MINI SEARCH ENGINE

Bachelor of Technology

in

ELECTRICAL AND ELECTRONICS ENGINEERING

Submitted by

V.SAI PRAVEEN-170060064

UNDER ESTIMATED GUIDANCE OF

M.RADHA

ASSISTANT PROFFESOR

DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

K L DEEMED TO BE UNIVERSITY

Guntur – 522 502

K L DEEMED TO BE UNIVERSITY

DEPARTMENT OF ELECTRICALS AND ELECTRONICS ENGINEERING

CERTIFICATE

This is to certify that this project-based lab report entitled **"MINI SEARCH ENGINE"** is a bonafide work done by V.SAI PRAVEEN **(170060064)** in partial fulfilment of the requirements for the award of degree in **BACHELOR OF TECHNOLOGY** in **ELECTRICAL AND ELECTRONICS ENGINEERING** during the Academic year 2018-2019.

Faculty in Charge

M.RADHA

Head of the Department

Dr. K. NARASIMHA RAJU

Project guide

M.RADHA (Assistant professor)

K L DEEMED TO BE UNIVERSITY

DEPARTMENT ELECTRICAL AND ELECTRONICS ENGINEERING

DECLARATION

We hereby declare that this project-based lab report titled **"MINI SEARCH ENGINE"** has been prepared by us in partial fulfilment of the requirements for the award of degree "**BACHELOR OF TECHNOLOGY in ELECTRICAL AND ELECTRONICS ENGNEERING**" during the Academic year 2017-2018.

We also declare that this project-based lab report is of our own efforts and it has not been submitted to any other university for the award of any degree.

V.SAI PRAVEEN (170060064)

ACKNOWLEDGEMENT

Our sincere thanks to M. RADHA MAM **in** the Lab for their outstanding support throughout the project for the successful completion of the work.

We express our gratitude to **DR.K. NARASIMHA RAJU,** Head of the Department for Computer Science of Engineering for providing us with adequate facilities, ways and means by which we are able to complete this project-based Lab.

We would like to place on record the deep sense of gratitude to the honourable Vice Chancellor, K L University for providing the necessary facilities to carry the project-based Lab.

Last, but not the least, we thank all Teaching and Non-Teaching Staff of our department and especially my classmates and my friends for their support in the completion of our project-based Lab.

V.SAI PRAVEEN (170060064)

INDEX

1. ABSTRACT

As we know the title itself describes that it’s a searching-based program and have to implement it by linked list approach. Our task is to design and implement an algorithm that searches a collection of documents. We will be provided with a set of 50 documents and a set of sample queries. You have the freedom to select the data structures and algorithms that you consider to be more efficient for this task. Of course, we will have to justify your decisions. In our approach we have to undergo two modules.1) Indexing Access the data by saving their contents like words/tokens in the data structure we selected and have to access the file by storing keywords.2) Retrieval In this module we have to search for the key words which stored in the documents by using operators.

2. **INTRODUCTION**

Initially we have to input the root path of the folder which we are going to search. Then the search engine should search all the files within that folder as well as the subfolders. Then it should read all the words in text files and store the words and the corresponding file paths in a suitable data structure. After finishing the storing the search engine is ready for search. Then the user can input a word alone or multiple words using & and | operations for search. then the search engine should return the corresponding file paths as the output. linked list is a linear collection of data elements, whose order is not given by their physical placement in memory. Instead, each element points to the next. It is a data structure consisting of a collection of nodes which together represent a sequence. In its most basic form, each node contains: data, and a reference (in other words, a link) to the next node in the sequence. This structure allows for efficient insertion or removal of elements from any position in the sequence during iteration. More complex variants add additional links, allowing more efficient insertion or removal of nodes at arbitrary positions. A drawback of linked lists is that access time is linear (and difficult to pipeline). Faster access, such as random access, is not feasible. Arrays have better cache locality compared to linked lists.

