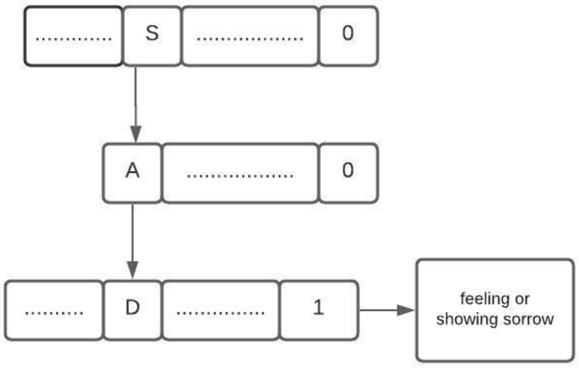
A call to the fopen() or freopen() function must be matched with a call to fclose() before the lifetime of the last pointer that stores the return value of the call has ended or before normal program termination, whichever occurs first.

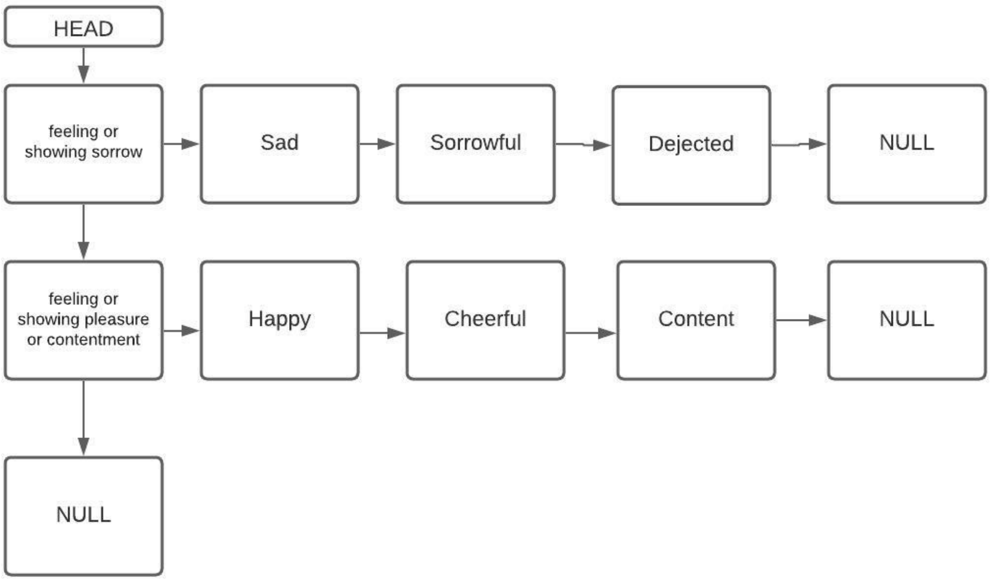
DESIGN

NODE STRUCTURES

1. DICTIONARY
   Trie Node structure:

```
struct trienode {
struct trienode *child[255];
int endofword;
char meaning[500];
};
```





HEAD
feeling or showing sorrow
Sad
Sorrowful
Dejected
NULL
feeling or showing pleasure or contentment
Happy
Cheerful
Content
NULL
NULL

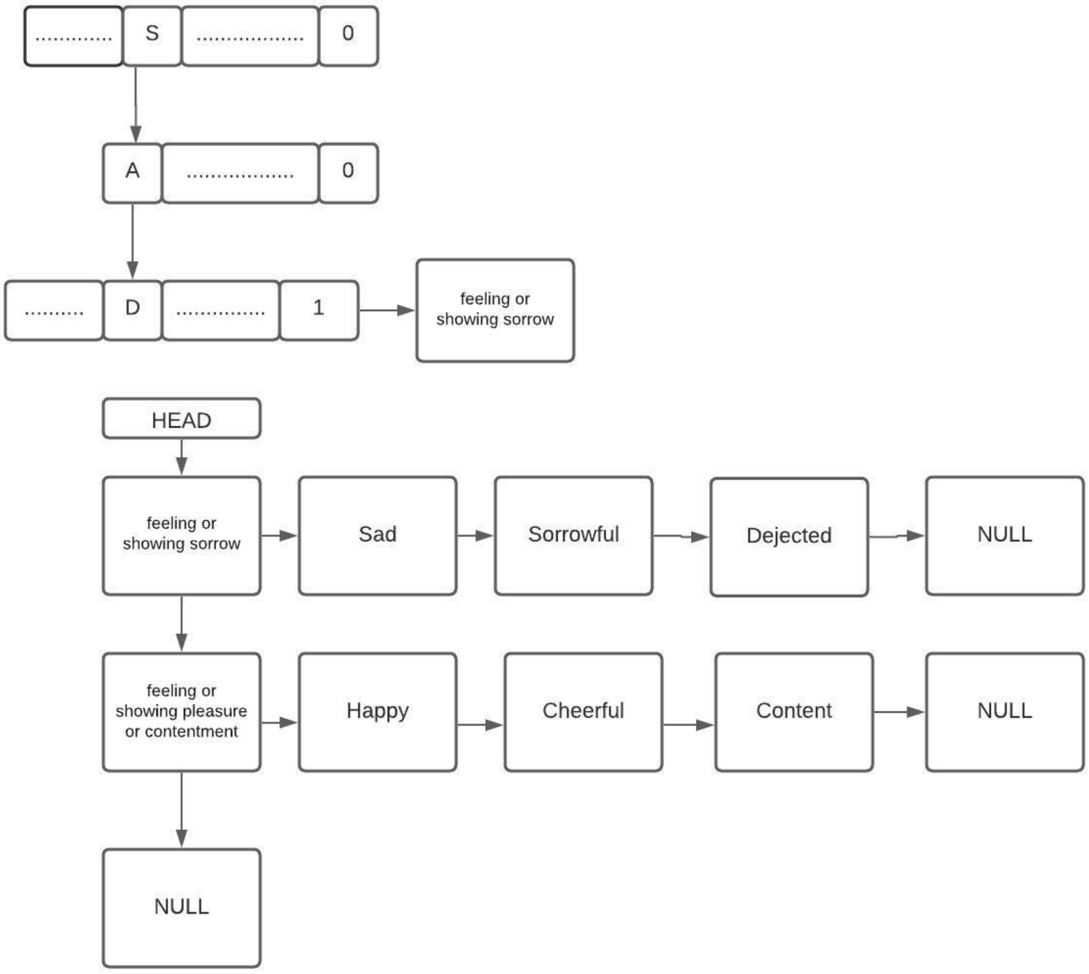
2. THESAURUS

Word Node Structure:

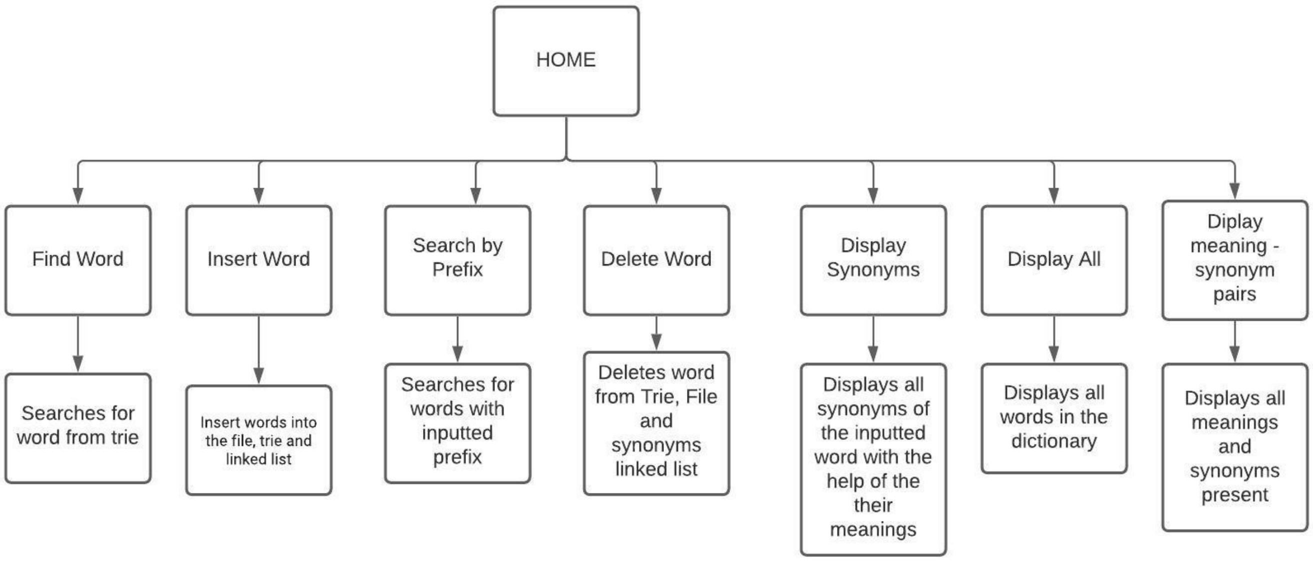
```
struct word
{
char w[200];
struct word* next;
};
```

