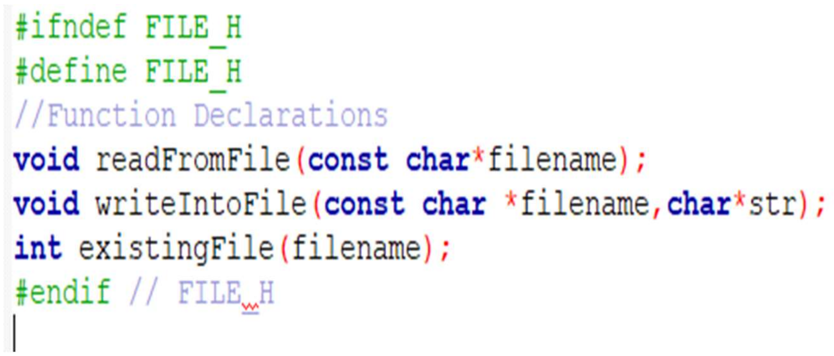
LAB 06

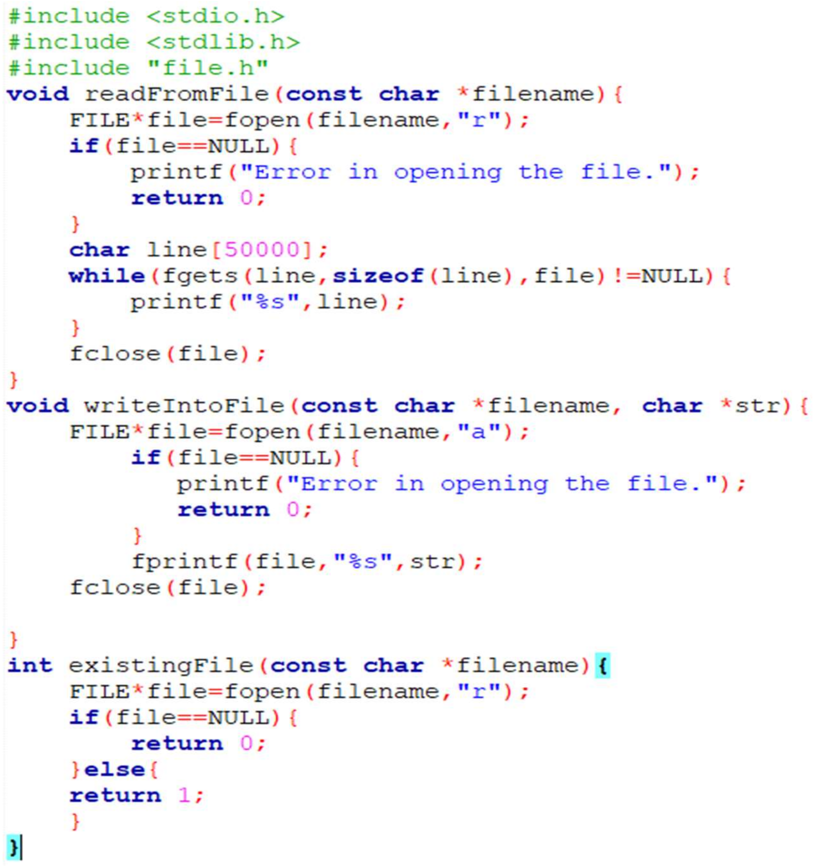
QUESTION 01:

Header file:



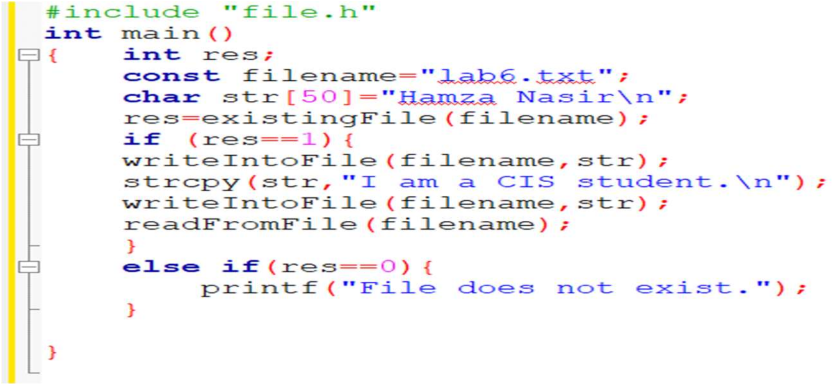
```
#ifndef FILE_H
#define FILE_H
//Function Declarations
void readFromFile(const char*filename);
void writeIntoFile(const char *filename,char*str);
int existingFile(filename);
#endif // FILE_H
```

C File:



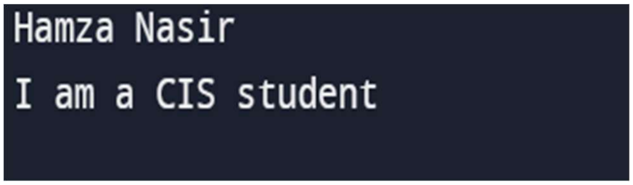
```
#include <stdio.h>
#include <stdlib.h>
#include "file.h"
void readFromFile(const char *filename){
    FILE*file=fopen(filename,"r");
    if(file==NULL){
        printf("Error in opening the file.");
        return 0;
    }
    char line[50000];
    while(fgets(line,sizeof(line),file)!=NULL){
        printf("%s",line);
    }
    fclose(file);
}
void writeIntoFile(const char *filename, char *str){
    FILE*file=fopen(filename,"a");
        if(file==NULL){
           printf("Error in opening the file.");
           return 0;
        }
        fprintf(file,"%s",str);
    fclose(file);

}
int existingFile(const char *filename){
    FILE*file=fopen(filename,"r");
    if(file==NULL){
        return 0;
    }else{
    return 1;
    }
}
```

Main file:

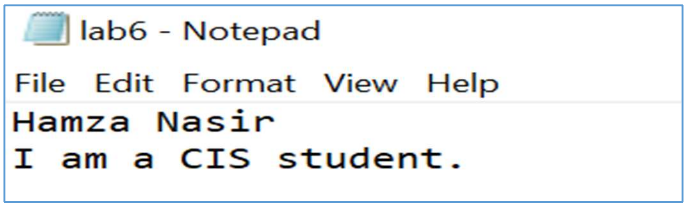


```
#include "file.h"
int main()
{     int res;
      const filename="lab6.txt";
      char str[50]="Hamza Nasir\n";
      res=existingFile(filename);
      if  (res==1){
      writeIntoFile(filename,str);
      strcpy(str,"I am a CIS student.\n");
      writeIntoFile(filename,str);
      readFromFile(filename);
      }
      else if(res==0){
          printf("File does not exist.");
      }

}
```

OUTPUT:



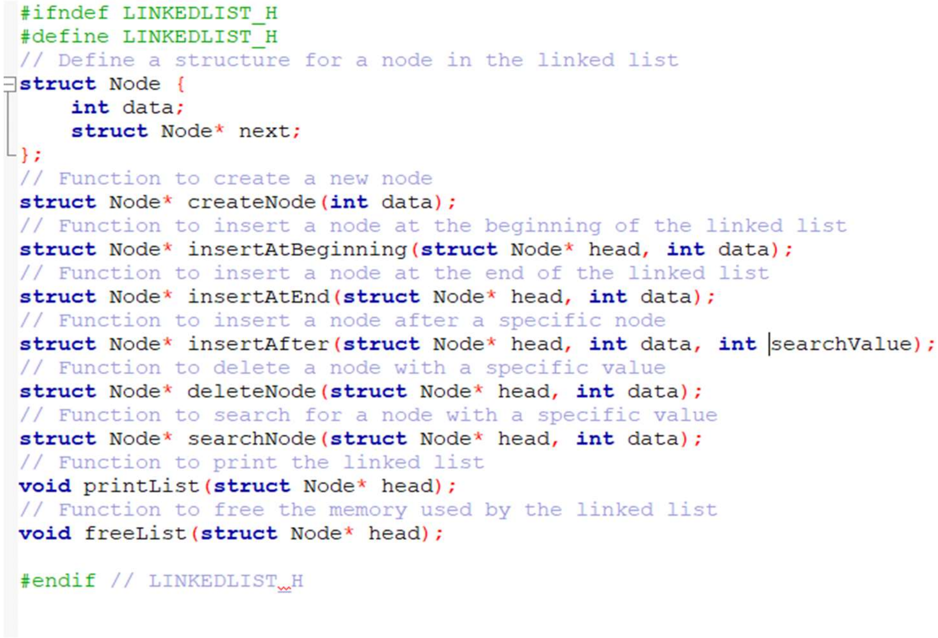
```
Hamza Nasir

I am a CIS student
```



```
Hamza Nasir
I am a CIS student.
```

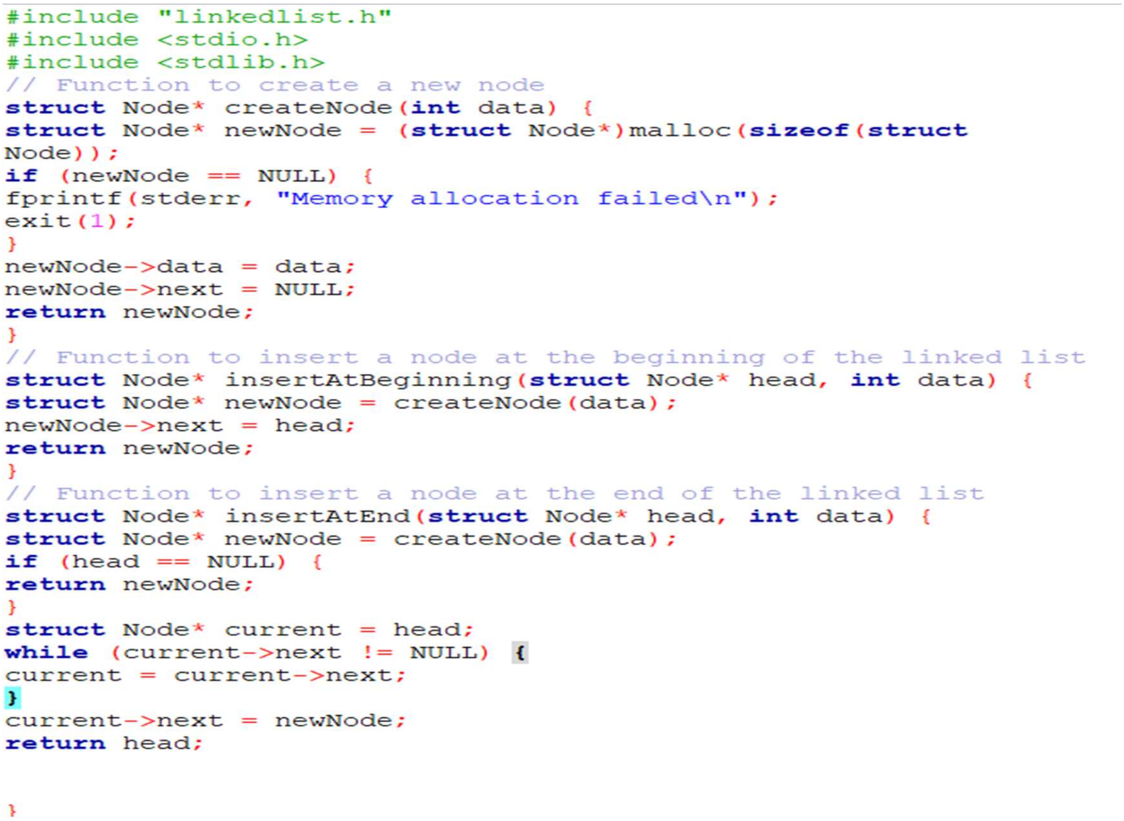
QUESTION 02:

Header file:

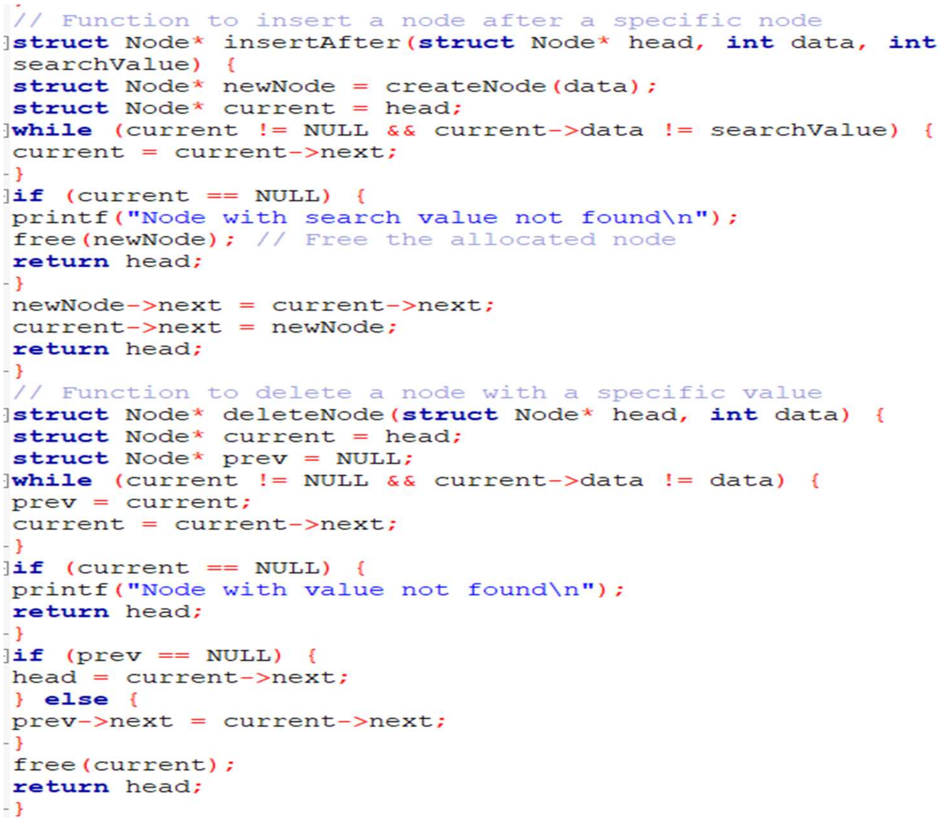


```
#ifndef LINKEDLIST_H
#define LINKEDLIST_H
// Define a structure for a node in the linked list
struct Node {
    int data;
    struct Node* next;
};
// Function to create a new node
struct Node* createNode(int data);
// Function to insert a node at the beginning of the linked list
struct Node* insertAtBeginning(struct Node* head, int data);
// Function to insert a node at the end of the linked list
struct Node* insertAtEnd(struct Node* head, int data);
// Function to insert a node after a specific node
struct Node* insertAfter(struct Node* head, int data, int searchValue);
// Function to delete a node with a specific value
struct Node* deleteNode(struct Node* head, int data);
// Function to search for a node with a specific value
struct Node* searchNode(struct Node* head, int data);
// Function to print the linked list
void printList(struct Node* head);
// Function to free the memory used by the linked list
void freeList(struct Node* head);

#endif // LINKEDLIST_H
```

