

|  |  |
| --- | --- |
| **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**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

[**Declaration Sheet** ii](#_Toc21883325)

[**Contents** iii](#_Toc21883326)

[Marking Scheme v](#_Toc21883327)

[**Question No. 1** 7](#_Toc21883328)

[A1.1 Static memory allocation 7](#_Toc21883329)

[A1.1.1 Intro: 7](#_Toc21883330)

[A1.1.2 Key features: 7](#_Toc21883331)

[A1.1.3 Example 7](#_Toc21883332)

[A1.1.4 Deletion of allocated memory 8](#_Toc21883333)

[A1.1.5 Advantages of Static memory allocation 8](#_Toc21883334)

[A1.1.6 Disadvantages of Static memory allocation 8](#_Toc21883335)

[A1.2 Dynamic memory allocation 8](#_Toc21883336)

[A1.2.1 Intro 8](#_Toc21883337)

[A1.2.2 Allocating a block of memory: malloc 9](#_Toc21883338)

[A1.2.3 Allocating multiple blocks of memory: calloc 9](#_Toc21883339)

[A1.2.4 Releasing the used space: free 9](#_Toc21883340)

[A1.2.5 Altering the size of a block: realloc 9](#_Toc21883341)

[A1.2.6 Advantages of Dynamic memory allocation 9](#_Toc21883342)

[A1.2.7 Disadvantages of Dynamic memory allocation 10](#_Toc21883343)

[A1.3 Comparative analysis 10](#_Toc21883344)

[**Question No. 2** 12](#_Toc21883345)

[B1.1 Classful addressing in networks 12](#_Toc21883346)

[B1.2 Data Structures Used 14](#_Toc21883347)

[B1.3 Validated C Program 14](#_Toc21883348)

[**Question No. 3** 20](#_Toc21883349)

[B2.1 Plagiarism rules and threshold 20](#_Toc21883350)

[B2.2 Pseudocode for checking plagiarized content 20](#_Toc21883351)

[B2.3 Validated C Program 20](#_Toc21883352)

|  |  |  |  |
| --- | --- | --- | --- |
|  | **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 | ValidatedC 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 | | | | | | | | | | | |

# **Question No. 1**

**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 play()

{

   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.

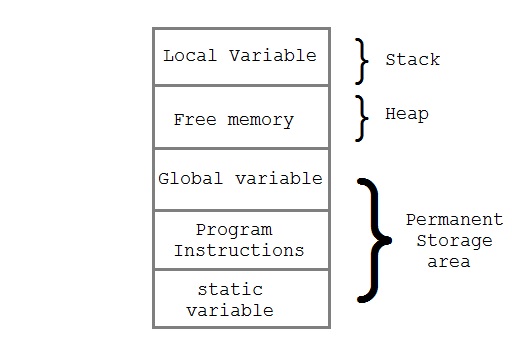
### A1.2.7 Disadvantages of Dynamic memory allocation

* 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. |

# **Question No. 2**

**Solution to Question No. 2:**

## B1.1 Classful addressing in networks

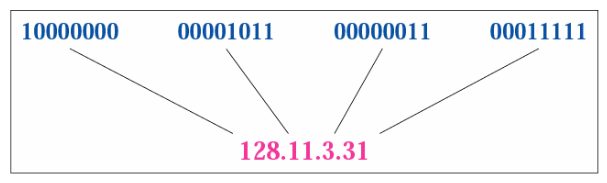
An IP address is a 32-bit address that identifies a connection to the Internet.

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.

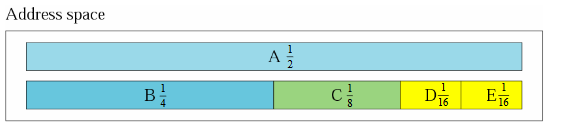
The IP addresses are universally unique.

The address space of IPv4 is 232  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.

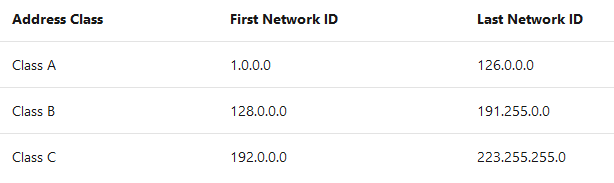
The minimum possible value for the left most octet in binaries is 11100000 (decimal equivalent is 224) and the maximum possible value for the leftmost octet is 11101111 (decimal equivalent is 239). Therefore for a "Class D" IPv4 address, leftmost octet must have a value between 224-239 (224.X.X.X to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



**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

## B1.3 Validated C Program

Code Snippet:

*/\* 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();

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*

        }

    }

}

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);

    }

}

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;

    }

}

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;

}

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:

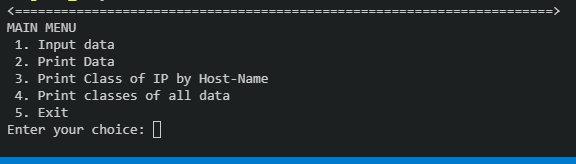


Figure 1 Menu of the program

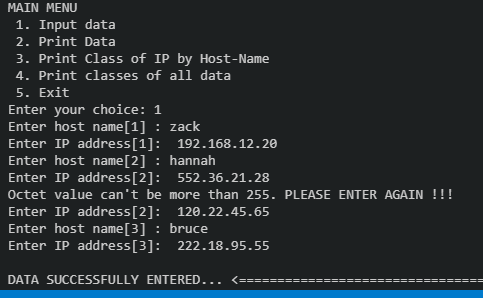


Figure 2 Inputting data using structures also checking if the value is <255

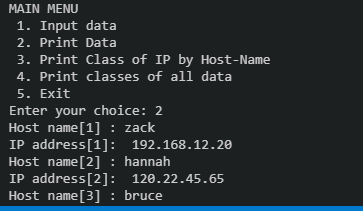


Figure 3 Printing the stored data from structures

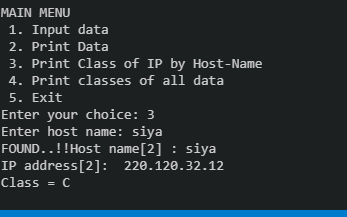


Figure 4 searching a data from its host-name

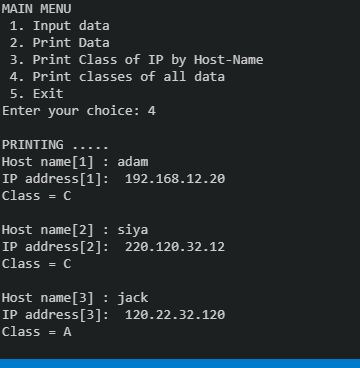


Figure 5 printing all the data with its class

# **Question No. 3**

**Solution to Question No. 3:**

## B2.1 Plagiarism rules and threshold

## B2.2 Pseudocode for checking plagiarized content

## 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 Printing LCS.*

*/\* global variables \*/*

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;

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 file 1*

        while (fgets(data2, 100, f2)); *// data2 is a string of the whole data of file 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 Printing 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: