```
//*****************************************************************//
//                                                                 //
//                        DNS DOSSIER                              //
//                                                                 //
//*****************************************************************//
//                                                                 //
//                  SERVER END TCP/IP APPLICATION                  //
//                                                                 //
//*****************************************************************//

#include <stdio.h>                    // Standard input and output
#include <sys/socket.h>                 // For socket(), connect(), send(), and recv()
#include <arpa/inet.h>                 // For sockaddr_in and inet_addr()
#include <stdlib.h>                   // For atoi() and exit()
#include <string.h>                   // For memset()
#include <unistd.h>                    // For close()
#include <netdb.h>                     // For gethostbyname()
#include <time.h>                     // For time_t
#define MAXCONNECTION 5                  // Maximum outstanding connection requests
#define RCVBUFSIZE 100                 // Size of receive buffer
#define SECURITYCODE "ABC123"              // Pre-shared  security code

void DieWithError(char *errorMessage);      // Error handling function
void HandleTCPClient(int clntSocket, int serverSock, char ip[]);
                                // TCP client handler function

/*** Structure for Documenting last served client ***/
struct{
   char ipAddress[16];
   time_t timeStamp;
} clientNode;

/*** Structure for Linked List ***/
struct Node{
   char domainName[20];
   int count;
   char ipAddr[65];
   struct Node *next;
}l;                                 // Structure variable "l"

typedef struct Node *nodePointer;          // nodePointer points to a structure of type linkedList

//************  Function Prototype Declarations  ***************//

void readInput(char *, nodePointer *);

char * addDomain(char *, nodePointer *, int);

void displayNode(nodePointer);
```

```
void * searchDomain(char *, nodePointer *, int);

void deleteDomain(char *, nodePointer *);

void  nodeFromStructure(nodePointer, int);

char * toString(char str[], int);

void timesRequested(char , nodePointer);

void writeLinkedListToFile(nodePointer);

char * resolveName(char name[]);

//******************     Global variables   *********************//

nodePointer head = NULL;
char buffer[RCVBUFSIZE];
char fileLocation[200];
char responseMessage[100];
char resolveNameMessage[100];
unsigned int  timeOut;
unsigned int oneTimeCount = 0;

//-------------------------------------------------------------//
//******************     MAIN FUNCTION      *********************//
//-------------------------------------------------------------//

int main(int argc, char *argv[]) {

   int serverSock;                  // Socket descriptor for server
   int clientSock;                  // Socket descriptor for client

   struct sockaddr_in serverAddr;         // Structure variable for Local address
   struct sockaddr_in clientAddr;         // Structure variable for Client address

   unsigned short serverPort;             // Server port
   unsigned int clientLen;               // Length of client address data structure


   char line[1024];                  // String variable to read input from file

   if (argc != 4)                    // Test for correct number of arguments
   {
      printf("\n\t\tNumber of command line parametes aren't proper. Terminating DNS service!");
      exit(1);
   }
```

```
printf("\n\t\tDNS Doisser listening on Port Number is: %s\n",argv[1]);
FILE *fp = fopen(argv[2],"r");

if( fp == NULL ){                          // If any error opening file, return -1
   if ((fp = fopen(argv[2],"w")) == NULL)
      return -1;
}

else{
   while(fgets(line,1024,fp)){              // If file opened properly, start reading line by line

      readInput(line, &head);
   }

   displayNode(head);                      // Calling the function to display the nodes in Linked List
}

printf("\n\n\t\tFile opened and data read success");
fclose(fp);                                // Close the file DOMAIN_IP_MAP.TXT

printf("\n\n\t\tWelcome to the DNS Doisser System \t\t\n");
