RAMAIAH UNIVERSITY OF APPLIED SCIENCES

ASSIGNMENT

Course Code	CSC202A
Course Name	Data structure and Algorithms
Programme	B. Tech
Department	Computer Science & Engineering
Faculty	Faculty of Engineering Technology

Name of the Student	SUBHENDU MAJI
Reg. No	18ETCS002121
Semester/Year	3RD / 2019
Course Leader/s	Vaishali R Kulkarni

Declaration Sheet			
Student Name	SUBHENDU MAJI		
Reg. No	18ETCS002121		
Programme	B. Tech	Semester/Year	3rd / 2019
Course Code	CSC202A		
Course Title	Data structure and Algorithms		
Course Date		To	
Course Leader	Vaishali R Kulkarni, Dr Pushphavathi T P, G. Roopa		

Declaration

The assignment submitted herewith is a result of my own investigations and that I have conformed to the guidelines against plagiarism as laid out in the Student Handbook. All sections of the text and results, which have been obtained from other sources, are fully referenced. I understand that cheating and plagiarism constitute a breach of University regulations and will be dealt with accordingly.

Signature of the Student		Date	
Submission date stamp (by Examination & Assessment Section)			

Signature of the Course Leader and date	Signature of the Reviewer and date

Contents

Faculty of Engineering and Technology			
Ramaiah University of Applied Sciences			
Department	Computer Science and Engineering	Programme	B. Tech
Semester/Batch	03/2019		
Course Code	CSC202A	Course Title	Data structure and Algorithms
Course Leader	Vaishali R Kulkarni, Dr Pushphavathi T P, G. Roopa		

Assignment - 01			
Register No	**18ETCS002121**	Name of Student	**SUBHENDU MAJI**

Sections		Marking Scheme	Marks		
			Max. Marks	First Examiner Marks	Moderator Marks
Part A					
	A 1.1	Static memory allocation	01		
	A 1.2	Dynamic memory allocation	02		
	A 1.3	Comparative analysis	02		
		Part-A Max Marks	**05**		
Part B 1					
	B 1.1 **1**	Classful addressing in networks	02		
	B 1.2	Data structures used	04		
	B 1.3	Validated C Program	04		
		Part-B 1 Max Marks	**10**		
Part B 2					
	B2.1	Plagiarism rules and threshold	02		
	B2.2	Pseudocode for checking plagiarized content	04		
	B2.3	Validated C Program	04		
		Part-B 2 Max Marks	**10**		

Course Marks Tabulation

Component- CET B Assignment	**First Examiner**	**Remarks**	**Second Examiner**	**Remarks**
A				
B.1				
B.2				
Marks (Max 25)				

Signature of First Examiner   Signature of Second Examiner

Solution to Question No. 1:

A1.1 Static memory allocation

A1.1.1 Intro:

Static memory allocation is an allocation technique which allocates a fixed amount of memory during compile time and the operating system internally uses a data structure known as Stack to manage this.

In Static Memory Allocation the memory for your data is allocated when the program starts. The size is fixed when the program is created. It applies to global variables, file scope variables, and variables qualified with static defined inside functions. This memory allocation is fixed and cannot be changed, i.e. increased or decreased after allocation. So, exact memory requirements must be known in advance.

A1.1.2 Key features:

- Variables get allocated permanently
- Allocation is done before program execution
- It uses the data structure called stack for implementing static allocation
- Less efficient
- There is no memory reusability

A1.1.3 Example

All the variables in the program below are statically allocated.

```
void fun()
{
   int a;
}
int main()
{
   int b;
   int c[10];
   return 1;
}
```

In this type of allocation, you strictly allocate memory for your data at compile time. This is also called simple memory allocation. It is mostly used and very easy to application.

A1.1.4 Deletion of allocated memory

Deletion of memory allocated to a program is as important as allocation otherwise it results in memory leakage. Statically allocated memory is automatically released on the basis of scope, i.e., as soon as the scope of the variable is over, memory allocated get freed.

A1.1.5 Advantages of Static memory allocation

- Simplicity of usage.
- Efficient execution time.
- Need not worry about memory allocation/re-allocation/freeing of memory
- Variables remain permanently allocated.

A1.1.6 Disadvantages of Static memory allocation

- Main disadvantage is wastage of memory.
- Memory can't be freed when it is no longer needed.

A1.2 Dynamic memory allocation

A1.2.1 Intro

C language requires the number of elements in an array to be specified at compile time. But we may not be able to do so always. Our initial judgement of size, if it is wrong, may cause failure of the program or wastage of memory space. The process of allocating memory at run time is known as **dynamic memory allocation**.

Although C does not inherently have this facility, there are four library routines known as "memory management functions" that can be used for allocating and freeing memory during program execution. These functions help us build complex application programs that use the available memory intelligently.

Function and their task

- **malloc** : Allocates request size of bytes and returns a pointer to the first byte of the allocated space.
- **calloc** : Allocates space for an array of elements, initializes them to zero and then returns a pointer to the memory.
- **free** : Frees previously allocated space.
- **realloc**: Modifies the size of previously allocated space

A1.2.2 Allocating a block of memory: malloc

A block of memory may be allocated using the function **malloc**. The **malloc** function reserves a block of memory of specified size and returns a pointer of type **void**. This means that we can assign it to any type of pointer. It takes the following form:

```
ptr = (cast-type *) malloc(byte-size)
```

ptr is a pointer of type *cast-type*.

Example,

```
x = (int *) malloc (100 *sizeof(int));
```

On successful execution of this statement, a memory space equivalent to "100 times the size of an int" bytes is reserved and the address of the first byte of the memory allocated is assigned to the pointer x of type of int.

A1.2.3 Allocating multiple blocks of memory: calloc

calloc is another memory allocation function that is normally used for requesting memory space at run time for storing derived data types such as arrrays and structures. while **malloc** allocates a single block of storage space, **calloc** allocates multiple blocks of storage, each of the same size, and then sets all bytes to zero. The general form of **calloc** is:

```
ptr = (cast-type *) calloc (n, elem-size);
```

A1.2.4 Releasing the used space: free

Dynamically allocated memory created with either **calloc()** or **malloc()** doesn't get freed on their own. You must explicitly use **free()** to release the space. Syntax:

```
free (ptr);
```

This statement frees the space allocated in the memory pointed by ptr.

A1.2.5 Altering the size of a block: realloc

The C library function void **realloc(void * ptr, size_t size)** attempts to resize the memory block pointed to by ptr that was previously allocated with a call to malloc or calloc. Syntax:

```
void *realloc(void *ptr, size_t size)
```

A1.2.6 Advantages of Dynamic memory allocation

- Data structures can grow and shrink according to the requirement.
    - We can allocate (create) additional storage whenever we need them.
    - We can de-allocate (free/delete) dynamic space whenever we are done with them.
- Dynamic Allocation is done at run time.

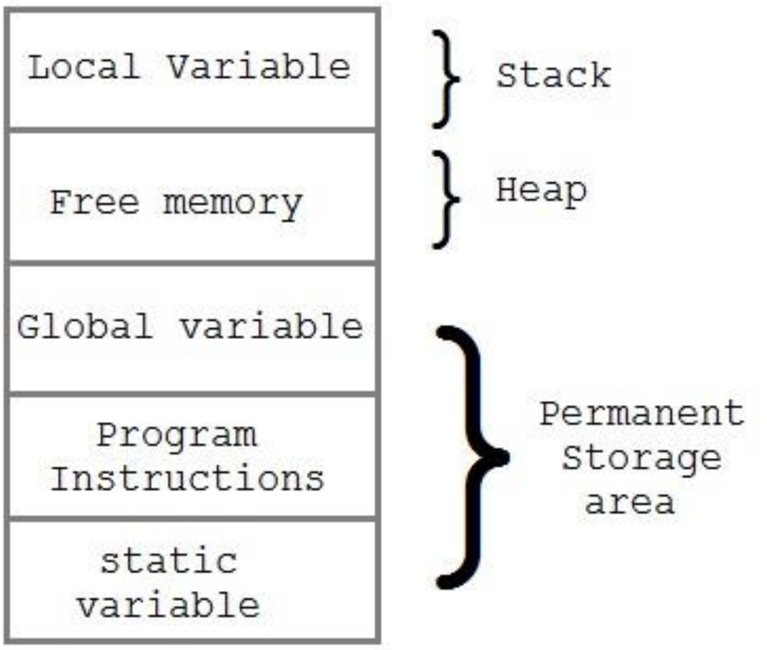
A1.2.7 Disadvantages of Dynamic memory allocation

- As the memory is allocated during runtime, it requires more time.
- Memory needs to be freed by the user when done. This is important as it is more likely to turn into bugs that are difficult to find.

A1.3 Comparative analysis

Stack memory is allocated during compilation time execution. This is known as static memory allocation.

Whereas, **heap memory** is allocated at run-time compilation. This is known as dynamic memory allocation.

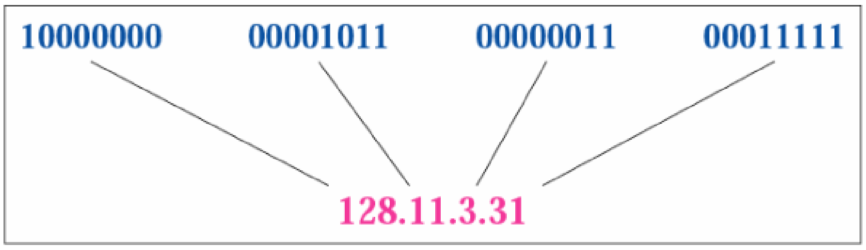


STATIC MEMORY ALLOCATION	DYNAMIC MEMORY ALLOCATION
• Memory is allocated before the execution of the program begins (During Compilation).	• Memory is allocated during the execution of the program.
• Variables remain permanently allocated.	• Allocated only when program unit is active.
• In this type of allocation Memory cannot be resized after the initial allocation.	• In this type of allocation Memory can be dynamically expanded and shrunk as necessary.
• Implemented using stacks.	• Implemented using heap.
• Faster execution than Dynamic.	• Slower execution than static.
• It is less efficient than Dynamic allocation strategy.	• It is more efficient than Static allocation strategy.
• Implementation of this type of allocation is simple.	• Implementation of this type of allocation is complicated.
• Memory cannot be reuse when it is no longer needed.	• Memory can be freed when it is no longer needed & reuse or reallocate during execution.

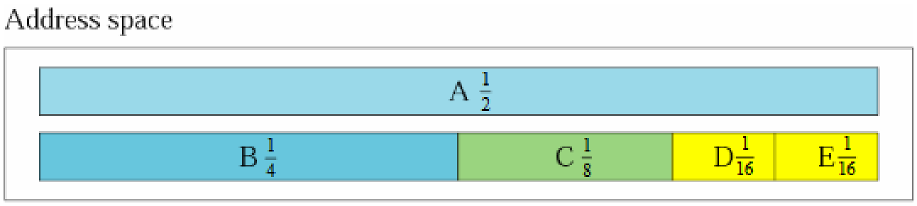
Solution to Question No. 2:

B1.1 Classful addressing in networks

An IP address is a 32-bit address that identifies a connection to the Internet.
IPv4 addresses 32 bit binary addresses (divided into 4 octets) used by the Internet Protocol (OSI Layer 3) for delivering packet to a device located in same or remote network.
The IP addresses are universally unique.
The address space of IPv4 is 2^{32} or 4,294,967,296.
IP address is written as a Binary (hexadecimal) or a Dotted-Decimal (w/out leading zeros) notation.



The IP address space (all possible IP values) is divided into five classes: A, B, C, D, and E.



Class A:

"Class A" IPv4 addresses are for very large networks. The left most bit of the left most octet of a "Class A" network is reserved as "0". The first octet of a "Class A" IPv4 address is used to identify the Network and the three remaining octets are used to identify the host in that particular network (Network.Host.Host.Host).

The 32 bits of a "Class A" IPv4 address can be represented as 0xxxxxxx.xxxxxxxx.xxxxxxxx.xxxxxxxx.

The minimum possible value for the leftmost octet in binaries is 00000000 (decimal equivalent is 0) and the maximum possible value for the leftmost octet is 01111111 (decimal equivalent is 127). Therefore for a "Class A" IPv4 address, leftmost octet must have a value between 0-127 (0.X.X.X to 127.X.X.X).

The network 127.0.0.0 is known as loopback network. The IPv4 address 127.0.0.1 is used by the host computer to send a message back to itself. It is commonly used for troubleshooting and network testing.

Computers not connected directly to the Internet need not have globally-unique IPv4 addresses. They need an IPv4 addresses unique to that network only. 10.0.0.0 network belongs to "Class A" is reserved for private use and can be used inside any organization.