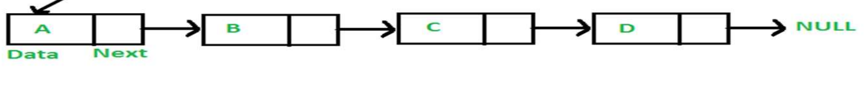
3. DESCRIPTION

The main features of this project include basic file handling operations; you will learn how to search data from file. so thoroughly go through the mini project, and try to analyze how things such as functions, pointers, files, and linked lists are implemented. The documents will be stored using files and given a set of texts and a query, the search engine will locate all the documents that contain the keywords in that query. The purpose of this project is to provide an overview of how a search engine works and to gain hands on experience in using linked list and files. A linked list is formed when many such nodes are linked together to form a chain. Each node points to the next node present in the order. The first node is always used as a reference to traverse the list and is called HEAD. The last node points to NULL. Linked lists are among the simplest and most common data structures. They can be used to implement several other common abstract data types, including lists, stacks, queues, associative arrays, and S-expressions, though it is not uncommon to implement those data structures directly without using a linked list as the basis.

The principal benefit of a linked list over a conventional array is that the list elements can be easily inserted or removed without reallocation or reorganization of the entire structure because the data items need not be stored contiguously in memory or on disk, while restructuring an array at run-time is a much more expensive operation. Linked lists allow insertion and removal of nodes at any point in the list, and allow doing so with a constant number of operations by keeping the link previous to the link being added or removed in memory during list traversal.

Linked lists are a way to store data with structures so that the programmer can automatically create a new place to store data whenever necessary. Specifically, the programmer writes a struct definition that contains variables holding information about something and that has a pointer to a struct of its same type (it has to be a pointer--otherwise, every time an element was created, it would create a new element, infinitely). Each of these individuals structs or classes in the list is commonly known as a node or element of the list.

Like arrays, Linked List is a linear data structure. Unlike arrays, linked list elements are not stored at contiguous location; the elements are linked using pointers.



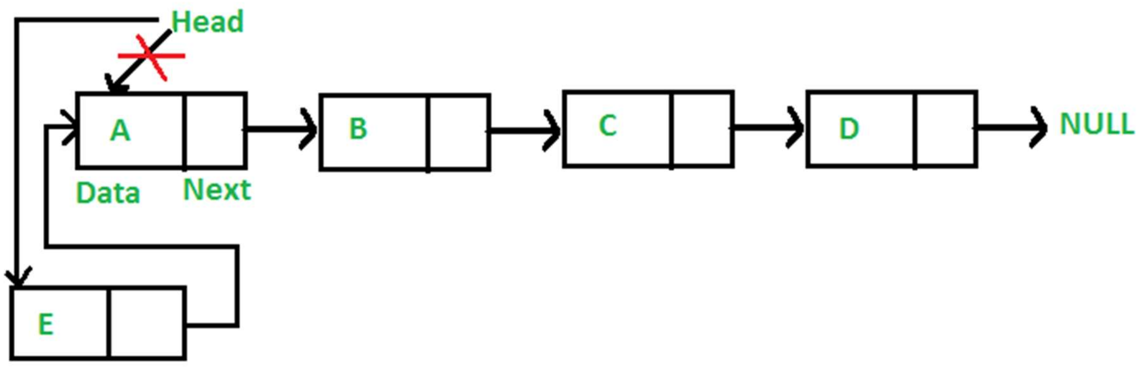
Arrays can be used to store linear data of similar types, but arrays have following limitations.
This function takes the start node and data to be inserted as arguments. New node is inserted at the end so, iterate through the list till we encounter the last node. Then, allocate memory for the new node and put data in it. Lastly, store the address in the next field of the new node as NULL.

1) The size of the arrays is fixed: So we must know the upper limit on the number of elements in advance. Also, generally, the allocated memory is equal to the upper limit irrespective of the usage.
2) Inserting a new element in an array of elements is expensive, because room has to be created for the new elements and to create room existing elements have to shifted.

On the other hand, since simple linked lists by themselves do not allow random access to the data or any form of efficient indexing, many basic operations—such as obtaining the last node of the list, finding a node that contains a given datum, or locating the place where a new node should be inserted—may require iterating through most or all of the list elements. The advantages and disadvantages of using linked lists are given below.

