YILDIZ TECHNICAL UNIVERSITY

ELECTRICAL- ELECTRONICS

FACULTY

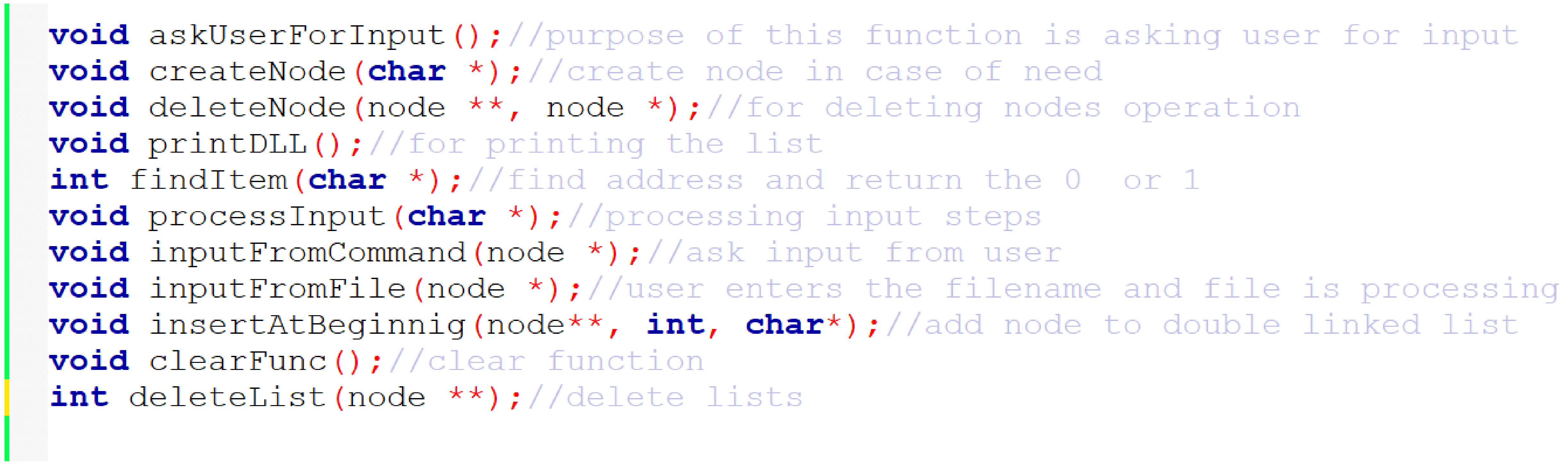
COMPUTER ENGINEERING

DEPARTMENT

KAMRAN BALAYEV 17011904

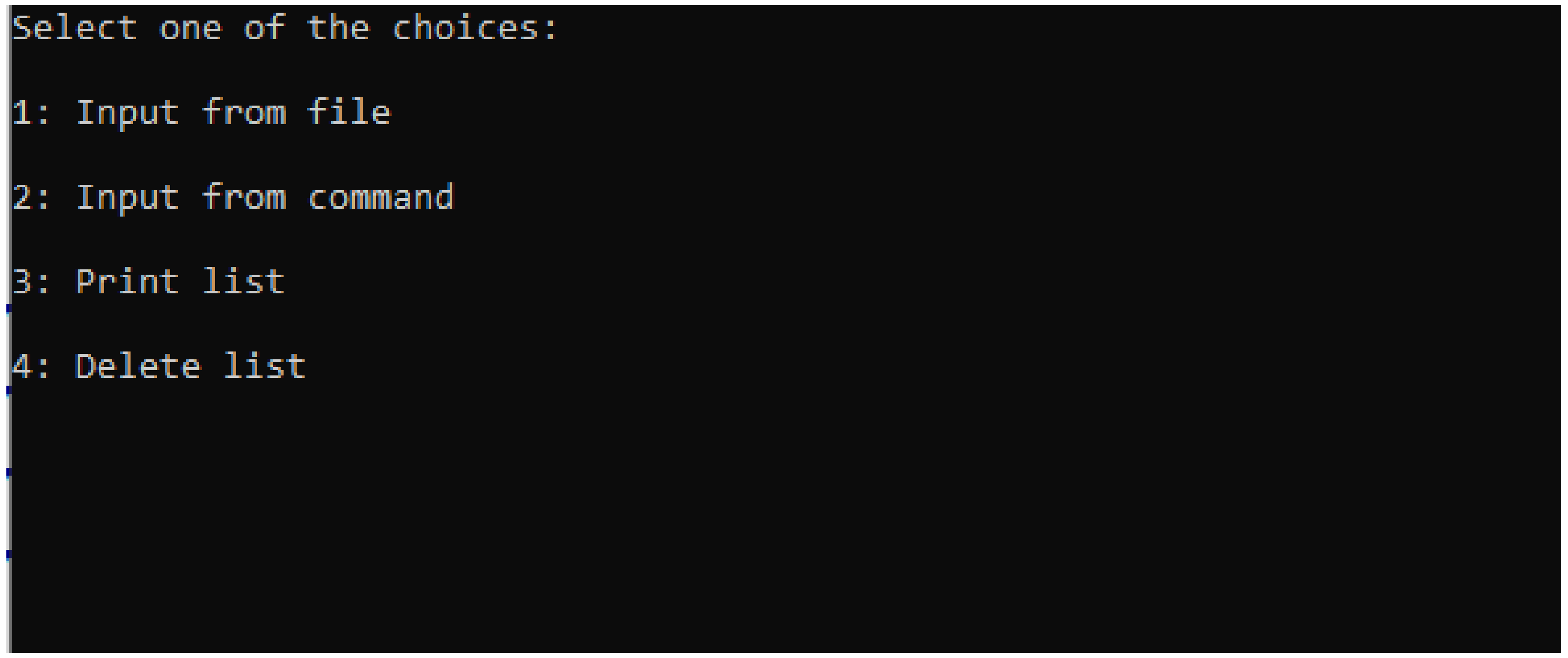
The short description of this assignment is creating cache buffer. I have used double linked list as a data structure . Each node has its own address, counter for address and next and previous pointer. We can ask counter size for addresses and add and delete nodes.

There are total of 11 functions in this application:



```c
void askUserForInput();//purpose of this function is asking user for input
void createNode(char *);//create node in case of need
void deleteNode(node **, node *);//for deleting nodes operation
void printDLL();//for printing the list
int findItem(char *);//find address and return the 0  or 1
void processInput(char *);//processing input steps
void inputFromCommand(node *);//ask input from user
void inputFromFile(node *);//user enters the filename and file is processing
void insertAtBeginnig(node**, int, char*);//add node to double linked list
void clearFunc();//clear function
int deleteList(node **);//delete lists
```

Application has 4 offers for the user:



```
Select one of the choices:

1: Input from file

2: Input from command

3: Print list

4: Delete list
```

In the first choice user should have a txt file in the same directory with application and then enter the name of txt file to the console . After these processes, program will get the datas from file into the buffer and strtok will be used for parsing operation. Parsed datas will be sent to the ***processInput*** function.

In the second choice user enters the T ,L ,requests and these addresses will be sent to ***processInput*** function. User have option for finishing the program by clicking the f key.

Third choice print the available lists.