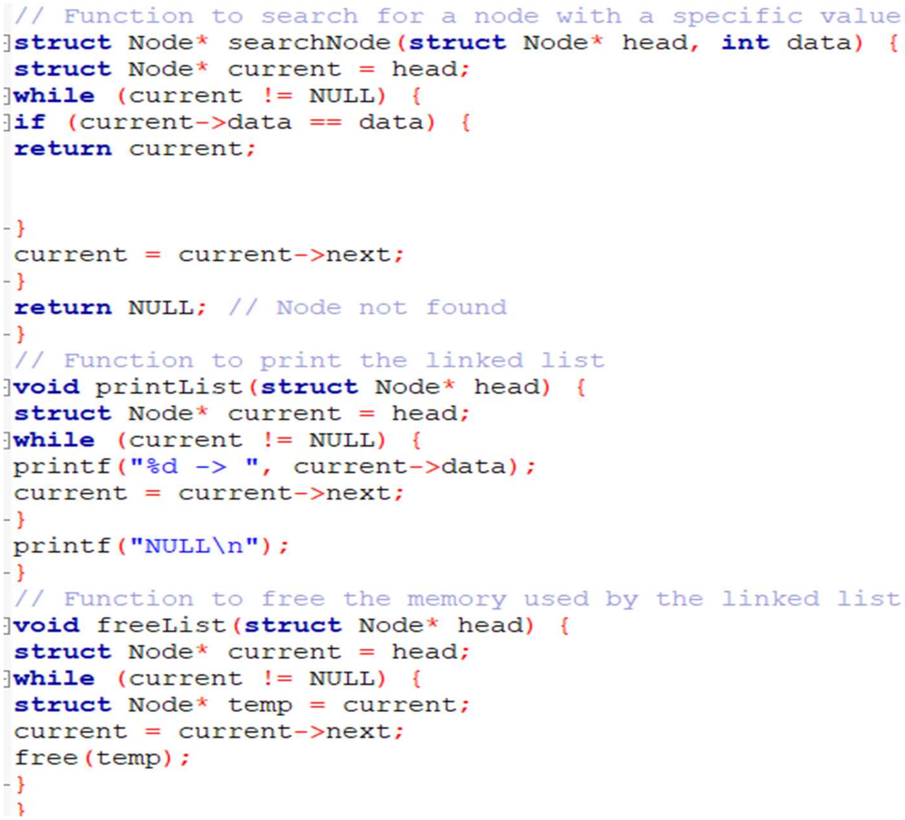
C file:



```
#include "linkedlist.h"
#include <stdio.h>
#include <stdlib.h>
// Function to create a new node
struct Node* createNode(int data) {
struct Node* newNode = (struct Node*)malloc(sizeof(struct
Node));
if (newNode == NULL) {
fprintf(stderr, "Memory allocation failed\n");
exit(1);
}
newNode->data = data;
newNode->next = NULL;
return newNode;
}
// Function to insert a node at the beginning of the linked list
struct Node* insertAtBeginning(struct Node* head, int data) {
struct Node* newNode = createNode(data);
newNode->next = head;
return newNode;
}
// Function to insert a node at the end of the linked list
struct Node* insertAtEnd(struct Node* head, int data) {
struct Node* newNode = createNode(data);
if (head == NULL) {
return newNode;
}
struct Node* current = head;
while (current->next != NULL) {
current = current->next;
}
current->next = newNode;
return head;


}
```

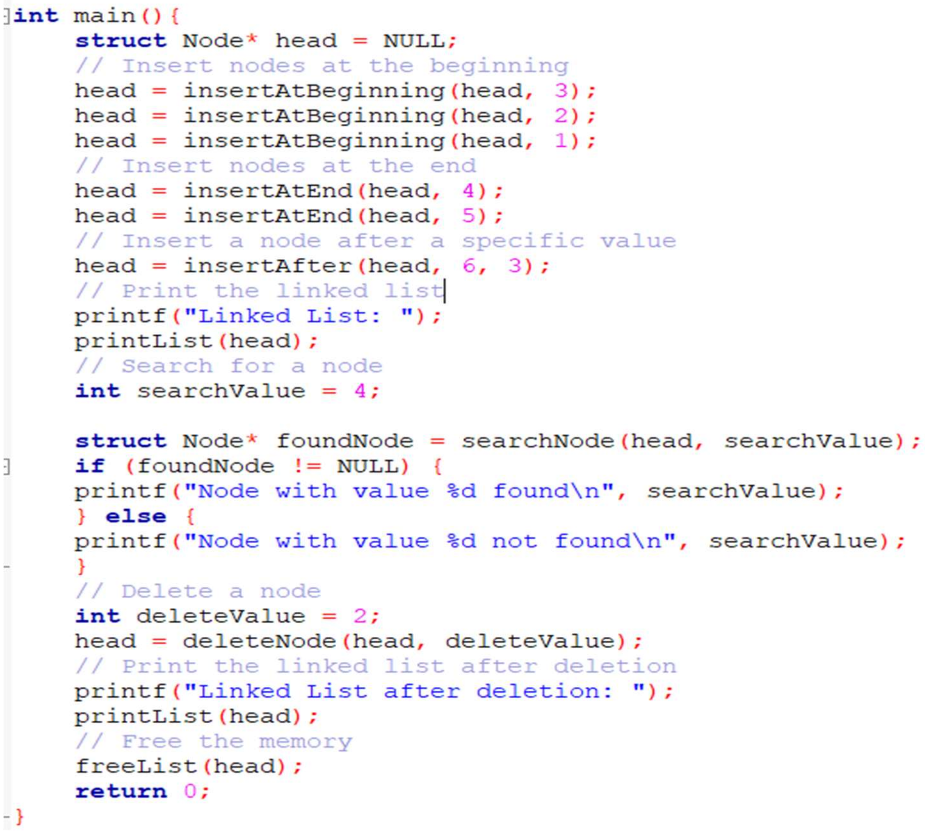


```
// Function to insert a node after a specific node
struct Node* insertAfter(struct Node* head, int data, int
searchValue) {
struct Node* newNode = createNode(data);
struct Node* current = head;
while (current != NULL && current->data != searchValue) {
current = current->next;
}
if (current == NULL) {
printf("Node with search value not found\n");
free(newNode); // Free the allocated node
return head;
}
newNode->next = current->next;
current->next = newNode;
return head;
}
// Function to delete a node with a specific value
struct Node* deleteNode(struct Node* head, int data) {
struct Node* current = head;
struct Node* prev = NULL;
while (current != NULL && current->data != data) {
prev = current;
current = current->next;
}
if (current == NULL) {
printf("Node with value not found\n");
return head;
}
if (prev == NULL) {
head = current->next;
} else {
prev->next = current->next;
}
free(current);
return head;
}
```