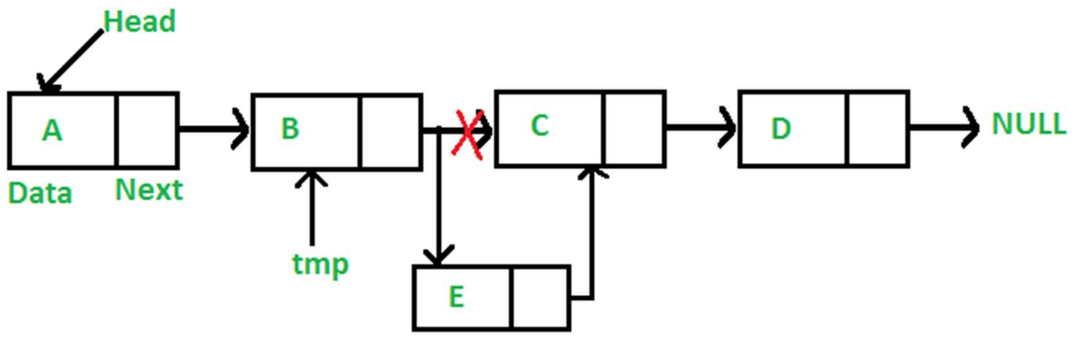
methods to insert a new node in linked list are discussed. A node can be added in three ways

1) At the front of linked list
2) After a given node
3) At end of linked list
    1) Add a node at the front:
    The new node is always added before the head of the given Linked List. And newly added node becomes the new head of the Linked List. For example, if the given Linked List is 10->15->20->25 and we add an item 5 at the front, then the Linked List becomes 5->10->15->20->25. Let us call the function that adds at the front of the list is push (). The push () must receive a pointer to the head pointer, because push must change the head pointer to point to the new node



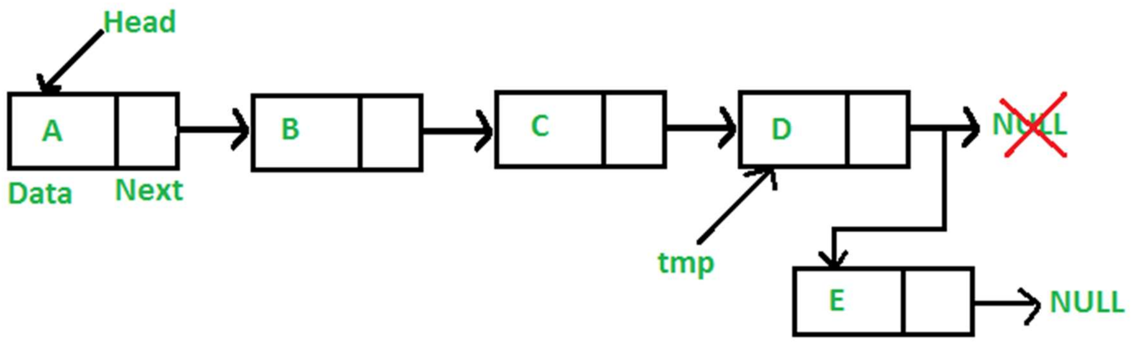
Time complexity of push () is O (1) as it does constant amount of work.

2) **Add a node after a given node**
   We are given pointer to a node, and the new node is inserted after the given node.



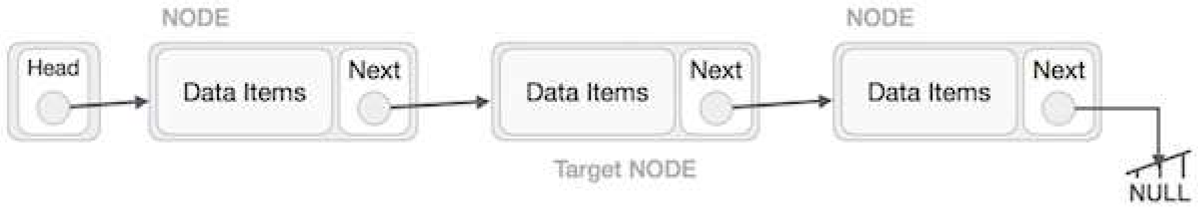
Time complexity of insert After () is O (1) as it does constant amount of work.

3) **Add a node at the end**
The new node is always added after the last node of the given Linked List. For example, if the given Linked List is 5->10->15->20->25 and we add an item 30 at the end, then the Linked List becomes 5->10->15->20->25->30.
Since a Linked List is typically represented by the head of it, we have to traverse the list till end and then change the next of last node to new node.