                                   // Print available operations at the Server
printf("\n\t\tRequest Code \t\tAction"
      "\n\t\t\t1. \t\t\tFind IP for a domain"
      "\n\t\t\t2. \t\t\tAdd a record to the list"
      "\n\t\t\t3. \t\t\tDelete a record from the list"
      "\n\t\t\t4. \t\t\tReport the most requested record(s)"
      "\n\t\t\t5. \t\t\tReport the least requested record(s)"
      "\n\t\t\t6. \t\t\tShutdown");

serverPort = atoi(argv[1]);                // First arg:  local port
strcpy(fileLocation,argv[2]);
timeOut = atoi(argv[3]);

/* Create socket for incoming connections */
if ((serverSock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0)
   DieWithError("socket() failed");

/* Construct local address structure */
memset(&serverAddr, 0, sizeof(serverAddr));     // Zero out structure
serverAddr.sin_family = AF_INET;                // Internet address family
serverAddr.sin_addr.s_addr = htonl(INADDR_ANY); // Any incoming interface
serverAddr.sin_port = htons(serverPort);        // Local port

/* Bind to the local address */
if (bind(serverSock, (struct sockaddr *) &serverAddr, sizeof(serverAddr)) < 0)
   DieWithError("bind() failed");

/* Mark the socket so it will listen for incoming connections */
```

```
   if (listen(serverSock, MAXCONNECTION) < 0)
      DieWithError("listen() failed");
   printf("\n");
   int i =0;

   //******************    Run forever   ******************//

   for (;;)
   {
      clientLen = sizeof(clientAddr);            // Set the size of the in-out parameter

      /* Wait for a client to connect */
      if ((clientSock = accept(serverSock, (struct sockaddr *) &clientAddr,&clientLen)) < 0)
         DieWithError("accept() failed");

      /* clientSock is connected to a client! */

      while(i==0){
         strcpy(clientNode.ipAddress, inet_ntoa(clientAddr.sin_addr));
         clientNode.timeStamp= time(NULL);
         i++;
      }

      HandleTCPClient(clientSock, serverSock,inet_ntoa(clientAddr.sin_addr));     // Calling TCP
handler
   }
}
//------------------------------------------------------------//
//******************   MAIN FUNCTION  ENDS  *********************//
//------------------------------------------------------------//


//********** Function to read lines from txt file into linked list data structure ***************//

void readInput(char ch[], nodePointer *first)
{
   char *word;                              // Temporary word

   nodePointer current = malloc(sizeof(l));        // Allocating memory for the current pointer to the
structure

   if((*first) == NULL){                        // Zero element in linked List test
      *first = current;
      (*first)->next = NULL;
   }
   else{                                  // If its not the first element
      (*current).next = *first;
      *first = current;
   }
```

```
    /* Extracting the data structure members from string read from the file */

    word = strtok(ch," ");
    strcpy((*first)->domainName, word);

    word = (strtok(NULL," "));
    (*first)->count = atoi(word);

    word = (strtok(NULL,"\n"));
    strcpy((*first)->ipAddr, word);


}

//********** Function to search a Domain Name **************//

void * searchDomain(char *dName, nodePointer *first, int type){

    char temp[100];
    int flag=0;                         // FLAG=0/1, Tells if a domain is found or not
    nodePointer cur = *first;

    while(cur!=NULL){                   // Search for domain till the end of the linked list

        if(strcmp((cur)->domainName,dName)==0){ // Compare the domain name received to those
existing in database
            printf("\n\n\t\tMatch Found");
            printf("\t\t%s %d %s",(cur)->domainName,(cur)->count,(cur)->ipAddr);
            flag = 1;                   // Set FLAG = 1, if domain is found
            break;
        }
        cur=cur->next;
    }


    if (flag == 0 && type == 1){        // If IP for the domain is NOT found
        printf("\n\n\t\tDomain record not found, trying gethostbyname() : ");

        if (resolveName(dName)!=NULL){
            strcpy(resolveNameMessage,resolveName(dName));
            /* Search to find its IP and add if found */
            strcpy(temp, dName);
            strcat(temp, " ");
            strcat(temp, resolveNameMessage);

            if(*first != NULL)          // Add domain
                addDomain(temp,&head,1);
            else
```

```
            addDomain(temp,&head,2);

          return (char *) resolveNameMessage;
       }
       else
          return NULL;
   }

   else if (flag == 1 && type==1){         // If domain match found and action requested is type 1
          cur->count += 1;
          return (char *) cur->ipAddr;    // Return the pointer to the node found
   }

   else if (flag == 1 && type == 2){       // If Domain match found and action requested is of type 2
          return (nodePointer) cur;       // Return the pointer of the searched node
       }

   else
      return NULL;                        // No match found
}

//*********  Function to read clients request to add new domain into existing linked list
***************//
char * addDomain(char *domainIp, nodePointer *first, int type)
{
   char *word;
   char domainPart[40];
   char ipPart[18];

   /* Split the string to obtain domain name and IP */
   word = strtok(domainIp," ");
   strcpy(domainPart, word);
   word = strtok(NULL, "\0");
   strcpy(ipPart,word);

   nodePointer nodeTOBeAdded = malloc(sizeof(l)); // Allocating memory for the current pointer to the
structure
   nodePointer matchPointer;

   /* If Action requested is ADD, Check for room for new IP if domain already exists */

   /*************** ASSUMPTION: MAXIMUM IP RECORDABLE IS 4 & ONLY ONE IP CAN
BE ADDED AT A TIME ***************/


   if (*first !=NULL && (matchPointer= searchDomain(domainPart, first, 2))!= NULL && type == 2)
{
      printf("\n\n\t\tDomain already exist in the linked list");
      printf("\n\t\tChecking Room for IP Address");
```

```
        char *tempComparator;
        char oldIpPart[64];
        strcpy(oldIpPart, matchPointer->ipAddr);

        // If record contains only 1 IP address for the domain to be added
        if (strlen(matchPointer->ipAddr)< 18 && strlen(ipPart)< 18 && strlen(matchPointer->ipAddr)
>6){

            if( strcmp(matchPointer->ipAddr,ipPart)!=0){
                strcat(matchPointer->ipAddr, " ");
                strcat(matchPointer->ipAddr, ipPart);
                printf("\n\t\tNew IP added for %s ",matchPointer->domainName);
                strcpy(responseMessage,"New IP added for ");
                strcat(responseMessage,matchPointer->domainName);
            }
            else{ // Record already exists
                printf("\n\t\tThe record to be added already exists  %s ",matchPointer->domainName);
                strcpy(responseMessage,"The record to be added already exists ");
                strcat(responseMessage,matchPointer->domainName);
            }
        }

        // If the record contains 2 IP addresses for the domain to be added
        else if (strlen(matchPointer->ipAddr)< 34 && strlen(ipPart)< 18 && strlen(matchPointer-
>ipAddr)>14){

            tempComparator = strtok(oldIpPart," ");

            if( strcmp(tempComparator,ipPart)!=0){
               tempComparator = strtok(NULL,"\0");

               if( strcmp(tempComparator,ipPart)!=0){

                  strcat(matchPointer->ipAddr, " ");
                  strcat(matchPointer->ipAddr, ipPart);
                  printf("\n\t\tNew IP added for %s %s",matchPointer->domainName, matchPointer-
>ipAddr);
                  strcpy(responseMessage,"New IP added for ");
                  strcat(responseMessage,matchPointer->domainName);
               }// inner if ends here
               else{
                  printf("\n\t\tThe record to be added already exists %s ",matchPointer->domainName);
                  strcpy(responseMessage,"The record to be added already exists ");
                  strcat(responseMessage,matchPointer->domainName);
               }
            }// outer if ends here
            else{
               printf("\n\t\tThe record to be added already exists %s ",matchPointer->domainName);
```

```
            strcpy(responseMessage,"The record to be added already exists ");
            strcat(responseMessage,matchPointer->domainName);
          }
        }// else-if ends here

        // If the record contains 3 IP addresses for the domain to be added
        else if (strlen(matchPointer->ipAddr)< 50 && strlen(ipPart)< 18 && strlen(matchPointer-
>ipAddr)>22){

          tempComparator = strtok(oldIpPart," ");
          if( strcmp(tempComparator,ipPart)!=0){

            tempComparator = strtok(NULL," ");
            if( strcmp(tempComparator,ipPart)!=0){

              tempComparator = strtok(NULL,"\0");
              if( strcmp(tempComparator,ipPart)!=0){