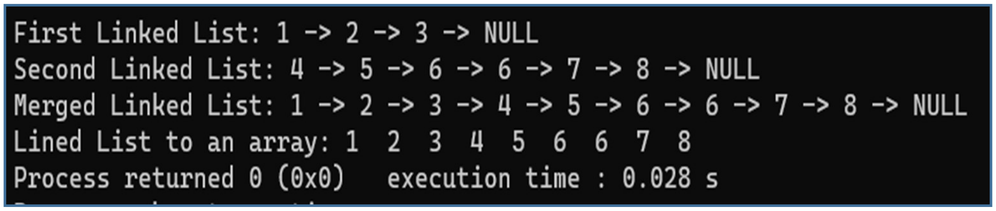
```
// Function to search for a node with a specific value
struct Node* searchNode(struct Node* head, int data) {
struct Node* current = head;
while (current != NULL) {
if (current->data == data) {
return current;


}
current = current->next;
}
return NULL; // Node not found
}
// Function to print the linked list
void printList(struct Node* head) {
struct Node* current = head;
while (current != NULL) {
printf("%d -> ", current->data);
current = current->next;
}
printf("NULL\n");
}
// Function to free the memory used by the linked list
void freeList(struct Node* head) {
struct Node* current = head;
while (current != NULL) {
struct Node* temp = current;
current = current->next;
free(temp);
}
}
```

<u>Main file:</u>



```
int main(){
    struct Node* head = NULL;
    // Insert nodes at the beginning
    head = insertAtBeginning(head, 3);
    head = insertAtBeginning(head, 2);
    head = insertAtBeginning(head, 1);
    // Insert nodes at the end
    head = insertAtEnd(head, 4);
    head = insertAtEnd(head, 5);
    // Insert a node after a specific value
    head = insertAfter(head, 6, 3);
    // Print the linked list
    printf("Linked List: ");
    printList(head);
    // Search for a node
    int searchValue = 4;

    struct Node* foundNode = searchNode(head, searchValue);
    if (foundNode != NULL) {
    printf("Node with value %d found\n", searchValue);
    } else {
    printf("Node with value %d not found\n", searchValue);
    }
    // Delete a node
    int deleteValue = 2;
    head = deleteNode(head, deleteValue);
    // Print the linked list after deletion
    printf("Linked List after deletion: ");
    printList(head);
    // Free the memory
    freeList(head);
    return 0;
}
```

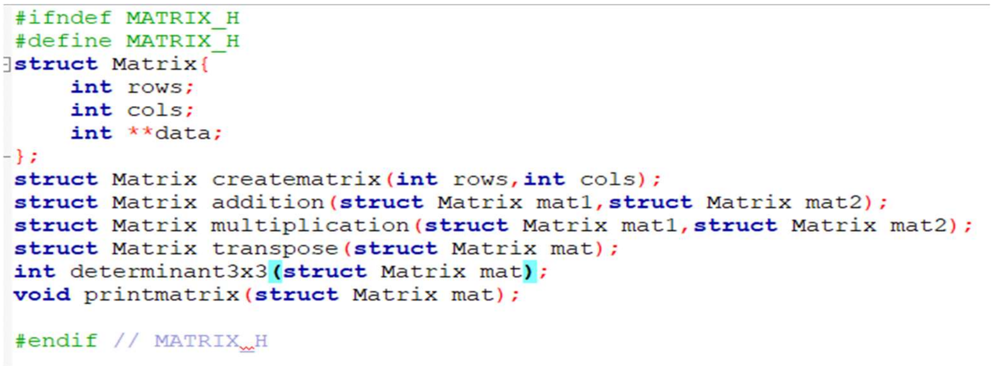
OUTPUT:



```
First Linked List: 1 -> 2 -> 3 -> NULL
Second Linked List: 4 -> 5 -> 6 -> 6 -> 7 -> 8 -> NULL
Merged Linked List: 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 6 -> 7 -> 8 -> NULL
Lined List to an array: 1  2  3  4  5  6  6  7  8
Process returned 0 (0x0)   execution time : 0.028 s
```