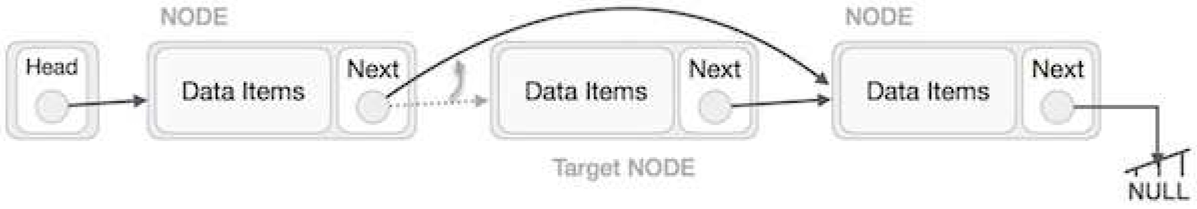
This function takes the start node (as pointer) and data to be deleted as arguments. Firstly, go to the node for which the node next to it has to be deleted, If that node points to NULL (i.e. pointer->next=NULL) then the element to be deleted is not present in the list. Else, now pointer points to a node and the node next to it has to be removed, declare a temporary node (temp) which points to the node which has to be removed. Store the address of the node next to the temporary node in the next field of the node pointer (pointer->next = temp->next). Thus, by breaking the link we removed the node which is next to the pointer (which is also temp). Because we deleted the node, we no longer require the memory used for it, free() will deallocate the memory.

Deletion is also a more than one step process. We shall learn with pictorial representation. First, locate the target node to be removed, by using searching algorithms.



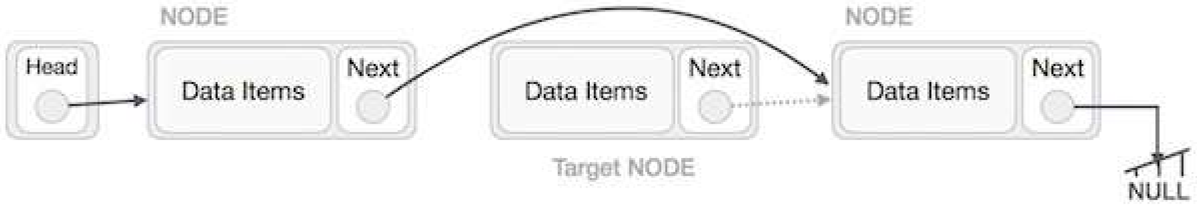
The left (previous) node of the target node now should point to the next node of the target node −

```
LeftNode.next −> TargetNode.next;
```

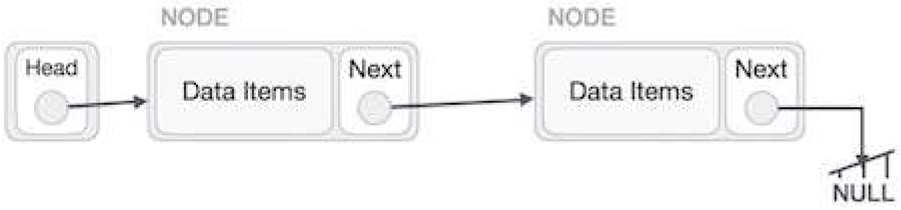


This will remove the link that was pointing to the target node. Now, using the following code, we will remove what the target node is pointing at.

```
TargetNode.next −> NULL;
```



We need to use the deleted node. We can keep that in memory otherwise we can simply deallocate memory and wipe off the target node completely.



A linked list is a set of dynamically allocated nodes, arranged in such a way that each node contains one value and one pointer. The pointer always points to the next member of the list. If the pointer is NULL, then it is the last node in the list.

This function takes the start node (as pointer) and data value of the node (key) to be found as arguments. First node is dummy node so, start with the second node. Iterate through the entire linked list and search for the key. Until next field of the pointer is equal to NULL, check if pointer->data = key. If it is then the key is

found else, move to the next node and search (pointer = pointer -> next). If key is not found return 0, else return 1.
Print - function takes the start node (as pointer) as an argument. If pointer = NULL, then there is no element in the list. Else, print the data value of the node (pointer->data) and move to the next node by recursively calling the print function with pointer->next sent as an argument.

A linked list is held using a local pointer variable which points to the first item of the list. If that pointer is also NULL, then the list is considered to be empty.

How to search element in linked list

1. Input element to search from user. Store it in some variable say keyToSearch.
2. Declare two variable one to store index of found element and other to iterate through list. Say index = 0; and struct node *curNode = head;
3. If curNode is not NULL and its data is not equal to keyToSearch. Then, increment index and move curNode to its next node.
4. Repeat step 3 till curNode! = NULL and element is not found, otherwise move to 5th step.
5. If curNode is not NULL, then element is found hence return index otherwise -1.

4. FUNCTIONAL REQUIREMENTS

The project consists of a basic part and an advanced part. All groups are required to first solve the assignments in the basic part. When a group has completed the basic part, they will have an initial search engine. After the basic part the group moves to the advanced part. Here, the group can continue in several directions, depending on interests and skills. A non-exhaustive list of suggestions for advanced assignments is given below.

The following sections is a detailed description of the project consisting of the following:

- Data Files A description of the format of the data files used for the search engine.
- The Program Index1 A basic and very simple search engine which is used as the starting point for the project.
- The Basic Part Description of the 4 assignments in the basic part.
- The Advanced Part The (non-exhaustive) list of suggestions for assignments in the advanced part.
- The Report Requirements and suggestions for the contents of the report.

5. CODE

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<malloc.h>
#include<limits.h>
#include<stdbool.h>
#include<Stdlib.h>
struct node {
   char *data;
   struct node* next;
   struct node* prev;
        char *key;
}*head,*last;
void insert(char* data)
{
   struct node * newNode;
   if(last==NULL)
   {
        head=(struct node *)malloc(sizeof(struct node));
      head->data=data;
```

```
        head->prev=NULL;
        head->next=NULL;
        last=head;
    }
    else
    {
        newNode = (struct node *)malloc(sizeof(struct node));
        newNode->data = data;
        newNode->prev = last;
        newNode->next = NULL;
        last->next = newNode;
        last = newNode;
    }
}
int search(char *item)
{
        int flag=0;
        struct node* temp=head;
        if(temp==NULL)
           printf("Empty");
        else
        {
          while(temp!=NULL)
          {
                    if(strcmp(temp->data,item)==0)
                    {
                              flag=1;
                              break;
                    }
                    temp=temp->next;
                    }
                    }
                    return flag;
}

void display()
{
          struct node* temp;
    if(head==NULL)
    {
          printf("empty");
          }
```

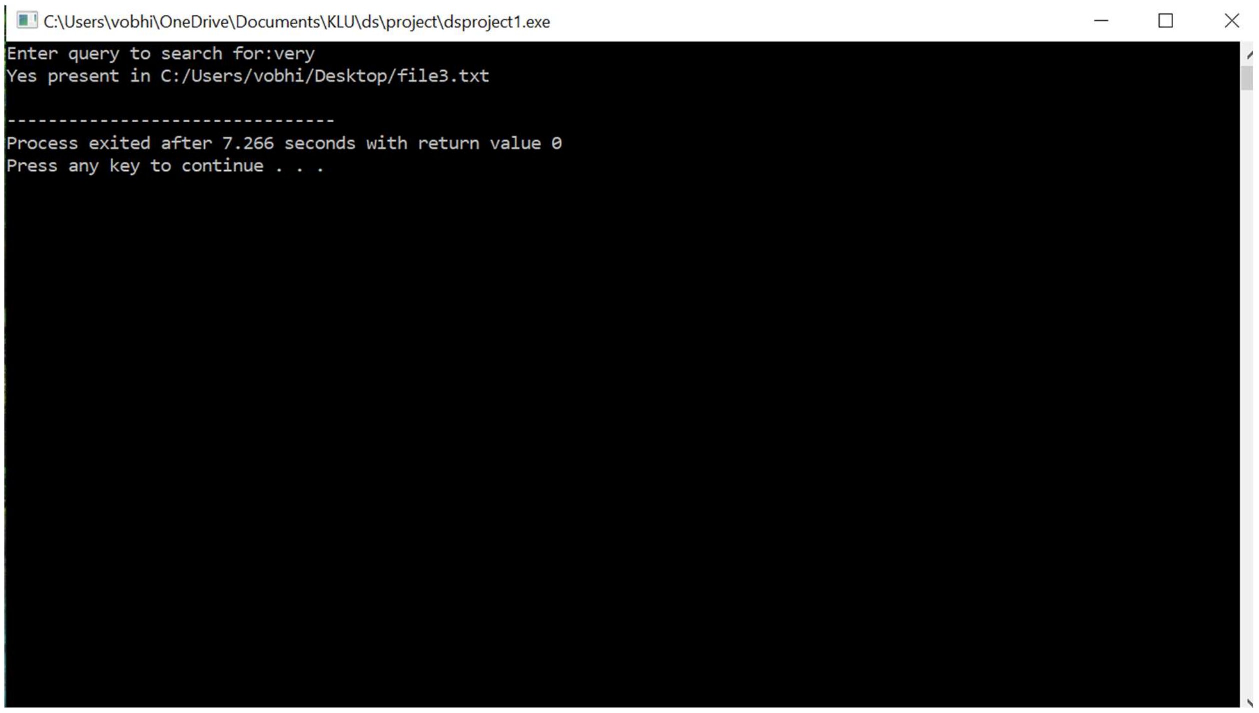
```
   else
   {
        temp=head;
        while(temp!=NULL)
        {
              printf("%s\t",temp->data);
              temp=temp->next;
      }
   }
}

int main()
{
        char query[1000];
        int i,h=0;
        char s[3][100]={"C:/Users/vobhi/Desktop/file1.txt",
"C:/Users/vobhi/Desktop/file2.txt",
"C:/Users/vobhi/Desktop/file3.txt"};
        printf("Enter query to search for:");
        scanf("%s",query);
        for(i=0;i<3;i++)
```

```
{
    {
            struct node *head=NULL;
            struct node *last=NULL;
            char line[10000];
            FILE *fp;
            fp=fopen(s[i],"r");
            char* token;
while(fgets(line,sizeof(line),fp))
{
token=strtok(line,"!,.- ");
while (token != NULL)
{
            if(token!=" ")
           insert(token);
            token=strtok(NULL,"!,.- ");
}
            }
            //display();
            int j=search(query);
            if(j==1)
```

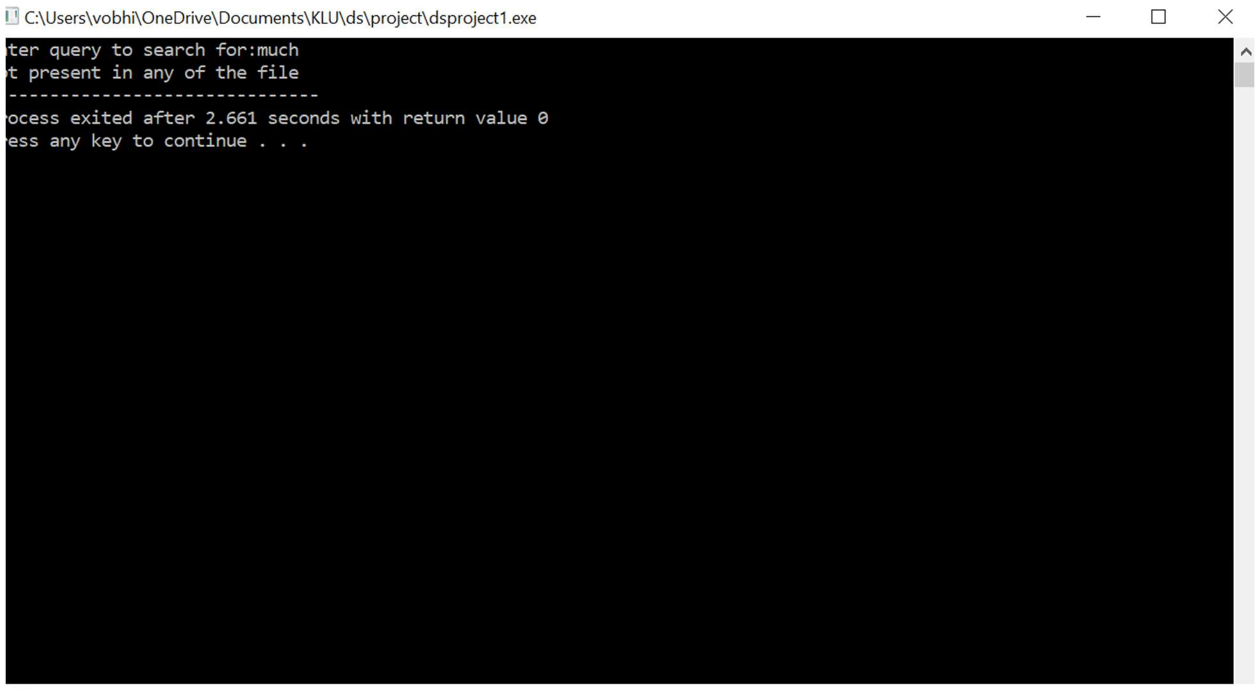
```
                    {
                       if(h==1)