Meaning Node Structure:

```
struct meaning
{
char sentence[500];
struct word* next_word;
struct meaning* next_meaning;
};
```



PROJECT WORKFLOW



HOME
Find Word
Searches for word from trie
Insert Word
Insert words into the file, trie and linked list
Search by Prefix
Searches for words with inputted prefix
Delete Word
Deletes word from Trie, File and synonyms linked list
Display Synonyms
Displays all synonyms of the inputted word with the help of the their meanings
Display All
Displays all words in the dictionary
Diplay meaning - synonym pairs
Displays all meanings and synonyms present

IMPLEMENTATION

header.h:

```
struct trienode {
    struct trienode *child[255];
    int endofword;
    char meaning[500];
};

struct stack{
    struct trienode *m;
    int index;
};

struct word
{
    char w[200];
    struct word* next;
};

struct meaning
{
    char sentence[500];
    struct word* next_word;
    struct meaning* next_meaning;
};

char ret[100];


void insert_meaning(struct meaning** ,char *,char
*);
void display_pairs(struct meaning* );
void insert_synonym(struct meaning** ,char *,char
*);
int check_meaning(struct meaning* ,char*);
struct trienode* getnode();
void del_thes(struct meaning*, char* , char*);
void find_syn(struct meaning* ,char* , char*);
void del_word(char *);
void add_to_file(char* ,char* );
void delete_trie(struct trienode *,char *);
int check(struct trienode *);
void search_pre(struct trienode * , char *);
char* search(struct trienode * , char *);
```

```
struct stack pop();
void push(struct trienode *,int );
void display(struct trienode *);
void file_to_dict(struct trienode* , struct
meaning** );
void insert(struct trienode* , char *, char *);
void destroy(struct meaning** );
```

implementation.c

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "header.h"

void insert_meaning(struct meaning** head,char
*wor,char *mean)
{
    struct meaning* node= (struct
meaning*)malloc(sizeof(struct meaning));
    struct word* new=(struct
word*)malloc(sizeof(struct word));

    if(node!=NULL && new!=NULL){
    strncpy(node->sentence,mean,450);
    node->next_meaning=NULL;
    strncpy(new->w,wor,190);
    new->next=NULL;
    node->next_word=new;
    }

    if(*head==NULL)
    {
        *head=node;
    }
    else
    {
        struct meaning* temp=*head;
        while(temp->next_meaning!=NULL)
        {
            temp=temp->next_meaning;
        }

        temp->next_meaning=node;
    }
}

void display_pairs(struct meaning* head)
{
    struct meaning* temp=head;
    while(temp!=NULL)
    {
        printf("\nMeaning:%s\n",temp->sentence);
        printf("Synonyms:");
        //printf("%s",temp->next_word->w);
        struct word* temp_word=temp->next_word;
        while(temp_word!=NULL)
        {
            printf("\n%s",temp_word->w);
            temp_word=temp_word->next;
        }
        printf("\n\n");
    temp=temp->next_meaning;
    }
}

void insert_synonym(struct meaning** head,char
*meaning,char *syn)
{
    struct word* node=(struct
word*)malloc(sizeof(struct word));

    if(node!=NULL){
    strncpy(node->w,syn,190);
    node->next=NULL;}

    struct meaning* temp1=*head;
    struct word* temp2;
    while(temp1!=NULL)
    {
        if(strncmp(temp1->sentence,meaning,450)==0)
        {
            temp2=temp1->next_word;
            while(temp2->next!=NULL)
                temp2=temp2->next;
            temp2->next=node;
            break;
        }
        temp1=temp1->next_meaning;
    }

    if(temp1==NULL)
        printf("\nMeaning not found");

}

int check_meaning(struct meaning* head,char*
meaning)
{
    struct meaning* temp= head;
    while(temp!=NULL)
    {
        if(strncmp(temp->sentence,meaning,450)==0)
        {
            return 1;
        }
        temp=temp->next_meaning;
    }
    return 0;
}

int length,top;
char word[100];

struct stack s[255];

struct trienode* getnode()
{
  struct trienode* temp;
  int i;
  temp=(struct trienode *)(malloc(sizeof( struct
trienode)));

  if(temp!=NULL){
  for(i=0;i<255;i++)
    temp->child[i]=NULL;
  temp->endofword=0;
  strncpy(temp->meaning,"",2);}

  return temp;
}

void insert(struct trienode* root, char *key, char
*m)
{
  struct trienode *curr;
  int i, index;

  curr = root;
  for(i=0;key[i]!='\0';i++)
   { index=(int)key[i];
      if(curr->child[index]==NULL)
        curr->child[index]=getnode();
    curr=curr->child[index];
   }
  curr->endofword=1;
  strncpy(curr->meaning,m,450);
}

//char ret[100]="Not Found";
void file_to_dict(struct trienode* root, struct
meaning** head)
{
     FILE* filePointer;
    int bufferLength = 255;
    char buffer[bufferLength];

    filePointer = fopen("file.txt", "r");

    while(fgets(buffer, bufferLength, filePointer))
{
        char word[50];
        char meaning[500];
        int x=0;
        while(buffer[x]!=':')
        {
            word[x]=buffer[x];
            x++;
        }
        word[x]='\0';
        x++;

        int y=0;
        while(buffer[x]!='\0')
        {
            if(buffer[x]=='\n')
             x++;
             else{
             meaning[y]=buffer[x];
            x++;
            y++;}
        }
        meaning[y]='\0';
         insert(root,word,meaning); //add word to
dictionary

        int result=check_meaning(*head,meaning);

        if(result!=0)
        insert_synonym(head,meaning,word);
        else
        insert_meaning(head,word,meaning);


         //call function to add word and meaning to
thesaurus
    }
    fclose(filePointer);
}

void display(struct trienode *curr)
{int i,j;

    for(i=0;i<255;i++)
    {
        if(curr->child[i]!=NULL)
        {
            word[length++]=(char)i;
            if(curr->child[i]->endofword==1)
            {
                //print the word
                printf("\n");
                for(j=0;j<length;j++)
                    printf("%c",word[j]);
            }
            display(curr->child[i]);
        }
    }
    length--;
    return;
}

void push(struct trienode *x,int k)
{
    ++top;
    s[top].m=x;
    s[top].index=k;
}

struct stack pop()
{
    struct stack temp;
    temp=s[top--];
    return temp;
}

char* search(struct trienode * root, char *key)
```

```
{

 int i,index;
 struct trienode *curr;
 curr=root;
  for(i=0;key[i]!='\0';i++)
   { index=(int)key[i];
      if(curr->child[index]==NULL)
          return("Not Found");
      curr=curr->child[index];
   }
   if(curr->endofword==1)
    return(curr->meaning);

   return("Not Found");
}

void search_pre(struct trienode * root, char *key)
{
 int i,index;
 struct trienode *curr;
 curr=root;
 length=0;
  for(i=0;key[i]!='\0';i++)
   { index=(int)key[i];
      if(curr->child[index]==NULL){
          printf("Prefix not found\n");
          return;
      }
      word[length++]=key[i];
      curr=curr->child[index];
   }
    if(curr->endofword==1)
        printf("%s",key);
    display(curr);
}

int check(struct trienode *x)
{
    int i,count=0;
    for(i=0;i<255;i++)
    {
        if(x->child[i]!=NULL) count++;
    }
    return count;
}
```

```
void delete_trie(struct trienode *root,char *key)
{
 int i,k,index;
 struct trienode *curr;
 struct stack x;

 curr =root;
 for(i=0;key[i]!='\0';i++)
   { index=(int)key[i];
      if(curr->child[index]==NULL)
      {
         printf("Word not found..");
         return;
      }
      push(curr,index);
      curr=curr->child[index];
    }
    curr->endofword=0;
    push(curr,-1);

    int ch=1;

    while(ch==1)
    {
        x=pop();
        if(x.index!=-1)
            x.m->child[x.index]=NULL;
        if(x.m==root)//if root
        return;

        k=check(x.m);

        if((k>=1)|| (x.m->endofword==1))break;
        else free(x.m);
    }
    return;
}

void add_to_file(char* word,char* meaning)
{
    char str[600]="";
    strncpy(str,"\n",2);
    strcat(str,word);
    strcat(str,":");
    strcat(str,meaning);
   FILE *ptr;
   ptr=fopen("file.txt","a");
   fputs(str,ptr);
   fclose(ptr);
}

void del_word(char *key)
{
    FILE *ptr,*tptr;
    ptr=fopen("file.txt","r");
    if(ptr == NULL)
    {
        printf("Unable to open file.\n");
        printf("Please check you have read/write
previleges.\n");
         return;
    }
    int line=-1;
    char str[200];
    char *pos;
    while ((fgets(str, 100, ptr)) != NULL)
    {
        line++;
         strncpy(str,strtok(str, ":"),190);
         //printf("'%s'\n",str);
        // Find first occurrence of word in str
        pos = strstr(str, key);

        if (pos != NULL)
        {
            break;
        }
    }
    fclose(ptr);
    //printf("line number %d",line);
    if(!pos)
        return;
    else
        printf("'%s' has been deleted\n",str);
    int ctr=-1;
    ptr=fopen("file.txt","r");
    tptr=fopen("temp.txt","w");
    if (!tptr)
    {
        printf("Unable to open a temporary file to
write!!\n");
```