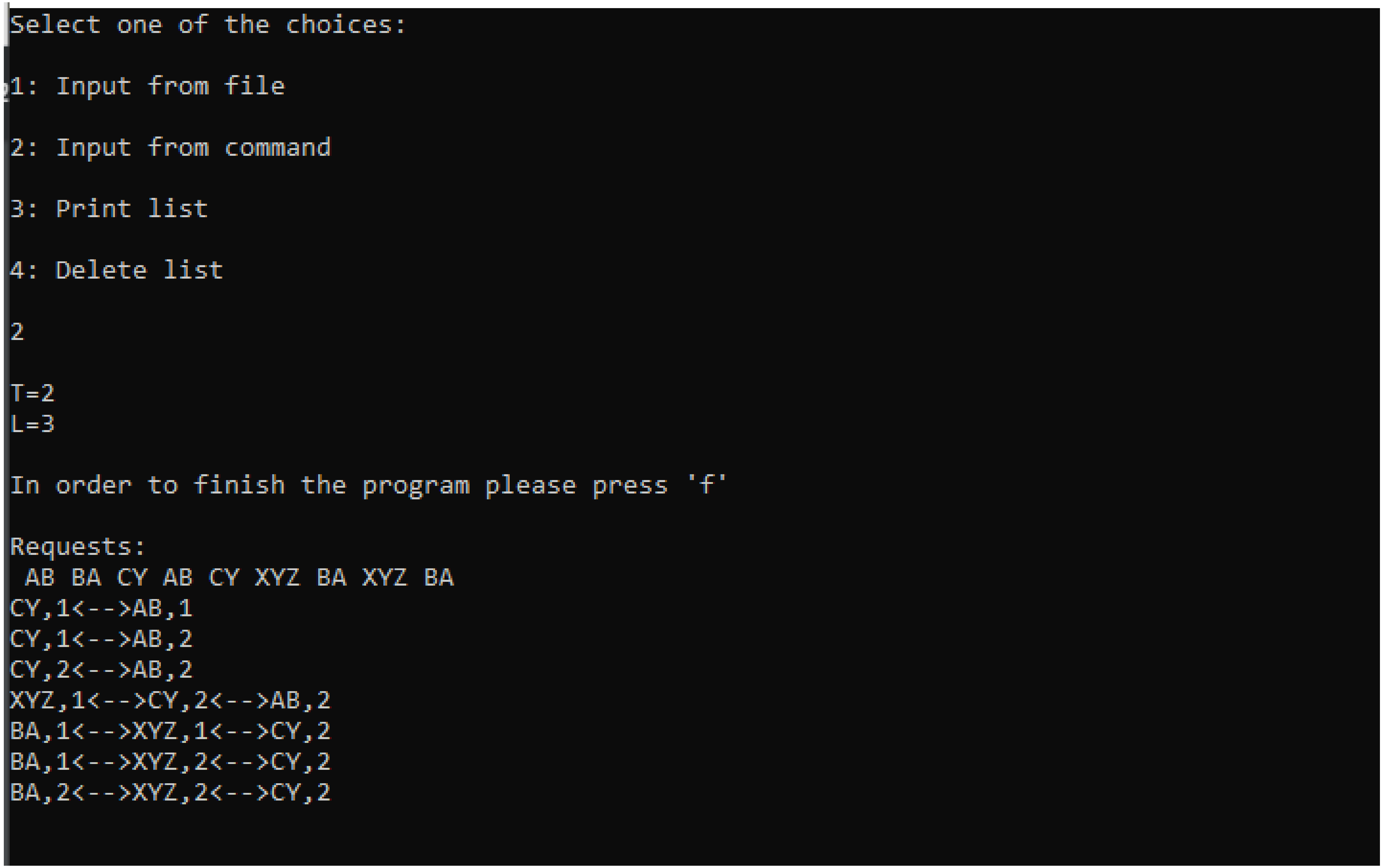
Fourth choice is deleting the available lists.

```c
/*
    KAMRAN BALAYEV 17011904
*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

//Node of a doubly linked list
typedef struct node {
        char address[30];    //address of page
        int counter;         //counter of address
        struct node* next; // pointer to next node
        struct node* prev; // pointer to previous node

    }node;

    void askUserForInput();//purpose of this function is asking user for input
    void createNode(char *);//create node in case of need
    void deleteNode(node **, node *);//for deleting nodes operation
    void printDLL();//for printing the list
    int findItem(char *);//find address and return the 0  or 1
    void processInput(char *);//processing input steps
    void inputFromCommand(node *);//ask input from user
    void inputFromFile(node *);//user enters the filename and file is processing
    void insertAtBeginnig(node**, int, char*);//add node to double linked list
    void clearFunc();//clear function
    int deleteList(node **);//delete lists


    node *tail;
    node *head = NULL;//head of linked list
    node *temp;/*purpose of this variable is storing the last element of linked list
                  if  the threshold is hung the last element will be deleted*/
    //node **headPointer = &head;
    int T, L, lCounter = 0;/*T is hug variable, L is length variable, lCounter will be used
                              for deleting last element*/


    int main() {
        askUserForInput();//cache buffer is starting
        return 0;
    }

    void askUserForInput() {
        int x;//choice key
        printf("Select one of the choices: \n");
        printf("\n1: Input from file \n");
        printf("\n2: Input from command \n");
        printf("\n3: Print list\n");
        printf("\n4: Delete list\n\n");
        scanf("%d", &x);
        switch (x)
        {
        case 1:
            printf("\n");
            inputFromFile(head);
            break;
        case 2:
            printf("\n");
            inputFromCommand(head);
            break;
        case 3:
            printf("\n");
            printDLL(head);
            break;
        case 4:
            printf("\n");
            clearFunc(head);
            break;
        default:
            printf("\nPlease select a proper case!\n\n");
            askUserForInput();
        }
    }

    void inputFromCommand(node *newNode) {
        char arr[700];//will be used for storing the requests
        head = NULL;
        tail = head;
        temp = head;
        printf("T=");
        scanf("%d", &T);
        printf("L=");
        scanf("%d", &L);
        printf("\nIn order to finish the program please press 'f' \n");
        printf("\nRequests:\n ");
        scanf("%s", &arr);
        createNode(&arr);
        scanf("%s", &arr);
        while (*arr != 'f') {
            scanf("%s",&arr);
            if (*arr == 'f')
                return;
            processInput(&arr);
            printDLL(newNode);
        }

    }

    void inputFromFile(node *newNode) {
        char arr1[500], arr2[500], *tk, filename[15];
        FILE *fp;
        int a = 0;
        head = NULL;
        tail = head;
        temp = head;
        printf("\nEnter name of a file you wish to see\n");
        scanf("%s", filename);
        /*  open the file for reading */
        fp = fopen(filename, "r");

        fgets(arr2,500,fp);
        strcpy(arr1, arr2);

        tk = strtok(arr1,",");
        tk = strtok(tk,"=");
        tk = strtok(NULL,"=");
        T = atoi(tk);

        strtok(arr2,"\n");
        tk = strtok(arr2,",");
        tk = strtok(NULL,",");
        tk = strtok(tk,"=");
        tk = strtok(NULL,"=");
        L = atoi(tk);
        fgets(arr1,500,fp);
        tk=strtok(arr1,"\n");
        tk = strtok(arr1," ");
        createNode(tk);
        fclose(fp);
        while(tk != NULL){
            tk = strtok(NULL," ");
            if(tk == NULL)
                return;
            processInput(tk);
            printDLL(newNode);
        }
    }

    void createNode(char *tk) {
        head = (node*)malloc(sizeof(node));//dynamic memory allocation
        head->next = NULL;
        head->prev = NULL;
        head->counter = 1;//set counter size 1
```