                strcat(matchPointer->ipAddr, " ");
                strcat(matchPointer->ipAddr, ipPart);
                printf("\n\t\tNew IP added for %s ",matchPointer->domainName);
                strcpy(responseMessage,"New IP added for ");
                strcat(responseMessage,matchPointer->domainName);
              }// inner if ends here
              else{
                printf("\n\n\t\tThe record to be added already exists %s ",matchPointer->domainName);
                strcpy(responseMessage,"The record to be added already exists ");
                strcat(responseMessage,matchPointer->domainName);
              }
            }// second-inner most if ends here
            else{
              printf("\n\n\t\tThe record to be added already exists%s ",matchPointer->domainName);
              strcpy(responseMessage,"The record to be added already exists ");
              strcat(responseMessage,matchPointer->domainName);
            }
          }// outer if ends here
          else{
            printf("\n\n\t\tThe record to be added already exists%s ",matchPointer->domainName);
            strcpy(responseMessage,"The record to be added already exists ");
            strcat(responseMessage,matchPointer->domainName);
          }
        }// else-if ends here

        else{
          printf("\n\n\t\tNo room for additional IP ");
          strcpy(responseMessage,"No room for additional IP for");
          strcat(responseMessage,matchPointer->domainName);

          }
```

```
        }

    /* If Domain to be added DOES not exist in the list, then ADD */
   else {
        printf("\n\n\t\tNew Domain added :%s \n",domainPart);
        strcpy(nodeTOBeAdded->domainName, domainPart);
        nodeTOBeAdded->count= 0;
        strcpy(nodeTOBeAdded->ipAddr, ipPart);
        nodeTOBeAdded->next = (*first);
        (*first) = nodeTOBeAdded;
        if (type ==2){
            strcpy(responseMessage,"New IP added for ");
            strcat(responseMessage,nodeTOBeAdded->domainName);
        }
    }
    return responseMessage;
}

//********** Function to display linked list nodes ****************//

void displayNode(nodePointer first){
    while(first != NULL){
        printf("\n\t\t%s %d %s",first->domainName,first->count,first->ipAddr);
        first=first->next;
    }
}

/* Returns node details as a string */

void nodeFromStructure(nodePointer node, int n){
    char * content = node->domainName;
    char nodeCount[8];
    char * count = toString(nodeCount,node->count);     // Converting the Integer variable, 'count' to
string

    /* Creating a string to be sent as a reply to the Client */
    if( n == 0){
        strcpy(buffer, content);
        strcat(buffer, " ");
        strcat(buffer,count);
    }
    else{
        strcat(buffer, content);
        strcat(buffer, " ");
        strcat(buffer,count);
    }

    strcat(buffer, " ");
```

```
}


//********* Function to return the most or least requested record ****************//

void timesRequested(char type, nodePointer first){

   nodePointer nextNode = first;
   nodePointer maxOrMinNode = first;       // Variable to store the node that is max/min  requested
   int count = 0;                          // Counter to find top most 3 domains in case of tie
                                           // between count

   switch(type){
      case '4':                            // When the requested action code is "4"
                                           // MOST requested record
               while(nextNode!=NULL){
                  // Searching through the Linked List till the end
                  // Check for finding greater of two values

                  if(nextNode->count > maxOrMinNode->count){
                     maxOrMinNode = nextNode;
                  }
                  nextNode = nextNode->next;
               }

               nextNode = first;
               while(nextNode != NULL){

                  if((nextNode->count == maxOrMinNode->count) && count <3){
                     nodeFromStructure(nextNode, count);
                     count++;
                  }
                  nextNode = nextNode->next;
               }


               break;

      case '5' :                           // When the requested action code is "5"
                                           // LEAST requested record
               while(nextNode!=NULL){
                  // Searching through the Linked List till the end
                  // Check for finding least of two values

                  if(nextNode->count < maxOrMinNode->count){
                     maxOrMinNode = nextNode;
                  }
                  nextNode = nextNode->next;
               }
```

```
                nextNode = first;
                while(nextNode != NULL)
                {
                   if((nextNode->count == maxOrMinNode->count) && count <3){
                      nodeFromStructure(nextNode, count);
                      count++;
                   }
                   nextNode = nextNode->next;
                }

                break;

        default:        break;
    }
}

//************       TCP Connection Handler  *************//

void HandleTCPClient(int clntSocket, int serverSock, char clientIp[])
{

printf("\n\n\n#############################################################################
################");
    printf("\n\t\tEntered TCP handler for %s ", clientIp);
    char *word;                         // Temporary word variable to obtain words from string
    int  action;                       // Action requested
    char dName[40];                     // Domain name
    time_t now;                        // Time variable
    char sec[5];
    long recvMsgSize;                   // Size of received message
    char *ipPointer;
    char *securityCode;                 // SECURITY CODE required to shutdown the Server
    int  statusCode = 0;                // Client request is outside the timeout window and request will be
processed directly

    // Receive message from client
    if ((recvMsgSize = recv(clntSocket, buffer, RCVBUFSIZE, 0)) < 0){
       DieWithError("recv() failed");}

    buffer[recvMsgSize]='\0';             //Terminating the receive buffer with null character

    now =time(NULL);                    // Current time

    if ((now - clientNode.timeStamp )< 10 && oneTimeCount != 0){
       // Calculates the time elapsed since previous query
       statusCode = 3;                  // Do not process the query, ask Client to wait till the timeout
     }
    else{
```

```
        strcpy(clientNode.ipAddress, clientIp);
        clientNode.timeStamp = now;         // Timestamp the current time for a Client
    }
    if( statusCode == 0){                   // Normal operation - Prcoess the query received from the Client

    switch(buffer[0]){

        case '4':                           // If arguments are 3, find most/least requested domain

                    if (buffer[2] == '4'){
                        printf("\n\n\t\tIncoming request from client  %s : Most requested domain\n", clientIp);
                        printf("\n\t\tDNS Dossier will present at most 3 most requested domains in case of tie
b/w counts\n");
                        timesRequested('4', head);
                        break;
                        // Function to print top most
                    }

                    else if (buffer[2] == '5'){
                        printf("\n\n\t\tIncoming request from client  %s : Least requested domain\n",
clientIp);
                        printf("\n\t\tDNS Dossier will present at most 3 Least requested domains in case of tie
b/w counts\n");
                        timesRequested('5', head);
                        break;
                        // Function to calculate the number of times a domain was requested the LEAST
                    }
                    else{
                        printf("\n\t\tTransmission Error via client");    // if command isn't in write format
                        strcpy(buffer,"Incorrect request passed by client ");
                        strcat(buffer,clientIp);
                    }
                    break;

        case '5':       // If arguments are 4 find IP or shutdown

                    if (buffer[2] == '6'){ // When requested action is SHUTDOWN
                        printf("\n\n\t\tIncoming request from client  %s : Shutdown DNS Dossier", clientIp);
                        securityCode=strtok(buffer+4,"#");


                        if (strcmp(SECURITYCODE, securityCode) == 0){   // Verify SECURITY CODE
                            printf("\n\n\t\tSecurity code verified");
                            printf("\n\n\t\tSaving LinkedList into the file");

                            writeLinkedListToFile(head);                // Write to file all the nodes
                            printf("\n\n\t\tLinked List written to the File\n");
                            statusCode = 1;
                            strcpy(buffer,"Server is going down. Connection will be lost!!");
```

```
                }
                else{
                    printf("\n\n\t\tSecurity code incorrect");
                    strcpy(buffer,"Security code entered is incorrect. Action Denied!");

                }

                break;

            }// end of if


            else if (buffer[2] == '1'){          // ACTION requested is Find IP for a domain
                                    action = (int)buffer[0];
                word = strtok(buffer,"#");
                word = strtok(NULL,"#");
                word = strtok(NULL,"#");
                strcpy(dName,word);
                printf("\n\n\t\tIncoming request from client  %s : Search IP for domain %s", clientIp,
word);
                if((ipPointer = searchDomain(word, &head, 1)) !=NULL){
                    strcpy(buffer, ipPointer);
                    buffer[strlen(ipPointer)]='\0';
                }

                else
                    strcpy(buffer,"IP not found");

                break;
            }

            else if(buffer[2] == '3'){          // ACTION requested is DELETE a specific domain
                word = strtok(buffer,"#");
                word = strtok(NULL,"#");
                word = strtok(NULL,"#");
                strcpy(dName,word);
                printf("\n\n\t\tIncoming request from client  %s : Delete domain %s", clientIp, word);
                deleteDomain(dName, &head);
                strcpy(buffer,responseMessage);
                break;
            }

            else {
                    printf("\n\t\tTransmission Error via client");    // if command isn't in write format
                    strcpy(buffer,"Incorrect request passed by client ");
                    strcat(buffer,clientIp);
                    break;
            }
```

```
    case '6':   if( buffer[2] == '2'){              // ACTION requested is ADD domain

                    word = strtok(buffer,"#");
                    word = strtok(NULL,"#");
                    word = strtok(NULL,"#");
                    printf("\n\n\t\tIncoming request from client  %s : Add domain %s", clientIp, word);
                    strcpy(dName,word);
                    strcpy(buffer,addDomain(dName, &head,2));
                    break;
                }
                else {
                    printf("\n\t\tTransmission Error via client");    // if command isn't in write format
                    strcpy(buffer,"Incorrect request passed by client ");
                    strcat(buffer,clientIp);
                    break;
                }



    default:    {
                    printf("\n\t\tTransmission Error via client");    // if command isn't in write format
                    strcpy(buffer,"Incorrect request passed by client ");
                    strcat(buffer,clientIp);
                }

  }

  /* Send received string and receive again until end of transmission */
  while (recvMsgSize > 0)      /* zero indicates end of transmission */
  {
    /* Echo message back to client */
    if (send(clntSocket, buffer, RCVBUFSIZE, 0) != RCVBUFSIZE)
      DieWithError("send() failed");

    /* See if there is more data to receive */
    if ((recvMsgSize = recv(clntSocket, buffer, RCVBUFSIZE, 0)) < 0)
      DieWithError("recv() failed");
  }
  printf("\n\t\tSent Response to the Client: %s", buffer);

printf("\n\n################################################################################
##############");

 }// end of if loop with STATUS code 0


  // Do not process the query, ask Client to wait till the timeout
```

```
    if(statusCode == 3){
        strcpy(buffer,"Another inquiry had been made ");
        strcat(buffer,toString(sec, (int) (now - clientNode.timeStamp)));
        strcat(buffer," seconds ago, wait ");
        strcat(buffer,toString(sec,timeOut));
        strcat(buffer," seconds before another submission" );
        if (send(clntSocket, buffer, RCVBUFSIZE, 0) != RCVBUFSIZE)
            DieWithError("send() failed");
        printf("\n\n\t\tSent Response to the Client: %s", buffer);

printf("\n\n################################################################################
##############");

    }