```
        fclose(ptr);
        return ;
    }
    while ((fgets(str, 100, ptr)) != NULL)
            {
                    ctr++;
                    /* skip the line at given line
number */
                    if (ctr != line)
                    {
                        //ctr++;
                        fprintf(tptr, "%s", str);
                        //ctr++;
                    }
            }
        fclose(ptr);
        fclose(tptr);
         remove("file.txt");
         rename("temp.txt","file.txt");
         return;
}

void find_syn(struct meaning* head,char* meaning,
char* word)
{
    struct meaning* temp=head;
    while(temp!=NULL)
    {
        if(strncmp(temp->sentence,meaning,450)==0)
        {
            struct word* temp2=temp->next_word;
            if(temp2->next==NULL)
            printf("\nNo synonyms for %s
available",word);

            else
            {
                printf("\nSynonyms of %s
are:",word);
               while(temp2!=NULL)
                {
                    if(strncmp(temp2->w,word,190)!
=0)
                   printf("\n%s",temp2->w);
                    temp2=temp2->next;
                }
            }
            break;
        }
        temp=temp->next_meaning;
    }
}

void del_thes(struct meaning* head,char* key, char*
ans)
{
    //char ans[500];
    //printf("word: %s\n",search(root,key));
    //strcpy(ans,search(root,key));
    //printf("print ans %s\n",ans);
    if(strncmp(ans,"Not found",450)==0)
    {
        //printf("compared to not found\n");
        return;
    }
    //printf("before declaration\n");
    struct meaning* temp=head;
    struct meaning* cur;
    while(temp!=NULL)
    {
        //printf("into the loop\n");
        if(strncmp(ans,temp->sentence,450)==0)
        {
            struct word* temp_word=temp-
>next_word;
            struct word* cur_word;
            if(strncmp(temp_word->w,key,190)==0)
            {
                temp->next_word=temp_word->next;
                //printf("Word %s has been deleted
from LL\n",temp_word->w);
                break;
            }
            while((temp_word!
=NULL)&&(strncmp(key,temp_word->w,190)!=0))
            {
                cur_word=temp_word;
                temp_word=temp_word->next;
            }

            if(temp_word!=NULL)
                cur_word->next=temp_word->next;
            //printf("Word %s has been deleted 
from LL\n",temp_word->w);
            free(temp_word);
            break;

        }
        temp=temp->next_meaning;
    }
    temp=head;
    if(temp->next_word==NULL)
    {
        head=temp->next_meaning;
        //printf("Meaning %s has been deleted from 
LL\n",temp->sentence);
        free(temp);
        return;
    }
    while(temp!=NULL && (strncmp(ans,temp-
>sentence,450)!=0))
    {
        cur=temp;
        temp=temp->next_meaning;
    }
    if(temp!=NULL && temp->next_word==NULL)
    {
        cur->next_meaning=temp->next_meaning;
        //printf("Meaning %s has been deleted from 
LL\n",temp->sentence);
        free(temp);
        return;
    }
}


void destroy(struct meaning** head )
{
    struct meaning* cur=*head;
    while(cur->next_meaning!=NULL)
    {
        struct word *temp2=cur->next_word;

        while(temp2->next!=NULL)
        {
            struct word *cur2=temp2;
            temp2=temp2->next;
            free(cur2);
        }

        free(temp2);
        cur=cur->next_meaning;
    }

    free(cur);
    *head=NULL;
}

void destroy_trie(struct trienode** root)
{
    struct trienode* temp= *root;

    int i;

    for(i=0;i<255;i++)
    {
        if((temp->child[i])!=NULL)
        {
            destroy_trie(&temp->child[i]);
            free(temp->child[i]);
        }
    }
}
```

dict_driver.c

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "header.h"


int length,top;
int main()
{
    char key[100];
    char m[500];
    struct trienode *root;
    int ch=1;

    root=getnode();
    struct meaning *head= NULL;
    file_to_dict(root,&head);
    while(ch!=0)
    {
    printf("\n\
n--------------------------------------------------
-------------------------------------------------")
;
    printf("\n1. Insert");
    printf("\n2. Display");
    printf("\n3. Search meaning");
    printf("\n4. Delete word");
    printf("\n5. Display prefix");
    printf("\n6. Display meaning-synonyms");
    printf("\n7. Display synonyms of a word\n");

    int y=scanf("%d",&ch);
    if(y!=1)
    return 0;

    switch(ch)
    {
    case 1: {
                printf("Enter the string: ");
                int x=scanf(" %[^\n]s",key);

                if(x!=1)
                 break;
                printf("Enter the meaning: ");
                x=scanf(" %[^\n]s",m);

                if(x!=1)
                break;

                insert(root, key,m);
               int result=check_meaning(head,m);
               //printf("word is %s and check is
%d",m,result);

               if(result!=0)
               insert_synonym(&head,m,key);
               else
               insert_meaning(&head,key,m);

                add_to_file(key,m);
             }break;

    case 2: {length=0;
              printf("The words stored in the
dictionary are: ");
              display(root);
             }break;

    case 3: {     printf("Enter the string for
search: ");
                int x=scanf(" %[^\n]%*c", key);

                if(x!=1)
                break;

               char search_value[500];
                strcpy(search_value,search(root,
key));
               if(strcmp(search_value,"Not Found")!
=0)
               {
                   printf("\nMeaning is:
%s",search_value);
               }
                else
               printf("%s",search_value);
             }break;
     case 4: {
             printf("Enter the word for deletion..\
n");
             int x=scanf("%s",key);

             if(x!=1)
             break;
             top=0;
             char search_value[500];
             strcpy(search_value,search(root,
key));

             if(strcmp(search_value,"Not Found")!
=0)
             {   delete_trie(root,key);
                 del_thes(head,key,search_value);
                 del_word(key);
             }

             else
             printf("\n%s",search_value);
     }
             break;

     case 5: {
             length=0;
             printf("Enter the string for
search ..\n");
             int x=scanf("%s",key);

             if(x!=1)
             break;

             search_pre(root, key);
     }break;

    case 6: {
                if(head!=NULL)
                 display_pairs(head);
            }break;

    case 7: {
                printf("Enter the string for
search: ");
                int x=scanf(" %[^\n]s",key);
                if(x!=1)
```

```
                break;

               char search_value[500];
                strcpy(search_value,search(root,
key));
               if(strcmp(search_value,"Not Found")!
=0)
               {
                   find_syn(head,search_value,key);
               }
               else
               printf("%s",search_value);
            }break;



    default: {
             ch=0;
             destroy(&head);
             destroy_trie(&root);
             //exit(0);
         }
    }
  }
  return 0;
}
```