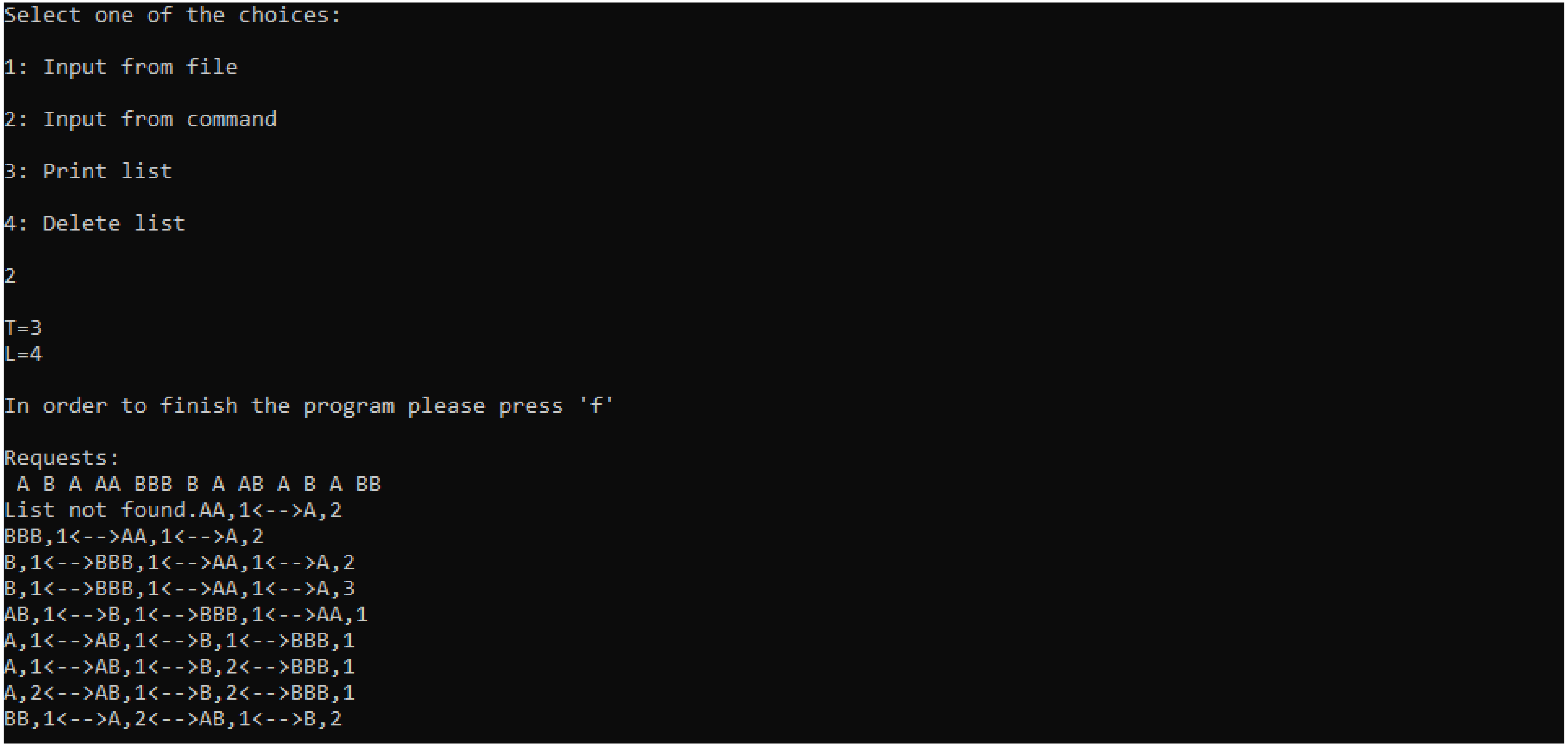
```c
        tail = head;//last and first element
        temp = head;//is referencing head
        strncpy(head->address, tk, 29);//add the address to the node
        lCounter++;
    }

    void processInput(char *tk) {
        if (findItem(tk))// if item exist in the list then start processing the data
        {
            temp->counter++;//incrementing the numerical value of address
            if (temp->counter > T) {// if counter is bigger than threshold
                insertAtBeginnig(&head, temp->counter, tk);//then add it to the head
                tail = temp->prev;
                deleteNode(&head, temp);
                temp = head;
                if (lCounter > L) {//if length counter is bigger than L
                    tail = tail->prev;
                    deleteNode(&head, tail->next);//remove the last element
                    lCounter--;//decrement the size of counter
                }
            }
        }
        else //else add node to the beginning of list and control the length size for deletion operation
        {
            insertAtBeginnig(&head, 1, tk);
            temp = head;
            lCounter++;
            if (lCounter > L) {
                tail = tail->prev;
                deleteNode(&head, tail->next);
                lCounter--;
            }
        }
    }

    /*findItem function is designed for searching address in the list and if the
    search operation is successful the function will return 1 (address is available in list), else it
    will return 0
    */
    int findItem(char *tk) {
        temp = head;
        while (temp != NULL) {
            if (strcmp(temp->address, tk) == 0)
                return 1;
            temp = temp->next;
        }
        return 0;
    }