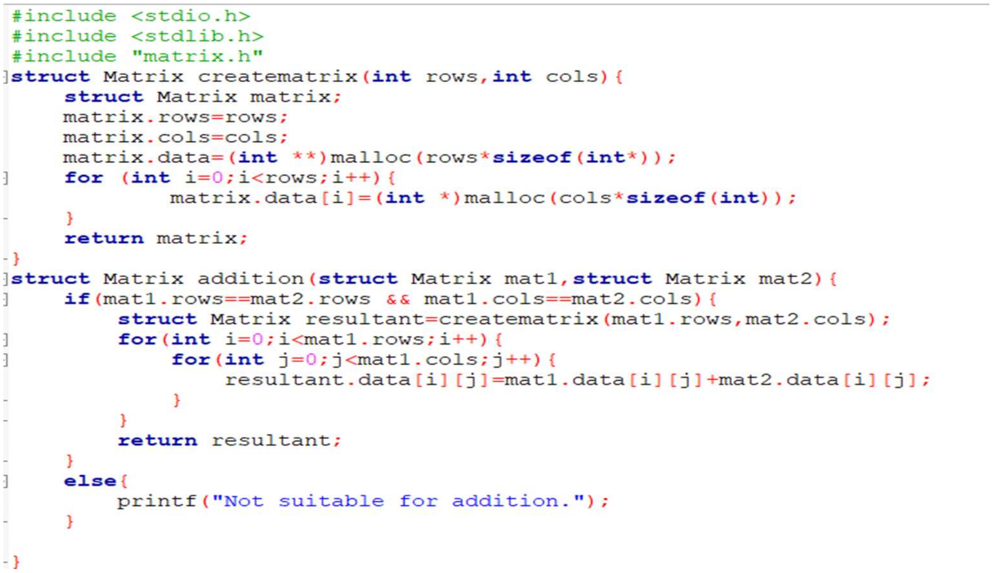
QUESTION 03:

Header file:

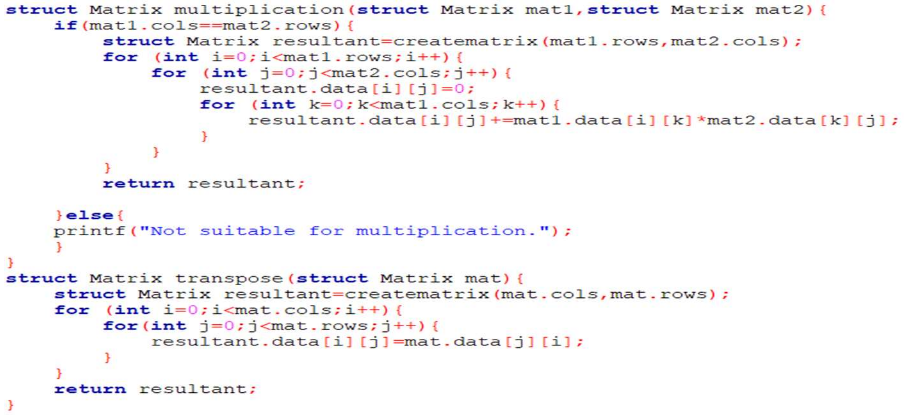


```c
#ifndef MATRIX_H
#define MATRIX_H
struct Matrix{
    int rows;
    int cols;
    int **data;
};
struct Matrix creatematrix(int rows,int cols);
struct Matrix addition(struct Matrix mat1,struct Matrix mat2);
struct Matrix multiplication(struct Matrix mat1,struct Matrix mat2);
struct Matrix transpose(struct Matrix mat);
int determinant3x3(struct Matrix mat);
void printmatrix(struct Matrix mat);

#endif // MATRIX_H
```