Class B:

"Class B" IPv4 addresses are used for medium-sized networks. Two left most bits of the left most octet of a "Class B" network is reserved as "10". The first two octets of a "Class B" IPv4 address is used to identify the Network and the remaining two octets are used to identify the host in that particular network (Network.Network.Host.Host).

The 32 bits of a "Class B" IPv4 address can be represented as 10xxxxxx.xxxxxxxx.xxxxxxxx.xxxxxxxx.

The minimum possible value for the leftmost octet in binaries is 10000000 (decimal equivalent is 128) and the maximum possible value for the leftmost octet is 10111111 (decimal equivalent is 191). Therefore for a "Class B" IPv4 address, leftmost octet must have a value between 128-191 (128.X.X.X to 191.X.X.X).

Network 169.254.0.0 is known as APIPA (Automatic Private IPv4 addresses). APIPA range of IPv4 addresses are used when a client is configured to automatically obtain an IPv4 address from the DHCP server was unable to contact the DHCP server for dynamic IPv4 address.

Networks starting from 172.16.0.0 to 172.31.0.0 are reserved for private use.

Class C:

"Class C" IPv4 addresses are commonly used for small to mid-size businesses. Three left most bits of the left most octet of a "Class C" network is reserved as "110". The first three octets of a "Class C" IPv4 address is used to identify the Network and the remaining one octet is used to identify the host in that particular network (Network.Network.Networkt.Host).

The 32 bits of a "Class C" IPv4 address can be represented as 110xxxxx.xxxxxxxx.xxxxxxxx.xxxxxxxx.

The minimum possible value for the leftmost octet in binaries is 11000000 (decimal equivalent is 192) and the maximum possible value for the leftmost octet is 11011111 (decimal equivalent is 223). Therefore for a "Class C" IPv4 address, leftmost octet must have a value between 192-223 (192.X.X.X to 223.X.X.X).

Networks starting from 192.168.0.0 to 192.168.255.0 are reserved for private use.

Class D:

Class D IPv4 addresses are known as multicast IPv4 addresses. Multicasting is a technique developed to send packets from one device to many other devices, without any unnecessary packet duplication. In multicasting, one packet is sent from a source and is replicated as needed in the network to reach as many end-users as necessary. You cannot assign these IPv4 addresses to your devices.

Four left most bits of the left most octet of a "Class D" network is reserved as "1110". The other 28 bits are used to identify the group of computers the multicast message is intended for.

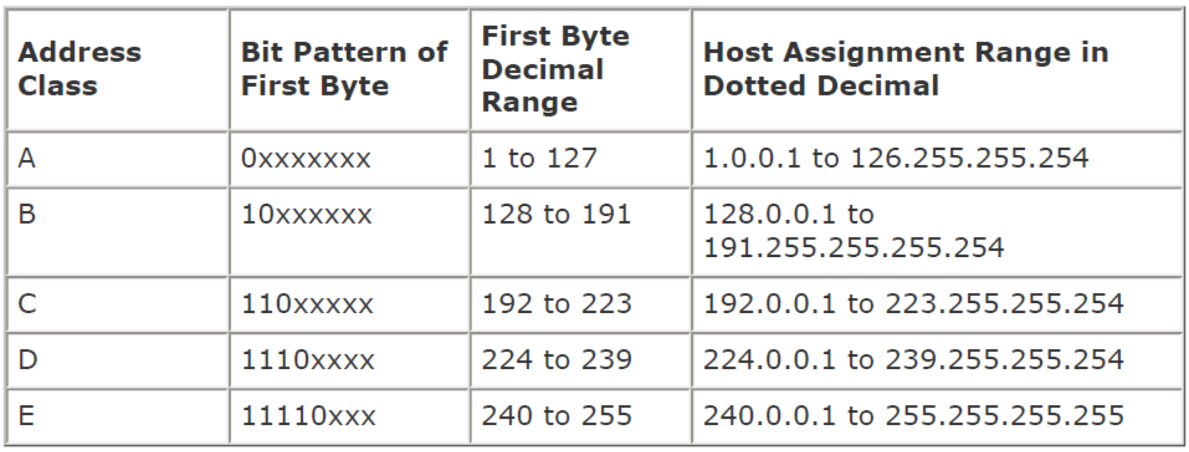
The minimum possible value for the left most octet in binaries is 11100000 (decimal equivalent is 224) and the maximum possible value for the leftmost octet is 11101111 (decimal equivalent is 239). Therefore for a "Class D" IPv4 address, leftmost octet must have a value between 224-239 (224.X.X.X to 239.X.X.X).

Class E:

Class E is used for experimental purposes only and you cannot assign these IPv4 addresses to your devices.

Four left most bits of the left most octet of a "Class E" network is reserved as "1111".

The minimum possible value for the left most octet in binaries is 11110000 (decimal equivalent is 240) and the maximum possible value for the leftmost octet is 11111111 (decimal equivalent is 255). Therefore for a "Class E" IPv4 address, leftmost octet must have a value between 240-255 (240.X.X.X to 255.X.X.X).



Address Class	Bit Pattern of First Byte	First Byte Decimal Range	Host Assignment Range in Dotted Decimal
A	0xxxxxxx	1 to 127	1.0.0.1 to 126.255.255.254
B	10xxxxxx	128 to 191	128.0.0.1 to 191.255.255.255.254
C	110xxxxx	192 to 223	192.0.0.1 to 223.255.255.254
D	1110xxxx	224 to 239	224.0.0.1 to 239.255.255.254
E	11110xxx	240 to 255	240.0.0.1 to 255.255.255.255

Problems with Classful Addressing:

The problem with this classful addressing method is that millions of class A address are wasted, many of the class B address are wasted, whereas, number of addresses available in class C is so small that it cannot cater the needs of organizations. Class D addresses are used for multicast routing and are therefore available as a single block only. Class E addresses are reserved.

Since there are these problems, Classful networking was replaced by Classless Inter-Domain Routing (CIDR) in 1993.

B1.2 Data Structures Used

Data Structure can be defined as the group of data elements which provides an efficient way of storing and organising data in the computer so that it can be used efficiently. Some examples of Data Structures are arrays, Linked List, Stack, Queue, etc.

Data Structures are the main part of many computer science algorithms as they enable the programmers to handle the data in an efficient way. It plays a vital role in enhancing the performance of a software or a program as the main function of the software is to store and retrieve the user's data as fast as possible

The only data-structure used in the program is **Arrays.**

An **array** is defined as the collection of similar type of data items stored at contiguous memory locations. Arrays are the derived data type in C programming language which can store the primitive type of data such as int, char, double, float, etc. It also has the capability to store the collection of derived data types, such as pointers, structure, etc. The array is the simplest data structure where each data element can be randomly accessed by using its index number.