```

```c
//insert new node to the linked list
void insertAtBeginnig(node** head, int counter, char* address)
{
    //memory allocation
    node* newNode = (node*)malloc(sizeof(node));
    //put the address
    strcpy(newNode->address, address);
    newNode->counter = counter;

    //Make next of new node as head and previous as NULL
    newNode->next = (*head);
    newNode->prev = NULL;

    //change previous of head node to new node
    if ((*head) != NULL)
        (*head)->prev = newNode;
    //move the head to point to the new node

    (*head) = newNode;
}


//print double linked list
void printDLL() {
    node *temp = head;
    if (head == NULL || head->next == NULL) {
        printf("List not found.");
    }
    else {
        while (temp != NULL) {
            printf("%s,%d", temp->address, temp->counter);
            if (temp->next != NULL) {
                printf("<-->");
            }
            temp = temp->next;
        }
        printf("\n");
    }
}


int deleteList(node **delNode) {
    if ((*delNode) == NULL || (*delNode)->next == NULL) {
        printf("\nList is not available!\n");
        return 1;
    }
    else {
        struct Node *temp;
        while ((*delNode)->next != NULL) {
            temp = (*delNode)->next;
            free((*delNode));
            (*delNode) = temp;
        }
        temp = NULL;
        free(temp);
        return 0;
    }


}

void clearFunc() {
    char a;
    printf("Are you sure for deleting the list (y (for yes) or n (for not)) ?");
    scanf("%s", &a);

    switch (a)
    {
    case ('y'):
        if(!deleteList(&head))
            printf("Deleted all list.");
        break;
    case ('n'):
        printf("Delete operation has been cancelled.");
        break;
    default:
        printf("Please select the proper choice:");
        clearFunc();
        break;
    }
}

void deleteNode(node ** headNode, node * dlt)
{

    if (*headNode == NULL || dlt == NULL)
        return;

    // If node to be deleted is head node
    if (*headNode == dlt)
        *headNode = dlt->next;

    // Change next only if node to be deleted is not the last node
    if (dlt->next != NULL)
        dlt->next->prev = dlt->prev;

    // Change previous only if node to be deleted is not the first node
    if (dlt->prev != NULL)
```

```c
            dlt->prev->next = dlt->next;


        free(dlt);
        return;
    }
```



```
Select one of the choices:

1: Input from file

2: Input from command

3: Print list

4: Delete list

2

T=2
L=3

In order to finish the program please press 'f'

Requests:
 AB BA CY AB CY XYZ BA XYZ BA
CY,1<-->AB,1
CY,1<-->AB,2
CY,2<-->AB,2
XYZ,1<-->CY,2<-->AB,2
BA,1<-->XYZ,1<-->CY,2
BA,1<-->XYZ,2<-->CY,2
BA,2<-->XYZ,2<-->CY,2
```



```
Select one of the choices:

1: Input from file

2: Input from command

3: Print list

4: Delete list

2

T=3
L=4

In order to finish the program please press 'f'

Requests:
 A B A AA BBB B A AB A B A BB
List not found.AA,1<-->A,2
BBB,1<-->AA,1<-->A,2
B,1<-->BBB,1<-->AA,1<-->A,2
B,1<-->BBB,1<-->AA,1<-->A,3
AB,1<-->B,1<-->BBB,1<-->AA,1
A,1<-->AB,1<-->B,1<-->BBB,1
A,1<-->AB,1<-->B,2<-->BBB,1
A,2<-->AB,1<-->B,2<-->BBB,1
BB,1<-->A,2<-->AB,1<-->B,2
```