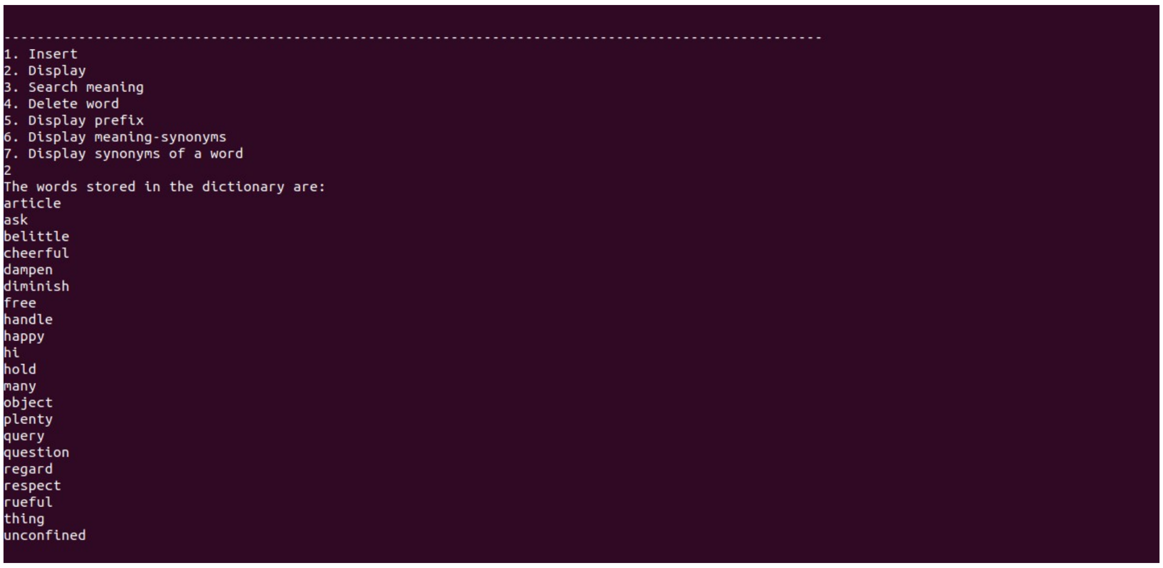
TESTING

Test case 1
FIO42-C. Close files when they are no longer needed

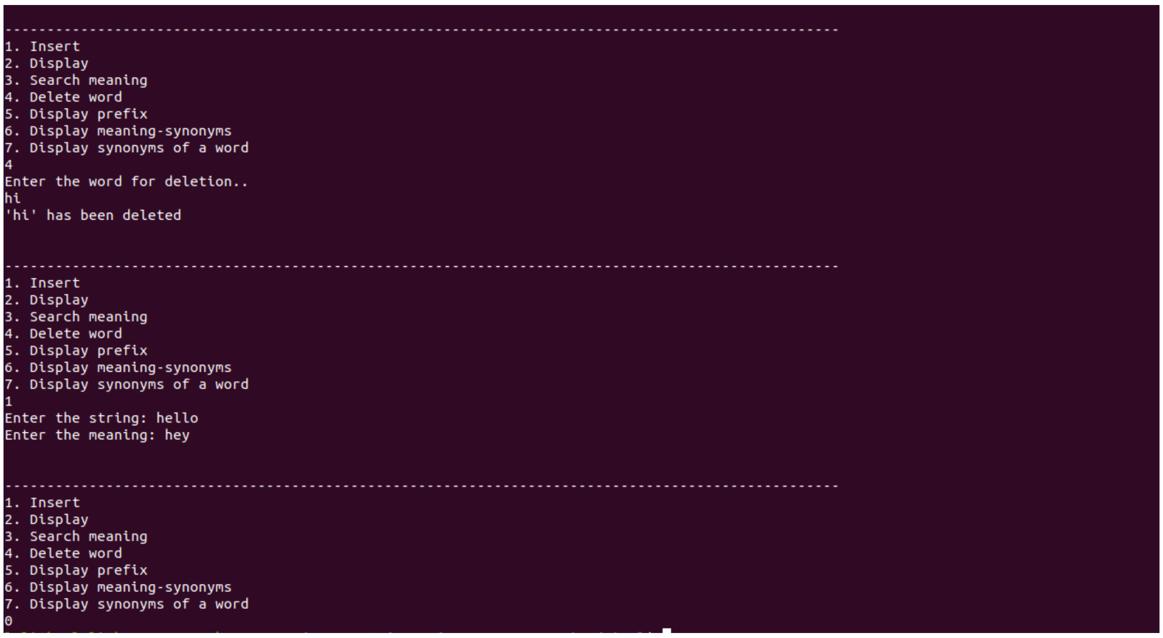
Compliant code with fclose (file delete function):

```
void del_word(char *key)
{
     FILE *ptr,*tptr;
     ptr=fopen("file.txt","r");
     if(ptr == NULL)
    {
        printf("Unable to open file.\n");
        printf("Please check you have read/write
previleges.\n");
          return;
    }
     int line=-1;
     char str[200];
     char *pos;
     while ((fgets(str, 100, ptr)) != NULL)
    {
        line++;
          strncpy(str,strtok(str, ":"),190);
          //printf("'%s'\n",str);
        // Find first occurrence of word in str
        pos = strstr(str, key);

        if (pos != NULL)
        {
            break;
        }
    }
     fclose(ptr);
     //printf("line number %d",line);
     if(!pos)
          return;
     else
          printf("'%s' has been deleted\n",str);
     int ctr=-1;
     ptr=fopen("file.txt","r");
     tptr=fopen("temp.txt","w");
     if (!tptr)
     {
        printf("Unable to open a temporary file to write!!\
n");
        fclose(ptr);
        return ;
    }
     while ((fgets(str, 100, ptr)) != NULL)
               {
                         ctr++;
                         /* skip the line at given line
number */
                         if (ctr != line)
                         {
                              //ctr++;
                              fprintf(tptr, "%s", str);
                              //ctr++;
                         }
               }
        fclose(ptr);
        fclose(tptr);
          remove("file.txt");
          rename("temp.txt","file.txt");
          return;
}
```

Sample input:



```
---------------------------------------------------------------------------------------------
1. Insert
2. Display
3. Search meaning
4. Delete word
5. Display prefix
6. Display meaning-synonyms
7. Display synonyms of a word
2
The words stored in the dictionary are:
article
ask
belittle
cheerful
dampen
diminish
free
handle
happy
hi
hold
many
object
plenty
query
question
regard
respect
rueful
thing
unconfined
```



```
------------------------------------------------------------------------------------------------
1. Insert
2. Display
3. Search meaning
4. Delete word
5. Display prefix
6. Display meaning-synonyms
7. Display synonyms of a word
4
Enter the word for deletion..
hi
'hi' has been deleted


------------------------------------------------------------------------------------------------
1. Insert
2. Display
3. Search meaning
4. Delete word
5. Display prefix
6. Display meaning-synonyms
7. Display synonyms of a word
1
Enter the string: hello
Enter the meaning: hey


------------------------------------------------------------------------------------------------
1. Insert
2. Display
3. Search meaning
4. Delete word
5. Display prefix
6. Display meaning-synonyms
7. Display synonyms of a word
0
```

File after program termination:

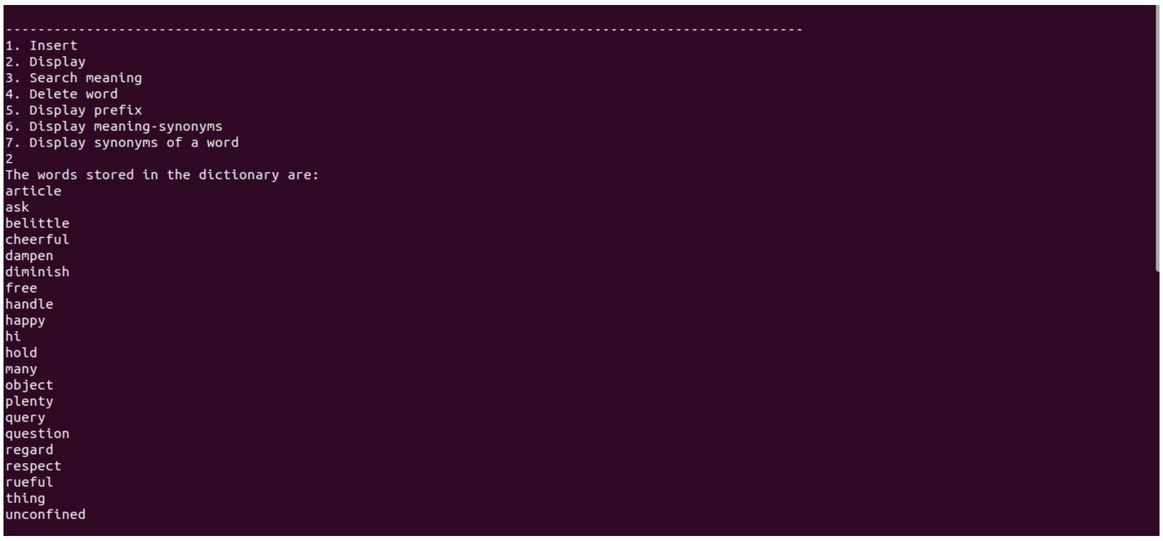
```
rueful:feeling, showing, or expressing sorrow, repentance, or regret
dampen:to dull or deaden
diminish:to dull or deaden
happy:delighted, pleased, or glad, as over a particular thing
cheerful:delighted, pleased, or glad, as over a particular thing
respect:esteem for or a sense of the worth or excellence of a person
regard:esteem for or a sense of the worth or excellence of a person
belittle:to regard or portray as less impressive or important
thing:an inanimate material object as distinct from a living sentient being
object:an inanimate material object as distinct from a living sentient being
article:an inanimate material object as distinct from a living sentient being
free:able to act or be done as one wishes
unconfined:able to act or be done as one wishes
ask:say something in order to obtain an answer or some information
query:say something in order to obtain an answer or some information
question:say something in order to obtain an answer or some information
handle:feel or manipulate with the hands
hold:feel or manipulate with the hands
plenty:a large or sufficient amount or quantity; more than enough
many:a large or sufficient amount or quantity; more than enough
hello:hey
```

From above, it is clear that line “hello:hey” has been added to the file at the end. The insert function was called after delete.