Advantages of C Array

- **Code Optimization**: Less code to the access the data.
- **Ease of traversing**: By using the for loop, we can retrieve the elements of an array easily.
- **Ease of sorting**: To sort the elements of the array, we need a few lines of code only.
- **Random Access**: We can access any element randomly using the array.

We can declare an array in the C language in the following way.

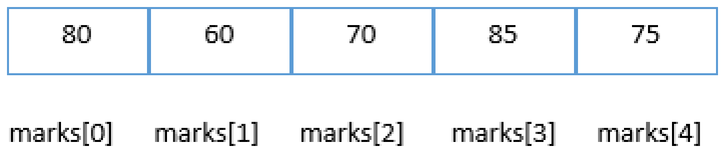
```
data_type array_name[array_size]
```

For Eg. , if we want to store the marks of a student in 6 subjects, then we don't need to define different variables for the marks in the different subject. Instead of that, we can define an array which can store the marks in each subject at the contiguous memory locations.

```
int marks[5];
```

The simplest way to initialize an array is by using the index of each element. We can initialize each element of the array by using the index.

```
marks[1]=60;
marks[2]=70;
marks[3]=85;
marks[4]=75;
```



B1.3 Validated C Program

Code Snippet:

```c
/* header files */
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define MAX 50 // no. of inputs user want to give

struct systemIP
{
    char host_name[20];
    char ip[20];
};

struct systemIP S[MAX]; // declaring variables of struct systemIP
int octet_1[MAX];

/* function protypes */
void get_data();
void print_data();
void get_octet();
char class_select(int);
void print_by_host_name();
void print_data_class();
```

```c
void main(int argc, char **argv) // main body
{

    while (1)
    {
        int ch;
        printf("<=============================================================>");
        printf("\nMAIN MENU\n 1. Input data\n 2. Print Data\n 3. Print Class of IP
by Host-Name\n 4. Print classes of all data\n 5. Exit ");
        printf("\nEnter your choice: ");
        scanf("%d", &ch);
        char temp;
        scanf("%c", &temp);
        switch (ch)
        {
        case 1:
            get_data();
            printf("\nDATA SUCCESSFULLY ENTERED... ");
            break;
        case 2:
            print_data();
            break;
        case 3:
            print_by_host_name();
            break;
        case 4:
            printf("\nPRINTING .....\n");
            for (int i = 0; i < MAX; i++)
            {
                print_data_class(i);
            }
            break;
        case 5:
            exit(1);
            break;
        default:
            printf("\nWrong Choice!!! ");
            break;
        }
    }
}
```

```
void get_data() // For Inputing data
{
    for (int i = 0; i < MAX; i++)
    {
        printf("Enter host name[%d] : ", i + 1);
        gets(S[i].host_name);
        label:
        printf("Enter IP address[%d]:  ", i + 1);
        gets(S[i].ip);
        get_octet();
        if (octet_1[i] > 255)
        {
            printf("Octet value can't be more than 255. PLEASE ENTER AGAIN !!!\n");
            goto label; // for asking IP
address again & again, if user inputs a value more than 255
        }
    }
}
```

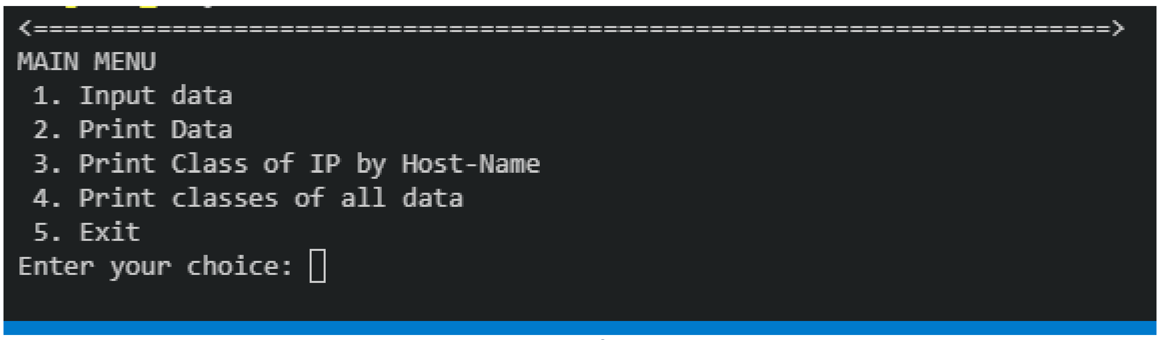
```c
void print_data() // For printing the inputed data
{
    for (int i = 0; i < MAX; i++)
    {
        printf("Host name[%d] : ", i + 1);
        puts(S[i].host_name);
        printf("IP address[%d]:  ", i + 1);
        puts(S[i].ip);
    }
}
```

```c
void get_octet() // For getting the first octet of the IP Address
{
    for (int i = 0; i < MAX; i++)
    {
        // atoi converts string to integer
        int int_octet = atoi(S[i].ip);
        octet_1[i] = int_octet;
    }
}
```

```c
char class_select(int oct) // For Getting the class of an IP Address
{
    char cl;
    if (oct >= 1 && oct <= 126)
    {
        cl = 'A';
    }
    else if (oct >= 127 && oct <= 191)
    {
        cl = 'B';
    }
    else if (oct >= 192 && oct <= 223)
    {
        cl = 'C';
    }
    else if (oct >= 224 && oct <= 239)
    {
        cl = 'D';
    }
    else if (oct >= 240 && oct <= 254)
    {
        cl = 'E';
    }
    return cl;
}
```

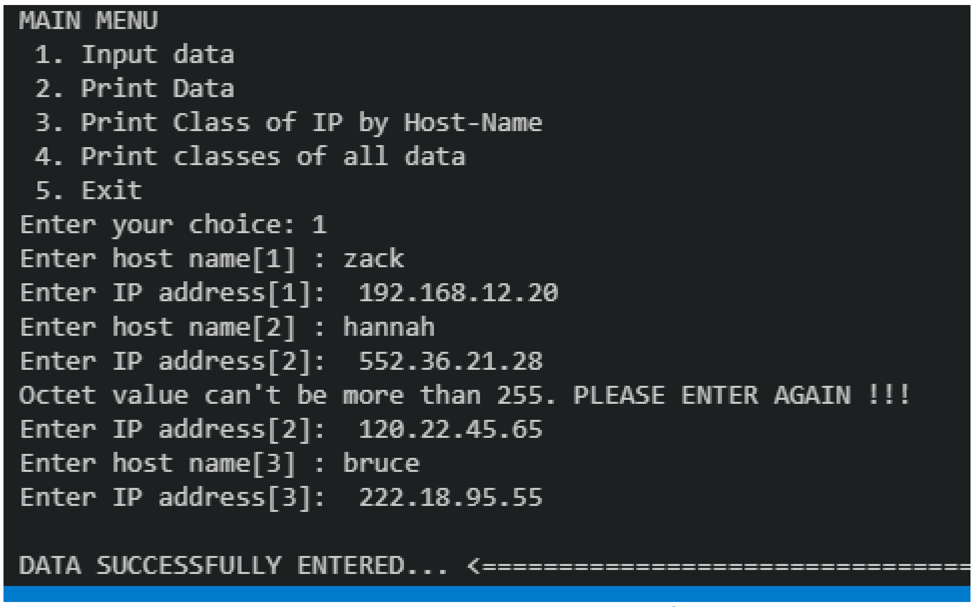
```c
void print_by_host_name() // Searching a data by host-name (Linear Search)
{
    char srch_hstnm[20];
    printf("Enter host name: ");
    gets(srch_hstnm);
    int pos, flag = 0;
    for (int i = 0; i < MAX; i++)
    {
        if (!(strcmp(srch_hstnm, S[i].host_name)))
        {
            flag = 1;
            pos = i;
            break;
        }
    }
    if (flag == 0)
    {
        printf("Host-Name Not Found!!!\n");
    }
    else
    {
        printf("FOUND..!!");
        print_data_class(pos);
    }
}
```

Screenshot of Output:



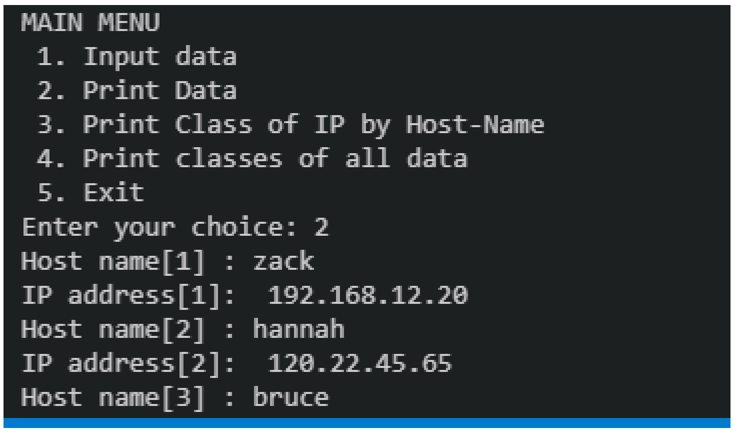
```
<========================================================================>
MAIN MENU
 1. Input data
 2. Print Data
 3. Print Class of IP by Host-Name
 4. Print classes of all data
 5. Exit
Enter your choice: 
```

Figure 1 Menu of the program



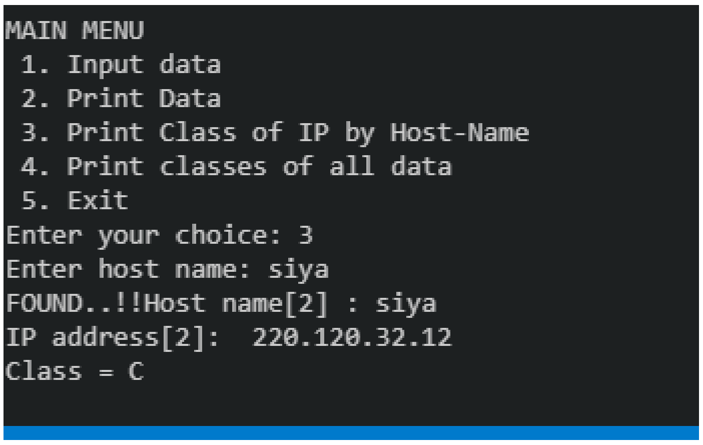
```
MAIN MENU
 1. Input data
 2. Print Data
 3. Print Class of IP by Host-Name
 4. Print classes of all data
 5. Exit
Enter your choice: 1
Enter host name[1] : zack
Enter IP address[1]:  192.168.12.20
Enter host name[2] : hannah
Enter IP address[2]:  552.36.21.28
Octet value can't be more than 255. PLEASE ENTER AGAIN !!!
Enter IP address[2]:  120.22.45.65
Enter host name[3] : bruce
Enter IP address[3]:  222.18.95.55

DATA SUCCESSFULLY ENTERED... <================================
```

Figure 2 Inputting data using structures also checking if the value is <255



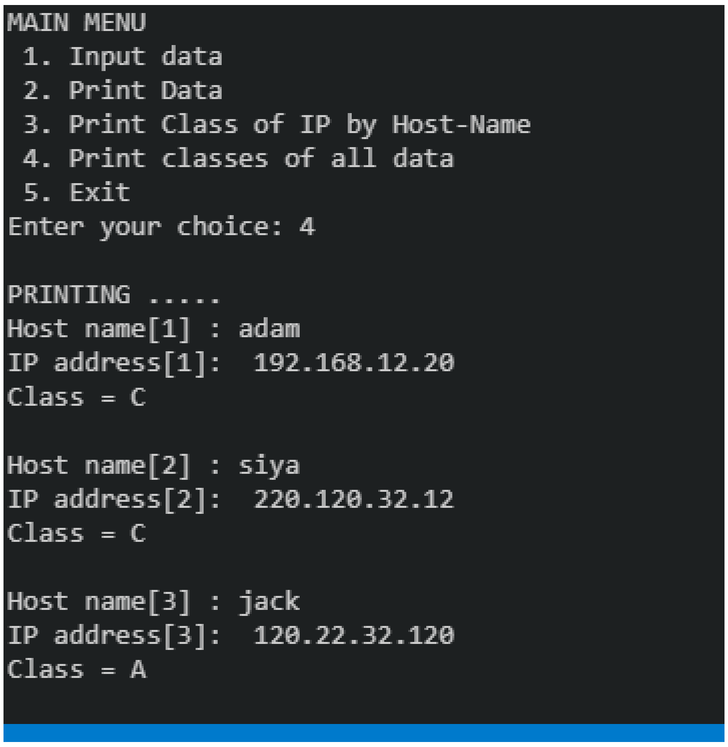
```
MAIN MENU
 1. Input data
 2. Print Data
 3. Print Class of IP by Host-Name
 4. Print classes of all data
 5. Exit
Enter your choice: 2
Host name[1] : zack
IP address[1]:  192.168.12.20
Host name[2] : hannah
IP address[2]:  120.22.45.65
Host name[3] : bruce
```