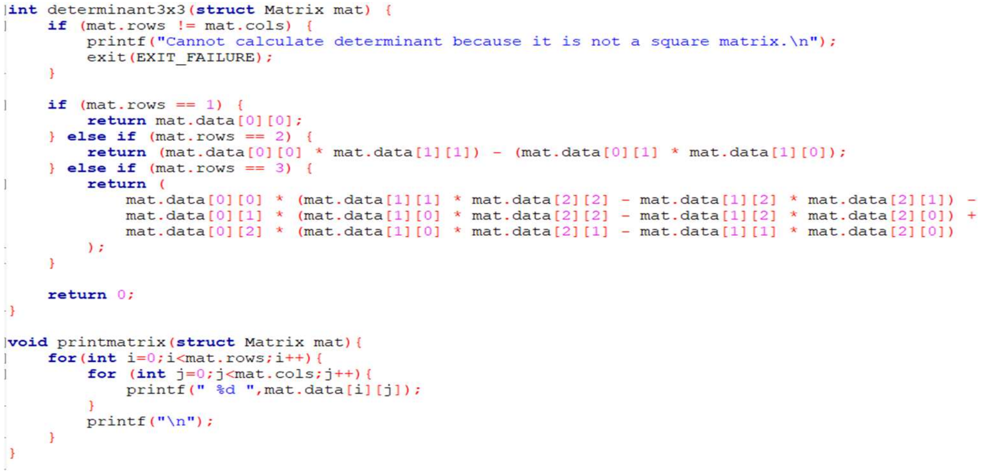
C File:



```c
#include <stdio.h>
#include <stdlib.h>
#include "matrix.h"
struct Matrix creatematrix(int rows,int cols){
    struct Matrix matrix;
    matrix.rows=rows;
    matrix.cols=cols;
    matrix.data=(int **)malloc(rows*sizeof(int*));
    for (int i=0;i<rows;i++){
            matrix.data[i]=(int *)malloc(cols*sizeof(int));
    }
    return matrix;
}
struct Matrix addition(struct Matrix mat1,struct Matrix mat2){
    if(mat1.rows==mat2.rows && mat1.cols==mat2.cols){
        struct Matrix resultant=creatematrix(mat1.rows,mat2.cols);
        for(int i=0;i<mat1.rows;i++){
            for(int j=0;j<mat1.cols;j++){
                resultant.data[i][j]=mat1.data[i][j]+mat2.data[i][j];
            }
        }
        return resultant;
    }
    else{
        printf("Not suitable for addition.");
    }

}
```

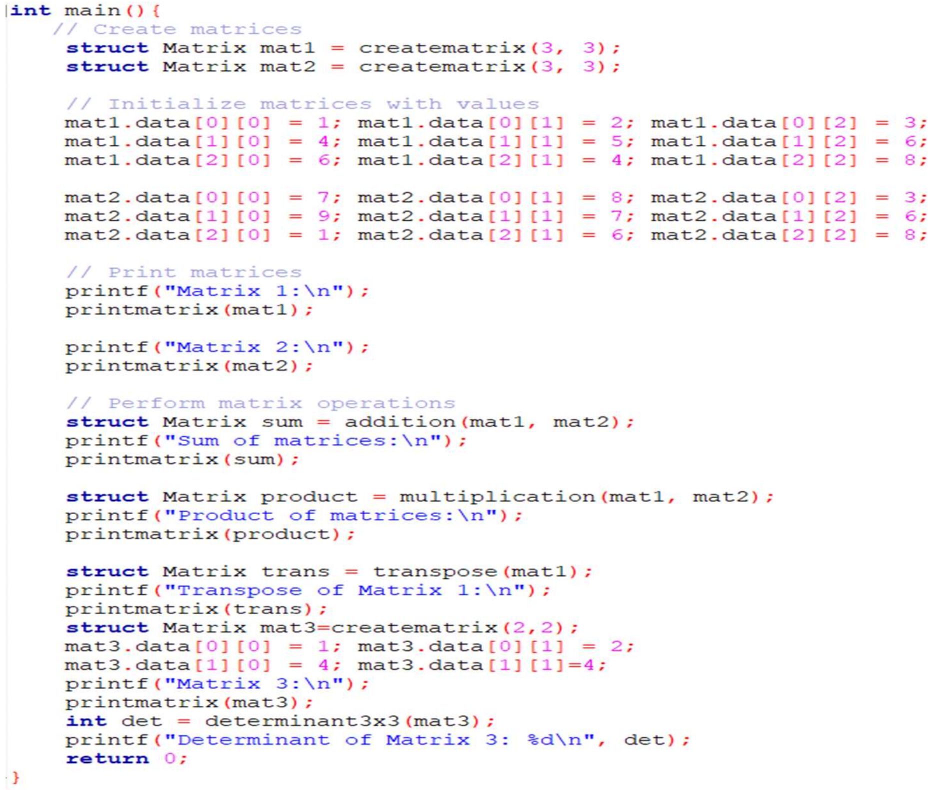


```
struct Matrix multiplication(struct Matrix mat1,struct Matrix mat2){
    if(mat1.cols==mat2.rows){
        struct Matrix resultant=creatematrix(mat1.rows,mat2.cols);
        for (int i=0;i<mat1.rows;i++){
            for (int j=0;j<mat2.cols;j++){
                resultant.data[i][j]=0;
                for (int k=0;k<mat1.cols;k++){
                    resultant.data[i][j]+=mat1.data[i][k]*mat2.data[k][j];
                }
            }
        }
        return resultant;

    }else{
    printf("Not suitable for multiplication.");
    }
}
struct Matrix transpose(struct Matrix mat){
    struct Matrix resultant=creatematrix(mat.cols,mat.rows);
    for (int i=0;i<mat.cols;i++){
        for(int j=0;j<mat.rows;j++){
            resultant.data[i][j]=mat.data[j][i];
        }
    }
    return resultant;
}
```