    // Save the linkedlist to the file, and SHUTDOWN the server

    if (statusCode == 1){
        printf("\n\t\tClosing Client Socket");
        close(clntSocket);
        printf("\n\t\tClosing Server listening socket");
        close(serverSock);
        exit(0);

    }
    oneTimeCount = 1;
    close(clntSocket);    /* Close client socket */
}

//******* Function to convert Integer value to String   **********//

char * toString(char * str, int num)
{
    int i, rem, len = 0, n;

    if(num >0){
        n = num;
        while (n != 0)      // Count the number of digits
        {   len++;
            n /= 10;
        }
        for (i = 0; i < len; i++)
        {   rem = num % 10; // Convert each digit to char and store in the char array
            num = num / 10;
            str[len - (i + 1)] = rem +'0' ;
        }
        str[len] = '\0';
    }
    else
```

```
        return "0";

    return str;
}


//*************** Function to SHUTDOWN the server  *********************//

void writeLinkedListToFile(nodePointer first){

    FILE *fp;
    char line[100];
    char str[8];
    printf("\n\n\t\tLocation is :%s",fileLocation);

    /* Open the file to write */
    if( (fp = fopen(fileLocation, "w") ) == NULL )
        DieWithError("\nError opening file to save linked list");

    else{
        while(first != NULL){          // Add all the nodes in the linked list
            strcpy (line, first->domainName);
            strcat (line, " ");
            strcat (line, toString(str,first->count));
            strcat (line, " ");
            strcat (line, first->ipAddr);
            strcat (line,"\n");
            fprintf(fp,"%s",line);
            first = first->next;        // Incrementing to next node
        }

        fclose(fp);                    // CLOSE FILE
    }

}

//********** Function to DELETE A RECORD on client's request   ************//

void deleteDomain(char *dName, nodePointer *first)
{

    nodePointer prev, cur;
    int flag =0;                // FLAG = 1, record found | FLAG = 0, recod NOT found

    prev = (*first);
    cur = (*first);

    if(cur->next == NULL && strcmp(dName,cur->domainName)== 0){
        // When the last node needs to be deleted
```

```
            *first = NULL;
            strcpy(responseMessage, cur->domainName);
            strcat(responseMessage, " deleted from the linked list");
            flag = 1;
        }// end of if


    else if(cur->next != NULL && strcmp(dName,cur->domainName)== 0){
    // When the HEAD node needs to be deleted
        *first = cur->next;
        strcpy(responseMessage, cur->domainName);
        strcat(responseMessage, " deleted from the linked list");
        flag = 1;
    }// end of else if

    else{   // When any node in between needs to be deleted
            cur = cur->next;
            while(cur != NULL){
                if(strcmp(dName,cur->domainName) == 0){
                    prev->next = cur ->next;
                    strcpy(responseMessage, cur->domainName);
                    strcat(responseMessage, " deleted from the linked list");
                    flag = 1;
                }

                prev = cur;
                cur = cur->next;

            }// end of while loop
    }// end of else

    if (flag ==0){
        strcpy(responseMessage, dName);
        strcat(responseMessage, " doesn't exist in the record database");

    }
    // Print HEAD for the linked list
    printf("\n\n\t\tNew Head after deletion is %s %s\n",(*first)->domainName, (*first)->ipAddr);

}


//*********** Resolving domain name  **********//
char * resolveName(char name[])
{
    struct hostent *hp = gethostbyname(name);

    if (hp == NULL) {                       // If gethostby name is unable to find the IP
        printf("gethostbyname() failed\n");
```

```
        return NULL;
    }
    else {
        printf(" Domain Name Resolved");                    // If gethostby name returns an IP address
        unsigned int i=0;

        while ( hp -> h_addr_list[i] != NULL) {          // Add it to the existing domain IP map
            if (i == 0){
                strcpy(responseMessage, inet_ntoa( *( struct in_addr*)( hp -> h_addr_list[i])));
                strcat(responseMessage, " ");
            }

            i++;
        }
        return responseMessage;
    }

}

//********* Die with error Function  **********//
void DieWithError(char *errorMessage)
{
    perror(errorMessage);
    exit(1);
}
```