Figure 3 Printing the stored data from structures



```
MAIN MENU
 1. Input data
 2. Print Data
 3. Print Class of IP by Host-Name
 4. Print classes of all data
 5. Exit
Enter your choice: 3
Enter host name: siya
FOUND..!!Host name[2] : siya
IP address[2]:  220.120.32.12
Class = C
```

Figure 4 searching a data from its host-name



```
MAIN MENU
 1. Input data
 2. Print Data
 3. Print Class of IP by Host-Name
 4. Print classes of all data
 5. Exit
Enter your choice: 4

PRINTING .....
Host name[1] : adam
IP address[1]:  192.168.12.20
Class = C

Host name[2] : siya
IP address[2]:  220.120.32.12
Class = C

Host name[3] : jack
IP address[3]:  120.22.32.120
Class = A
```

Figure 5 printing all the data with its class

Solution to Question No. 3:

B2.1 Plagiarism rules and threshold

What is plagiarism?

- turning in someone else's work as your own
- copying words or ideas from someone else without giving credit
- failing to put a quotation in quotation marks
- giving incorrect information about the source of a quotation
- changing words but copying the sentence structure of a source without giving credit
- copying so many words or ideas from a source that it makes up the majority of your work, whether you give credit or not.

The rules and threshold of plagiarism in my program are as follows:

- If there are some words in a document that are found to be copied in the same manner then the document is plagiarized, for that I have used LCS (Longest Common Subsequence) algorithm.
- The threshold can be defined by the user, by default the program will calculate how much of the content in the document is plagiarized. It will show "*plagiarized*" if percentage is more than 30%.
- If the document is one-to-one copied, then the program will output 100% plagiarized.
- If the document contains parts of the original document, then the percentage that is copied is shown in the output along with the copied content.
- I am comparing file1 to file2, vice-versa can also be done by simply changing parameters in call function.

B2.2 Pseudocode for checking plagiarized content

```
Pseudocode:

read_files:
            1. Begin
            2. Create file pointer f1 ,f2
            3. Open file1 and file2 in read mode
            4. Read files and store as a string in data1/2
               4.1 eg: while (fgets(data1, 100, f1))
            5. Now, data1 will have the whole document as a string
            6. Close the opened files
            7. End
Process_data:
        1. Begin
        2. Call read_files() function
        3. Using strtok() splitting data1 across .(full stop)
        4. Store each sentences in ar1 as an element. (ar1 will be
           a list of string)
```

```
   5. Split further to create an array of strings(words) of
      full document, lets it be X[i]
   6. Create another list which contains nth element as list
      which contains words of nth sentence.
   7. Repeat steps from 3 to 6 for data2
   8. if flag=1
      8.1 print common sequences sentence-wise
      8.2 call percentator(percentage_calculator) function

   9. if flag=2
      9.1 print LCS(Longest common subsequence of full
 document)
      9.2 call percentaor(percentage_calculator) function
   10.    End
```

percentator:

```
 1. Begin
 2. Read num1 and num2
 3. float percent = ((float)num1 / (float)num2) * 100;
 4. print 0.3f of percent
 5. if percent>30
        5.1     print plagiarized
      else
        5.2 print plagiarized
 6. End
```

max:

```
 1. Begin
 2. Read num1 and num2
 3. return (num1 > num2) ? num1 : num2;
 4. End
```

LCS:

```
 1. Begin
 2. Initialize int L[m + 1][n + 1];
 3.  for (int i = 0; i <= m; i++)
     {
         for (int j = 0; j <= n; j++)
         {
             if (i == 0 || j == 0)
                   L[i][j] = 0;
                 else if (!(strcmp(X[i - 1], Y[j - 1])))
                    L[i][j] = L[i - 1][j - 1] + 1;
             else
                 L[i][j] = max(L[i - 1][j], L[i][j - 1]);
         }
     }
 4. length of lcs can be printed by printing L[m][n]
 5. For printing lcs:
 6. Initialize int index = L[m][n];
 7. Initialize int i = m, j = n;
 8. while (i > 0 && j > 0)
     {
```

```
                if (!(strcmp(X[i - 1], Y[j - 1])))
                {
                    strcpy(lcs[index - 1], X[i - 1]);
                    i--;
                    j--;
               index--;
          }
               else if (L[i - 1][j] > L[i][j - 1])
                    i--;

                else
                    j--;
           }
       9. print lcs[L[m][n]]
       10.  return L[m][n]
       11.  End
```

main:

```
       1. Begin
       2. Create a menu and put it in an infinite loop to print menu
          every time any choice executes (exit condition will be
          EXIT(1))
       3. read choice
       4. if user wants to print common sequences  sentence-wise ,
          then flag1=1
       5. else if user wants to print lcs of whole document , then
          flag2=1
       6. else exit(1)
       7. call process_data()
       8. reassign flag1=0, flag2=0
       9. End
```

B2.3 Validated C Program

Code Snippet:

```
/* header files */
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

/* function prototypes */
void read_files();  // For reading data from file1 and file2
void process_data(); // For processing the data and calling LCS function
void percentator(); // For calculating similarity percentage
int max(int num1, int num2); // For calculating MAX of two numbers
int LCS(char **X, char **Y, int m, int n); // For calculating Length of LCS and Pri
nting LCS.

/* global variables */
```

```c
char lcs[60][60];
char data1[100];
char data2[100];

int count = 0;

char *X[100];
int M = 0;
char *Y[100];
int N = 0;

int flag1 = 0;
int flag2 = 0;
```

```c
void main() // main body
{
    while (1) // For Printing Main-Menu repeatedly
    {
        int ch;
        printf("<==============================================================>");
        printf("\nMAIN MENU\n 1.Print Common Sequences per Sentences\n 2.
Print LCS\n 3. Exit ");
        printf("\nEnter your choice: ");
        scanf("%d", &ch);
        printf("<==============================================================>");
        if (ch == 1)
        {
            flag1 = 1;
        }
        else if (ch == 2)
        {
            flag2 = 1;
        }
        else if (ch == 3)
        {
            exit(1);
        }
        else
        {
            printf("INVALID CHOICE !!!!");
        }
        process_data();
        flag1 = 0;
        flag2 = 0;
    }
}
```

```
void read_files()   // For reading data from file1 and file2
{
    FILE *f1, *f2;
    f1 = fopen("test1.txt", "r"); // Opening file1 in reading mode
    f2 = fopen("test2.txt", "r"); // Opening file2 in reading mode

    if (f1 == NULL || f2 == NULL)
    {
        printf("\n File Failed to Open.");
    }
    else
    {
        while (fgets(data1, 100, f1)); // data1 is a string of the whole data of fi
le 1
        while (fgets(data2, 100, f2)); // data2 is a string of the whole data of fi
le 2
        fclose(f1);
        fclose(f2);
        printf("\n <== File Read Successfully ==> \n");
    }
}
```