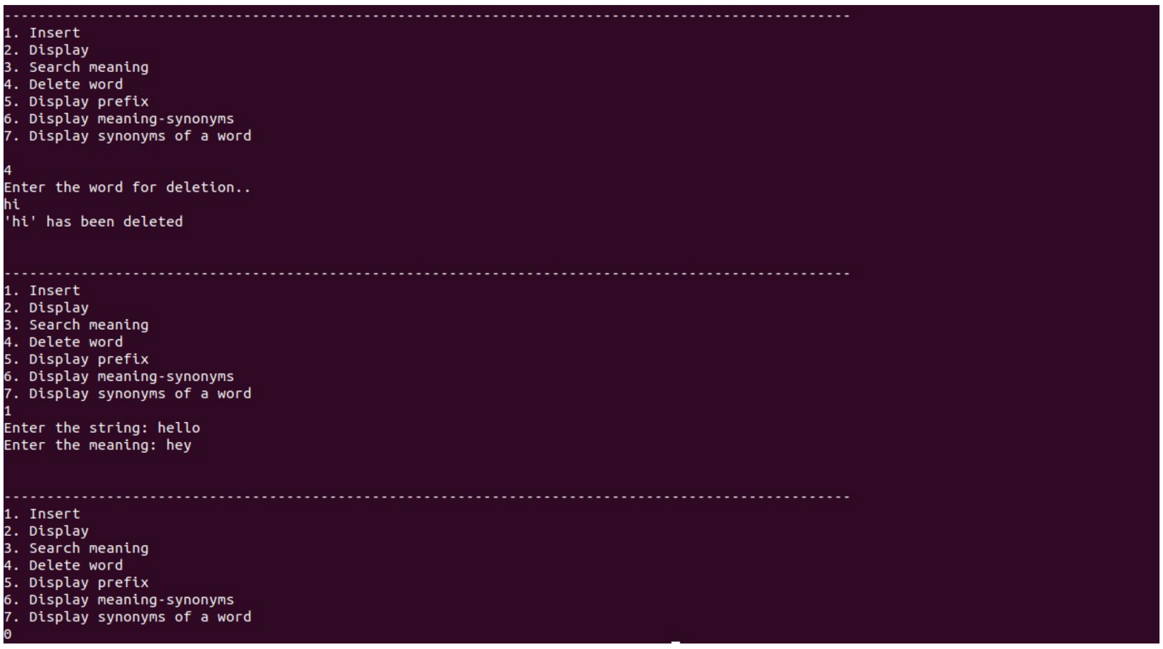
Non-Compliant code without fclose (delete function):

```
void del_word(char *key)
{
     FILE *ptr,*tptr;
     ptr=fopen("file.txt","r");
     if(ptr == NULL)
    {
        printf("Unable to open file.\n");
        printf("Please check you have read/write
previleges.\n");
          return;
    }
     int line=-1;
     char str[200];
     char *pos;
     while ((fgets(str, 100, ptr)) != NULL)
    {
        line++;
          strncpy(str,strtok(str, ":"),190);

        pos = strstr(str, key);

        if (pos != NULL)
        {
            break;
        }
    }

     if(!pos)
          return;
     else
          printf("'%s' has been deleted\n",str);
     int ctr=-1;
     ptr=fopen("file.txt","r");
     tptr=fopen("temp.txt","w");
     if (!tptr)
     {
        printf("Unable to open a temporary file to write!!\
n");

        return ;
    }
     while ((fgets(str, 100, ptr)) != NULL)
               {
                         ctr++;
                         /* skip the line at given line
number */
                         if (ctr != line)
                         {
                              //ctr++;
                              fprintf(tptr, "%s", str);
                              //ctr++;
                         }
               }

          remove("file.txt");
          rename("temp.txt","file.txt");
          return;
}
```

Sample input:



```
----------------------------------------------------------------------------------------------------
1. Insert
2. Display
3. Search meaning
4. Delete word
5. Display prefix
6. Display meaning-synonyms
7. Display synonyms of a word
2
The words stored in the dictionary are:
article
ask
belittle
cheerful
dampen
diminish
free
handle
happy
hi
hold
many
object
plenty
query
question
regard
respect
rueful
thing
unconfined
```



```
---------------------------------------------------------------------------------------------------
1. Insert
2. Display
3. Search meaning
4. Delete word
5. Display prefix
6. Display meaning-synonyms
7. Display synonyms of a word

4
Enter the word for deletion..
hi
'hi' has been deleted


---------------------------------------------------------------------------------------------------
1. Insert
2. Display
3. Search meaning
4. Delete word
5. Display prefix
6. Display meaning-synonyms
7. Display synonyms of a word
1
Enter the string: hello
Enter the meaning: hey


---------------------------------------------------------------------------------------------------
1. Insert
2. Display
3. Search meaning
4. Delete word
5. Display prefix
6. Display meaning-synonyms
7. Display synonyms of a word
0
```

File after program termination:

rueful:feeling, showing, or expressing sorrow, repentance, or regret
dampen:to dull or deaden
diminish:to dull or deaden
happy:delighted, pleased, or glad, as over a particular thing
cheerful:delighted, pleased, or glad, as over a particular thing
respect:esteem for or a sense of the worth or excellence of a person
regard:esteem for or a sense of the worth or excellence of a person
belittle:to regard or portray as less impressive or important
thing:an inanimate material object as distinct from a living sentient being
object:an inanimate material object as distinct from a living sentient being
article:an inanimate material object as distinct from a living sentient being
free:able to act or be done as one wishes
unconfined:able to act or be done as one wishes
ask:say something in order to obtain an answer or some information
query:say something in order to obtain an answer or some information
question:say something in order to obtain an answer or some information
handle:feel or manipulate with the hands
hold:feel or manipulate with the hands
plenty:a large or sufficient amount or quantity; more than enough
many:a large or sufficient amount or quantity; more than enough

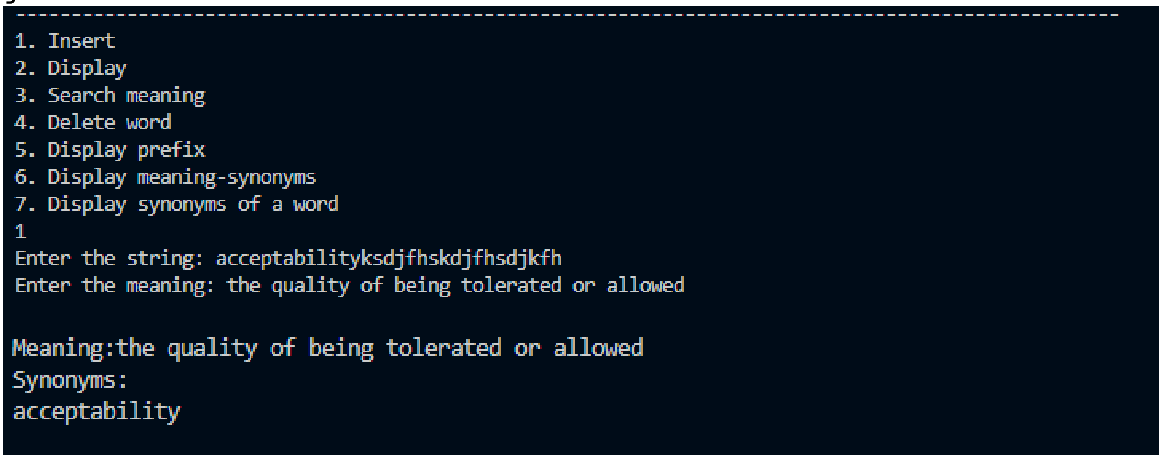
Input “hello:hey” added after delete function not added to file.