                            printf(", %s",s[i]);
                       else
                       {
                            printf("Yes present in %s\n",s[i]);
                            h=1;
                    }
          }
    }
          else if(i==2&&h==0)
          {
             printf("Not present in any of the file");
             //exit(0);
          }
            fclose(fp);
            }
       return 0;
    }
```

OUTPUT:

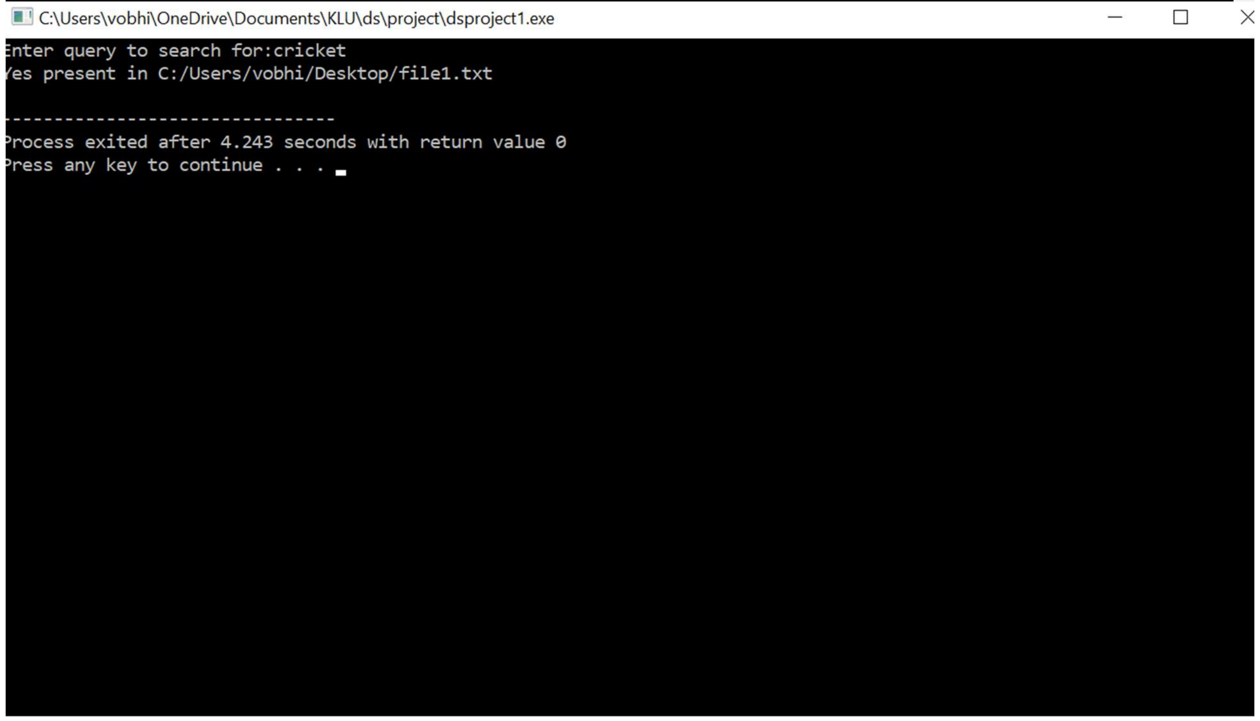
SCREEN SHOTS:


C:\Users\vobhi\OneDrive\Documents\KLU\ds\project\dsproject1.exe

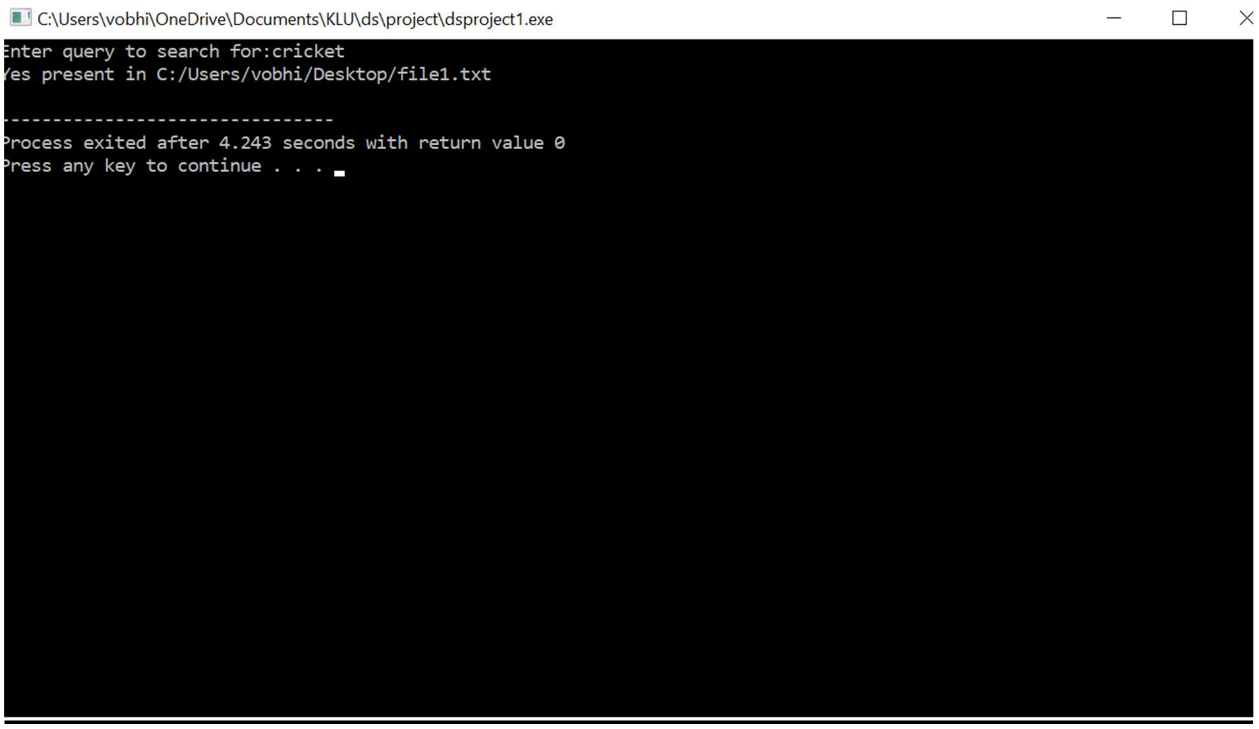
```
Enter query to search for:very
Yes present in C:/Users/vobhi/Desktop/file3.txt

--------------------------------
Process exited after 7.266 seconds with return value 0
Press any key to continue . . .
```


C:\Users\vobhi\OneDrive\Documents\KLU\ds\project\dsproject1.exe

```
ter query to search for:much
t present in any of the file
------------------------------
ocess exited after 2.661 seconds with return value 0
ess any key to continue . . .
```


C:\Users\vobhi\OneDrive\Documents\KLU\ds\project\dsproject1.exe

```
nter query to search for:cricket
es present in C:/Users/vobhi/Desktop/file1.txt

--------------------------------
rocess exited after 4.243 seconds with return value 0
ress any key to continue . . .
```



C:\Users\vobhi\OneDrive\Documents\KLU\ds\project\dsproject1.exe

```
nter query to search for:cricket
es present in C:/Users/vobhi/Desktop/file1.txt

--------------------------------
rocess exited after 4.243 seconds with return value 0
ress any key to continue . . .
```

7. Conclusion

We have done the code for “MINI SEARCH ENGINE” for a set of documents with implementation.