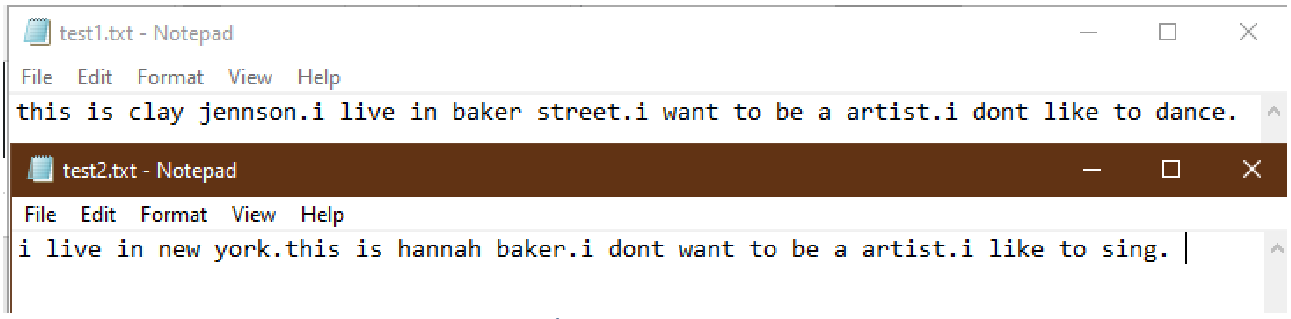
```
void process_data() // For processing the data and calling LCS function
{
    read_files(); // reading files (calling function)
    /* data processing for file 1 */
    int i = 0;
    char *ar1[10];
    int n1 = 0;
    char *p = strtok(data1, ".");
    while (p != NULL)
    {
        n1++;
        ar1[i++] = p;
        p = strtok(NULL, ".");
    }
    // for (i = 0; i < n1; i++)
    //     printf("==> %s\n", ar1[i]);

    int u = 0;
    M = 0;
    i = 0;
    int j = 0;
    char *x[n1][20];
    int m[n1];
    m[0] = 0;
    while (u < n1)
    {
        i = 0;
        char *r = strtok(ar1[u], " ");
        while (r != NULL)
        {
            M++;
            m[u] += 1;
            x[u][i] = r;
            X[j++] = r;
            i++;
            r = strtok(NULL, " ");
        }
        u++;
        m[u] = 0;
    }
    /* X[i] contains all the words of the file 1 */
    //  for (int i = 0; i <M ; i++)
    //  printf("%s\n", X[i]);

    /* x[i][j] contains all the words of each line as an element of file 1 */
    /* eg: x[3][3] = {{"this","is","pen"},{"boy","man","girl"},{"orange","yellow","
blue"}} */
    // for (i=0;i<n1;i++){
    //  for(j=0;j<m[i];j++){
    //       printf(" %s ",x[i][j]);
    //          }printf("\n");
    //      }

    /* data processing for file 2 */
    j = 0;
    char *ar2[20];
    int n2 = 0;
    char *q = strtok(data2, ".");
    while (q != NULL)
    {
        n2++;
        ar2[j++] = q;
        q = strtok(NULL, ".");
    }
    // for (i = 0; i < n2; i++)
    //     printf("==> %s\n", ar2[i]);

    u = 0;
    N = 0;
    i = 0;
    j = 0;
    char *y[n2][10];
    int n[n2];
    n[0] = 0;
    while (u < n2)
    {
        i = 0;
        char *s = strtok(ar2[u], " ");
        while (s != NULL)
        {
            N++;
            n[u] += 1;
            y[u][i] = s;
            Y[j++] = s;
            i++;
            s = strtok(NULL, " ");
        }
        u++;
        n[u] = 0;
    }
    /* Y[i] contains all the words of the file 2 */
    //  for (int i = 0; i <N ; i++)
    //  printf("%s\n", Y[i]);

    /* y[i][j] contains all the words of each line as an element of file 2 */
    /* eg:   y[3][3] = {{"this","is","pen"},{"boy","man","girl"},{"orange","yellow"
,"blue"}} */
    // for (i=0;i<n2;i++){
    //  for(j=0;j<n[i];j++){
    //          printf(" %s ",y[i][j]);
    //             }printf("\n");
    //          }
    /* PRINTING */
    if (flag1 == 1)
    {
        printf("COMMON SEQUENCES PER SENTENCES ---------\n");
        for (int i = 0; i < n1; i++)
        {
            LCS(x[i], Y, m[i], N);
            memset(lcs, '\0', sizeof lcs);
        }
        percentator(count, M);
    }
    if (flag2 == 1)
    {
        printf("\nLONGEST COMMON SUBSEQUENCE  -------- \n");
        int t = LCS(X, Y, M, N);
        percentator(t, M);
    }
    count = 0;
```

```
}
void percentator(int num1, int num2) // For calculating similarity percentage
{
    printf("\nPLAGIARISED PERCENTAGE => ");
    float percent = ((float)num1 / (float)num2) * 100;
    printf(" %0.3f %%", percent);
    if (percent > 30)
        printf("\n\t....PLAGIARISED....\n");
    else
        printf("\n\t....NOT PLAGIARISED....\n");
}
```

```
int max(int num1, int num2) // For calculating MAX of two numbers
{
    return (num1 > num2) ? num1 : num2;
}
```

```
int LCS(char **X, char **Y, int m, int n) // For calculating length of LCS and Prin
ting LCS.
{   /* for calculating length of LCS */
    int L[m + 1][n + 1];
    for (int i = 0; i <= m; i++)
    {
        for (int j = 0; j <= n; j++)
        {
            if (i == 0 || j == 0)
                L[i][j] = 0;
            else if (!(strcmp(X[i - 1], Y[j - 1])))
                L[i][j] = L[i - 1][j - 1] + 1;
            else
                L[i][j] = max(L[i - 1][j], L[i][j - 1]);
        }
    }

    // printing length of LCS
    // printf("\nLength of lcs :: %d", L[m][n]);

    /* For Printing LCS */
    int index = L[m][n];
    int i = m, j = n;
    while (i > 0 && j > 0)
    {
        if (!(strcmp(X[i - 1], Y[j - 1])))
        {
            strcpy(lcs[index - 1], X[i - 1]);
            i--;
            j--;
            index--;
        }
        else if (L[i - 1][j] > L[i][j - 1])
            i--;

        else
            j--;
    }
    // Printing the LCS
    for (i = 0; i < L[m][n]; i++)
    {
        count++;
        printf("%s\t", lcs[i]);
    }
    printf("\n");

    return L[m][n]; // returning the length of the LCS
}
```