<u>Test Case 2:</u>

STR31-C. Guarantee that storage for strings has sufficient space for character data and the null terminator

Compliant Code

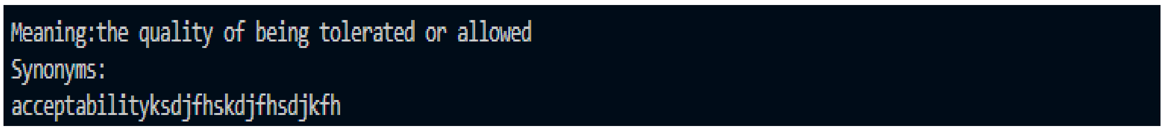
```
void insert_synonym(struct meaning** head,char *meaning,char
*syn)
{
     struct word* node=(struct word*)malloc(sizeof(struct
word));

     if(node!=NULL){
     strncpy(node->w,syn,15);
     node->next=NULL;}

     struct meaning* temp1=*head;
     struct word* temp2;
     while(temp1!=NULL)
     {
          if(strncmp(temp1->sentence,meaning,450)==0)
          {
               temp2=temp1->next_word;
               while(temp2->next!=NULL)
                    temp2=temp2->next;
               temp2->next=node;
               break;
          }
          temp1=temp1->next_meaning;
     }

     if(temp1==NULL)
          printf("\nMeaning not found");

}
```



```
--------------------------------------------------------------------------------------------
1. Insert
2. Display
3. Search meaning
4. Delete word
5. Display prefix
6. Display meaning-synonyms
7. Display synonyms of a word
1
Enter the string: acceptabilityksdjfhskdjfhsdjkfh
Enter the meaning: the quality of being tolerated or allowed

Meaning:the quality of being tolerated or allowed
Synonyms:
acceptability
```

Non-Compliant Code

```
void insert_synonym(struct meaning** head,char *meaning,char
*syn)
{
     struct word* node=(struct word*)malloc(sizeof(struct
word));
     strcpy(node->w,syn);
     node->next=NULL;
     struct meaning* temp1=*head;
     struct word* temp2;
     while(temp1!=NULL)
     {
          if(strcmp(temp1->sentence,meaning)==0)
          {
               temp2=temp1->next_word;
               while(temp2->next!=NULL)
                    temp2=temp2->next;
               temp2->next=node;
               break;
          }
          temp1=temp1->next_meaning;
     }

     if(temp1==NULL)
          printf("\nMeaning not found");

}
```



```
Meaning:the quality of being tolerated or allowed
Synonyms:
acceptabilityksdjfhskdjfhsdjkfh
```

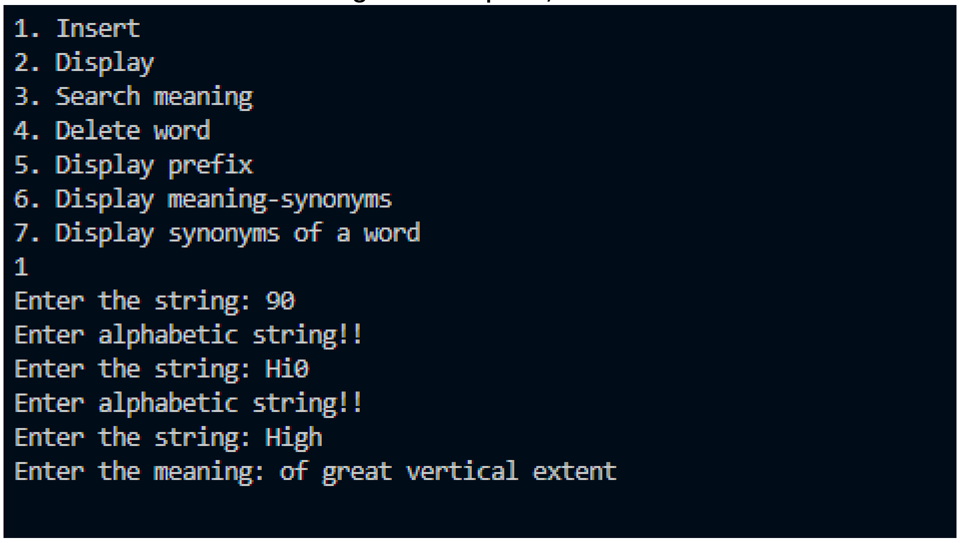
When strcpy is used, the field word takes any number of characters as input, which might lead to buffer overflow. By using strncpy and specifying the limit as 13 we avoid this vulnerability.

Test Case 3:
STR02-C. Sanitize data passed to complex subsystems

Compliant Code

```
input:
                  printf("Enter the string: ");
                  int x=scanf(" %[^\n]s",key);
                  int check=1;
                  for(int i=0;key[i]!='\0';i++)
               {
                       if(!isalpha(key[i]))
                       {
                            check=0;
                            printf("Enter alphabetic
string!!\n");
                            break;
                       }
                  }
```

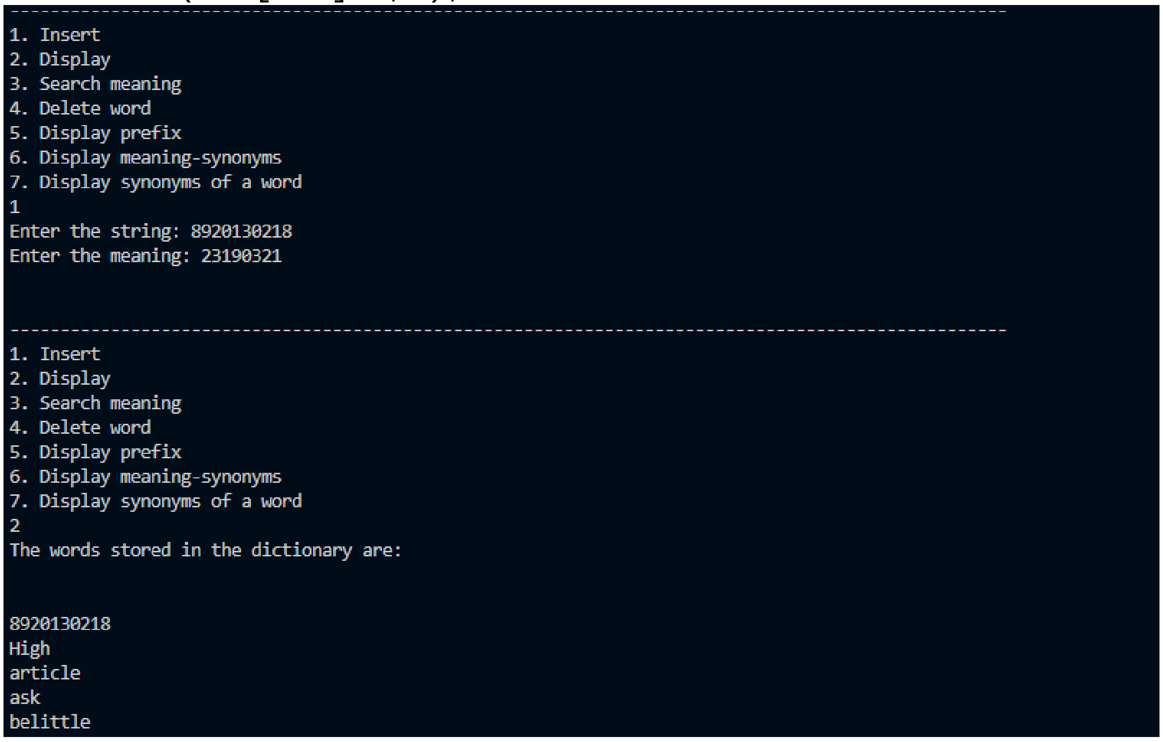
```
                    if(check==0)
                    goto input;
```



```
1. Insert
2. Display
3. Search meaning
4. Delete word
5. Display prefix
6. Display meaning-synonyms
7. Display synonyms of a word
1
Enter the string: 90
Enter alphabetic string!!
Enter the string: Hi0
Enter alphabetic string!!
Enter the string: High
Enter the meaning: of great vertical extent
```

Non-compliant Code

```
printf("Enter the string: ");
int x=scanf(" %[^\n]s",key);
printf("Enter the meaning: ");
x = scanf(" %[^\n]s",m);
```

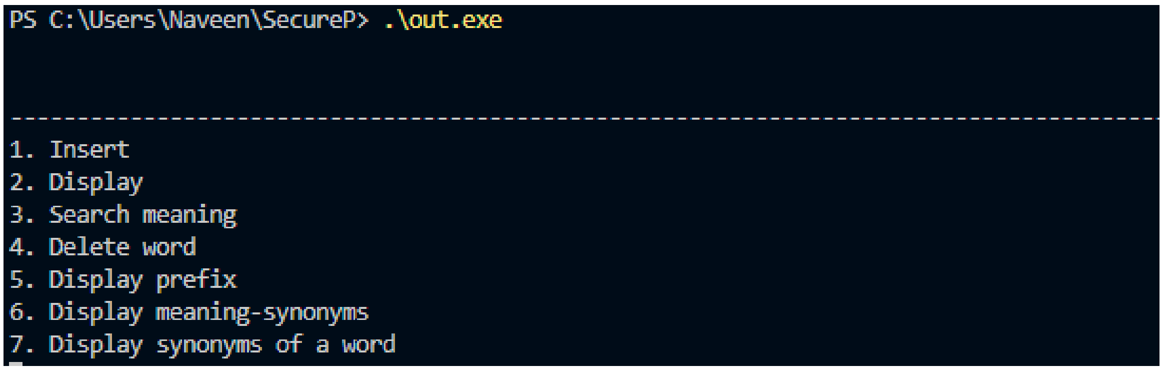


```
----------------------------------------------------------------------------------------------------
1. Insert
2. Display
3. Search meaning
4. Delete word
5. Display prefix
6. Display meaning-synonyms
7. Display synonyms of a word
1
Enter the string: 8920130218
Enter the meaning: 23190321


----------------------------------------------------------------------------------------------------
1. Insert
2. Display
3. Search meaning
4. Delete word
5. Display prefix
6. Display meaning-synonyms
7. Display synonyms of a word
2
The words stored in the dictionary are:


8920130218
High
article
ask
belittle
```