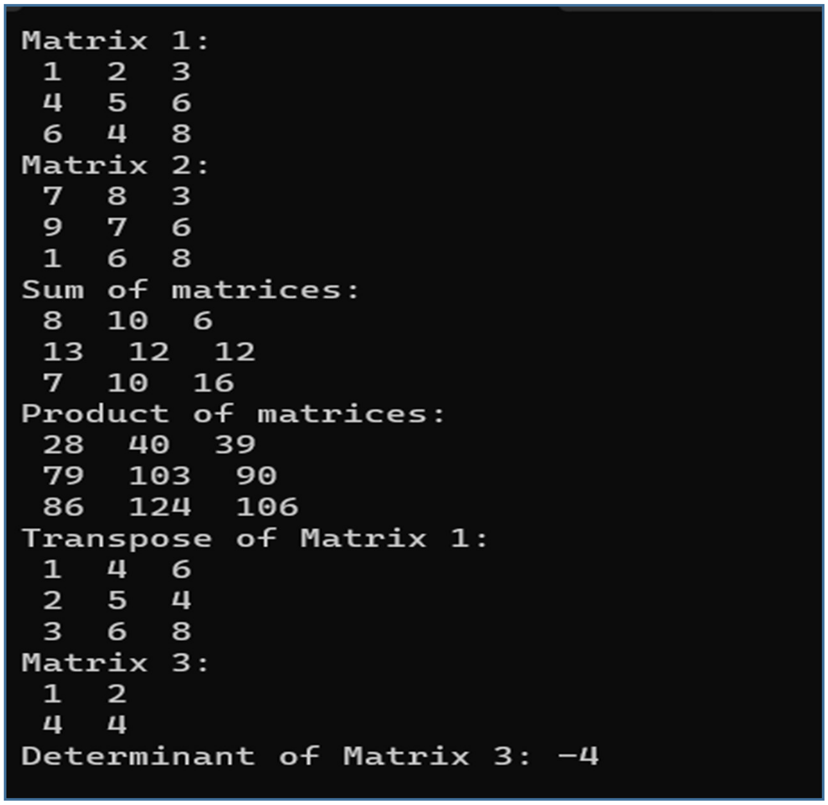
```
int determinant3x3(struct Matrix mat) {
    if (mat.rows != mat.cols) {
        printf("Cannot calculate determinant because it is not a square matrix.\n");
        exit(EXIT_FAILURE);
    }

    if (mat.rows == 1) {
        return mat.data[0][0];
    } else if (mat.rows == 2) {
        return (mat.data[0][0] * mat.data[1][1]) - (mat.data[0][1] * mat.data[1][0]);
    } else if (mat.rows == 3) {
        return (
            mat.data[0][0] * (mat.data[1][1] * mat.data[2][2] - mat.data[1][2] * mat.data[2][1]) -
            mat.data[0][1] * (mat.data[1][0] * mat.data[2][2] - mat.data[1][2] * mat.data[2][0]) +
            mat.data[0][2] * (mat.data[1][0] * mat.data[2][1] - mat.data[1][1] * mat.data[2][0])
        );
    }

    return 0;
}

void printmatrix(struct Matrix mat){
    for(int i=0;i<mat.rows;i++){
        for (int j=0;j<mat.cols;j++){
            printf(" %d ",mat.data[i][j]);
        }
        printf("\n");
    }
}
```

<u>Main file:</u>



```
int main(){
   // Create matrices
    struct Matrix mat1 = creatematrix(3, 3);
    struct Matrix mat2 = creatematrix(3, 3);

    // Initialize matrices with values
    mat1.data[0][0] = 1; mat1.data[0][1] = 2; mat1.data[0][2] = 3;
    mat1.data[1][0] = 4; mat1.data[1][1] = 5; mat1.data[1][2] = 6;
    mat1.data[2][0] = 6; mat1.data[2][1] = 4; mat1.data[2][2] = 8;

    mat2.data[0][0] = 7; mat2.data[0][1] = 8; mat2.data[0][2] = 3;
    mat2.data[1][0] = 9; mat2.data[1][1] = 7; mat2.data[1][2] = 6;
    mat2.data[2][0] = 1; mat2.data[2][1] = 6; mat2.data[2][2] = 8;

    // Print matrices
    printf("Matrix 1:\n");
    printmatrix(mat1);

    printf("Matrix 2:\n");
    printmatrix(mat2);

    // Perform matrix operations
    struct Matrix sum = addition(mat1, mat2);
    printf("Sum of matrices:\n");
    printmatrix(sum);

    struct Matrix product = multiplication(mat1, mat2);
    printf("Product of matrices:\n");
    printmatrix(product);

    struct Matrix trans = transpose(mat1);
    printf("Transpose of Matrix 1:\n");
    printmatrix(trans);
    struct Matrix mat3=creatematrix(2,2);
    mat3.data[0][0] = 1; mat3.data[0][1] = 2;
    mat3.data[1][0] = 4; mat3.data[1][1]=4;
    printf("Matrix 3:\n");
    printmatrix(mat3);
    int det = determinant3x3(mat3);
    printf("Determinant of Matrix 3: %d\n", det);
    return 0;
}
```

OUTPUT:



```
Matrix 1:
 1  2  3
 4  5  6
 6  4  8
Matrix 2:
 7  8  3
 9  7  6
 1  6  8
Sum of matrices:
 8  10  6
 13  12  12
 7  10  16
Product of matrices:
 28  40  39
 79  103  90
 86  124  106
Transpose of Matrix 1:
 1  4  6
 2  5  4
 3  6  8
Matrix 3:
 1  2
 4  4
Determinant of Matrix 3: -4
```