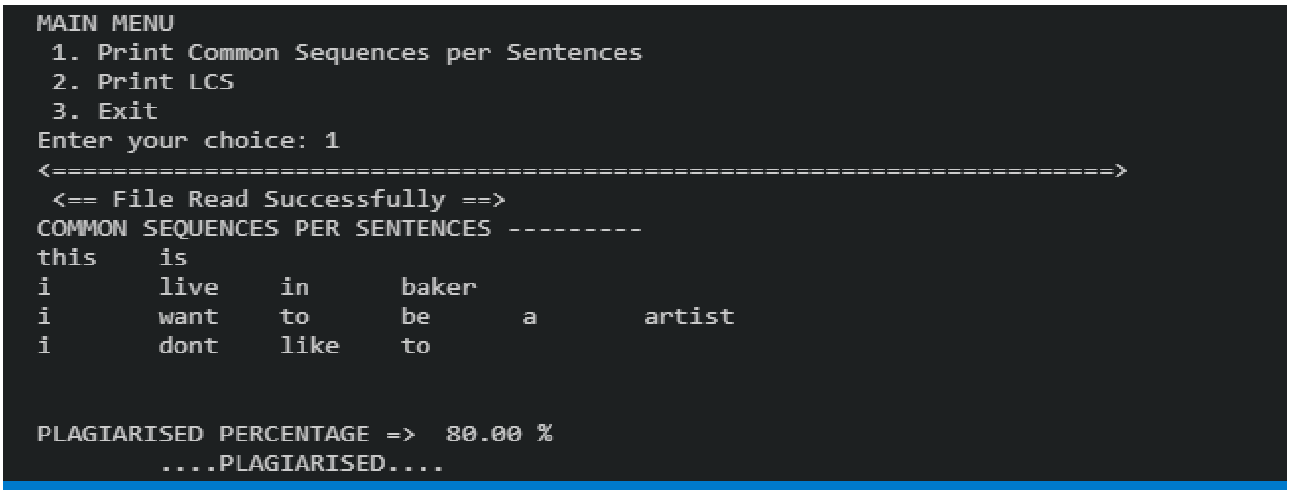
Screenshot of Output:



```
test1.txt - Notepad
File  Edit  Format  View  Help
this is clay jennson.i live in baker street.i want to be a artist.i dont like to dance.
```

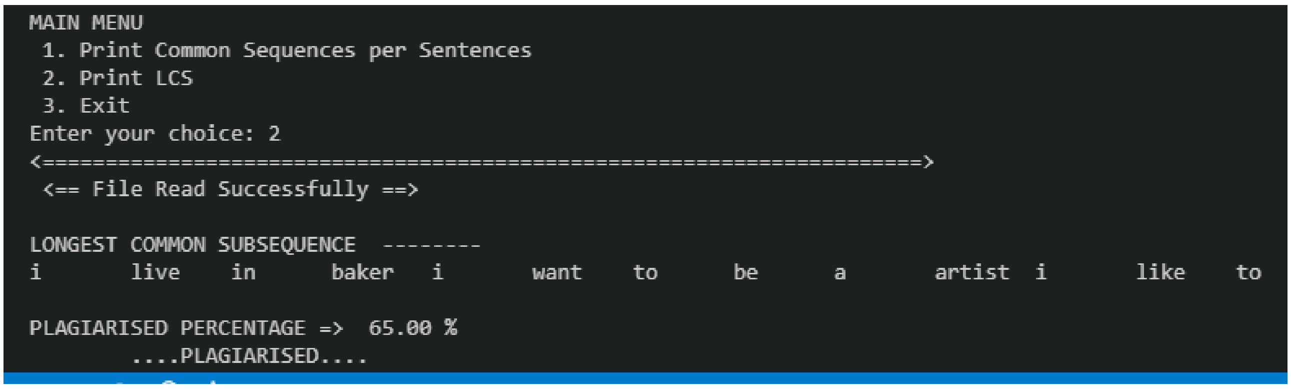
```
test2.txt - Notepad
File  Edit  Format  View  Help
i live in new york.this is hannah baker.i dont want to be a artist.i like to sing.
```

Figure 6 files content used to compare



```
MAIN MENU
 1. Print Common Sequences per Sentences
 2. Print LCS
 3. Exit
Enter your choice: 1
<======================================================================>
 <== File Read Successfully ==>
COMMON SEQUENCES PER SENTENCES ---------
this    is
i       live    in      baker
i       want    to      be      a       artist
i       dont    like    to


PLAGIARISED PERCENTAGE =>  80.00 %
        ....PLAGIARISED....
```

Figure 7 printing common words sentence-wise



```
MAIN MENU
 1. Print Common Sequences per Sentences
 2. Print LCS
 3. Exit
Enter your choice: 2
<========================================================================>
 <== File Read Successfully ==>

LONGEST COMMON SUBSEQUENCE  --------
i       live    in      baker   i       want    to      be      a       artist  i       like    to

PLAGIARISED PERCENTAGE =>  65.00 %
        ....PLAGIARISED....
```

Figure 8 printing lcs

- https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-2000-server/cc940018(v=technet.10)?redirectedfrom=MSDN
- https://www.javatpoint.com/c-array
- https://en.wikipedia.org/wiki/Classful_network
- https://www.plagiarism.org/article/what-is-plagiarism

For complete source code refer to :

- Subhendu Maji (2019)
https://github.com/subhendu17620/RUAS/tree/master/sem%2003/DSA%20lab/assignment