Test Case 4:
MEM35-C. Allocate sufficient memory for an object
Compliant Code:

```
     struct meaning* node= (struct
meaning*)malloc(sizeof(struct meaning));
     struct word* new=(struct word*)malloc(sizeof(struct
word));
     if(node==NULL || new==NULL)
```

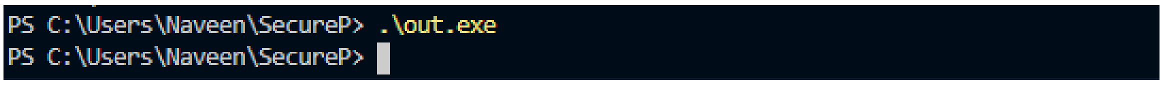
```
	{
		printf("Malloc Failed\n");
	}
```



```
PS C:\Users\Naveen\SecureP> .\out.exe


-----------------------------------------------------------------------------------
1. Insert
2. Display
3. Search meaning
4. Delete word
5. Display prefix
6. Display meaning-synonyms
7. Display synonyms of a word
```

Non-Compliant Code:

```
	struct meaning* node= (struct meaning*)malloc(10);
	//insufficient memory allocated
	struct word* new=(struct word*)malloc(sizeof(struct
word));
	if(node==NULL || new==NULL)
	{
		printf("Malloc Failed\n");
	}
```



```
PS C:\Users\Naveen\SecureP> .\out.exe
PS C:\Users\Naveen\SecureP>
```

Test case 5
MEM31-C. Free dynamically allocated memory when no longer needed
Compliant Code:

```
void insert_synonym(struct meaning** head,char *meaning,char
*syn)
{
	struct word* node=(struct word*)malloc(sizeof(struct
word));

	if(node!=NULL){
	strncpy(node->w,syn,190);
	node->next=NULL;}

	struct meaning* temp1=*head;
	struct word* temp2;
	while(temp1!=NULL)
	{
		if(strncmp(temp1->sentence,meaning,450)==0)
		{
			temp2=temp1->next_word;
			while(temp2->next!=NULL)
```

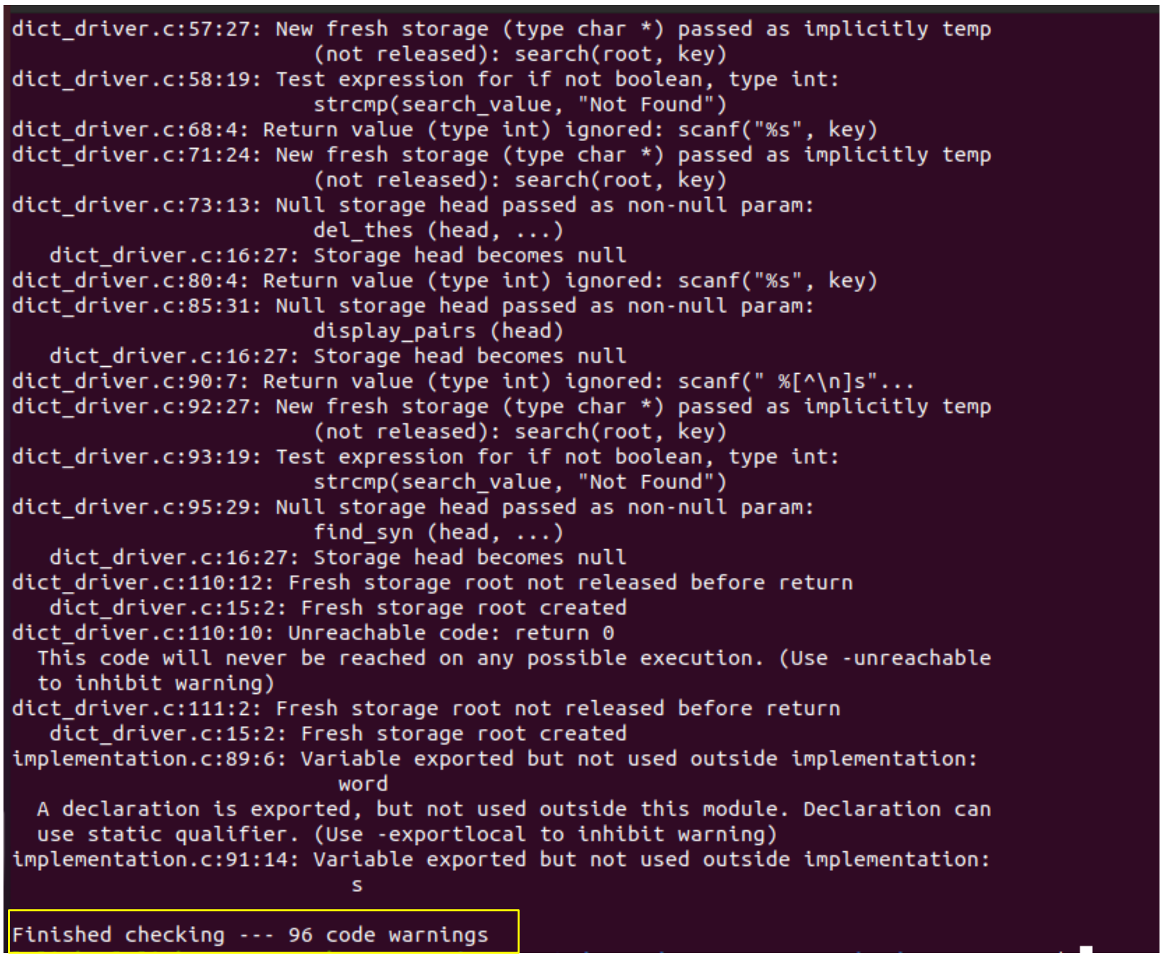
```
                    temp2=temp2->next;
               temp2->next=node;
               break;
          }
          temp1=temp1->next_meaning;
     }

     if(temp1==NULL)
          printf("\nMeaning not found");
     free(node);
}
```

<u>Non-Compliant Code:</u>

```
void insert_synonym(struct meaning** head,char *meaning,char
*syn)
{
     struct word* node=(struct word*)malloc(sizeof(struct
word));

     if(node!=NULL){
     strncpy(node->w,syn,190);
     node->next=NULL;}

     struct meaning* temp1=*head;
     struct word* temp2;
     while(temp1!=NULL)
     {
          if(strncmp(temp1->sentence,meaning,450)==0)
          {
               temp2=temp1->next_word;
               while(temp2->next!=NULL)
                    temp2=temp2->next;
               temp2->next=node;
               break;
          }
          temp1=temp1->next_meaning;
     }

     if(temp1==NULL)
          printf("\nMeaning not found");

}
```

STATIC ANALYSIS REPORT

Splint warnings generated



```
dict_driver.c:57:27: New fresh storage (type char *) passed as implicitly temp
                        (not released): search(root, key)
dict_driver.c:58:19: Test expression for if not boolean, type int:
                        strcmp(search_value, "Not Found")
dict_driver.c:68:4: Return value (type int) ignored: scanf("%s", key)
dict_driver.c:71:24: New fresh storage (type char *) passed as implicitly temp
                        (not released): search(root, key)
dict_driver.c:73:13: Null storage head passed as non-null param:
                        del_thes (head, ...)
   dict_driver.c:16:27: Storage head becomes null
dict_driver.c:80:4: Return value (type int) ignored: scanf("%s", key)
dict_driver.c:85:31: Null storage head passed as non-null param:
                        display_pairs (head)
   dict_driver.c:16:27: Storage head becomes null
dict_driver.c:90:7: Return value (type int) ignored: scanf(" %[^\n]s"...
dict_driver.c:92:27: New fresh storage (type char *) passed as implicitly temp
                        (not released): search(root, key)
dict_driver.c:93:19: Test expression for if not boolean, type int:
                        strcmp(search_value, "Not Found")
dict_driver.c:95:29: Null storage head passed as non-null param:
                        find_syn (head, ...)
   dict_driver.c:16:27: Storage head becomes null
dict_driver.c:110:12: Fresh storage root not released before return
   dict_driver.c:15:2: Fresh storage root created
dict_driver.c:110:10: Unreachable code: return 0
  This code will never be reached on any possible execution. (Use -unreachable
  to inhibit warning)
dict_driver.c:111:2: Fresh storage root not released before return
   dict_driver.c:15:2: Fresh storage root created
implementation.c:89:6: Variable exported but not used outside implementation:
                          word
  A declaration is exported, but not used outside this module. Declaration can
  use static qualifier. (Use -exportlocal to inhibit warning)
implementation.c:91:14: Variable exported but not used outside implementation:
                          s

Finished checking --- 96 code warnings
```

92 warnings were totally generated when the code was compiled on splint

Warnings handled:

1. Null dereferences:

   A common cause of program failures is when a null pointer is dereferenced. It occurs either because of missing a null check after a memory allocation function call, such as malloc or realloc, or it occurs because of invalid/wrong pointer assignments. Splint detects these errors and shows warnings such as:
   implementation.c:10:13: Arrow access from possibly null pointer node:
   node->sentence
   These types of vulnerabilities are dealt with by handling the cases where a pointer can be NULL and running the program accordingly.

```
struct meaning* node= (struct
meaning*)malloc(sizeof(struct meaning));
    struct word* new=(struct
word*)malloc(sizeof(struct word));

    if(node!=NULL && new!=NULL){
    strcpy(node->sentence,mean);
    node->next_meaning=NULL;
    strcpy(new->w,wor);
```

2. Vulnerabilities in string functions:

   Considering strcpy() as an example,
   The strcpy() function does not specify the size of the destination array, so buffer overrun is often a risk. Using strcpy() function to copy a large character array into smaller one is dangerous, but if the string will fit, then it will not worth the risk. If destination string is not large

enough to store the source string then the behavior of strcpy() is unspecified or undefined.
To tackle this we use strncpy(). The strncpy() function is similar to strcpy() function, except that at most n bytes of src are copied. If there is no NULL character among the first n character of src, the string placed in dest will not be NULL-terminated. If the length of src is less than n, strncpy() writes additional NULL character to dest to ensure that a total of n character are written.
Stack smashed error when appropriate size input not given.

```
strncpy(node->sentence,mean,450);
```

3. Ignoring return value of scanf()

   The scanf() function returns number of parameters when the scan is successful and 0 otherwise. We ensure that the program proceeds only when this is successful.
   dict_driver.c:80:4: Return value (type int) ignored: scanf("%s", key)

```
printf("Enter the word for deletion..\n");
int x=scanf("%s",key);
if(x!=1)
     break;
```

4. Implicit type conversion:

   We avoid implicit type conversion as this might lead to erroneous values stored in the variables. We also provide bool values as the loop conditions instead of int values.
   implementation.c:173:4: Assignment of int to char: word[length++] = i

```
word[length++]=(char)i;
```

implementation.c:157:12: Test expression for if not boolean, type int: result
Test expression type is not boolean or int. (Use -predboolint to inhibit warning)

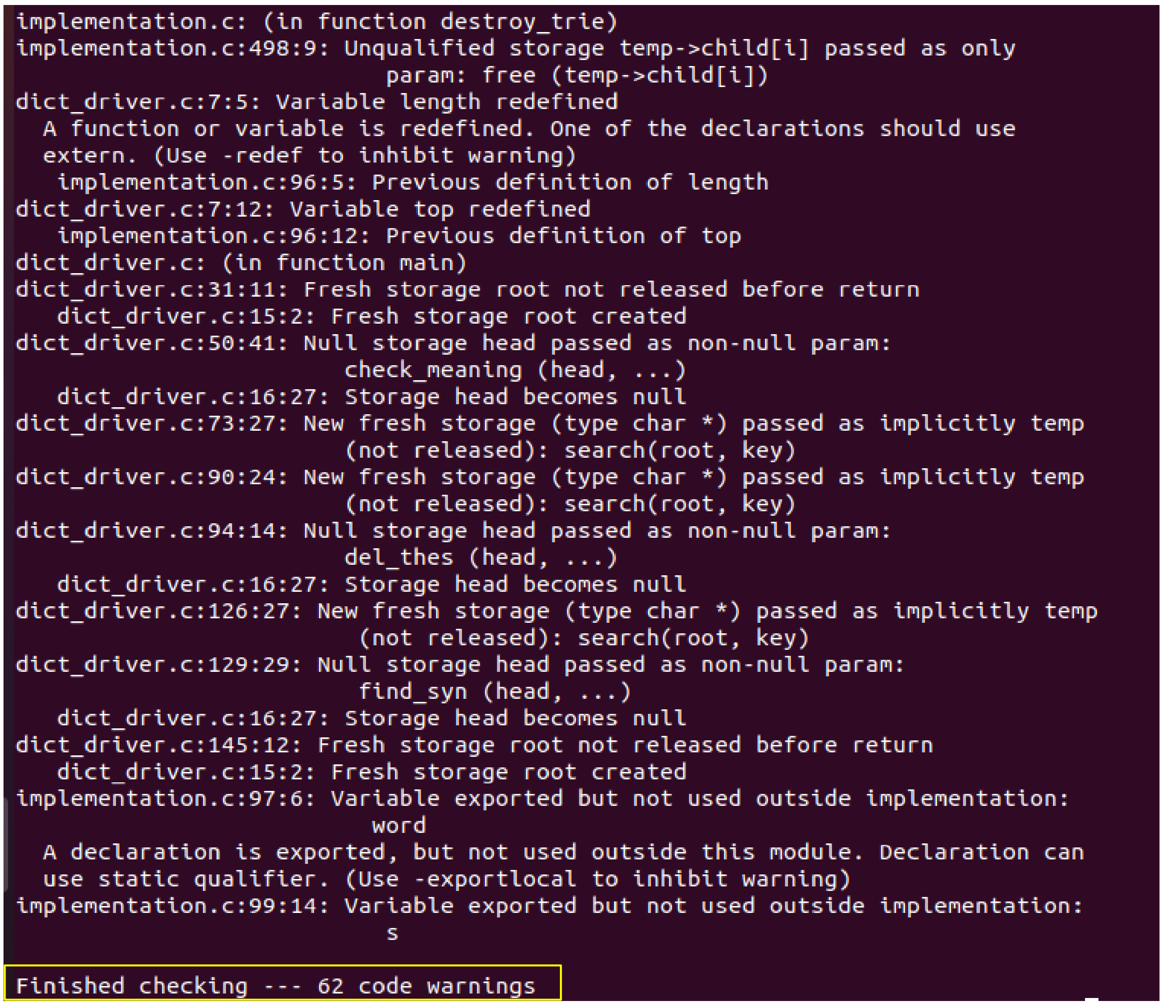
```
int result=check_meaning(*head,meaning);
if(result!=0)
        insert_synonym(head,meaning,word);
```

5. Unreachable code return 0:

   We changed the structure of the main while loop such that the program doesn't quit abruptly and only quits as it reaches the final return 0 instruction.
   dict_driver.c:110:10: Unreachable code: return 0

```
int ch=1;
while(ch!=0)
………
default: ch=0;
```

Splint Warnings after corrections made



```
implementation.c: (in function destroy_trie)
implementation.c:498:9: Unqualified storage temp->child[i] passed as only
                           param: free (temp->child[i])
dict_driver.c:7:5: Variable length redefined
  A function or variable is redefined. One of the declarations should use
  extern. (Use -redef to inhibit warning)
   implementation.c:96:5: Previous definition of length
dict_driver.c:7:12: Variable top redefined
   implementation.c:96:12: Previous definition of top
dict_driver.c: (in function main)
dict_driver.c:31:11: Fresh storage root not released before return
   dict_driver.c:15:2: Fresh storage root created
dict_driver.c:50:41: Null storage head passed as non-null param:
                        check_meaning (head, ...)
   dict_driver.c:16:27: Storage head becomes null
dict_driver.c:73:27: New fresh storage (type char *) passed as implicitly temp
                        (not released): search(root, key)
dict_driver.c:90:24: New fresh storage (type char *) passed as implicitly temp
                        (not released): search(root, key)
dict_driver.c:94:14: Null storage head passed as non-null param:
                        del_thes (head, ...)
   dict_driver.c:16:27: Storage head becomes null
dict_driver.c:126:27: New fresh storage (type char *) passed as implicitly temp
                         (not released): search(root, key)
dict_driver.c:129:29: Null storage head passed as non-null param:
                         find_syn (head, ...)
   dict_driver.c:16:27: Storage head becomes null
dict_driver.c:145:12: Fresh storage root not released before return
   dict_driver.c:15:2: Fresh storage root created
implementation.c:97:6: Variable exported but not used outside implementation:
                          word
  A declaration is exported, but not used outside this module. Declaration can
  use static qualifier. (Use -exportlocal to inhibit warning)
implementation.c:99:14: Variable exported but not used outside implementation:
                           s

Finished checking --- 62 code warnings
```

Warnings reduced to 62 after making the require changes in code

CONCLUSION

In this project, we made a Dictionary and Thesaurus using C language. We used a trie to implement the dictionary and used a linked list of linked lists to implement the thesaurus. We included functionalities such as search word, display words by prefix, display synonyms etc.
Once we completed the error-free code, we learnt that our code is still vulnerable to security threats. Hence we implemented several CERT recommendations to protect our program for these threats. For example we implemented input sanitization, dealt with dangerous or unnecessary code and prevented situations of buffer overflow.
Through this project we learnt how to code a complete and secure program.